UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Robert M. Lake

TITLE OF THESIS: Dynamic Motion Control of an Articulated Figure

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1990

(Signed) .....
Permanent Address:
11610 111 Avenue
Edmonton, Alberta
Canada T5G 0E1

Dated 20 April 1990

University of Alberta

DYNAMIC MOTION CONTROL OF AN ARTICULATED FIGURE

by

Robert M. Lake

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring, 1990

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Dynamic Motion Control of an Articulated Figure** submitted by **Robert M. Lake** in partial fulfillment of the requirements for the degree of **Master of Science.**

.....

Supervisor

.....

.....

.....

Date

ABSTRACT

Dynamics is becoming an increasingly popular method for producing realistic animation. While the motion of simple objects such as blocks and spheres is easily controlled using this technique, applying dynamics to articulated figures such as humans presents two major difficulties. The first is specifying the motion in a language familiar to the animator. Animators want to work with a natural motion language rather than directly entering the forces and torques required for a particular motion. The onus should be on the animation system to translate high-level motion commands into the necessary forces and torques. The second issue involves controlling the figure's motion. What magnitude and combination of force and torque is required to execute a specific motion within a fixed time interval?

This thesis focuses on issues related to articulated figure animation and motion control. A new animation system, designed to dynamically control articulated figure motion by performing interpolations along quaternion curves, is presented. The system uses ballroom dancing as an example. Motion sequences are entered using an easy to learn, high-level, ballroom dance notation language. The system decomposes these motion sequences into series of primitive movements and activates a motion control model to execute each movement within an animator-specified time interval. A description of the model structure is provided along with results from a set of animation experiments.

Acknowledgements

I extend my appreciation and thanks to the following:

My supervisor, Dr. Mark Green, for his support and encouragement throughout the course of this work. The members of my committee - Dr. Barry Joe, Dr. Hong Zhang, and Dr. Jacques Bobet for their critical evaluation of this thesis. Dr. Bill Armstrong for his assistance with the dynamics package. Ken Hruday for reading an earlier draft of this thesis and offering numerous comments and improvements. Cindy Webber for her assistance with defining dance positions. The Department of Computing Science for providing me with the time to complete this work. The University of Alberta and the University of Alberta Dance Club for the technical and artistic knowledge incorporated in this thesis. My family for their support and encouragement. Lastly, Chris Shaw for exploring quaternions, discussing how we move, video assistance, and proofreading. Chris also inducted me as an honorary member of the Canadian Institute for Advanced Bogosity Research (CIABR), surely a profound tribute a person can receive only once in a lifetime.

Table of Contents

List of Figures

Chapter 1

Introduction

Computer graphics and mathematics can be used to accurately model the behavior, shape, and texture of objects. Over the years, computer graphics has advanced from simple pen plotter drawings to sophisticated images containing a multitude of colors displayed on high quality raster devices. The introduction of better, faster hardware has enabled the complexity of an image to evolve from scenes composed of a few dozen polygons to modern images constructed from many hundreds of thousands of polygons. The development of sophisticated modeling techniques based on the laws of nature has resulted in surface properties of objects changing from a uniform plastic look to a very realistic appearance with the simulation of optical properties such as reflection, refraction, and transparency. Despite these technological achievements, modern computer-generated animation still lacks one important feature - the lifelike representation of a human figure and its behavior.

The realistic modeling and animation of human figures remains one of the most challenging problems in computer graphics. Although computer graphics has been applied to television animation for a number of years, few commercial productions have attempted to represent human figures and their motion.

There are two fundamental reasons for the lack of realistic human animation. Designing a complete body model and animating it in a natural manner are both difficult tasks. Unlike the underlying bone, human flesh is not rigid. As a figure moves through a sequence of postures, the muscles on the limbs and tissues around the joints change shape in a manner difficult to consistently model. The human body is an immensely complex figure containing over 200 bones and several hundred degrees of freedom. A human figure is capable in moving in such a multitude of ways that scientists are still learning how to define and measure movement. Because human movement is a familiar activity, people are well-trained to distinguish realistic from unnatural movement. Computer-generated animation sequences of human motion must

therefore meet a very high standard to be acceptable.

Specifying human movement in computer animation systems also raises interesting issues. How can a realistic movement sequence be defined using minimal effort and a language familiar to the animator? How should an animation system translate these high-level descriptions into a set of movement instructions? How should these movement instructions be translated into a series of time-related motion sequences? What method should be employed to execute the motion and how can it be controlled?

This thesis focuses on issues related to articulated figure animation and motion control. A new model, designed to produce realistic and controlled human movement based on ballroom dancing, is presented. All motion is generated by modeling the behavior of objects under the influence of forces and torques. Movement sequences are derived from symbols entered using a high-level, easy to learn, ballroom dance notation language.

1.1. Importance of Modeling and Animating Articulated Figures

Computer-generated figure models and animation sequences have many applications in a variety of fields. In medicine and related disciplines, accurate human models can be used to study human anatomy and physiology. In commercial animation, including life-like figures enhances the interest and quality of animation sequences [Lasseter87]. Employing computer systems in commercial animation can help reduce the cost of hand-drawn animation and produce more accurate motion. An animation system modeling any articulated figure motion can be used in biomechanics to study movement and gait patterns [Alexander84b] [McMahon84a]. Many techniques and algorithms used to animate articulated figures can also be applied to the field of robotics [Herbison-Evans84b] [Miura84].

Human figure animation systems have extensive uses in the dance world. Computer-based editors and interpreters can produce animation by reading and translating into visual form a movement description based on one or more notation languages [Dransch86] [Sealey80]. Systems like these can then be used to train dance annotators to record movement using these notations [McNair82]. Choreographers can com-

pose and edit new sequences before hiring performers, thus saving production costs [Herbison-Evans85]. In conjunction with video, previous dance masterpieces can be viewed, archived, and stored for future reference. Dance can also provide computing with an artistic flavor [Herbison-Evans82].

A human figure animation system based on forces and torques has a variety of applications. In sports, feasibility studies of dangerous activities (for example, a dive off a cliff) can be done without risking injury. Simulations can be used to explore new methods to maximize performance while avoiding injury [McMahon84a]. Ergonomic analysis can benefit from experiments on the interaction between a human figure and a simulated working environment [Dooley82]. Evaluation studies can be performed on how a human figure reacts to forces resulting from automobile accidents [Willmert82]. In ballroom dancing, these systems can be used to explore interaction between dance partners and illustrate the principles of lead, follow, and frame [Green90].

1.2. Methods for Animating Articulated Figures

Three-dimensional articulated figure animation systems require specifying the figure's support position and orientation of all body segments during each frame. Three translational terms describe the support position and three rotational terms describe the orientation angle of body segments. A common procedure is to let one segment define the figure's support position and have the orientation angles of all other segments defined relative to the orientation of adjoining segments. Thus, each segment has three rotational degrees of freedom and the segment defining the figure's support position has an additional three translational degrees of freedom. Many different animation techniques exist for determining these positions and orientations.

Two methods involve measuring the body position and limb angles from a live human subject. *Rotoscoping* is the digitization of the joint coordinates from a film or video recording. *Instrumentation* is the recording of a motion sequence using special instrumentation (for example, a set of goniometers) attached to the subject. Many recent commercial animation sequences have been produced using one or both of these techniques. However, both methods have serious deficiencies. Motion recording

instrumentation attached to a subject often inhibits movement. Animation obtained from these techniques is usually limited to the sequences generated by the human figure acting under the imposed conditions. Changing the set of environmental conditions (for example, replacing a sidewalk with a skating rink) likely requires re-recording the subject's motion under the new conditions.

Key-frame animation systems simulate motion by having the animator create an ordered set of *key frames* describing the appearance and position of one or more objects at specific times [Reeves81] [Sturman84]. Interpolation algorithms read the key frames and create a series of *inbetween frames* containing object positions required to link successive key frames. Motion results when all key frames and inbetween frames are rapidly played back in sequential order.

While key framing can be applied to articulated figure animation, several serious problems exist with these systems [Catmull78]. Human figures can perform motion not easily represented by the linear and cubic spline interpolation methods commonly used by these systems. This requires either the development of more sophisticated interpolation methods, or that the user hand-specify the motion. Two-dimensional systems have difficulty generating correct inbetween frames because key frame information is lost when three dimensional scenes are transformed into two dimensional images. Lengthy animation sequences can consume vast amounts of computer resources because key frame systems often require large amounts of space for frame storage.

Kinematic systems describe motion by the positions, velocities, and accelerations occurring at each joint of the articulated figure [Calvert82] [Wilhelms85]. These systems do not account for the forces and torques responsible for producing the motion. Motion is obtained either by showing the position of the figure through a series of key and inbetween frames, or by solving the kinematic equations of motion for each degree of freedom. Unlike key framing, kinematically calculated positions work well in three dimensions.

Most articulated figure animation systems are kinematic systems. While these systems produce realistic animation sequences, kinematic systems have several disadvantages. Although kinematic motion specification is fairly efficient and easy to implement, it is awkward to use with complex figures containing

many degrees of freedom. Correct and realistic motion descriptions are often difficult to produce. Even if the figure moves in accordance to a realistic kinematic motion specification, the figure cannot react to sudden, unexpected forces such as those resulting from a shove.

Dynamic systems produce motion by accounting for forces (responsible for translational motion) and torques (responsible for rotational motion) acting on each body segment [Armstrong87] [Wilhelms85]. Assuming quantities such as the mass and rotational inertia of each body segment are known, physics and mechanical engineering can be applied to model the resulting acceleration. The acceleration can then be integrated twice to obtain the velocity and position of the body segment. In the case of articulated figures, the dynamic equations describing motion for each degree of freedom are usually complex second order differential equations which are solved numerically.

The major advantage to dynamic analysis is its ability to correctly predict motion based on mechanical principles. Dynamic analysis can accurately predict motion in other environments such as the moon, whereas a realistic kinematic description may be impossible to obtain. Dynamic analysis can also be used to explore interactions between one or more objects. Until recently, dynamic analysis was much more computationally expensive than kinematic analysis. However, efficient solution techniques now exist for producing near real time dynamic motion specifications [Armstrong85a] [Armstrong85b].

The main disadvantage with dynamic analysis is that motion must be specified in terms of forces and torques. While some forces (such as gravity) are obvious, most people do not have an intuitive feel for the magnitude of the forces and torques required to produce a particular motion. Despite this, a growing number of computer scientists believe dynamic analysis is the proper method to use for animation purposes.

Several other methods are used for animating articulated figures. *Notation systems* usually consist of two components. Movement commands are entered using an *editor* designed for a particular notation language [Brown76] [Dransch86] [Singh83]. These commands are then parsed by an *interpreter* which produces the desired motion [Herbison-Evans86] [Politis85] [Wolofsky74]. *Goal directed* models define a sequence of goal positions for the figure [Badler87]. The animation system attempts to sequentially orient

the figure to each goal position. *Knowledge base* systems apply artificial intelligence theory to determine motion sequences from definitions and relationships stored in object and motion knowledge bases [Drewery86] [Ridsdale86]. The description of motion with each model uses a higher level of abstraction than the level used by the methods previously introduced. However, all motion produced by these models is output using either a kinematic or a dynamic specification.

1.3. Thesis Overview

The remainder of this thesis describes a hierarchical and controllable dynamically-based human figure animation system. This system, hereafter referred to as the *Ballroom Dance Animation System* (or *BDAS*), uses a hierarchical motion control model to generate ballroom dance animation from a high-level movement notation language.

Chapter 2 highlights previous work done in animating articulated body motion and outlines the goals of this thesis. Since the construction of *BDAS* relies on concepts from other fields, Chapter 3 provides a survey of the relevant material.

The remaining chapters describe the structure of the animation model and discuss the results of experiments performed with the system. Chapter 4 presents a recently proposed hierarchical motion control model incorporated into *BDAS*. Chapter 5 introduces the movement notation language supported by *BDAS* and describes the system's main components. Chapter 6 discusses the motor control model applied to control the figure's motion. Chapter 7 presents the experimental results and other observations. Finally, Chapter 8 summarizes the work and outlines directions for future research.

Chapter 2

Articulated Figure Animation and Movement Notation Systems

Within the past thirty years, a number of systems have appeared which produce articulated figure animation using a variety of techniques. These systems have had applications ranging from the study of how a human reacts to an environment to issues related to recording human movement performed in artistic productions. Highlights of this work are presented with special emphasis on systems which inspired the design of the animation model introduced in Chapter 5.

2.1. Early Work

Ergonomic analysis (the study of how a human interacts with its environment) provided some of the earliest applications in computer graphics for modeling a human figure and its motion. One of the earliest figures used for ergonomic analysis was William Fetter's Landing Signal Officer (LSO), developed for Boeing in 1959 [Fetter82]. The LSO was a fixed database that gave the position and size of a landing services officer on an aircraft carrier, as viewed from the cockpit of the simulated flying aircraft. It was displayed as a 12-point, 2-dimensional figure. Further studies with 30 point figures simulated passenger movement down the aisles of commercial aircraft.

Fetter developed several additional models [Fetter82]. The seven jointed "First Man", used for studying the instrument panel of a Boeing 747, enabled many pilot motions to be displayed by articulating the figure's pelvis, neck, shoulders, and elbows. Possibly the first use of computer graphics in commercial advertising took place in 1970 when this figure was used for a Norelco television commercial. The addition of twelve extra joints to "First Man" produced "Second Man". This figure was used to generate a set of animation film sequences based on a series of photographs produced by Eadweard Muybridge [Muybridge55]. "Third Man and Woman" was a hierarchical figure series with each figure differing by an order of magnitude in complexity. These figures were used for general ergonomic studies. The most

complex figure had 1000 points and was displayed with lines to represent the contours of the body. In 1977, Fetter produced "Fourth Man and Woman" figures based on data from biostereometric tapes. These figures could be displayed as a series of colored polygons on raster devices.

Cyberman (*Cyber*netic *man*-model) was developed by Chrysler Corporation for modeling human activity in and around a car [Blakeley80]. A 15-link stick figure (with or without a wireframe outline) could be interactively manipulated by the user. Reach attempts were performed either by the operator from one of 36 operator eye locations, or by an associated supplementary program. No provision was made for enforcing realistic positions, and the user was responsible for determining the comfort and feasibility of a position after each operation. Results of each reach attempt were displayed on a vector device.

Combiman (*Com*puterized *bi*omechanical *man*-model) was designed for research in aircraft design and evaluation by the Aerospace Medical Research Laboratory and the University of Dayton Research Institute [Bapu80]. The model consisted of 33 links, with 2 of these providing a seat-based reference point. Reach attempts were performed by the operator specifying and initiating the limb used in the operation. All angular movements were constrained to realistic values during the operation. Although the system indicated success or failure with each reach operation, the operator was required to determine the amount of clearance (or distance remaining to the goal).

Sammie (*S*ystem for *A*iding *M*an *M*achine *I*nteraction *E*valuation) was developed by the University of Nottingham, Nottingham, England for general ergonometric design and analysis [Kingsley81]. The user defined the environment by either building objects from simple primitives, or by defining the vertices and edges of irregular shaped objects. The human model was based on a measurement survey of a general population group. Users could, however, alter the various limb and segment lengths. After attempting a one-step reach, the system displayed the final position of the model and indicated whether or not the reach was successful. If the reach failed, the amount of distance between the object and the model was given. Notification was provided when maximum comfort or the maximum limb angle was obtained. This helped to decide if a position could be achieved or maintained.

Boeman was designed in 1969 by the Boeing Corporation in Seattle, Washington [Dooley82]. The human model was based on data representing a 50th-percentile human model obtained from studies by Dreyfus, Dempster, and Hertzberg. All input data concerning the model and its environment were read from computer cards. Output was displayed on a plotter. Although this program is dated by the hardware it used, the model's database provided a basis for many subsequent anthropometric modeling programs.

Buford was developed at Rockwell International in Downey, California to find reach and clearance areas around a model positioned by the operator [Dooley82]. The figure represented a 50th-percentile human model and was covered by CAD-generated polygons. The user could interactively design the environment and change the body position and limb sizes. However, repositioning the model was done by individually moving the body and limb segments.

2.2. Key Frame Animation Systems

Several key frame animation systems have been developed for representing a human model and animating its movement. These systems have also been used for exploring interpolation and inbetweening techniques.

BBOP, produced at the New York Institute of Technology's Graphics Lab, displayed the figure as a jointed structure with each node represented by a transformation matrix [Sturman84]. Joints (or limbs) could be scaled, translated, or rotated by using function keys to select appropriate parameters which were set through the manipulation of a joystick. All parameter changes were reflected in updates to the screen. Key frames were created by storing values for every joint. This implementation, though simple, required much storage. A motion editor allowed the animator to specify the interpolation method used for key frame inbetweening.

EM, also produced at NYIT, was a more complicated key frame animation system [Sturman84]. This system represented a figure using a structure similar to the one used by *BBOP*. Unlike *BBOP*, *EM* used a geometric modeling language to define parameters controlling the joint transformations and shape of the

body parts. The model's motion could be influenced by coordinating these parameters with respect to constants and other parameters. Interaction modes were configured according to each parameter and joint in the tree structure. Although this made animating a model more difficult, the system could display more complicated motions.

Other key frame animation systems explored different methods of inbetweening and object representation. Ron Baecker's 1969 Ph.D. thesis described efforts to provide more animator control by using P-curves for inbetweening [Baecker69]. William Reeves controlled inbetween motion using moving point constraints [Reeves81]. Craig Reynolds' *ASAS* (*A*ctor/*S*criptor *A*nimation *S*ystem) was an extension of LISP and could be used for both object definition and action specification [Reynolds82].

2.3. Kinematic and Dynamic Systems

Kinematic systems produce articulated figure motion through the specification of velocities and accelerations acting on the figure's limbs. Position is then calculated as a function of time. Dynamic systems take this one step further by applying forces and torques to the figure. Equations are then solved to obtain the acceleration, velocity, and position of the figure. During the past decade, a number of systems based on these methods have been used to produce articulated figure animation.

2.3.1. A Goal-Directed Finite State Machine

In 1982, David Zeltzer and Charles Csuri of Ohio State University developed an extensible and general model whereby the animator could define complex articulated objects and kinematically simulate their motion [Zeltzer82b] [Zeltzer82a]. The eventual goal of the project was to have a goal-directed, extensible, and interactive system capable of producing animation of real or imaginary objects in close to real time. General motion commands input by the user were parsed by the system into many primitive, sequential movements which would then be applied to the figure.

Articulated figures were represented as skeletons of arbitrary complexity. Skeletons were defined using a language based on a context free grammar. Every skeletal description contained a *declarations* and

a *description* block. The declarations block assigned a name to each joint and set any rotational constraints. As an example, the declarations block for a jaw was:

jaw: x -60 0 y -10 10

Two structures were used in the description block. *Limb* structures defined a sequence of connected segments by listing the names of the segments in a string enclosed by round parentheses. *Compound* structures defined joints where two or more segments met. These structures were enclosed by begin/end blocks. The first string following the *begin* statement was the name of the joint and all remaining entries up to the *end* statement were interpreted as dependent joints. For example, the following structure defined a wrist as a compound joint with five limbs and each limb having two or more joints:

```
                    begin
    wrist
        (thumb1 thumb2)
        (index1 index2 index3)
        (middle1 middle2 middle3)
        (ring1 ring2 ring3)
        (little1 little2 little3)
end
```

The parser generated a binary transformation tree and a symbol table. The tree represented the skeleton and the symbol table held information about each joint. These structures were subsequently used by other components of the motion simulation system.

The motion animation system was a hierarchy of three levels modeled after synergic control systems of animals. Kinematic studies of human and mechanical motion were applied to the decomposition of movement commands. The *task manager* accepted commands from the user and parsed them into a series of component skills. These skills were placed into a *movement queue* for subsequent processing by the middle level. Every skill represented a type of motion (such as walking, grasping, and jumping) the figure could perform.

The middle and lower levels were implemented as finite state machines. Skills were executed at the middle level by a series of motor programs. Every motor program handled a particular type of motion by executing a fixed set of low-level local motor programs (LMPs). The LMPs were procedures which per-

formed the actual motion by modifying the rotational angles of the joints represented in the skeletal database. Each LMP handled a specific movement primitive. Motion synchronization was done by having the LMP return a feedback signal to the controlling program. At the end of each time frame, the graphics display was updated to reflect the changed state of the skeletal database. All motion within the model was kinematically produced.

As an example, a command requiring a 'walk' had the walk motor program placed on the movement queue by the task manager. Each time a walk was performed a set of eight LMPs were executed. These LMPs consisted of four LMPs responsible for regulating the left swing, left stance, right swing, and right stance of the hips and leg, and four LMPs responsible for regulating the swinging of the arms.

Most of the work on the system was focused on the lower levels. By having the user input component skills directly to the task manager, interesting animation sequences of a skeletal figure walking, somersaulting, jumping, and running were produced.

2.3.2. A Low-Level Kinematic/Dynamic System

In 1985, Jane Wilhelms of the University of California, Berkeley published a thesis describing an articulated figure animation system *Deva* which could produce both kinematic and dynamic specified motion [Wilhelms85] [Wilhelms86] [Wilhelms87].

Deva was a low-level system where the animator defined articulated figure motion by creating a set of motion control functions responsible for the motion of each degree of freedom. These functions represented either the position and orientation, or the magnitude of the applied force or torque, as a function of time.

Deva consisted of two components. Dynamic and kinematic calculations and all output to the display device were handled within the *Deva* animation system. Control functions governing the motion of the body were created and modified with the motion control editor *Virya*.

The animator defined the articulated figure structure from a language based on the body grammar

proposed by Zeltzer [Zeltzer82b]. The system then produced an ASCII file containing information about the names of the body segments, the degrees of freedom available to all joints in the figure, and the connectivity between the body segments and joints. Joints could either be revolute or sliding. Joints with multiple degrees of freedom were treated as a series of joints with a single degree of freedom. The notation could describe arbitrary articulated figures representable by a tree-like structure. Figures generally had no more than 12 limbs or 36 degrees of freedom due to *Virya* limitations and the computational overhead of the dynamic analysis routines.

Deva converted the contents of this file into a database for its internal use. *Deva*-internal database functions allowed the animator to add or modify information pertaining to the physical properties of each segment. These values included the length, mass, center of mass, inertia tensor matrix, and orientation to the segment's parent link. Once the figure was specified, motion was produced either kinematically through a series of key frame positions read from an ASCII file, or kinematically and dynamically by interpolating motion control functions for each degree of freedom. In addition to using these motion control functions for dynamic analysis, *Deva* also accounted for gravity acting on each segment of the body and used springs and dampers to simulate ground reaction forces and joint limits.

Virya was used primarily for displaying and interactively creating and editing the motion control function associated with each degree of freedom. A motion control function consisted of a second-derivative continuous-cubic interpolatory spline. All displayed motion control functions had time represented along the horizontal axis. The vertical axis represented position for kinematic functions, and force or torque for dynamic functions.

During dynamical analysis, a degree of freedom could be in one of five modes. *Relaxed* mode enabled the degree of freedom to interact freely with the environment without any user specified restrictions. *Dynamic* mode caused *Deva* to apply pseudo-muscular torques required to perform simple motions. *Freeze* caused the degree of freedom to be maintained at a particular position through the application of a strong restorative spring and damper. *Balance* resulted in the degree of freedom maintained at a particular

orientation with respect to the world frame. *Hybrid Kinematic-Dynamic* mode resulted in the application of forces and torques to the degree of freedom based upon the current velocity and position of the limb. These modes could be set as a function of time within the *Virya* editor. *Deva* could also run in pure kinematic mode with all the other modes turned off.

The equations of motion for the figure were derived using the Gibbs-Appell formulation and solved numerically with fourth order Runge-Kutta integration techniques. Output from the dynamics calculations was either in the form of *Virya* kinematic control functions or updates to the *Deva* data structures. Kinematic control functions could be modified by the animator at a later time to produce kinematic motion. The data structures were directly used by *Deva* to update the display of the figure.

Most experiments with *Deva* involved simple motions such as making an object fall onto a frictional and frictionless floor, testing joint limits, balancing a man, raising and lowering an arm, raising a leg from a horizontal position, and performing a sit up. While the resulting motion appeared realistic, the solution technique used in *Deva* was computationally expensive for bodies composed of many degrees of freedom. Animation sequences lasting a few seconds required many hours or days to compute.

This problem was somewhat alleviated by Wilhelms and Forsey in 1988 with the construction of *Manikin* [Forsey88]. This system, designed for the interactive manipulation of an articulated figure, used a computationally efficient recursive formulation of the dynamics equations of motion [Armstrong85a] [Armstrong85b]. The computational complexity of this solution method increased linearly in the number of links. The main difficulty with *Manikin* was finding the precise forces and torques required to handle collisions and reach goals. Additional computational power was also necessary for the system to run in near-real time.

2.3.3. A Near-Real Time Dynamic System

A system which dynamically manipulated articulated figures in close to real time was produced in 1985 at the University of Alberta [Armstrong86b] [Armstrong87]. This system was based on the same for-

mulation and solution used by *Manikin* for the equations of motion of articulated figures.

The system was composed of two modules. Interaction with the user and display of the articulated figure model were handled by the *front end*. All dynamic computations were performed within the *dynamic analysis* module. This module either resided on the same processor as the front end, on a different processor, or on a network of processors [Armstrong86a]. Both modules communicated with each other through a series of packet exchanges using either a pipe or a local area network.

Motion was controlled using a set of global and limb motion processes. Global motion processes included a balance procedure to maintain the figure in an upright position, and a friction process to simulate horizontal ground friction.

One of four limb processes controlled the motion of each limb. *Free swing* removed all forces and torques from a limb, thereby leaving the limb to freely interact with the environment. *Friction* generated a velocity-dependent force or torque to retard a limb's motion. *Maintain* held a limb at its current angular position with respect to an adjoining limb. *Move* applied torques produced by torque functions to move a limb from one position to another. Torque functions were based on results obtained from biomechanical studies of torques produced by muscle contraction.

This system was designed primarily to allow the animator to experiment with simple movements on an articulated figure model in close to real time. Results indicated real-time dynamic animation could be achieved using a network of four SUN 3 workstations.

2.3.4. Motion Specification by a Structured Language

In 1986, Danny Cachola and Gunther Schrack of the University of British Columbia discussed a system which enabled a model's motion to be described by a structured language [Cachola86].

A figure was constructed using a grammar to define the components connecting each joint. These components represented either a single link or a previously defined submodel of the body. Joint definitions included any rotational constraints limiting joint movement. The parser assigned each joint a unique iden-

tifier for subsequent motion commands and then built a tree structure to map the figure's links and joints to the nodes and arcs.

Motion verbs were defined either explicitly or implicitly. Explicitly defined motion commands consisted of a motion name followed by a list of affected joints. The joint list contained the identifier of the joint and time-ordered sequence of key frame positions giving the joint angle and the interpolation technique used to reach the orientation. Frame identification numbers were defined either within the motion command or passed as a parameter. Once a motion was defined, it could be implicitly executed by a procedure call from other motion commands. Other commands allowed synchronization of concurrently executing motion commands and the application of a motion command to a subset of affected limbs.

Animation was produced by executing scene procedures composed of functions displaying static components (such as the background and inanimate objects) and functions executing dynamic components. A dynamic component consisted of a model name, the time period allotted for the animation, and a set of motion verbs describing the action to be performed. All motion within the system was produced kinematically.

2.3.5. Animating Multi-Legged Animal Motion

PODA, a computer animation system for simulating the motion of multi-legged animals using dynamics, kinematics, and spline interpolations, was first introduced in 1986 by Michael Girard of Ohio State University [Girard85] [Girard87].

Limbs were positioned using either forward kinematics (solving for the position of the end-effector given the joint angles) or inverse kinematics (solving for the joint angles given the position of the end-effector). A third mode allowed the user to adjust the joint angles with the end-effector clamped at a desired position. The angles were solved using inverse kinematics calculated by means of a pseudoinverse Jacobian.

PODA defined a posture sequence as a series of limb postures. The path within a posture sequence

was given by a spline interpolation of the joint angles or a three dimensional interpolating spline of the end-effector coordinates. Speed was controlled by reparameterizing the arc lengths of an interpolating spline curve giving position as a function of time, and by expressing postures as a function of distance and time.

The limb motion was coordinated using gait specification terminology from biomechanics and robotics. This involved specifying time intervals describing the support and non-support phases of each limb. In addition to handling a wide range of gaits, the system also attempted smooth transitions from one form of gait to another.

The animator specified horizontal motion using cubic splines. Vertical motion during the non-support phases was calculated using simple dynamics based on the gravity acting on the body. Additional dynamics were applied to handle angular motion such as turning and banking.

Non-periodic movement such as dance was animated by displaying sequences of body trajectories. A body trajectory consisted of the union of a series of trajectories belonging to each body limb (including the pelvis). Interpolations of vertical motion such as hops or jumps were based on the height of the body specified in postures prior to lift-off and immediately following contact with the ground.

2.3.6. A Dynamic Motion System

In 1987, Paul Isaacs and Michael Cohen of Cornell University described a system which produced dynamic motion based on a combination of inverse dynamics incorporated with kinematic constraint checking and behavior functions [Isaacs87].

DYNAMO accepted input describing the physical and behavior characteristics of a linked object. Physical characteristics included quantities such as the size, shape, mass, moments of inertia, and center of mass of each link. Behavior characteristics were specified by behavior functions. These included environmental factors such as gravity, and key frame path sequences giving the acceleration of a linkage based on a time-specified input path.

Dynamic simulation at each time increment took place in four phases. First, the behavior functions were executed to obtain the force acting on a linkage or the motion a linkage was about to undergo. Next, joint forces were calculated from spring and dampers acting on each degree of freedom and added to the internal force and torque vectors for the joint. Finally, the equations of motion were formulated and solved for the object.

The equations were derived from D'Alembert's principle of virtual work. A kinematic constraint consisted of explicitly specifying the acceleration of a degree of freedom, thereby necessitating its removal from the system of equations. Once the accelerations of the remaining degrees of freedom were found, inverse dynamics was applied to find the unknown forces acting on each link. The new positions and velocities for each link would be subsequently calculated and the solution checked for violation of any kinematic constraints present within the degrees of freedom. If none occurred, the model was updated with the new solution. Otherwise, a constraining acceleration for the affected degree of freedom was specified and the degree of freedom removed from the equations. The equations would be reformulated and the constraint checking process repeated.

This system produced realistic motion for simple objects such as a small tree swaying in the wind and an arm catching and throwing a ball, but computation time was exponentially dependent upon the number of degrees of freedom in the object.

2.3.7. A Hybrid Kinematic/Dynamic Walking Model

In 1989, Armin Bruderlin and Thomas Calvert of Simon Fraser University presented a goal directed walking model which incorporated both kinematic and dynamic motion control [Bruderlin89]. One of the primary objectives of the *K*eyframe-*L*ess *A*nimation of *W*alking (*KLAW*) system was to produce realistic human gait at a variety of speed and step lengths.

In addition to the body height and mass, three fundamental locomotion parameters (forward velocity, step frequency, and step length) were provided by the animator. If none of these parameters were specified,

the system computed normalized parameters based on the height of the body model. The animator could further customize the locomotion by specifying other attributes such as the amount of rotation of the pelvis and the height of toe clearance during the swing phase. Thus, all input was specified in terms and quantities familiar to the animator.

Control of the locomotion cycle was maintained by a hierarchical model consisting of three levels. The highest (conceptual abstraction) level transformed the fundamental locomotion parameters into a number of step constraints required by the lower levels. These constraints included the maximum angle of the legs from a vertical position based on a compass gait [McMahon84a] [McMahon84b], and the length of a telescopic segment simulating knee flexion and plantar flexion of the ankle.

The middle (gait refinement) level operated similarly to the middle level in Zeltzer's finite state machine motor control model [Zeltzer82b]. This level oversaw the changes in the model's state during the gait cycle. A state change occurred whenever a limb entered either the stance or swing phase of a gait cycle.

The lowest (physical abstraction) level generated the motion through the application of kinematic and dynamic motion algorithms. The dynamic algorithms were used as a starting point for obtaining the motion of the legs. These algorithms consisted of a limited constrained set of equations applied to the body and legs. The equations were derived by the Lagrangian method and solved using numerical integration. A technique referred to as the *virtual leg* principle displayed the stance leg during the gait cycle. This technique superimposed a human leg over the telescopic stance leg by calculating the angles of the leg based on the motion of the foot during the stance phase. Foot rotation and arm swings were simulated using kinematic algorithms. Further "cosmetic" kinematic algorithms were applied to the body to animate various determinants of gait [McMahon84a] [McMahon84b].

Preliminary results with *KLAW* indicated the system could produce a wide range of realistic appearing human walks in close to real time. However, since the figure had been modeled to walk on a stiff floor surface with a high coefficient of friction, the system likely could not produce an accurate description of

gait resulting from surfaces similar to skating rinks or foam mats.

2.4. Movement Notation Systems

Throughout recorded history music and dance have been a part of culture. Many of the musical masterpieces composed by artists within the past four centuries are known because of a musical notation language. Until recently, dance has not enjoyed the benefits of a notation language or archival system. Nearly all of the classics composed in the past have been either entirely lost or altered to an unknown degree because demonstration and repetition were the only means for passing these works on from generation to generation. This shortcoming has been somewhat alleviated in the twentieth century with the development of movement notation languages and video.

While video is useful for recording and archiving dance sequences, it has several limitations [Herbison-Evans85]. A tradeoff exists between recording the flow and overall pattern of the dance versus focusing on the subtle movements of the head, arms, and feet of the dancers. Video offers a two-dimensional interpretation of a three-dimensional work. Some movements may be concealed if the performance involves more than one dancer. Since video is very dependent upon how the performer interprets the choreographer's work, some of the choreographer's intent may be lost in the final production.

2.4.1. Movement Notation Languages

The recent development of movement notation languages has provided a means for translating dance into symbolic form and has helped to enhance dance literacy. Although many notation languages for describing body movement have been proposed, two languages are in common use throughout the dance world [Royce77].

Labanotation, developed by Rudolf Laban, is used throughout the United States and Europe [Laban75]. This notation language has a high level of accuracy and is capable of recording almost any type of human movement. Movement is described along a vertical staff with time running upward. The columns of the staff correspond to the various major body parts and the movement of these parts is

indicated by the direction of the symbols. There are approximately 1500 different symbols in this language.

Benesh notation, developed in the United Kingdom by Rudolf and Joan Benesh in 1947, is used primarily in the Commonwealth countries [Benesh56]. This notation language is more suited towards dance, particularly ballet. A Benesh score has many similarities to a musical score. Movement is described using a horizontal staff with time running discontinuously from left to right. Body parts are projected onto the staff with the upper lines corresponding to the head and arms, and the lower lines to the legs and feet. A series of symbols along these lines indicate the direction of movement of the analogous body parts. Although many of the symbols are compound, approximately 52,000 symbols fully describe the language.

While movement notation languages solve many problems introduced by video, they present problems of their own. Both notation languages require many months of study to learn. As a result, few dancers and choreographers can read or write movement notation. It is also difficult to record a score in real time while a movement sequence is performed.

Within the past twenty years several attempts have been made to remedy these problems through the construction of movement notation editors and interpreters.

2.4.2. Labanotation Editors and Interpreters

In 1974, Zella Wolofsky at Simon Fraser University developed an editor which converted an alphanumeric description of Labanotation into a numerical language describing body position and orientation [Wolofsky74]. This body description could then be converted into the graphical representation of a stick figure. Successive body descriptions were separated by time intervals and an interpolation algorithm was constructed to supply the motion between key positions. The research group at SFU continued this work and by 1977 they had designed a graphical editor capable of reading a score and converting it into alphanumeric code. In 1978, Calvert and Chapman built a macro processor which translated various higher level movement descriptions into Labanotation scores [Calvert78].

Research efforts at the University of Pennsylvania occurred in the opposite order. A Labanotation editor was developed first in 1976 by Maxine Brown [Brown76]. By observing that a movement description consists of the concurrent movement of body parts, in 1978 Badler et al. designed an interpreter based on a set of parallel processors [Badler78]. Each processor operated on one or more limbs.

In 1977, G.J. Savage and J.M. Officer of the University of Waterloo developed a Labanotation system named *CHOREO-L* [Savage78]. The original version, *CHOREO*, used Massine notation and resembled a key-frame animation system. *CHOREO-L* accepted scores entered with an acoustic pen acting on a graphical menu overlaid on an acoustic tablet. The resulting Labanotation score was displayed on the screen for verification. A data base of record information about the symbols was constructed during the movement description. Each record contained the start and end time for a particular body motion. A time-ordered sequential interpolation of the records produced animation.

Another Labanotation-based editor, *NOTATE*, was completed by Sealey at the University of Iowa in 1979 [Sealey80]. The design of this editor focused predominantly on user interface and portability issues. Further expansion to this editor resulted in the completion of *NOTATE II* in 1983 [Politis87]. In conjunction with a computer assisted instruction system *CLIP (C*omputerized *L*abanotation *I*nstructional *P*rogram), *NOTATE II* has been primarily used in teaching Labanotation.

These Labanotation systems are generally more suited for introducing students to Labanotation than for studying movement simulation. An algorithmic approach is difficult to implement because Labanotation leaves too much for interpretation with common motion commands (such as walking and jumping). Much extra work is required to produce a system capable of being used for serious movement studies.

2.4.3. Benesh Notation Editors and Interpreters

Following this active research on movement with Labanotation editors and interpreters, similar investigations were made with editors and interpreters handling Benesh movement notation. Most notable were efforts at the University of Waterloo and the University of Sydney.

An interactive graphics editor for preparing and editing Benesh notated dance scores was published in 1982 by Baldev Singh at the University of Waterloo [Singh83]. This editor was designed to provide a powerful and convenient user interface for use by people not familiar with computers. The system was later extended and named *ChoreoScribe* [Dransch86]. An interpreter was developed for *ChoreoScribe* by Don Herbison-Evans while on leave at the University of Waterloo in 1986 [Herbison-Evans86].

Possibly the first commercial application of a dance score notation system was the microcomputer-based program *MacBenesh* at the University of Waterloo in 1984a. The interpolation of dance scores within *MacBenesh* was more difficult than with *ChoreoScribe* because semantic information retained in *ChoreoScribe* had to be omitted in *MacBenesh.*

Research into computer applications of Benesh movement notation at the University of Sydney began in 1979 with the development of the *NUDES* (*N*umerical *U*tility *D*isplaying *E*llipsoid *S*olids) language for displaying the motion of ellipsoid-shaped solids [Herbison-Evans87]. *NUDES* was a low-level animation system which took a detailed description of the body and displayed the figure as a series of ellipsoids. This system was used by a professional choreographer in 1980 to produce animation of the first six bars of the solo female waltz from the 1836 ballet *Les Sylphides* [Herbison-Evans84a]. In 1986, a Benesh Interpreter *BI* was developed by George Politis. *BI* took, as input, a score generated by a Benesh graphical editor designed on a SUN workstation [Politis82] [Politis85] [Politis86]. The interpreter processed each score, calculated all necessary explicit and implicit information, interpolated motion between adjacent frames, and produced output suitable as input to the *NUDES* animation system.

2.5. Other Systems

In 1987, Badler et al. introduced *POSIT* - a system for positioning articulated figures by constraining limb locations [Badler87]. An articulated figure's position was defined by assigning one or more limbs a goal position and weight value measuring the importance of reaching the goal. *POSIT* then constructed a reach tree based on the configuration of limbs assigned to a goal position. Each tree node contained the displacement of the node from its goal (scaled by goal weight), and the sum of the weighted displacements for

all nodes in the subtree defined from the node. A constraint-satisfaction algorithm balanced these displacements and the resulting position of the figure was updated on the display device.

Several recent animation models have been based on expert system technology. These systems plan and execute goal-directed motion by deducing motion sequences from definitions and relationships stored in object and motion knowledge bases. Expert-based animation systems have included the *Director's Apprentice* [Ridsdale86] at Simon Fraser University and *GEMS* [Drewery86] at the University of Toronto.

During the past decade, several models have modeled facial expressions and speech. Frederic Parke developed a facial animation model which used a parameter set based on both the underlying structure and anatomy of a face, and on observations of facial characteristics [Parke82]. This model produced good facial images from relatively simple parameter sets. A similar facial expression animation model was developed by Stephen Platt and Norman Badler of the University of Pennsylvania [Platt81]. This model emphasized the underlying muscle structure of a face and used a movement notation language based on the Facial Action Coding System (FACS). A model for animating both speech and expression appeared in 1986 [Pearce86]. The face was based on Parke's model and input was specified textually. Another model which produced facial expression based on muscle structure and a limited parameter set appeared in 1987 [Waters87]. This model could handle a wide variety of facial topologies.

Other systems in recent years have attempted to model the grasping motion of hands [Magnenat-Thalmann88] and muscle contraction in arms [Chadwick89].

2.6. Thesis Goals

The dynamically based articulated figure animation systems discussed in this literature review have yet to solve several important issues. How can dynamically based motion be specified using minimal effort and a vocabulary familiar to the animator? Previous systems (for example, *DEVA*) have the animator entering motion sequences in terms of dynamic and kinematic motion generating functions. These functions then act on each limb or degree of freedom. While this approach can be used to study and modify the

resulting motion, the motion vocabulary consists of forces, torques, velocities, and accelerations - quantities unfamiliar to most animators. This approach also becomes tedious and unmanageable for complicated movements requiring the coordination of many limbs, especially when the movement must be performed within a specified time limit.

A method needs to be developed to allow a system to accept both simple and complex movement commands from an easy to use, high-level motion vocabulary. The system should translate these commands into the necessary lower-level dynamic components. All forces and torques should have reasonable magnitudes and produce natural appearing movement within animator-specified time limits. Control of the figure's movement should be performed automatically by the system rather than by having the animator guessing the required forces and torques.

This thesis presents a new animation system which addresses these issues. The system has been designed to dynamically control the motion of an articulated figure and allow motion sequences to be entered using an easy to learn, high-level motion vocabulary. This motion vocabulary consists of the movements defined in ballroom dancing. Each movement is entered using a ballroom dance notation language. In addition to specifying both simple and complex movements, this language can be used to assign each motion a time limit for completion.

The system controls the figure's movement by activating an internal motion control model. This control model breaks complex movements into smaller components and activates a set of low-level motion processes to move or maintain the limbs as required. All torques are generated automatically and are based on muscle torque functions derived from biomechanics. The movement of each limb is controlled by performing interpolations along a quaternion curve defined for the limb. These interpolations are used to adjust the magnitude and direction of the applied torque. As a result, interesting and complex motions may be performed at different rates of speed.

Chapter 3

Background Material

Construction of dynamically-based articulated figure animation systems relies on concepts and theory developed in other scientific fields. This chapter provides a background of the relevant theory from rotational algebra, mechanical engineering, and physics.

3.1. Coordinate Systems and Rotation Matrices

Articulated figures are normally represented using a tree-like structure. The tree consists of links, with arcs connecting each link. Each body segment has a parent link and possibly one or more child links. One segment (usually the upper or lower torso for human figures) is mapped as the root of the tree.

To represent the figure analytically, a coordinate system is required. When dealing with articulated figures, a useful technique is to define a coordinate system for each link and one for the world space. The inertial (world) coordinate system remains constant relative to all links, while each link's coordinate system moves with the corresponding link. The primary advantage with this approach is that several vector quantities (such as the vector representing limb length) remain constant as the limb rotates.

Each coordinate system is composed of three mutually orthogonal unit vectors which define a frame. The components of each of these vectors are arranged as one-column matrices. Frames are commonly represented using a right-handed coordinate system. A limb is rotated by rotating the frame defined by these orthogonal unit vectors. The rotation axis is either one of these unit vectors, or an axis defined by the vector sum of two or more of these vectors. The new coordinate system is obtained by multiplying these unit vectors by a 3×3 orthogonal rotation matrix.

Rotations by ϕ, θ, and ψ degrees about the *X*, *Y*, and *Z* axes are defined by the following standard orthogonal rotation matrices:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Any orientation may be achieved through the multiplication of these rotation matrices. Because matrix multiplication, generally, does not commute, the order these rotations are applied is important. A common method (and the one adopted in *BDAS*) is to define orientations by first rotating about the *Z* (roll) axis, then rotating about the rotated *Y* (yaw) axis, and finally rotating about the doubly rotated *X* (pitch) axis. The rotation matrix which describes this new orientation is the matrix product $R = (R_z(\psi)R_y(\theta)R_x(\phi))$ and it has the following form:

$$R = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}.$$

R represents the transformation of a column vector from the frame of the link (represented by this rotation matrix) to the frame representing the parent of this link. The angles ψ, θ, and ϕ are known as the *Euler angles* of the transformation. Since *R* is the product of three orthogonal matrices, *R* is also orthogonal since the set of all real orthogonal 3×3 matrices form a group called the Orthogonal group O(3). Therefore, the inverse of *R* is its transpose. This transpose represents the transformation of column vectors from the parent frame of the link (represented by *R*) to the frame of the link represented by *R*.

By ascending the tree from the link to the root, a rotation matrix can be obtained giving the transformation of vectors from the frame of the link (represented by *R*) to vectors in the inertial frame. The transformation matrices for all visited nodes are multiplied together from left to right to produce the desired matrix.

A transformation matrix can be decomposed into its Euler angle components by first noting R_{31} is equal to $-\sin\theta$. The absolute value of $\cos\theta$ can be calculated using the trigonometric identity $\sin^2\theta + \cos^2\theta = 1$. If $\cos\theta$ is not equal to zero then the values of ψ and ϕ can be determined by:

$$\cos\psi = R_{11}/\cos\theta \qquad \cos\phi = R_{33}/\cos\theta$$

$$\sin\psi = R_{21}/\cos\theta \qquad \sin\phi = R_{32}/\cos\theta.$$

When $\cos\theta$ is zero, roll cannot be distinguished from pitch. In this case, the roll angle ψ is set to zero and the value of ϕ is determined by:

$$\cos\phi = R_{22}/\cos\theta \qquad \sin\phi = -R_{23}/\cos\theta.$$

Thus, the process of converting a rotation matrix to a set of Euler angles is very ill-defined because it involves uncertainties with taking the sign of square roots and inverse trigonometric functions, as well as an assumption when the cosine of the yaw angle is zero.

3.2. Quaternions

A better and more general method for expressing orientations and rotations is obtained by using a set of four-dimensional numbers called *quaternions*. Quaternions, discovered in 1843 by Sir William Rowan Hamilton, are an extension to complex numbers and consist of one real component and three imaginary components. Every quaternion can be expressed in the form:

$$\lambda + \lambda_x \mathbf{i} + \lambda_y \mathbf{j} + \lambda_z \mathbf{k}$$

where λ, λ_x, λ_y, and λ_z are scalars and **i**, **j**, and **k** are imaginary units with the properties:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1, \qquad \mathbf{ij} = \mathbf{k}, \qquad \mathbf{ji} = -\mathbf{k},$$

and a cyclic permutation of $\mathbf{i} \rightarrow \mathbf{j} \rightarrow \mathbf{k} \rightarrow \mathbf{i}$.

A more convenient method for expressing quaternions is the form $\lambda + (\lambda_x, \lambda_y, \lambda_z)$, where λ is a scalar and λ_x, λ_y, λ_z are the components of a three dimensional vector. This may be rewritten as $[\lambda, \Lambda]$ with λ representing the scalar and Λ the vector component of the quaternion.

3.2.1. The Algebra of Quaternions

Quaternions form a commutative group under addition and a non-commutative group under multiplication. Quaternion addition is performed by adding the corresponding scalar to scalar and vector to vector components:

$$[\lambda_1, \Lambda_1] + [\lambda_2, \Lambda_2] = [\lambda_1 + \lambda_2, \Lambda_1 + \Lambda_2].$$

The rule for multiplying quaternions requires several operations involving scalar multiplications, inner dot

products, and outer cross products of the vector components:

$$[\lambda_1, \Lambda_1][\lambda_2, \Lambda_2] = [\lambda_1\lambda_2 - \Lambda_1\cdot\Lambda_2, \lambda_1\Lambda_2 + \lambda_2\Lambda_1 + \Lambda_1\times\Lambda_2].$$

Quaternions with a zero vector component are called *real quaternions*. Real quaternions multiply like real numbers and can consequently be mapped to the real numbers. By defining the mapping $[\lambda, \mathbf{0}] \equiv \lambda$, the product of two real quaternions can be expressed as:

$$[\lambda_1, \mathbf{0}][\lambda_2, \mathbf{0}] = [\lambda_1\lambda_2, \mathbf{0}] \equiv \lambda_1\lambda_2.$$

Therefore, the product of a real number and a quaternion is:

$$\lambda_1[\lambda_2, \Lambda_2] = [\lambda_1, \mathbf{0}][\lambda_2, \Lambda_2] = [\lambda_1\lambda_2, \lambda_1\Lambda_2].$$

Quaternions with a zero scalar component are called *pure quaternions*. A *unit quaternion* is a pure quaternion in which $\|\Lambda\| = 1$.

The *conjugate* of a quaternion $[\lambda, \Lambda]$ is the quaternion $[\lambda, -\Lambda]$. Multiplying a quaternion by its conjugate gives:

$$[\lambda, \Lambda][\lambda, -\Lambda] = [\lambda\lambda + \Lambda\cdot\Lambda, \mathbf{0}] = \lambda^2 + \|\Lambda\|^2.$$

The square root of this value is the *norm* of the quaternion $[\lambda, \Lambda]$ and is denoted by $\|[\lambda, \Lambda]\|$ A quaternion whose norm is equal to 1 is called a *normalized quaternion*.

The inverse of a quaternion is a quaternion which satisfies the condition:

$$[\lambda, \Lambda][\lambda, \Lambda]^{-1} = [1, \mathbf{0}] \equiv 1.$$

Since

$$[\lambda, \Lambda][\lambda, -\Lambda] = \|[\lambda, \Lambda]\|^2,$$

the inverse of $[\lambda, \Lambda]$ is the quaternion

$$[\lambda, \Lambda]^{-1} = \frac{[\lambda, -\Lambda]}{\|[\lambda, \Lambda]\|^2}$$

provided, of course, $[\lambda, \Lambda]$ is not the null quaternion. Thus, any quaternion other than the null quaternion has an inverse. Like rotation matrices, quaternions have left and right inverses.

The *distance* between two quaternions is measured by taking the norm of the difference of the

quaternions:

$$d([\lambda_1, \Lambda_1], [\lambda_2, \Lambda_2]) = \|[\lambda_1 - \lambda_2, \Lambda_1 - \Lambda_2]\|.$$

3.2.2. Relationship between Rotations and Quaternions

Leonhard Euler proved in 1752 that any three dimensional orientation can be expressed as a single rotation around an axis from a reference position. Let $\mathbf{n} = (x, y, z)$ define a unit vector pointing along the rotation axis and let θ represent the amount of counterclockwise rotation about $\mathbf{n}$. Using spherical trigonometry, a normalized quaternion describing this rotation can be derived with the scalar part, λ, equal to $\cos(\theta/2)$, and the vector part, $\Lambda\ (= \lambda_x, \lambda_y, \lambda_z)$, equal to $\mathbf{n}$ multiplied by $\sin(\theta/2)$ [Altmann86]. The components λ and Λ of this normalized quaternion are called the *Euler-Rodrigues rotational parameters*.

Normalized quaternions and quaternions consisting of Euler-Rodrigues parameters (*Rodrigues* quaternions) can be viewed as the same. An orientation defined by the normalized quaternion $[\lambda, \Lambda]$ is identical to the one defined by $[-\lambda, -\Lambda]$ because adding 2π radians to the rotation angle θ changes the signs of λ and Λ. Alternatively, this negative quaternion expresses a rotation of $360 - \theta$ degrees along the same axis with the rotation axis pointing in the opposite direction.

Normalized quaternions form a sub-group of the quaternion group. A homomorphic mapping exists between the sub-group of normalized quaternions and the set of all real orthogonal 3×3 matrices with determinant $+1$, known as the Special Orthogonal group SO(3). This mapping can be made isomorphic by standardizing the quaternions so either λ is greater than zero, or if λ is equal to zero, Λ points in the direction of the positive hemisphere of the unit sphere.

The orientation defined by a normalized quaternion can be converted into a three dimensional non-homogeneous rotation matrix by performing 15 additions and 9 multiplications:

$$R[\lambda, \Lambda] = \begin{bmatrix} 1-2\lambda_y{}^2-2\lambda_z{}^2 & 2\lambda_x\lambda_y+2\lambda\lambda_z & 2\lambda_x\lambda_z-2\lambda\lambda_y \\ 2\lambda_x\lambda_y-2\lambda\lambda_z & 1-2\lambda_x{}^2-2\lambda_z{}^2 & 2\lambda_y\lambda_z+2\lambda\lambda_x \\ 2\lambda_x\lambda_z+2\lambda\lambda_y & 2\lambda_y\lambda_z-2\lambda\lambda_x & 1-2\lambda_x{}^2-2\lambda_y{}^2 \end{bmatrix}.$$

This rotation matrix can be converted back into a normalized quaternion by noting that the sum of the

diagonal entries equals $4\lambda^2 - 1$. If this value is equal to -1 then λ equals zero; otherwise set λ equal to the positive square root. Similar operations can be applied to the other matrix entries to obtain the values for λ_x, λ_y, and λ_z. The most efficient matrix to quaternion conversion algorithm requires only one square root, three divisions, and a few addition and binary shifting operations [Shoemake85].

Euler angles may be converted to quaternion form by noting rotations by ψ, θ, and ϕ degrees about the roll, yaw, and pitch axes can be described by the following normalized quaternions [Shoemake85]:

$$\Lambda_{roll} = (\cos(\psi/2),\ 0,\ 0,\ \sin(\psi/2))$$

$$\Lambda_{yaw} = (\cos(\theta/2),\ 0,\ \sin(\theta/2),\ 0)$$

$$\Lambda_{pitch} = (\cos(\phi/2),\ \sin(\phi/2),\ 0,\ 0).$$

The normalized quaternion corresponding to the rotation matrix $R_z(\psi)R_y(\theta)R_x(\phi)$ is obtained by the quaternion product $\Lambda_{roll}\Lambda_{yaw}\Lambda_{pitch}$. The quaternion product has components:

$$\lambda = \cos(\psi/2)\cos(\theta/2)\cos(\phi/2) + \sin(\psi/2)\sin(\theta/2)\sin(\phi/2)$$

$$\lambda_x = \sin(\psi/2)\cos(\theta/2)\cos(\phi/2) - \cos(\psi/2)\sin(\theta/2)\sin(\phi/2)$$

$$\lambda_y = \cos(\psi/2)\sin(\theta/2)\cos(\phi/2) + \sin(\psi/2)\cos(\theta/2)\sin(\phi/2)$$

$$\lambda_z = \cos(\psi/2)\cos(\theta/2)\sin(\phi/2) - \sin(\psi/2)\sin(\theta/2)\cos(\phi/2).$$

An orientation defined by a normalized quaternion can be decomposed into its Euler angle representation by converting the quaternion to a 3×3 non-homogeneous rotation matrix and then applying the rotation matrix to Euler angle conversion algorithm described earlier.

Although using normalized quaternions to describe rotations is slightly redundant when compared to using Euler angles (quaternions consist of four components while three dimensional rotations require only three parameters), there are significant advantages to working with quaternions. Quaternions uniquely specify all orientations and are continuous for all orientations. Some orientations defined by Euler angles are not uniquely determined (for example, when the yaw angle is zero). Quaternions are free from *gimbal lock* - the loss of one rotational degree of freedom when two rotation axes are superimposed on each other.

Finally, unlike Euler angles, quaternions can uniquely determine rotation axes.

3.3. Dynamic Analysis

Dynamics deals with the physical laws governing the motion of objects. A fundamental application of dynamics is to predict the motion of a particular system given a series of forces and constraints acting upon it. Dynamics is based on three laws of motion formulated by Sir Isaac Newton [Fowles86]:

1. Every body continues in its state of rest or of uniform motion in a straight line, unless compelled by a force to change that state.
2. Change of motion is proportional to the applied force and occurs in the direction of the force.
3. To every action there is always an equal and opposite reaction.

The simplest application of these laws is to a system consisting of a single particle. Such a system may be viewed as having a body consisting of a mass and an infinitesimal size so all motion may be regarded as translational. The relationship between the force acting on the particle and its change of motion, as described by Newton's second law, is expressed as:

$$\mathbf{F} = m\mathbf{a}$$

where **F** is the vector sum of all forces acting on the particle, m is the mass of the particle, and **a** is the particle's resulting acceleration. Throughout this thesis, vector and scalar quantities are denoted by bold and italic type respectively.

3.3.1. Dynamically Moving Reference Frames

Force, acceleration, velocity, and other physical quantities used in dynamics are vectors which are represented by a system of coordinates within a reference frame defined from three mutually orthogonal unit vectors. As previously mentioned, reference frames may either be inertial (fixed relative to the earth) or non-inertial (move with the system).

The equation describing particle motion in a moving reference frame differs from that of an inertial

frame. Consider a particle at position $\mathbf{r}_i$ in an inertial frame (where $\mathbf{r}$ is the offset from the origin of the frame), and at position $\mathbf{r}_m$ in a moving frame. The origin $\mathbf{r}_o$ of the moving frame is given by:

$$\mathbf{r}_i = \mathbf{r}_m + \mathbf{r}_o$$

Taking the first and second derivatives of this equation with respect to time, the velocity and acceleration vectors are:

$$\mathbf{v}_i = \mathbf{v}_m + \mathbf{v}_o$$

and

$$\mathbf{a}_i = \mathbf{a}_m + \mathbf{a}_o.$$

$\mathbf{v}_o$ and $\mathbf{a}_o$ represent the velocity and acceleration of the origin of the moving frame and $\mathbf{v}_m$ and $\mathbf{a}_m$ represent the velocity and acceleration of the particle in the moving frame. Since, according to Newton's second law, the relationship between the acceleration and the force acting on the particle in the inertial frame is:

$$\mathbf{F}_i = m\mathbf{a}_i,$$

the equation of motion for this particle in the moving frame is:

$$\mathbf{F}_i - m\mathbf{a}_o = m\mathbf{a}_m,$$

or

$$\mathbf{F}_m = m\mathbf{a}_m$$

where $\mathbf{F}_m = \mathbf{F}_i - m\mathbf{a}_o$.

Inertial terms such as $-m\mathbf{a}_o$ are called *fictitious* forces because they are only present within moving frames and are not real forces acting on the system. Fictitious forces must be considered when constructing dynamics formulations based on moving frames.

Now consider the motion of a particle in a frame undergoing both a translation and a rotation with respect to the inertial system. If ω is the angular velocity of the moving frame, the velocity of the particle caused by its rotation about an axis is given by the cross product:

$$\mathbf{v}_{rot} = \omega \times \mathbf{r}_m$$

and the velocity of the particle with respect to the inertial frame can therefore be generalized as:

$$\mathbf{v}_i = \mathbf{v}_m + \omega \times \mathbf{r}_m + \mathbf{v}_o.$$

Thus, the velocity of a particle in an inertial frame can be described by three components - the velocity of the particle in the moving frame, the rotational velocity of the particle as a result of being in a rotating frame, and the velocity of the origin of the moving frame.

The time derivative for any vector $\mathbf{q}_i$ in an inertial frame can be generalized as a sum of the time derivative of the vector in a moving frame added to the term $\omega \times \mathbf{q_m}$ [Goldstein59]:

$$\dot{\mathbf{q}}_i = \dot{\mathbf{q}}_m + \omega \times \mathbf{q}_m.$$

The relationships for higher time derivatives are obtained from differentiating the above equation. For example, the second derivative of $\mathbf{r}_i$ gives:

$$\mathbf{a}_i = \mathbf{a}_m + 2(\omega \times \mathbf{v}_m) + \dot{\omega} \times \mathbf{r}_m + \omega \times (\omega \times \mathbf{r}_m) + \mathbf{a}_o.$$

The resulting equation of motion is:

$$\mathbf{F}_i = m\mathbf{a}_m + 2m(\omega \times \mathbf{v}_m) + m\dot{\omega} \times \mathbf{r}_m + m\omega \times (\omega \times \mathbf{r}_m) + m\mathbf{a}_o$$

or

$$\mathbf{F}_m = m\mathbf{a}_m$$

for $\mathbf{F}_m = \mathbf{F}_i - 2m(\omega \times \mathbf{v}_m) - m\dot{\omega} \times \mathbf{r}_m - m\omega \times (\omega \times \mathbf{r}_m) - m\mathbf{a}_o$.

The above is a generalized equation of motion for a particle in a frame undergoing both a rotation and a translation. Several fictitious forces are present in this equation. The term $2m(\omega \times \mathbf{v}_m)$ is the Coriolis force, $m\dot{\omega} \times \mathbf{r}_m$ a transverse force, $m\omega \times (\omega \times \mathbf{r}_m)$ a centripetal force, and the last term is a force resulting from the acceleration of the moving frame.

3.3.2. Rigid Bodies

Describing the motion of a rigid body is slightly more complicated than that of a particle. Unlike a particle, the mass of a rigid body is not concentrated at one point. A further complication is introduced by the motion resulting from the rotation of the body.

When studying the translational motion of a rigid body, the total force acting on the body is equal to the total force acting on all particles composing the body. The point on the body where the motion resulting from the sum of the forces applied to this point is the same as the motion resulting from the forces applied to all points on the body is called the *center of mass* for the body. This point allows Newton's second law for the motion of a rigid body to be expressed as:

$$\mathbf{F} = m\mathbf{a}_{cm}$$

where $\mathbf{F}$ is the sum of all forces acting on the body, m is the total mass of the body, and $\mathbf{a}_{cm}$ is the acceleration of the center of mass.

Rotational motion for a body about a fixed axis is more complicated because the motion depends upon the point at which the force is applied and the distribution of mass in the body. The rotational analog of force is called *torque* and is defined as:

$$\tau = \mathbf{r} \times \mathbf{F}$$

where $\mathbf{r}$ is a vector from the center of the rotation to the point of application of the force.

The rotational counterpart to Newton's second law describing the relationship between the torque acting on a body and its change of motion is expressed as:

$$\tau = I\alpha$$

where I is the moment of inertia and α is the angular acceleration of the body. Unlike its translational counterpart, rotational motion depends upon the shape and distribution of mass in the body. A torque applied to a long, slender rod produces a different rotational speed for an axis of rotation parallel to the long axis of the rod and passing through the center than for an axis perpendicular to one end of the rod. The distribution of mass is represented in a 3×3 matrix called the *inertia tensor* matrix of the body. This matrix consists of nine integrations performed over the body and has the following form:

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

The diagonal entries are called the *moments of inertia* about the *x*, *y*, and *z* axes and the off-diagonal entries are the *xy*, *xz*, and *yz products of inertia*.

The equations for a rigid body can be simplified by choosing a set of axes so the off-diagonal products of inertia vanish. The set of axes for which this happens are called the *principal axes* of the body and the three diagonal moments are referred to as the *principal moments* of the body. It can be proven that every rigid body has a set of principal axes for any given point [Fowles86]. This set can be found by diagonalizing the inertia tensor matrix. The equations describing rigid body motion are often formulated in a non-inertial frame defined by the principal axes.

3.3.3. Articulated Figures

Formulating and solving the equations of motion for articulated figures is much more difficult than for particles and rigid bodies because of interactions between different parts of the figure. The equations are usually quite complicated and consist of terms involving external influences, control techniques, and internal interactions from adjacent limbs. All dynamic formulations for articulated figures produce a large set of second order differential equations, with an equation representing each degree of freedom in the system. Solving these equations can be computationally expensive because even a simple articulated human figure has approximately 50 degrees of freedom. In many instances, the equations cannot be integrated and must be solved numerically.

3.3.4. Dynamics Equations for Articulated Figures

Articulated figure motion can be obtained dynamically using equations described by Armstrong and Green [Armstrong85a] [Armstrong85b]. These equations assume the figure is represented by a tree-like structure, with the root of the tree connected to one of the body segments (usually the upper body). Neighboring links are connected by revolute joints with three degrees of freedom. A joint consisting of three translational and three revolute degrees of freedom connects the upper body to the inertial (world) frame.

Each link is represented within a moving frame whose axes consist of the principal axes for the body

part represented by the link. Four transformation matrices are associated with each link. Two matrices convert vector representations from the link's frame to the frame of the parent, and from the link's frame to the inertial frame. The other two matrices perform the inverse transformations.

In the equations to follow, superscripts denote a vector belonging to the appropriate link number. Vectors are denoted in bold type, scalars in italics with the link number subscripted, and rotation matrices by upper-case letters. All vectors are represented either in the frame of link *r* or in the inertial frame.

The following vectors and matrices are represented in the frame of link *r*: $\mathbf{a}^r$, ω^r, and $\dot{\omega}^r$ are the acceleration, angular velocity and angular acceleration of link *r*. $\mathbf{c}^r$ is a vector from the proximal hinge of link *r* (the point at which link *r* connects to its parent) to the center of mass of link *r*. $\mathbf{l}^s$ is a vector from the proximal hinge of link *r* to the proximal hinge of child link *s*. $\mathbf{f}^r$ and $\mathbf{g}^r$ are the force and torque exerted by link *r* on its parent at the proximal hinge. $\mathbf{p}_E^r$ is a vector from the proximal hinge of link *r* to the point where an external force is applied. I^r is the moment of inertia matrix for link *r*, formulated about the proximal hinge.

Three vectors are represented in the inertial frame. $\mathbf{f}_E^r$ is an external force acting on the point $\mathbf{p}_E^r$ (represented in the frame of link *r*), $\mathbf{g}_E^r$ is an external torque acting on link *r*, and $\mathbf{a}_G$ is the acceleration of gravity.

m_r is a scalar quantity denoting the mass of link *r*. The rotation matrix R_I^r converts vectors from the frame of link *r* to the inertial frame, and $R_I^{r\,T}$ performs the inverse transformation. R^r converts vectors from the frame of link *r* to the frame of the parent of link *r*. $R^{r\,T}$ performs the inverse transformation.

The first equation of motion relates the angular acceleration of the link in terms of the torques applied to the link:

$$I^r\ \dot{\omega}^r = \mathbf{g}_\Sigma^r - m_r\ \mathbf{c}^r \times \mathbf{a}^r + \sum_{s\ \varepsilon\ S_r} \mathbf{l}^s \times R^s\ \mathbf{f}^s \qquad (1)$$

where

$$\mathbf{g}_\Sigma^r = -\ \omega^r \times (\ I^r\ \omega^r\ ) - \mathbf{g}^r + \sum_{s\ \varepsilon\ S_r} R^s\ \mathbf{g}^s + R_I^{r\,T}\ \mathbf{g}_E^r \qquad (2)$$

$$+ m_r \, \mathbf{c}^r \times R_I^{r\,T} \, \mathbf{a}_G + \mathbf{p}_E^r \times R_I^{r\,T} \, \mathbf{f}_E^r .$$

The second term in (1) is a fictitious torque caused by the acceleration of the moving frame. The rightmost term is the sum of the torques at the proximal hinge of link *r* resulting from the forces applied to the child links. In equation (2), the leftmost term is a fictitious torque caused by the rotation of the moving frame. The second term is the negative of the torque exerted by link *r* on its parent at the proximal hinge, as per Newton's third law. The next two terms are torques from the child links and an external torque. Finally, the bottom terms of (2) represent the torques caused by gravity and an external force applied to the point $\mathbf{p}_E^r$.

The next equation of motion relates the force acting on a link in terms of the acceleration of the link:

$$\mathbf{f}^r = \mathbf{f}_\Sigma^r - m_r \, \mathbf{a}^r + m_r \, \mathbf{c}^r \times \dot{\omega}^r + \sum_{s \, \varepsilon \, S_r} R^s \, \mathbf{f}^s \tag{3}$$

where

$$\mathbf{f}_\Sigma^r = - m_r \, \omega^r \times (\, \omega^r \times \mathbf{c}^r) + R_I^{r\,T} \, (\, \mathbf{f}_E^r + m_r \, \mathbf{a}_G \,) . \tag{4}$$

In equation (3), $-m_r \, \mathbf{a}^r$ is a fictitious force coming from the acceleration of frame *r*. The next term is a force resulting from the frame rotating with an angular acceleration and the last term represents the forces from all child links. The first term of equation (4) is the centripetal force caused by the rotation of the frame. The last term consists of the components of the external force and gravity acting on the link.

The last equation of motion relates the acceleration of the proximal hinge of a child *s* of link *r* to the acceleration of the proximal hinge of link *r*:

$$R^s \, \mathbf{a}^s = \mathbf{a}^r + \dot{\omega}^r \times \mathbf{l}^s + \omega^r \times (\, \omega^r \times \mathbf{l}^s) . \tag{5}$$

3.3.5. Solving the Dynamics Equations

Several methods exist for solving these and similar equations. Wilhelms [Wilhelms85] [Wilhelms86] [Wilhelms87] animates human figure motion using the Gibbs-Appell formulation. Although this technique provides for simulation of both revolute and sliding joints, the computational complexity of this method is quadratic in the number of links since the inversion of a large, usually sparse matrix is

required during each step of the simulation.

A much faster, although more restrictive, solution technique was formulated by Armstrong and Green [Armstrong85a] [Armstrong85b]. This method solves the equations using a recursive formulation based on the following two linear relationships:

$$\dot{\omega}^r = K^r \mathbf{a}^r + d^r$$

$$\mathbf{f}^r = M^r \mathbf{a}^r + \mathbf{f}'^r.$$

The first relationship is between the acceleration of a link and the magnitude of its angular acceleration, and the second is between the acceleration of a link and the reactive forces present on its parent.

By recursively considering these relationships between adjacent links, the equations are solved by computing the "recursive" coefficients of the above linear relationships in an inward pass from the leaves of the tree to the root. Once the root is reached, its acceleration is determined since there is no force present on the parent of this link (because there is no parent). The angular acceleration of all the child links are then calculated in an outward pass from the root to the leaves. Each link's angular acceleration is then multiplied by the time step to obtain its angular velocity. A second time step multiplication produces the incremental rotation vector.

Because many quantities in the above equations vary slowly over each time step, the authors recommend dividing the computations into two bands: a *fastband* where certain computations are performed at every time step, and a *slowband* where other computations occur once for every *n* iterations of the fastband.

The primary advantage with the Armstrong-Green solution method is that computational complexity increases linearly with respect to the number of links. This has allowed for some dynamic simulations to be performed in near real-time [Armstrong86b] [Armstrong87]. However, this method has been developed for articulated figures having only revolute joints with three degree of freedom. Joints with limited or no movement along certain degrees of freedom (for example, the knee) must be simulated using tools such as springs and dampers. Articulated figures with sliding joints cannot be modeled, and non-spherical joints on other figures must be constrained. The benefits resulting from reduced computational costs and the ability

to perform near real-time dynamics, however, greatly outweigh these restrictions.

3.3.6. Springs and Dampers

Springs and dampers are used in *BDAS* to provide upward restorative ground forces and internal link torques. Consider a force whose magnitude is dependent upon the displacement from an equilibrium position, and whose direction opposes the direction of this displacement. Such forces are exerted by springs obeying Hooke's Law:

$$\mathbf{F} = -k\mathbf{x}$$

where $\mathbf{x}$ is the displacement from the equilibrium position and k is a proportionality constant referred to as the *stiffness* of the spring. Springs whose force can be expressed in terms of a proportionality constant are called *linear springs*.

Since frictional forces are always present in mechanical systems, there is a viscous retarding force which opposes the motion produced by the spring force. This force (for example, air resistance) can be assumed to vary linearly with the speed of the object. This retarding force can be expressed as:

$$\mathbf{F} = -c\dot{\mathbf{x}}$$

where $\dot{\mathbf{x}}$ is the speed of the object and c is a proportionality constant called the *friction* of the spring.

The behavior of the spring is determined by the stiffness and friction values and whether or not gravity influences the mass attached to the spring. When gravity is present, the resulting motion is either non-oscillatory with the displacement from equilibrium decaying exponentially to zero, or oscillatory with the amplitude decaying exponentially over time.

3.3.7. Frictional Ground Forces

Frictional ground forces oppose forces applied to move stationary objects and cause deceleration of objects already in motion. Two types of frictional ground forces act on bodies. Frictional forces acting between objects at rest (with respect to each other) are called forces of *static friction*. The maximum force of static friction is the minimum force required to move the object. Frictional forces acting between objects

in motion (with respect to each other) are called forces of *kinetic friction*. The force required to initiate movement is usually greater than the force required to sustain uniform motion.

Each type of force is approximately independent of the surface contact area and is proportional to the magnitude of the normal force. The ratio of the magnitude of the maximum force of static friction to the magnitude of the normal force is called the *coefficient of static friction*. The corresponding ratio of the magnitude of the force of kinetic friction to the magnitude of the normal force is called the *coefficient of kinetic friction*. These coefficients are abbreviated by μ_s and μ_k and may be expressed mathematically as:

$$\mu_s = F_s / N \quad \text{and} \quad \mu_k = F_k / N.$$

In general, μ_s is greater than μ_k. Because calculating the forces of static friction for each contact point involves calculating the reaction forces at each contact point of the figure model, *BDAS* makes the simplifying assumption that these two coefficients of friction have the same value.

Chapter 4

Motion Control

One of the major problems with human figure animation is describing and controlling the figure's motion. Most human figure movement is obtained from simultaneous coordination of the motion of many limbs. Simple key frame systems are inappropriate for this type of animation since animators are seldom able to control the motion of more than three limbs at a time.

Animators also want to work with natural units of control. A figure's motion should not be controlled by the animator directly specifying the force and torque, or velocity and acceleration. Nor should motion be controlled by the animator directly entering the coordinate position of each segment. Instead, motion commands should be entered using a vocabulary familiar to the animator. The animation system should then translate these motion commands into a lower level description suitable for kinematic or dynamic analysis.

Most articulated figure animation systems within the past decade have approached these issues by implementing models which organize motion specification and control in a hierarchical manner. In all cases, the higher levels of the model define motion in a vocabulary known to the animator, and the lower levels parse the motion description into the components required to generate animation. The high-level motion descriptions supported by these systems have ranged from the symbolic form found in movement notation languages, to English commands found in knowledge based systems. All these systems parse high-level motion descriptions into suitable low-level key frame, kinematic, or dynamic specifications.

BDAS addresses the motion description and control issue by implementing a recently proposed hierarchical motion control model formulated by Mark Green [Green90]. This motion control model proposes that movement be generated from the execution of low-level motion verbs. Movement commands entered by the animator use a high-level description language and are parsed by the system into these

motion verbs. In addition to this motion specification structure, the model also proposes how the environment should interact with the figure, how a figure should behave, and how personality characteristics should influence the figure's motion.

4.1. Green's Hierarchical Motion Control Model

Green's hierarchical motion control model, shown in Figure 4.1, is composed of six levels. Each level consists of one or more separate processes which execute in parallel and communicate with processes

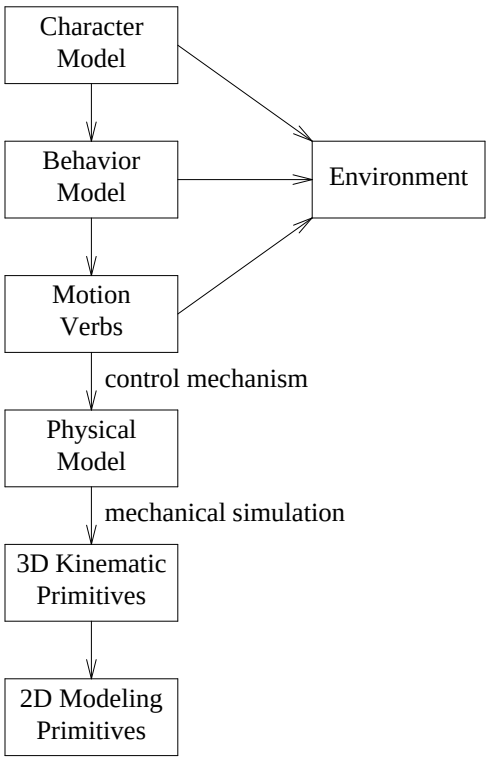


Figure 4.1 Green's Hierarchical Motion Control Model

in other levels. The two lowest levels produce the graphical primitives generated by the graphics system and the three dimensional modeling primitives used to represent the shapes of the objects. These are common to most modeling systems and are not described further.

The physical model represents the anatomy of the object and the physics required to produce motion. The anatomy component contains information such as the number of limbs and body parts, their mass,

inertia, length, how they are connected to each other, and what (if any) constraints limit their motion. The physical component is responsible for producing object motion. This component consists of reach algorithms, dynamic solutions of the equations of motion, kinematic solutions based on the velocities and accelerations of each limb, or key frames and inbetweening algorithms.

The motion verb level defines the motion vocabulary of the object. This level is responsible for generating input necessary for the physical model to produce the desired motion. An object's motion vocabulary consists of a library of motion verbs. Each motion verb corresponds to a particular motion the object can perform. For example, a motion verb of an articulated two or four-legged animal may consist of a single forward step.

Different objects may have similar or very different motion vocabularies. For instance, one would expect the motion vocabulary of a man to be very different from that of a fish. In all cases, however, a motion verb consists of a set of virtual processes which drive the physical model. Whenever these virtual processes are active, the verb is marked as *active*. Otherwise, the verb is marked *inactive*.

The behavior level controls the status of the motion verbs and is responsible for setting the short term goals of the object. As with motion verbs, this level may also consist of one or more processes executing in parallel. Each behavior level process is either executed in an arbitrary manner or according to a priority determined from examining the state of the environment. The environmental approach is often preferred since this allows for greater control of the object's motion.

Each behavior process is assigned a series of triggers that determine when it should start execution. An internal priority is also set to resolve conflicts arising when triggers for two behavior processes are concurrently satisfied. These triggers are related to the current state of the environment and to the current state of the figure.

The character level is responsible for setting the long term goals, personality, and characteristics of the object. Without this level, all objects containing similar behavior functions and motion vocabularies would execute in an identical fashion. This level gives each object a "personality" by influencing the firing

of the triggers and the setting of the priorities in the behavior level.

4.2. The Hierarchical Structure of Ballroom Dances

Green's hierarchical motion control model can be applied to a wide variety of motion studies. The example used in this thesis is the motion produced by human subjects performing ballroom dancing. This area of movement was selected for a number of reasons.

Ballroom dancing provides an interesting challenge to human figure animation. The patterns and dances that define ballroom dancing provide a rich repertoire of human motion and include motion found in normal activities (such as walking) as well as motion suited for artistic purposes (such as swirls and body dips). Unlike many other dances, ballroom dancing requires close interaction with a partner. Every motion sequence is decided by the male partner through a process called *leading*. The female partner is responsible for correctly interpreting the leads given by the male. This can produce interesting synchronization problems between the two partners.

Ballroom dancing also has a well-defined notation language [Thornhill-Geiger81]. The symbols composing this language represent common ballroom dance movements such as forward, sideward, and backward steps. A detailed description of this notation language is provided in Chapter 5. Unlike the Labanotation and Benesh notation languages, the ballroom dance language is designed for recording movement specific to ballroom dances rather than movement in general.

All ballroom dance motions are classified into general categories known as *dances*. The complete set of ballroom dances constitutes a wide range of human figure movements and styles. All dances have individual characteristics defined by factors such as the tempo and timing of the music, the procedure used to step down on a foot, and movement required among selected body parts. For example, the *waltz* consists of a smooth progression of long, slow steps danced to 3/4 time music. The *cha cha*, on the other hand, consists of the rapid execution of many small, quick steps while rocking the hips and is danced in 4/4 time.

Each dance can be divided into a set of motion sequences called *patterns*. A pattern consists of a

well-defined movement which usually requires between 2 to 10 seconds to execute. Commonly occurring patterns are given names to aid identification of the movement sequence. For example, the *forward basic* pattern in the *foxtrot* is defined by two forward steps (starting with the left foot) followed by one left side step. A ballroom dance performance is usually composed of the sequential execution of these patterns.

All ballroom dance patterns may be decomposed into a sequence of *positions*. A position defines the location and orientation of the body after a specified time interval. The movement to a position during this time period is called a *step*. Nearly all patterns are composed of the transitions between five fundamental positions. These are known as first, second, third, fourth, and fifth position (see Chapter 5 for more details). Although the orientation of one or more limbs may vary slightly from pattern to pattern, the general limb and body orientations defining each position remain essentially invariant.

In summary then, the movement sequences defining ballroom dancing may be hierarchically decomposed into sets of *dances*, *patterns*, and *positions*. Each dance has a set of well-defined patterns and each pattern consists of a set of fundamental positions.

4.3. Mapping the Structure of Ballroom Dancing to the Motion Control Model

A divide and conquer approach is applied toward developing a ballroom dance animation model. Rather than generating animation starting at the dance level, the ideas outlined in Green's hierarchical motion control model are applied to the development of motion verbs producing the dance patterns and position transitions. A wide and interesting range of human motion can result from the construction of higher controlling levels (consisting of the combination of patterns and dances) on top of the motion verbs.

All the fundamental components of ballroom dancing outlined in the previous section can be mapped onto Green's hierarchical motion control model between the physical model and the character model. Dance patterns are mapped to the motion verb level, with each motion verb defining a pattern. Because a pattern consists of a series of transitions between fundamental positions, it is useful to introduce an extra level into the model by having each pattern motion verb initiate sequential execution of low-level position

verbs responsible for performing single position transitions. These low-level verbs then interact directly with the physical model.

Adding this extra level removes the redundancy that would be present if each pattern motion process tried to execute an entire pattern without the aid of a structured position level. The added level also simplifies the structure of each motion verb and facilitates debugging of the low-level verbs since each position verb handles a more specific type of movement.

The behavior level controls the short-term motion of the human figure through the sequential selection of dance patterns. A pattern is selected based on a set of parameters and priorities associated with each pattern. Parameters are based on factors such as the location of the dancer, the difficulty of the pattern, the friction of the floor, and the number of dancers on the floor. A pattern's priority is set by conditions such as the last time the pattern was executed.

Virtual processes at this level are most frequently triggered by the completion of the previous dance pattern. Environmental factors such as external interrupts (for example, collision with the wall) and internal interrupts (for example, loss of synchronization with the step) also trigger these processes.

The character level is responsible for ensuring that no two dancers behave identically. In addition to influencing parameters used by the behavior level for selecting patterns, this level customizes each dancer's movement by assigning each dancer motion characteristics such as the size of a normal step and how well the dancer keeps to the beat.

A minimum of five components are required to implement a ballroom dance animation model based on Green's hierarchical motion control model. These components are a *Dance Library and Pattern Editor* for creating and editing dances and patterns, a *Gesture Editor* for defining dance positions, a *Vocabulary Editor* for assigning a dance vocabulary to each dancer, an *Environment Editor* for setting and changing environmental parameters, and a *Dance Interpreter* for overseeing the execution of the character, behavior, motion verb, and lower levels of the model.

Chapter 5

The Ballroom Dance Animation System

The Ballroom Dance Animation System (*BDAS*) is a computer animation project whose goal is to produce dynamically controlled motion from an easy to learn ballroom dance notation language. Although this system is currently used as a tool for studying the dynamic motion of a human model, potential applications for this system include use as a teaching device for ballroom dance students and as a choreography tool for professional ballroom choreographers.

5.1. Overview

BDAS consists of six major components divided into two levels (Figure 5.1). The Top Level allows the user to switch between the five modules composing the Lower Level. Modules in the Lower Level define the components required to implement the ballroom dance motion model. The modules are accessed from the Top Level by a series of pull-down menus. The design of the Lower Level modules have been influenced by Green's hierarchical motion control model, presented in Chapter 4.

Dances and patterns are created and modified using the Dance Library and Pattern Editor. This editor consists of an easy to learn, menu-driven interface. The animator creates and modifies dance patterns by selecting symbols representing movements and timings defined by a subset of a ballroom dance notation language. This editor defines the motion verb vocabulary available to the human figures.

Dance positions are defined with the Gesture Editor. This editor enables the animator to define human figure goal positions by interactively manipulating, in three dimensions, the limbs of a model displayed on the screen.

The Vocabulary Editor assigns a dance vocabulary to each dancer. This vocabulary is represented by *dance names* and *dance patterns* previously defined by the Dance Library and Pattern Editor.

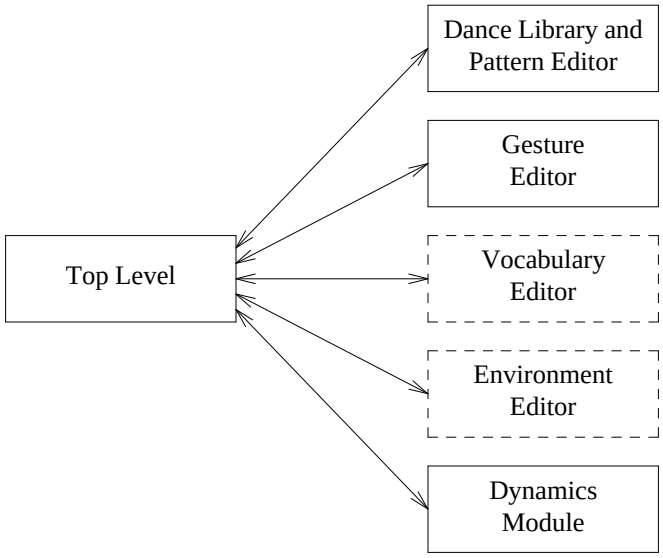


Figure 5.1 Software Architecture

Conditions in the dance environment are set by the Environment Editor. These conditions influence both the nature of the dance and the behavior of the dancers in terms of pattern selection. Environmental variables include the friction and size of the floor, and the number of dancers per square meter.

All motion generated in *BDAS* is dynamically produced by the Dynamics Module. This module gives each dancer a pattern to perform and executes the motion processes governing the dancer's motion. The Dynamics Module also simulates the environment defined by the Environment Editor.

Currently, only the Dance Library and Pattern Editor, Gesture Editor, and Dynamics Module have been installed. A subset of the Environment Editor has been implemented under the Dynamics Module to allow for animation tests under varying environmental conditions. The Vocabulary and Environment Editors have not yet been incorporated within the system because of difficulties encountered with debugging the critical low-level dynamic motion motor processes. The system is also limited to animating only one dancer.

5.2. Ballroom Dance Notation Language

All ballroom dance patterns in *BDAS* are entered and displayed using a notation language found in manuals published by the National Council of Dance Teacher Organizations, Inc. (N.C.D.T.O.) [Thornhill-Geiger81]. This notation language defines patterns using a grid. The rows of the grid correspond to body part descriptions and step timings. For example, a row may be used to describe the position of the head after each step or the number of beats required to execute the step. Each column defines a goal position (in terms of limb and body orientation) and the length of time required to reach this position. Although the time required to reach a goal position for a step may differ from step to step, the overall flow of time is from left to right.

F O X T R O T								
Pattern: Forward Basic								
Commence Facing Line of Dance								
Step Number	1	2	3	4	5	6	7	8
Foot Positions	LF FWD	RF FWD	LF SWD	RF CL/LF	LF FWD	RF FWD	LF SWD	RF CL/LF
Alignment	F LOD	----	----	----	----	----	----	----
Amount of Turn	NT	----	----	----	----	----	----	----
Arms	NP	----	----	----	----	----	----	----
Rise and Fall								
Footwork	H	H-T	T	T-H	H	H-T	T	T-H
Beats	2	2	1	1	2	2	1	1
Beats and Bars	1-2	3-4	2	2	3-4	3-2	3	4
Timing	S	S	Q	Q	S	S	Q	Q
Musical Counts	1-2	3-4	1	2	3-4	1-2	3	4
Dance Position	CP	----	----	----	----	----	----	----
CBM and Sway	SCBM	SCBM	SWR	SWR	SCBM	SCBM	SWR	SWR
Lead and Follow								
Head Position	NP	----	----	----	----	----	----	----

Figure 5.2 Forward Basic in Foxtrot

Figure 5.2 shows a sample description of a pattern called the *Forward Basic* in the foxtrot. The first step begins with the body facing *line of dance* (FLOD). This term refers to a counterclockwise direction of movement. The head looks forward and the arms are extended to a normal ballroom dance position (NP). The shoulders are parallel to the partner's, thereby forming a closed position (CP). This configuration is maintained throughout the pattern.

Step 1 is a forward step with the left foot (LF FWD). The step is executed by stepping down on the heel of the foot (H) and requires two beats to complete (the timing for each step may also be marked by indicating S for *slow* steps and Q for *quick* steps). Step 2 is a forward step with the right foot (RF FWD). This step is performed by stepping down on the heel and then gently rising to the toe (H-T). Like its predecessor, step 2 also requires two beats to complete. During both of these steps, the body undergoes slight contra-body movement (SCBM). This term refers to a small rotation of the upper body and arms towards the partner.

Step 3 is a side step with the left foot (LF SWD) executed entirely on the toe of the foot (T). Step 4 closes the right foot to the left foot (RF CL/LF) with the dancer first stepping down on the toe and then gently falling onto the heel of the foot (T-H). Each of the last two steps requires one beat to complete. During each interval, the body undergoes a gentle sway to the right (SWR).

All four steps are performed very smoothly with very little rise and fall (a term referring to a pronounced lift onto the toes followed by a descent back to a flat foot position). Throughout the motion, the body faces the same direction and undergoes no turning (NT). Steps 5 to 8 repeat the movement sequence defined by steps 1 to 4.

This example shows how interesting and complex movement can be defined using a limited and abbreviated notation language. A complete description of the notation language (which consists of 143 symbols) may be found in the N.C.D.T.O. manual. Descriptions of other terms used in ballroom dancing may be found in ballroom textbooks [Romain79]. Although this extensive use of abbreviations may be intimidating to novices, with experience these descriptions can be quickly translated into their associated

movement sequences.

BDAS supports a subset of the N.C.D.T.O. ballroom dance notation language. Since elementary ballroom movement consists of positioning the head, arms, body, legs, and feet along a particular direction over a specific time period, *BDAS* has implemented the rows corresponding to *Head Position*, *Foot Position*, *Footwork*, *Amount of Turn*, and *Number of Beats*. Arms are assumed to always be in a normal ballroom dance position. The upper body, unless otherwise specified, is assumed to be positioned in the direction of the movement. Symbols indicating subtle upper body movements, such as sway, are not implemented at this time.

Steps requiring limbs to deviate from a normal position are indicated by extra symbols in the row marking the position of the feet. For example, if a right forward step requires the toe to point outwards, the *Foot Position* entry describing this step would contain symbols for *right foot forward* and *toe out*.

BDAS has made one modification to the N.C.D.T.O. notation language. Instead of referring to N.C.D.T.O. terms such as SWD to describe a side step outwards and CL/{R,L}F to represent a side step to close the feet back together, *BDAS* bases most of its foot descriptions in terms of the five fundamental ballroom dance foot positions. These are known as *1st*, *2nd*, *3rd*, *4th*, and *5th* position and are described in detail in the forthcoming symbol tables. This modification helps make the patterns easier to read since most ballroom dancers use these foot positions to describe dance steps. Nearly every elementary ballroom dance pattern is composed of transitions between these foot positions.

The following tables describe the subset of the N.C.D.T.O. notation symbols supported by *BDAS*. Each entry contains the symbol's abbreviated name, full name, and description of the movement, position, or timing involved. All of these symbols may be accessed using the Dance Library and Pattern Editor.

Head (and Arm) Positions:

LL	Look Left	Rotate head 45 degrees to the left.
NP	Normal Position	Look straight ahead.
LR	Look Right	Rotate head 45 degrees to the right.

As previously mentioned, *BDAS* assumes the arms are always in a normal ballroom dance position.

Foot (or Leg) Selectors:

LF	Left Foot	Select left foot and leg for movement.
RF	Right Foot	Select right foot and leg for movement
HOLD	Hold	Hold current position for the entire step.

Fundamental ballroom dance positions:

Select one of these symbols when either **LF** or **RF** is chosen:

1ST	1st Position	Both feet are parallel directly beside each other and point forward.
2ND	2nd Position	Both feet are parallel approximately shoulder width apart and point forward.
3RD	3rd Position	Rotate the left leg approximately 45 degrees counterclockwise or the right leg 45 degrees clockwise and rest the heel of the rotated foot against the inside middle edge of the other foot. Body rotates to allow both feet to be slightly turned out.
FWD	4th Position	Extend one leg forward approximately 6 to 12 inches to assume a forward walk position. Feet are parallel and move in parallel tracks.
BWD	4th Position	Extend one leg backward approximately 6 to 12 inches to assume a backward walk position. Feet are parallel and move in parallel tracks.
5TH	5th Position	Rotate the left foot approximately 45 degrees counterclockwise or the right foot 45 degrees clockwise and place the toe of the rotated foot at the heel of the other foot. Body rotates to allow both feet to be slightly turned out.

Other foot positions not representable by the above but which occasionally occur in elementary ballroom dancing:

SIP	Step in Place	Shift weight but do not change location of the feet.
XIF	Cross Front	Cross the specified foot in front of the other foot.
XIB	Cross Behind	Cross the specified foot behind the other foot.
UNX	Uncross	Rotate the body approximately 180 degrees to have the feet uncross to the first position.

Ballroom dance position modifiers:

Select one or more of the following modifiers to the above dance positions:

CBMP	CBMP	An abbreviation for contra-body movement position. Rotate the upper body approximately 45 degrees towards the traveling foot.
DIAG	Diagonally	Move diagonally forward or backward.
LWD	Leftward	Move leftward (to the side).
RWD	Rightward	Move rightward (to the side).
NW	No Weight	Move to the required position but do not change weight.
SS	Small Step	Move to the required position, but cover about half the normal distance.
TO	Toe Out	Rotate the foot corresponding to moving leg so the toe points outward.
PAUSE	Pause	Hold current position for a fraction of a beat.

Foot Positions:

Select one of the following symbols to indicate the footwork applied to the step:

H	Heel	Step down on the heel and end on a flat foot.
T	Toe	Step down on the toe and remain on the toe.
H-T	Heel-Toe	Step down on the heel and rise on the toe.
T-H	Toe-Heel	Step down on the toe and end on a flat foot.

Turns:

Turns may be specified as **1/8**, **1/4**, **3/8**, **1/2**, **5/8**, or **3/4** of a full revolution. The body may either turn **L** (counterclockwise from the dancer's point of view) or **R** (clockwise from the dancer's point of view).

Timing:

Either **1**, **2**, **3**, or **4** beats may be selected as the time required to reach the new position. The number of beats per second is set by the animator prior to running animation sequences.

5.3. Dance Library and Pattern Editor

All dances and patterns associated with ballroom dancing are created and entered using the Dance Library and Pattern Editor. This editor consists of six top-level commands for creating and deleting dances, creating, modifying, and deleting patterns, and viewing a selected dance pattern. When the animator first enters the editor, a pull-down menu appears listing these commands.

Dances are organized by directory name. A dance is created by selecting the *Create Dance* entry from the pull-down command menu. *BDAS* prompts for a dance name and creates a dance directory reflect-

ing the entered name. All patterns belonging to the dance are subsequently assigned to this directory.

A dance is deleted by selecting the *Delete Dance* entry from the pull-down command menu. *BDAS* provides a list of dances to delete and requests confirmation of the action. The selected dance directory and all patterns within this directory are removed following an affirmative confirmation.

Dance patterns are created and modified by the *Add Pattern* and *Modify Pattern* pull-down menu commands. *Add Pattern* requests the name of the new pattern while *Modify Pattern* requests the name of a pattern to modify from a set of existing patterns. In either case, *BDAS* moves into the *Pattern Editor*, shown in Figure 5.3.

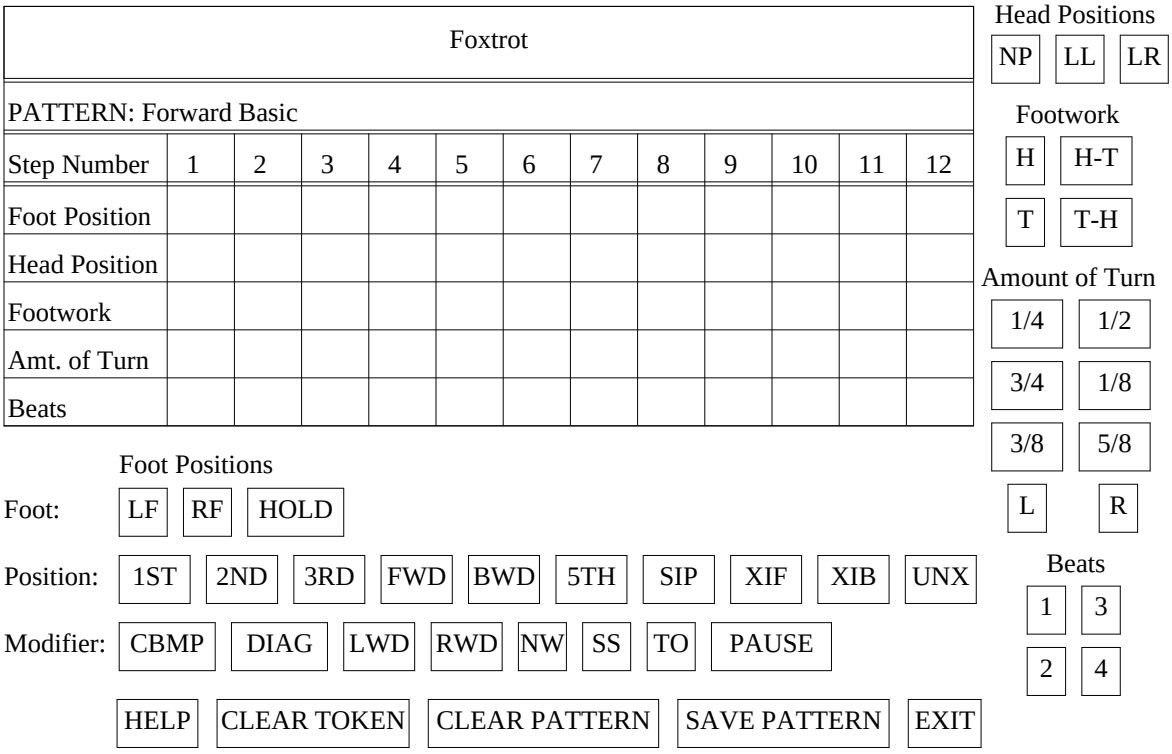


Figure 5.3 Pattern Editor Screen Layout

The Pattern Editor is used to add new patterns or edit existing patterns. Each pattern consists of at most 12 steps arranged in a grid-like fashion. If a previous pattern is being modified, the grid columns are filled with the symbols defining the pattern.

Patterns are defined by filling the grid columns with one or more symbol names. All symbol boxes are organized by row name and color-coded to aid category identification. A symbol is selected by using the mouse to point at the box corresponding to the symbol name and pressing the mouse button. The active symbol is marked by changing its symbol box background color to white. No more than one symbol from each category may be active for each step.

Once all symbols corresponding to a step have been selected, they are entered into the step column by pointing at any row belonging to the step column and pressing the mouse button. All active symbols are then inserted into these column boxes. In the case of a pattern undergoing modification, a previously defined column box is replaced by the new symbol (or cleared if the corresponding category has no active symbol). This process continues until the entire pattern is entered. The conclusion of a pattern is marked by the end of Step 12, or by a step with a blank column.

Commands at the bottom of the screen access on-line help, clear all active symbols, clear the grid entries defining the pattern, save the defined pattern, and return control to the Top Level. Each command is accessed by pointing at the command box and clicking the right button.

Patterns may be viewed on a step-by-step basis by selecting the *View Pattern* command from the pull-down command menu. After obtaining the dance and pattern name from two pop-up menus, *BDAS* transfers control to the *Pattern Analyzer*, shown in Figure 5.4.

The large window, shown in this figure, displays the human figure model. This model consists of sixteen body segments defining the head, neck, upper and lower torso, upper and lower arms, upper and lower legs, hands, and feet. Each of these body segments is covered with either four or six polygons. The polygons are shaded with different colors to facilitate identification of sides.

A step sheet listing the dance and pattern name, current step number, and symbol names composing the step appears to the right of the large window. Steps are displayed in a forward progression by continually clicking the right mouse button. Each step shows the current list of symbols in the step sheet and the position of the human figure model at the conclusion of the step. Previous steps are accessed by clicking

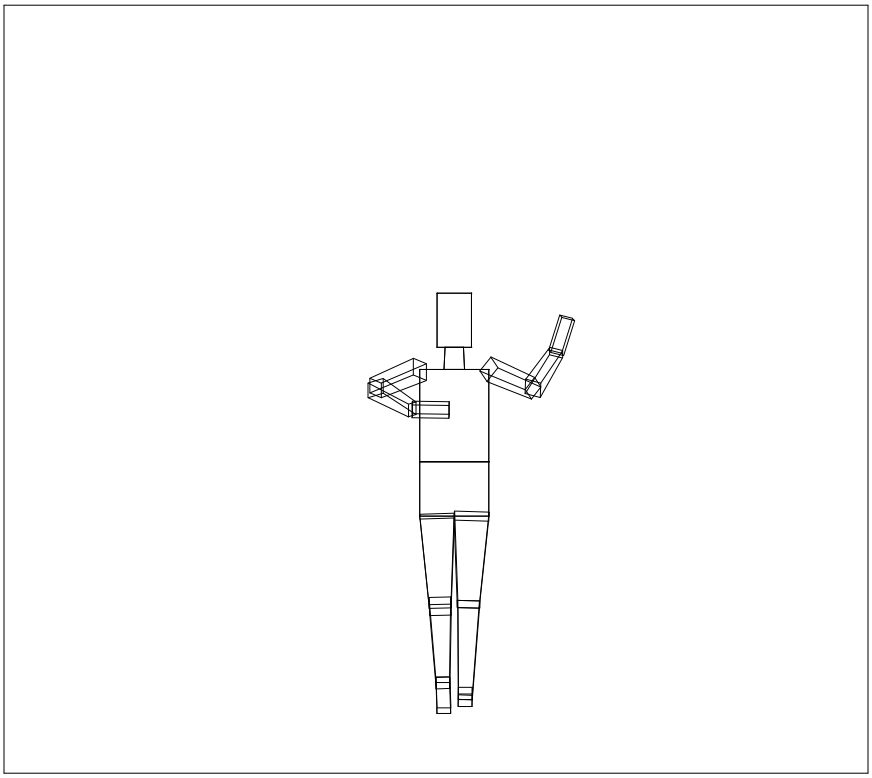


Figure 5.4 Pattern Analyzer Screen Layout

the left mouse button.

The human figure model is positioned by combining the body and limb orientation angles defined for each symbol name composing the step. Each orientation angle is set using the Gesture Editor. Since the limbs affected by *Foot Position*, *Footwork*, and *Head Position* form independent sets, a simple merge of these sets defines the position of all segments of the figure with the exception of the upper and lower torso. Applying the rotation specified by *Amount of Turn* to the upper and lower torso produces the final orientation of the figure model. This entire procedure is outlined in more detail in Section 5.4.

Commands at the bottom of the screen access on-line help, set the view angle and view point, and return control to the Top Level. As before, each of these commands is accessed by moving the pointer into the command box and clicking the right button.

The figure is displayed from one of six view points - front (default), rear, left, right, top, and bottom. The active view point is changed in a cyclical manner by continually invoking the *View Point* command. View points not on the above list are set using the *View Angle* command. First, the pointer is positioned over one of the three angle identifiers located under the step sheet. Clicking the right mouse button while pointing at an angle selects the direction to modify (which is displayed with a highlighted color). Dragging the mouse across the pad while holding down the left button sets the angle. While the angle is set, the human figure is displayed from the latest view angle. Repositioning the pointer over the angle identifier and repressing the right button terminates the operation.

The final command in the Dance Library and Pattern Editor pull-down command menu is *Delete Pattern*. This command deletes a single pattern from a dance. Upon invocation, *BDAS* provides a list of patterns to delete and requests confirmation of the action. The selected pattern is removed following an affirmative confirmation.

5.4. Gesture Editor

The second major component of the animation model is the Gesture Editor. This module is used to assign a positional meaning to the dance notation symbols supported by *BDAS*. Prior to entering this editor, a pop-up menu appears requesting the symbol category to edit. Entries in this menu represent the symbol categories defined in the Pattern Editor that directly affect the position of specific body segments. These categories include *Head Position*, *Foot Position*, *Body Modifier*, and *Footwork*. Once a category is selected, control is transferred to the Gesture Editor.

5.4.1. Overview

Figure 5.5 shows the screen layout for the Gesture Editor. The large window contains a graphical representation of a human figure similar to the one shown in the Pattern Analyzer. Directly below this window is a row of symbol names belonging to the symbol category selected prior to entering the editor, and a row of general purpose commands. A menu consisting of 16 boxes is located to the right of this window.

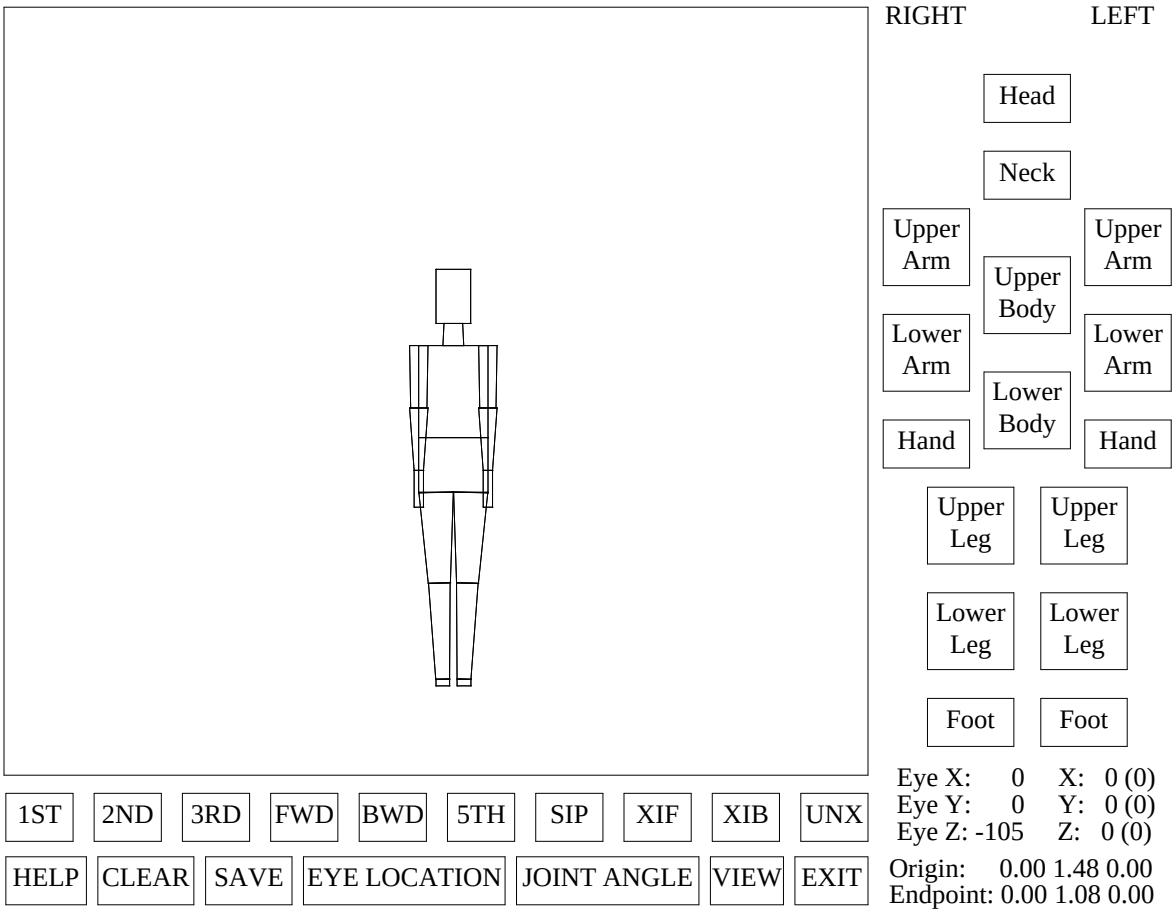


Figure 5.5 Gesture Editor Screen Layout

Each box in this menu corresponds to a body segment. The boxes are arranged to form a skeletal outline of the human figure with each box titled with a body segment name. This allows easier mapping between boxes and body segments. In addition to the current inertial frame eye angles, a listing of the current pitch (*X*), yaw (*Y*), roll (*Z*) angles and the location of the origin and endpoint for the active limb is found in a table below the limb menu.

A position is defined by first pointing at a box corresponding to a notation symbol and pressing the mouse button. This symbol becomes active and is marked active by having the symbol box displayed with a background color. If a position is already associated with the symbol, the display figure is updated to reflect the position; otherwise the figure is displayed with all internal limb angles set to zero. Only one

symbol is allowed to be active at any time.

A similar procedure is performed to activate a limb from the limb menu. Once a limb becomes active, its orientation is set by modifying the *X*, *Y*, and *Z* rotation angles. These angles are set by pointing at one of the angles listed in the angle table and clicking the right mouse button. The affected angle is displayed with a highlighted color. Its value is changed by dragging the mouse across the pad. Angle values increase for left to right movements, and decrease for movements in the opposite direction. As the mouse moves, the rotation angle values are continually updated. The angles outside the brackets display the current orientation of the limb with respect to the frame of its parent. Angles inside the brackets display the current orientation of the limb with respect to the inertial (world) frame. In both cases, the limb's orientation is obtained by first applying roll (a rotation about the frame's *Z* axis), then yaw (a rotation about the rotated *Y* axis), and finally, pitch (a rotation about the doubly rotated *X* axis).

Changing the orientation angles of a segment affects the orientation angles with respect to the inertial frame of all dependent (or child) segments. For example, rotating the left upper leg of the figure results in rotation of the left lower leg and left foot. The parent/child relationships between the body segments are explained further in the next section.

While the angle is set, both the location of the limb's origin and endpoint and the orientation of the human figure model are continuously updated and displayed on the screen. Realigning the pointer over the angle identifier and pressing the right mouse button terminates the angle setting operation. Similar angle setting operations are performed until the figure reaches the desired position for the active symbol.

When defining the symbol position, the animator must carefully select the limbs affected by the symbol. In particular, the sets of affected limbs for each symbol category should be disjoint. Otherwise, unexpected results may occur when the symbols are combined to form a step. A step consists of the addition of the orientation angles given by the symbols representing the *Foot Position*, *Footwork*, *Body Modifier*, and *Head Position* categories of the step. Because these additions are performed for each limb, the sets must be disjoint for the assigned orientations to be maintained.

Commands at the bottom of the screen access on-line help, zero all internal joint angles, save the current configuration, set the eye location, set the joint angle of the active segment, set the view point, and return control to the Top Level. The eye location refers to the imaginary distance from the viewer's eye to the display figure. This distance is set by positioning the pointer over the eye distance settings under the limb menu and using the dragging operation to change the *X*, *Y*, and *Z* values. As with the previous screens, all bottom commands are accessed by moving the pointer into the command box and clicking the right mouse button.

5.4.2. Internal Structure

Throughout the Gesture Editor and *BDAS*, the human figure is represented by the tree-like structure shown in Figure 5.6.

This tree consists of a series of nodes connected by arcs. Nodes represent body segments while arcs represent links connecting segments. Each body segment, with the exception of the upper body, has a parent segment and possibly one or more child segments. All arrows point from child segments to parent segments.

Each segment is represented using a local coordinate system. These coordinate systems have their origin set to the proximal hinge of the segment (the point where the segment connects to its parent). Although quantities such as mass and inertia are not used within the Gesture Editor, the set of axes used by each frame are the segment's principal axes. The inertial frame is defined using a separate coordinate system.

Each segment is associated with an internal structure containing the length of the segment and the three Euler angles defining the segment's orientation with respect to its parent. The three Euler angles for the upper body represent the orientation of the upper body with respect to the inertial frame.

The Euler angles defining segment orientation with respect to the inertial frame are calculated in an inward pass towards the root of the tree. Let ψ_r, θ_r, and ϕ_r define the Euler angles used in the rotations

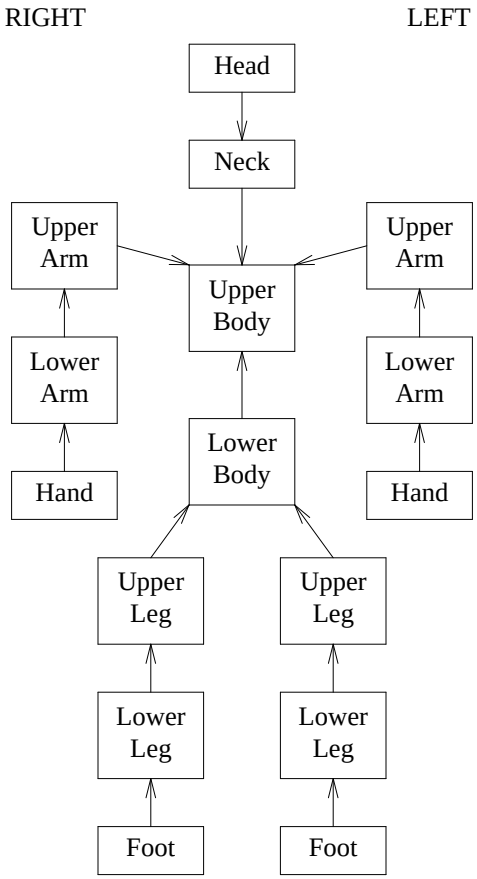


Figure 5.6 Tree Structure of the Human Figure Model

about the *Z*, rotated *Y*, and doubly rotated *X* axis to obtain the orientation of segment r with respect to its parent segment $r-1$. The matrix representing the transformation of column vectors from the frame of segment r to the frame of segment $r-1$ is given by the matrix product:

$$R^r = R_z(\psi_r)R_y(\theta_r)R_x(\phi_r),$$

where R^r has the form of the matrix R defined in Section 3.1 and the three matrices on the right hand side of the equation are the standard orthogonal rotation matrices, also defined in Section 3.1.

The transformation matrix giving the orientation of segment r with respect to the inertial frame is obtained from the matrix product:

$$R_I^r = R^1 \; R^2 \; \ldots \; R^{r-1}R^r,$$

where R^1 transforms vectors from the frame of the root segment to the inertial frame. Once the

transformation matrix R_I^r is known, the Euler angles defining the orientation of link r with respect to the inertial frame are found by applying the rotation matrix to Euler angle decomposition algorithm outlined in Section 3.1.

Calculating the inertial frame representation of the origin and endpoint of a segment (these points refer to the endpoints on the principal axis passing through the length of the segment) requires noting a point $\mathbf{p}^r$ in segment r can be expressed in the frame of the segment $r-1$ by:

$$\mathbf{p}^{r-1} = R^r \mathbf{p}^r + \mathbf{O}_r^{r-1},$$

where $\mathbf{O}_r^{r-1}$ is the location of the proximal hinge of segment r expressed in the frame of segment $r-1$. The inertial frame coordinates of the origin and endpoint of the segment are obtained by repeating this procedure until the inertial frame is reached.

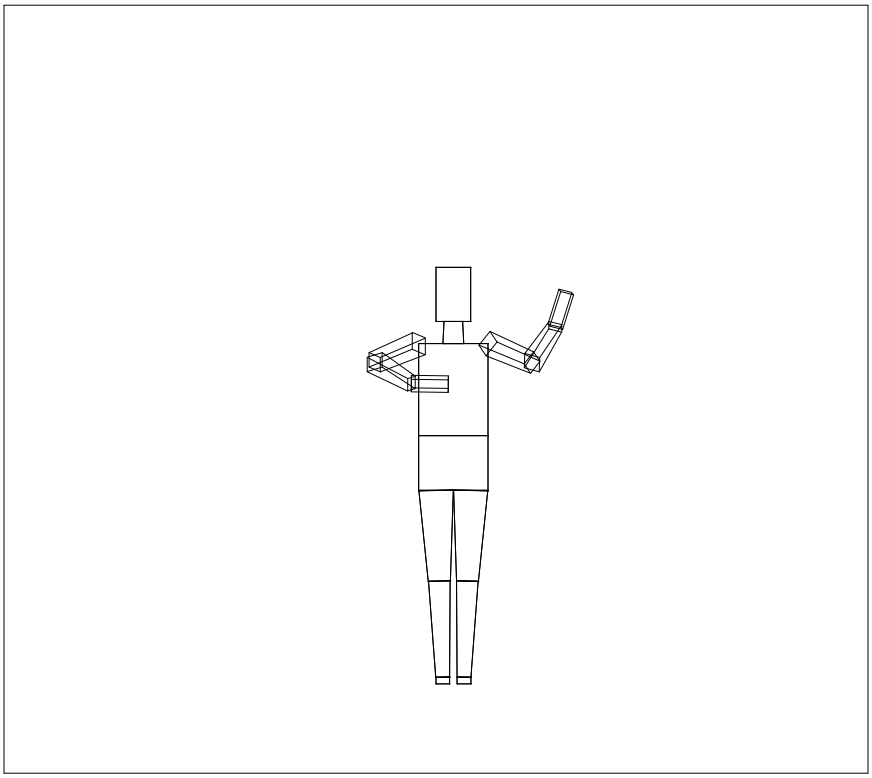
5.5. Dynamics Module

All ballroom dance animation is performed within the Dynamics Module. The primary purpose of this module is to read and interpret the patterns performed by the dancer, oversee the dynamics computations, and update the figure display. The human figure control model, described in the next chapter, is a subcomponent of this module.

5.5.1. Overview

Figure 5.7 shows the screen layout for the Dynamics Module. As with the Gesture Editor and Pattern Analyzer, the large window contains a graphical representation of a human figure. Directly below this window is a collection of commands used to access on-line help, play back the last animation sequence either as a collection of frames or one frame at a time, set the view point, read a dance pattern, execute a dance pattern, and return control to the Top Level.

A parameter table is located immediately to the right of the figure display window. This table displays the values of several variables which affect the dynamics computations. All these parameters, with the exception of *Elapsed Time*, may be set by pointing at the parameter name, pressing the right mouse



Fox Trot	
Box Step	
Step No.	1
Foot Position	LF FWD
Head Position	NP
Footwork	H
Amt. of Turn	
Beats	2

Gravity: 9.81
Normal Force: 1.0
Floorspring: 40
Floordamper: 5
Mu: 0.6

Mainspring: 50000
Maindamper: 80
Movespring: 10000
Movedamper: 80

Slowfreq: 2
Deltat: 0.0010
Nbps: 1.0
Time Limit: 2.0
Elapsed Time: 0.0

HELP | PLAYBACK | SINGLE STEP | VIEW | PATTERN | MOVE | EXIT

Figure 5.7 Dynamics Module Screen Layout

button, and using the dragging operation to change the parameter value. The function of each of these variables is explained in the remaining sections of this chapter.

A small window above the parameter table displays the current dance pattern and step number. This window only appears while the dynamics computations are in progress.

An animation sequence is generated by using the *Pattern* command to select a dance and pattern to execute. This, in effect, simulates the behavior level of the motion control model. Eventually, when the Vocabulary Editor and Environment Editor are complete, all dances and patterns will be selected automatically by the behavior module of the system. At this time, however, all patterns must be specified by the animator and only one pattern is executed per animation sequence.

Prior to starting the animation sequence, a time limit may be set by modifying the parameter table *Time Limit* variable. The step execution speed may be varied by setting *Nbps*, the number of beats per second.

After modifying the parameter table values, the pattern is executed by invoking the *Move* command. This starts the dynamics computations and activates the motion control model described in Chapter 6. Throughout the execution of the pattern, the dance pattern and step window provides feedback showing progress made in the execution of the steps. While this is happening, updates are made to the figure display at regular intervals. The elapsed time is regularly updated in the *Elapsed Time* parameter table entry. Once the time limit is reached, all previously displayed frames may be played back either in succession or one frame at a time.

5.5.2. Internal Structure

The Dynamics Module solves the equations of motion presented in Chapter 3 using the Armstrong-Green solution method [Armstrong85a] [Armstrong85b]. Fastband computations are executed every *Deltat* seconds and slowband computations occur once every *Slowfreq* executions of the fastband computations. Before beginning an animation sequence, several variables and constants are initialized. Physical properties of each body segment are set using data from anthropometric studies of human figures [Hanavan64] or data derived mathematically [Lien84]. These properties include scalar, vector, and matrix quantities giving the length, mass, center of mass, and moment of inertia for each segment.

The human figure moves in an environment consisting of an infinitely long dance floor. No provision is made for collision detection between two or more limbs. This implies that under certain circumstances (such as when the figure falls onto the floor or interpolates a step incorrectly) limbs can freely pass through each other.

Prior to starting the dynamics calculations, the motor control model reads the selected dance pattern. At the beginning of every dance step, the model assigns a goal position to each limb and activates a set of

low-level motor processes. Two of these processes use a spring and damper combination to maintain a limb at its current position or move a limb to a new position. The strengths of these low-level springs and dampers are determined by the parameter table values *Mainspring*, *Maindamper*, *Movespring*, and *Movedamper*. These and other motor processes are discussed in the next chapter.

Once the dynamic computations are underway, the figure display window is updated every 0.1 seconds of simulated time. Each time this window is updated, all segment joint angles are saved for future playback purposes. After updating the figure's position and rotation matrices during each slowband cycle, the Dynamics Module makes two important procedure calls. The first call is to a procedure which models a dance floor and is responsible for maintaining the figure on the floor. This procedure, discussed in Section 5.5.3, generates upward restorative forces on the figure and simulates horizontal frictional forces. The second procedure referenced is the motion control model, discussed in Chapter 6.

5.5.3. Ground Model

Gravity continuously exerts a downward force on the human figure. Unless a restraining floor is implemented in the model, the human figure soon disappears from view below the screen. The animation system requires a ground model to restrain the downward velocity of the figure and make the simulated environment more realistic. However, designing a reasonable floor model which simulates reaction forces such as horizontal friction is a non-trivial problem.

Several difficulties arise when trying to simulate ground reaction forces. Ensuring that each limb of the human figure is always above or on the floor is impossible when the time sampling is discrete. A limb positioned 1 millimeter above the floor may suddenly be several centimeters below the floor at the next time sampling. Ground reaction forces must stop the descent of a body before it descends too noticeably below floor level. The magnitude of the restorative upward force must be set so that, for small ground displacements, the body is not shot far upwards into space. Because the body is constantly oscillating between this upward and downward motion, the time sampling interval and restorative upward force must ensure that this bouncing of the body is not visible while the body is in a stationary position.

The implementation of horizontal frictional forces also presents difficulties. Friction is applied only when an object contacts the ground. Considerable excess tangential slippage can occur for sliding motions if the frictional model is not applied when the ground reaction forces have temporarily pushed the object off the ground.

Other problems with implementing a ground model involve calculating the normal force and how it should be distributed throughout the body. A body pressed into the floor, or hitting the floor with a high velocity, has a greater normal force than one resting on the floor. Although it is relatively simple to determine the contact points for a body, assigning a distribution of force to these points depends upon the position of the body and where the gravitational forces are being directed. For example, forces assigned to the contact points differ when a human is crouched on all four limbs versus when the human has most of the weight on the legs and is lightly touching the floor with the hands.

The floor model implemented in *BDAS* is based on the spring and damper ground model and floor contact algorithm used by Wilhelms in *Deva* [Wilhelms85]. At the beginning of each iteration, floor contact is checked for the eight points defining the corners of each limb by comparing their inertial frame coordinates with that of the floor level. It is especially important to check all corners of the feet for floor contact because many ballroom dance steps are performed by stepping down either heel first or toe first.

If the figure contacts with the floor, the normal force is estimated by first computing the mass of the figure times the acceleration due to gravity. The vertical momentum of the body divided by a time constant is then added to this product, and the entire result is negated. Mathematically, this is expressed as:

$$\mathbf{W}_{body} = -m_{body} * \mathbf{a}_G - \frac{m_{body} * \mathbf{v}_{down}}{time\_constant}.$$

Each point is marked indicating contact or no contact with the floor. The current depth of all marked points are summed to determine the total depth. Once the total depth is known, each point marked in contact with the ground is allocated a percentage of the normal force according to the following formula:

$$\mathbf{N}\% = \mathbf{W}_{body} * depth\%.$$

This amount is only an estimate of the force required to stop the downward motion. If the point continues to descend below its original contact point, a restorative spring force is applied:

$$\Delta y = Contact\_Point - Current\_Position$$

$$\mathbf{F}_{spring} = \mathbf{N}\% * Floorspring * \Delta y.$$

A damping term is also included:

$$\mathbf{F}_{damper} = m_{body} * \mathbf{v}_{down} * depth\% * Floordamper,$$

where $\mathbf{v}_{down}$ is the contact point's inertial frame velocity and *Floordamper* is the frictional damping component of the spring and damper representing the floor (this is not to be confused with horizontal friction).

The total normal force applied to the contact point is the vector sum of these forces:

$$\mathbf{N}_{total} = \mathbf{N}\% + \mathbf{F}_{spring} - \mathbf{F}_{damper}.$$

Two mutually perpendicular horizontal frictional forces are applied to each contact point by multiplying the total normal force for the point by the coefficient of friction (often about 0.6 [Halliday74]). This gives:

$$\mathbf{F}_{hor} = \mu * \mathbf{N}_{total}.$$

The sign of each horizontal frictional force is set to oppose the direction of both horizontal velocity vector components.

The Dynamics Module allows the animator to adjust several parameter table variables used in the ground model. *Gravity* sets the downward acceleration of gravity acting on the human figure. This value may be changed to model conditions on other worlds. *Normal Force* sets the time constant determining if the normal force is strengthened or weakened for each contact point. *Floorspring* sets the stiffness of the floor and *Floordamper* determines the amount of friction applied to the floor spring. Each setting, in turn, affects the magnitude of horizontal friction applied to all contact points. *Mu*, the coefficient of friction, determines the amount of friction associated with the floor surface.

Chapter 6

Motor Control

Studies of biological motor systems suggest control programs should be organized in a hierarchical manner where the high levels constitute some form of a task description, and the low levels consist of a series of control modules capable of activating a sequence of muscles and joints to perform a particular motion. Many scientists have argued that such a control structure may be the only efficient means of controlling complicated systems such as those represented by articulated, three dimensional human figures [Zeltzer82a].

Figure 6.1 shows the multi-level approach used to define the motor control structure implemented in the model. This hierarchical structure is similar to the skeletal control model used by Zeltzer [Zeltzer82a] in his studies of human gait. The upper levels, which function near the animator level, transform a series of high-level task descriptions (such as dance patterns) into a sequence of low-level primitive movements. The lower levels consist of biological motor programs responsible for executing small, well-defined primitive movements. All low-level motor programs operate under the control and supervision of the upper levels.

6.1. Upper Levels

The task manager constitutes the top level of the motor control structure. Its function is to oversee dance pattern execution and to periodically report motion difficulties to the behavior level. The task manager receives a pattern from the behavior level and decomposes it into a series of steps. Each step defines a new body position and orientation. All steps are placed into a *step queue* for sequential execution by the lower levels of the model.

The task manager is called each a time a new step is executed. If the step queue is empty, the task

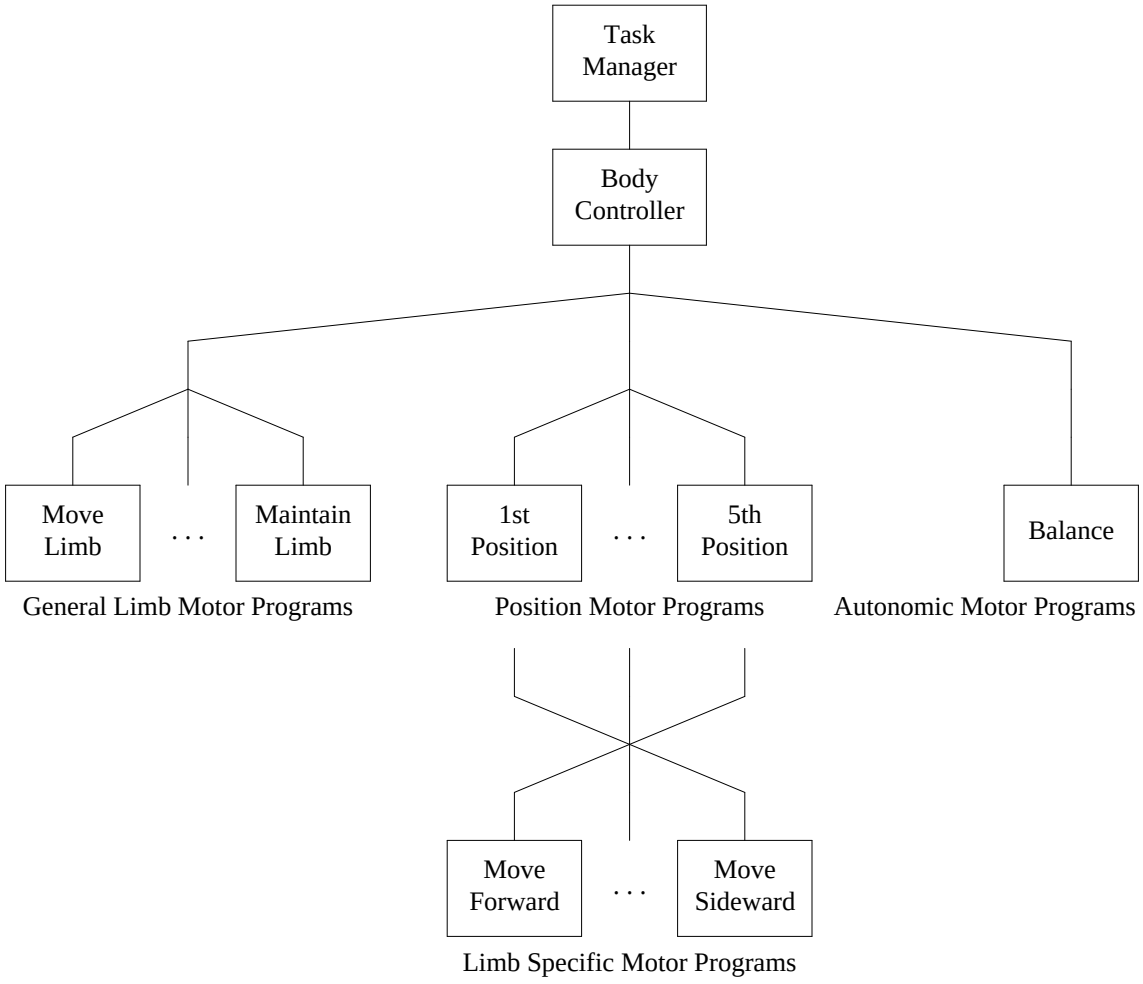


Figure 6.1 Motor Control Structure

manager notifies the behavior level of pattern completion and awaits arrival of the next pattern. When the step queue is not empty, the next step is removed and information is extracted describing the time required to complete the step and the final body position and orientation. Each link is then assigned a quaternion-defined goal position. This quaternion is obtained by converting the Euler angles defining the limb's orientation at the end of the step into a normalized quaternion. The normalized quaternion represents the orientation of the link with respect to its parent (or the inertial frame in the case of the upper body). The rotation matrix giving the link's current orientation with respect to its parent is also converted into a normalized quaternion and the link's motion state is determined by measuring the distance between these two quatern-

ions.

A limb may be in one of three motion states. *Free swing* is a null state where all internal torques are set to zero. This state allows a limb to move freely without motion constraints. *Move* results in the application of torques to move the limb as smoothly as possible from its current position to the new goal position within the specified time limit. *Maintain* attempts to hold a link at its current angular position with respect to its parent. Relatively strong restorative torques are applied to the link whenever it deviates from this position. Maintaining a link's position by implementing a firm clamp on the link is not the best solution for two reasons. First, from a biological point of view, this results in unnatural motion. Most limbs react to sudden external forces by "giving" a bit before moving back to their normal position. A limb should move out of its maintained position if a strong enough external force is applied. Second, a clamp adds a constraint to the dynamics equations that would require their reformulation.

Unless a link is explicitly set to *free swing*, links whose quaternion distance exceeds a system-set ε have their motion state set to *move*. Otherwise, all links whose quaternion distance is less than ε have their state set to *maintain*.

Based on the number of beats per second and the number of beats required to perform the step, the task manager calculates the time necessary to complete the step and initializes a timer for each link whose state has been set to *move*. All timers contain information giving the time the movement was initiated, the length of time required to complete the movement, and the time an interrupt alarm will go off. These interrupt alarms are used by the body controller to synchronize the motion of the moving limbs.

When the task manager has set the states, goal positions, and timers for each link, control is passed to the body controller. The body controller is responsible for activating and supervising the execution of the motor programs required to perform each step. In contrast to the task manager being executed at the beginning of each step, this module is executed many times during the step.

During its execution cycle, the body controller receives the rotation matrices describing the current position of each link marked *move*. These matrices are converted to normalized quaternions and the

distance between each link's current position quaternion and its goal position quaternion is computed. If the distance between these two quaternions is less than ε, the body controller changes the state of the limb to *maintain*. This module then services all pending timer interrupts for the limbs marked *move*. Each limb's interrupt handler compares the current position of the limb to its goal position. Adjustments are made to the limb's torque generating function if the limb's movement has been lagging or proceeding too quickly relative to where it should be since the movement began (the criteria for deciding where a limb should be at a given time are described in Section 6.4). The timer is then reset to produce another interrupt after a fixed time interval. After servicing all timer interrupts, the body controller initiates human figure movement by executing a series of low-level motor programs.

6.2. Low-Level Motor Programs

The human figure is driven by three types of low-level motor control programs. *General limb* motor programs act on the limb according to the state of the limb. *Position* motor programs apply internal torques to assist moving the limbs to the goal position defined for a specific dance position. *Autonomic* motor programs simulate functions humans perform either subconsciously or automatically.

6.2.1. General Limb Motor Programs

Every link is attached to a set of general limb motor programs. This motor program set consists of three motion processes - *free swing*, *move limb*, and *maintain limb*. Every link has one motor program from its set active at all times. The active program is determined from the current motion state of the limb.

6.2.1.1. Free Swing Motor Program

The *free swing* motor program is a null process which removes all internal torques from the link, thus allowing the limb to move freely without constraints.

6.2.1.2. Move Limb Motor Program

The *move limb* motor program attempts to move the limb as smoothly as possible from its current

position to the goal position within the specified time limit by applying a series of internal torques.

Prior to assigning an internal torque to the limb, a transformation quaternion is calculated based on the quaternions giving the limb's current orientation and goal orientation. Let $[\lambda_c, \Lambda_c]$, $[\lambda_g, \Lambda_g]$, and $[\lambda_t, \Lambda_t]$ define quaternions representing the current orientation, goal orientation, and transformation from the current to goal orientation. The transformation quaternion is computed by first calculating the minimum distance from the current orientation to the goal orientation:

$$d([\lambda_c, \Lambda_c], [\lambda_g, \Lambda_g]) = \min(d([\lambda_c, \Lambda_c], [+\lambda_g, +\Lambda_g]), d([\lambda_c, \Lambda_c], [-\lambda_g, -\Lambda_g])).$$

Both $[+\lambda_g, +\Lambda_g]$ and $[-\lambda_g, -\Lambda_g]$ must be considered because these two quaternions represent the same orientation. $[\lambda_g, \Lambda_g]$ is then set to the closest goal quaternion. $[\lambda_t, \Lambda_t]$ is computed by multiplying each side of the equation:

$$[\lambda_c, \Lambda_c][\lambda_t, \Lambda_t] = [\lambda_g, \Lambda_g]$$

by the left inverse of $[\lambda_c, \Lambda_c]$ to produce:

$$[\lambda_t, \Lambda_t] = [\lambda_c, \Lambda_c]^{-1}[\lambda_g, \Lambda_g].$$

$[\lambda_t, \Lambda_t]$ gives the least amount of rotation and the axis of rotation required to reach the orientation defined by the goal quaternion from the orientation defined by the current quaternion. Let $\mathbf{n}_t$ represent the normalized vector pointing along the rotation axis specified by Λ_t.

The internal torques applied to the link are computed using functions obtained from biomechanical studies on muscle contraction. The functions are similar to formulas used to express the force acting across parallel elastic muscle elements [Hatze81]:

$$\tau_{tot} = \tau_s - \tau_d$$

where

$$\tau_s = \alpha\,(e^{\beta\,\delta} - 1)$$

and

$$\tau_d = \gamma\,\omega.$$

τ_s and τ_d are non-linear springs and linear dampers consisting of five parameters. α, β, and γ are

constants set by the animator or animation system. δ is the quaternion distance between the current position and the goal position. ω is the angular velocity of the link (with respect to its parent). Assigning α and β from the domain of non-negative real numbers produces a family of exponential curves for τ_s, all having value zero when δ is equal to zero. α serves as a scalar multiplier and β controls the shape of the curve.

These functions are used empirically to obtain reasonable torque values for a given motion; no effort is made to model actual muscle contraction. Since different muscles can be modeled using different force and torque generating functions [Hatze81], a more complete motion control model should activate the appropriate set of torque generating functions for the muscles involved with the movement being performed.

The principal axes for each frame are used when evaluating the torque functions. For each axis *i*, α_i is determined by:

$$\alpha_i = I_{ii} * n_i * Movespring * \frac{1}{Movetime^2}.$$

I_{ii} is the moment of inertia value for the link along axis *i*, n_i is the component of the normalized vector $\mathbf{n}_t$ pointing along axis *i*, *Movespring* is a constant set by the animator, and *Movetime* is the amount of time required to perform the movement.

The initial amount of torque required to rotate a limb is based on the square of the inverse of *Movetime*. Since many limb movements start from a stationary position, the rotational distance θ_i about each principal axis (assuming a constant angular acceleration $\dot{\omega}_i$) can be expressed as a function of time by:

$$\theta_i = \frac{1}{2}\dot{\omega}_i t^2.$$

For limb movements with an initial angular velocity of zero, $\dot{\omega}_i$ is inversely proportional to t^2. Since $\tau_i = I_{ii}\dot{\omega}_i$, a similar relationship exists between τ_i and t^2. *Move limb* uses this inverse square relationship to obtain reasonable torque approximations for identical movements performed at differing time rates.

All three axes use the same value for β. γ_i is calculated from:

$$\gamma_i = I_{ii} * Movedamper,$$

where *Movedamper* is a constant set by the animator. This value is multiplied by ω_i to obtain the retarding frictional torque τ_d acting along the principal axis. Both γ and the previously computed value of α depend upon the rotational inertia *I* of the link. This allows torque values with similar constants to have similar effects on each of the principal axes.

In addition to calculating τ_s and τ_d for each principal axis, a gravitational torque term is applied to τ_{tot}. This term is determined by converting the vector giving the downward acceleration due to gravity from the inertial frame to the frame of the link. The vector components resulting from the cross product of the link's center of mass vector with the converted gravitational force vector are then subtracted from τ_{tot}. This gravitational term is necessary because the torque function would otherwise be unable to move a limb against gravity to its goal position. Movement would stop at the point where the upward torque generated by τ_{tot} matched the downward torque applied by gravity.

A vector sum of the components of τ_{tot} acting along each of the principal axes produces a torque pointing in the direction of $\mathbf{n}_t$ (the axis of rotation transforming the current quaternion to the goal quaternion) operating along an axis of rotation specified by λ_t. These internal torques perform a spherical interpolation of the shortest great circle arc between the current quaternion and the goal quaternion.

One problem with this interpolation method is that some great circle arcs contain orientations outside the rotation range of normal human limbs. However, if the shortest natural arc or series of arcs giving the shortest natural path can be found, Bezier curves can be constructed and spliced together to form a smooth interpolation path [Shoemake85]. In this case, the interpolation method consists of reaching a series of sub-goal quaternions along these curves. The main difficulty with this method is finding a satisfactory and efficient means of detecting if a quaternion is outside the limit of the limb's rotational range. This problem is ignored by *BDAS* since the shortest great circle arc between two positions is nearly always within the range of natural movement.

6.2.1.3. Maintain Limb Motor Program

The *maintain limb* motor program attempts to maintain the limb at the angular position (with respect to its parent) specified by its goal quaternion. This motor program operates similarly to the *move limb* process, except α and γ depend on the animator specified values *Mainspring* and *Maindamper*, β is set to a value higher than its *move limb* counterpart, and interrupt alarms are not used.

6.2.2. Position Motor Programs

The second class of motor programs driving the figure are *position* motion processes. These processes help the general limb motor programs move the human figure to a specific dance position. Position motor programs are attached to dance positions, and not all dance positions have these motor programs. Unlike the general limb motor programs, these processes are capable of simultaneously acting on more than one body part. Furthermore, the entire human figure has access to only one set of position motor programs rather than one set per limb.

Position motor programs operate by application of internal torques on the body during specific time intervals. Because the length of time required to perform a step depends on the tempo of the music, time intervals are expressed as percentage intervals of the step rather than in seconds. For example, a time interval for the application of a particular torque to raise a leg may be specified as the interval defining 40 to 60 percent completion of the step as illustrated below:

```
step_elapsed_time = (elapsed_time – step_start_time) / step_move_time;
if (step_elapsed_time >= 40% && step_elapsed_time <= 60%)
        raiseleg(left_or_right_leg, torque_magnitude);
```

During their execution, position motor programs may change the state of various limbs. For example, an implementation of the ballistic walking model described by McMahon [McMahon84a] [McMahon84b] may require a leg marked *move* be temporarily changed to *free swing* during the gait cycle executed by the 4th position forward motion process.

Position motor programs apply torques to the limbs by executing a set of *limb specific* motor pro-

grams. Limb specific processes form a pool of primitive motion functions available to all position motor programs. Each limb specific motor program applies a specific internal torque to a particular body limb. These motor programs are responsible for performing such actions as making the human figure move in a particular direction or temporarily raising a leg off the ground.

Nearly all limb specific motor programs are invoked with the magnitude of the applied force or torque given as a parameter from the position motor program. Other execution parameters may include the force or torque direction and the limb the vector acts on. Most position specific motor programs require the active leg (left or right) be given as an execution parameter. All execution time intervals are built into these processes.

At this time, all force and torque magnitudes used by these motor programs are derived empirically and are totally frozen within these processes. Although these values generally give realistic motion only for the more common dance tempos, more sophisticated algorithms need to be developed and incorporated into these programs to allow proper handling of a wider range of tempos.

6.2.3. Autonomic Motor Programs

Autonomic motor programs simulate functions humans perform either subconsciously or automatically. These processes generally operate independently of the dance related motion sequence. Currently the only autonomic motor program implemented is *balance*.

Balance is based on a comparison between the vertical orientation of the upper body to a general upright orientation. A restorative torque, whose magnitude is dependent upon the vector distance between these two orientations, is applied to the upper body whenever the distance is non-zero. Setting an upper limit on the magnitude of this restorative torque allows for gravitational torques to cause the body to fall over whenever the displacement exceeds a certain limit (typically a distance representing about 20 degrees from an upright position).

The figure can be made to stand upright by applying stiffness to the lower body, legs, and feet. Most

of this stiffness occurs in the form of the internal torques generated from setting the motion state of these limbs to *maintain*.

While this procedure has generally been satisfactory for ballroom dancing (since most patterns require an upright upper body orientation), it is not satisfactory for motion in general. Different types of motions (such as diving or bending over) require different types of balance. A more realistic model of balance should account for the limbs which contact the ground and the distribution of mass over those limbs.

Although autonomic motor programs generally operate at all times while dance patterns are processed, they can be temporarily deactivated by position motor programs. For example, since walking is based on falling forward [Alexander84a], the 4th position motor process, at the beginning of its execution, temporarily disables the *balance* autonomic motor process. At a slightly later time, balance is restored to allow the figure to catch itself from falling over.

6.3. Intermediate Goal Positions

So far the lowest level of the dynamics component of the animation system has consisted of a series of motor programs responsible for reaching goal frames. The general limb motor programs move the limbs to the desired orientation and the position motor programs handle the special forces and torques required to assist the general limb processes.

Transitions to some goal positions may require an extensive set of procedures in the corresponding motor programs because of the complicated nature of the movement. For example, consider the transition from 1st position into 4th position in the forward left step. Depending on the dance, the final position may resemble either the midpoint of the gait cycle (as in the waltz) or a position similar to 1st with a slight bend in the right knee causing the right foot to be suspended approximately half an inch off the ground (as in the foxtrot). The 4th position motor program must be capable of handling different dances in different manners and, in the case of the foxtrot, have sufficient knowledge to prevent the general limb motor programs from reaching the goal position by simply bending the right knee.

This problem can be solved by allowing complicated transitions to consist of a series of smaller transitions to one or more *intermediate* goal positions. These intermediate positions help eliminate any ambiguity that exists in the transition from the initial to final position. The problem outlined in the preceding example is handled by defining 4th position to consist of an intermediate and final goal position. These two positions represent the position at the midpoint of the gait cycle and the position defined by the bending of the right knee. The 1st to 4th position transition for the foxtrot then consists of sequentially reaching the intermediate and final goal position. The waltz transition, on the other hand, is performed by setting the intermediate position as its final position.

Intermediate goal positions are defined with the Gesture Editor in the same manner as a regular dance position. To simplify their implementation during the dynamics calculations, transitions to all intermediate goal positions are allocated equal time slices. If n represents the number of intermediate and final goal positions and t denotes the time required to reach the final goal position, all motor programs are allocated a time slice of t/n seconds to reach every intermediate and final goal position.

Since a dance step consists of combining the goal positions defined by the associated *Head position*, *Foot Position*, *Footwork*, *Body Modifier*, and *Amount of Turn* symbols, limbs associated with different symbols may reach their goal positions at different times. For example, suppose the dancer begins a step in 1st position looking straight ahead and is to move forward into 4th position while rotating the head 45 degrees to the left. If the entire step requires t seconds and the 4th position has one intermediate goal position, the legs and feet will have $t/2$ seconds to reach each goal position while the head will have t seconds to rotate its required amount.

Providing the animator with more control of movement and having the motor processes handle smaller movement sequences are two important factors favoring the use of intermediate goal positions. However, treating intermediate goal positions the same as final goal positions in terms of the magnitude of torques applied can lead to unnatural motion. Once again, consider the forward transition from 1st position to 4th position in the foxtrot. Since this sequence uses one intermediate goal position (the position resem-

bling the midpoint of the gait cycle), the torque functions apply little torque to the legs when the legs are near their intermediate goal. Thus, upon reaching their intermediate goal position at the completion of the allocated time interval, the legs are likely moving with little velocity. When the final goal position is read, the torques generated by the torque functions are once again large because of the increase in distance between the current position of the legs and their new goal position. This produces an acceleration of the limbs. The overall effect is a slow down followed by an acceleration of movement during the gait cycle.

The solution to this problem (which is not implemented in *BDAS*) lies with adding look ahead knowledge to the task manager and body controller so the torques used by the motor programs are based on the distance and time to the final position in the movement sequence. Further knowledge should also be incorporated as to *what* the motion is, so smooth transitions may occur between motion sequences.

6.4. Low-Level Motor Control

One of the major difficulties with using dynamics for human figure animation is correctly specifying the magnitude of the forces and torques required to produce a desired motion. Ballroom dancing introduces a further complication to this problem because not only should the motion appear smooth and natural, but it should be performed in a time frame governed by the tempo of the music. For example, motion at a rate of one beat per second should be performed half as fast as motion at a rate of two beats per second. In both cases, limbs should initiate movement at the beginning of the sequence and reach their goal position at the end of the allocated time period.

Low-level motion control is performed for each step by assigning a simple feedback system to each limb marked *move* [Grodins63] [Milhorn66] [DiStefano76]. These feedback systems perform spherical interpolations along the shortest great circle arc connecting the quaternion defining the starting orientation and the quaternion defining the goal orientation. All feedback systems are activated simultaneously and at least 10 times (at equal intervals) while an intermediate or final goal position is approached.

Let $[\lambda_s, \Lambda_s]$ and $[\lambda_g, \Lambda_g]$ denote the starting and goal quaternions, and $[\lambda_t, \Lambda_t]$ denote an interpo-

lation quaternion located at position u along the ($[\lambda_s, \Lambda_s]$, $[\lambda_g, \Lambda_g]$) shortest great circle arc. The domain of u includes all real numbers between 0 and 1 and $u = f(t)$ for an arbitrary function f. The following formula from four dimensional geometry provides a spherical linear interpolation along the ($[\lambda_s, \Lambda_s]$, $[\lambda_g, \Lambda_g]$) arc [Pletinckx89]:

$$[\lambda_t, \Lambda_t] = \frac{\sin(1-u)\theta}{\sin\theta}[\lambda_s, \Lambda_s] + \frac{\sin u\theta}{\sin\theta}[\lambda_g, \Lambda_g]$$

where $[\lambda_s, \Lambda_s]\cdot[\lambda_g, \Lambda_g] = \cos\theta$. Combining the relationship $u = f(t)$ (where t is the elapsed time expressed as a percentage interval of the step) with this formula allows intermediate orientations to be expressed as a function of time.

f is any increasing function satisfying the restrictions $f(0) = 0$, $f(allocated\_time) = 1$, and $0 \le f(t) \le 1$. Since time versus position plots of many simple motion sequences such as raising an arm produce curves resembling the distribution curve of the standard normal function, *BDAS* uses the latter as the curve expressing the desired intermediate orientations as a function of time. A more general method for approximating these and other motion curve shapes requires the construction of spline curves.

Each limb's feedback system monitors the amount of torque applied to the limb based on the limb's current position. Figure 6.2 displays the structure of a limb motor feedback system.

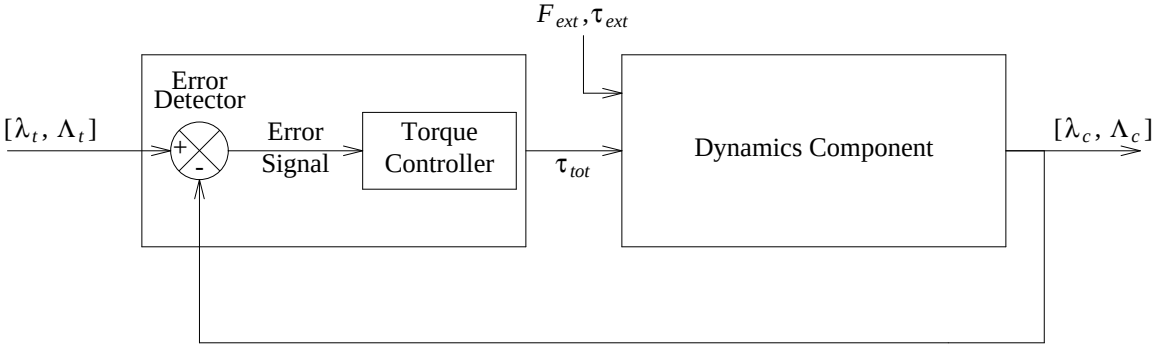


Figure 6.2 Limb Motor Feedback System

This system consists of two main components. On the left, the *controller* is responsible for controlling the torques applied to the component on the right, the *controlled system*.

The controller contains two subcomponents - the *error detector* and the *torque controller*. The current quaternion $[\lambda_c, \Lambda_c]$ and interpolation quaternion $[\lambda_t, \Lambda_t]$ (representing the desired orientation at time *t*) are fed as input signals to the error detector. An error signal β_e is generated by measuring the current quaternion distance from the goal against the interpolated quaternion distance from the goal:

$$\beta_e = d([\lambda_c, \Lambda_c], [\lambda_g, \Lambda_g]) - d([\lambda_t, \Lambda_t], [\lambda_g, \Lambda_g]).$$

The error signal is calculated using the above expression instead of measuring the distance between $[\lambda_c, \Lambda_c]$ and $[\lambda_t, \Lambda_t]$ because external disturbances may cause the limb to deviate from the $([\lambda_s, \Lambda_s], [\lambda_g, \Lambda_g])$ great circle arc.

This error signal is then used by the limb's torque controller to regulate the amount of internal torque applied to the limb by adjusting the β parameter of the limb torque generating function:

$$\beta = \max(\beta + \beta_e, 0).$$

A positive value for β_e produces an increase in the magnitude of the generated torques and a negative value results in either a decrease in the torque magnitude, or the magnitude remaining at zero. At the beginning of each step, all limbs whose state is set to *move* are assigned a β value of nearly zero to allow the torques to increase in a natural manner, thereby reducing the likelihood of a large displacement occurring between $[\lambda_c, \Lambda_c]$ and $[\lambda_t, \Lambda_t]$.

The *controlled system* is represented by the Dynamics Module. This component receives input from the torque controller and disturbances in the form of external torques and forces acting on the limb. At the conclusion of the dynamics computations the limb's new orientation is fed back to the error detector.

Interpolations along the $([\lambda_s, \Lambda_s], [\lambda_g, \Lambda_g])$ great circle arc are used primarily for regulating the application of internal torques required to move $[\lambda_c, \Lambda_c]$ to $[\lambda_g, \Lambda_g]$. This arc is also used for determining the magnitude and direction of the initial internal torques applied at the beginning of each step. Because external forces and torques may displace $[\lambda_c, \Lambda_c]$ from the $([\lambda_s, \Lambda_s], [\lambda_g, \Lambda_g])$ arc during the progression of the step, all subsequent internal torques are applied to move $[\lambda_c, \Lambda_c]$ along the $([\lambda_c, \Lambda_c], [\lambda_g, \Lambda_g])$ arc.

Chapter 7

Experimental Results

This chapter discusses the experiments performed and the problems encountered with the Ballroom Dance Animation System. All experiments were performed on an IRIS 3130 with a floating point accelerator. Unless otherwise stated, time samplings were performed every 0.01 seconds and all rotation matrices and limb positions were updated every 0.02 seconds of simulated time. Output to the screen appeared every 0.1 seconds of simulated time.

One of the major difficulties encountered with *BDAS* prior to conducting these experiments has been finding stable values for the spring and damper constants governing the motion of the limbs. Most values result in limb oscillations which cause the numerical instabilities introduced in the integration routines to destroy the simulation. This problem was addressed by using trial and error to find a reasonably stable set of values, and by increasing the moments of inertia for each link by a factor of 300.

7.1. Arm Reaches to Dance Position

The first set of experiments tested the general limb motor programs by having the figure perform an arm reach to dance position. Figures 7.1 to 7.4 show the animation sequences generated when the figure was given one, two, three, and four seconds to reach this position. Motion in all cases started with the figure standing in 1st position with the hands resting at the side. All limbs, with the exception of the upper arms, lower arms, and hands, were set to *maintain*. The remaining limbs were initially marked *move* and allowed to change to *maintain* once they reached their goal position. During each motion sequence, the *balance* process maintained the figure in an upright position.

In all sequences, the movement of the arms to reach the final position progressed smoothly and appeared very natural. All limbs either obtained or nearly reached their goal position by the end of the

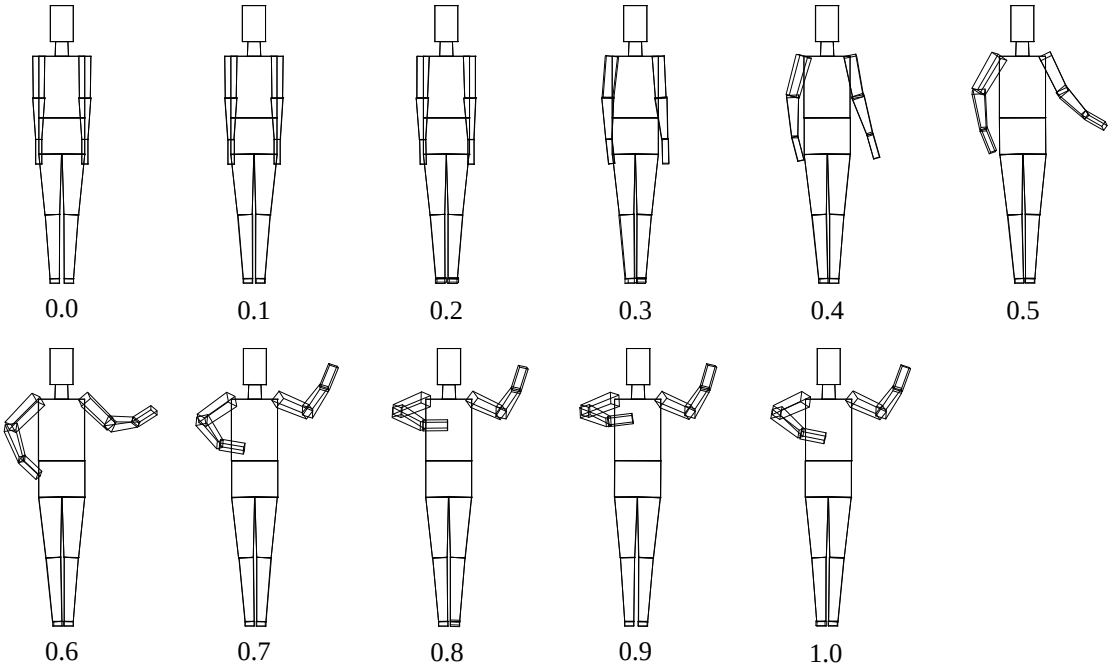


Figure 7.1 Reaching Dance Position in 1 Second

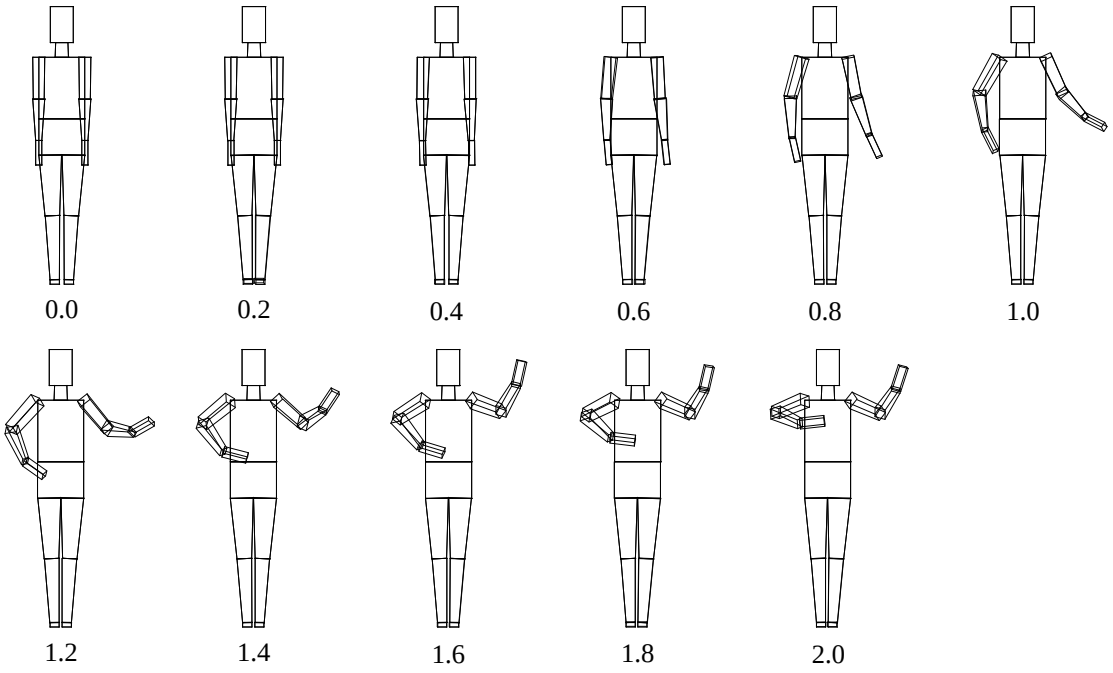


Figure 7.2 Reaching Dance Position in 2 Seconds

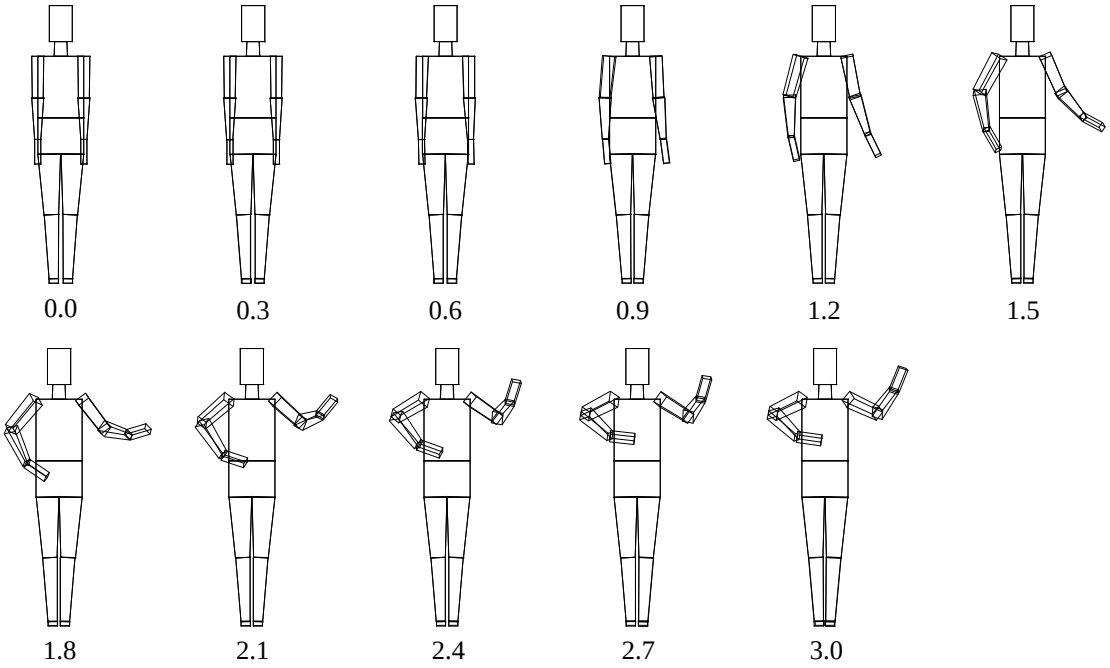


Figure 7.3 Reaching Dance Position in 3 Seconds

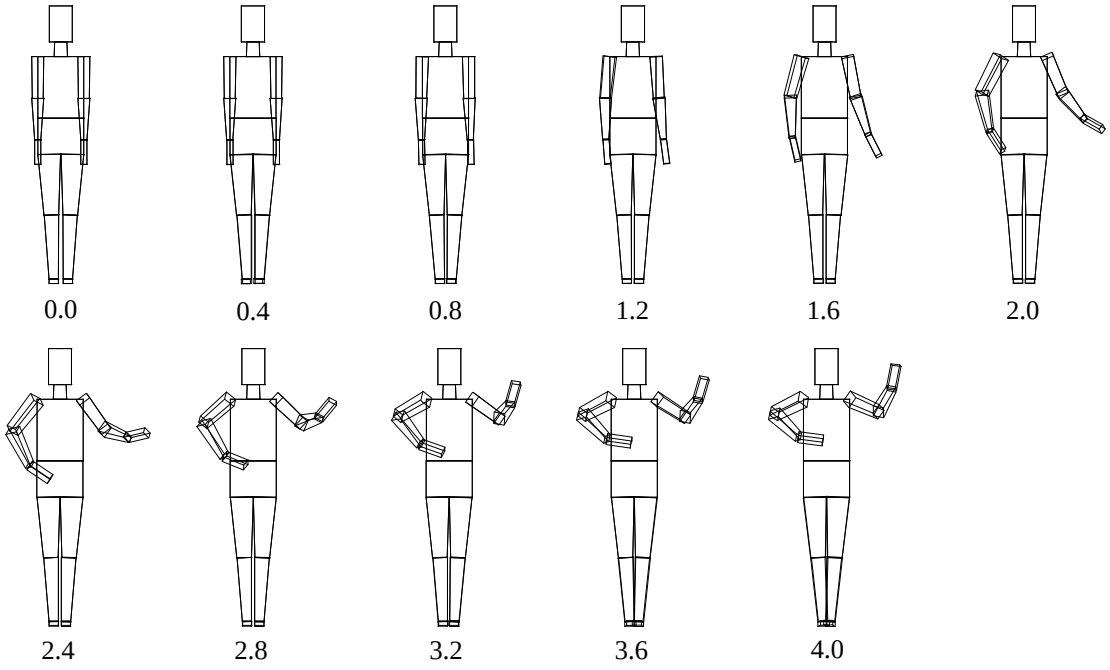


Figure 7.4 Reaching Dance Position in 4 Seconds

allotted time. The only unnatural movement appearing in some sequences was a slight jerk by some limbs resulting from the *move* to *maintain* state transfers. This jerking motion can be eliminated by reducing the difference between the *Movespring* and *Mainspring* constants. These sequences show that the torques applied to the limbs by the torque functions are strong enough to counteract the influence of gravity and to allow the limbs to maintain either an upright or a horizontal position with respect to the floor. A comparison of the four motion sequences show the figures at the corresponding time percentage intervals to have many similarities with the orientations of the arms.

These experiments show that, in the absence of interaction with other figures or the floor, realistic motion can be achieved in different time frames using the implemented motion control model.

7.2. Forward Step into 4th Position

The next set of experiments tested the fundamental position motor programs responsible for making the figure take a forward, backward, and side step. All experiments for these fundamental position transitions were conducted for motion frequencies of 1 and 2 seconds per step. Since most of the effort spent developing and debugging the position motion processes focused on the faster step rate, all the animation displayed in the next three sections show better movement for this speed. The motion processes operated at the slower step rate by setting all torques generated within these motor processes to be inversely proportional to the square of the time required to complete the motion.

The first of these experiments tested the motor program responsible for making the figure take a single step forward into 4th position. This movement was composed of one intermediate goal position representing the midpoint of the gait cycle (the point where the legs reach their maximum distance apart from each other), and a final goal position representing the end of the cycle (the point where the legs are back together again). The final goal position had one leg in a support stance and the other leg resting slightly above ground. This leg configuration at the end of the 4th position movement is common to many ballroom dances. In all experiments performed, the figure started in 1st position with the arms in a normal ballroom dance position.

The forward step was executed with the assistance of the 4th position forward motor program. As much as possible, this motion process applied torques based on biomechanical studies of human gait [Alexander84b] [Alexander84a] [McMahon84a]. At the beginning of each gait cycle, the 4th position forward motion process temporarily deactivated the figure's balance and applied a small internal torque to the upper body to make the figure fall slightly forward. While the figure was falling forward, additional torque was applied to the foot of the support leg to release weight from the swing leg. This torque made the support foot press into the ground, thereby raising the ankle of the leg and freeing the swing leg. Pressing the foot into the ground also helped anchor the support leg, thus reducing slippage. The swing leg was then swung forward with the assistance of the *move* motion process. Once the leg commenced its forward swing, the balance process was reactivated to prevent the figure from falling over. The 4th position motor program was then deactivated and the general limb motor programs completed the gait cycle.

Figures 7.5 and 7.6 show the animation sequences generated by this motion process. The motion appeared fairly natural despite a slight unnatural bend in the support leg while the figure was falling forward. This can be rectified by increasing the stiffness applied to the figure's leg joints. Although these figures do not show slippage, both forward steps had the initial support leg slip backwards about one and a half times the length of the foot. The figure was also pulled slightly backwards by the balance process during the last half second of the 2 steps per second animation rate. The net forward movement was about one and a half foot lengths for the faster rate, and about one foot length for the slower rate. Both the torque functions and gravity rotated the swing leg to the support leg slightly faster than desired for the faster gait.

This experiment was extended for both cycles by having the figure take 12 consecutive forward steps. Throughout the 1 second per step rate, the figure managed to maintain a relatively good position, although some steps showed more slippage than others. The results were quite different, however, for the 2 second per step rate. The figure started to slide about after the third step and was never able to completely recover despite efforts to continue the gait. The motion appeared very unnatural because the balance process con-

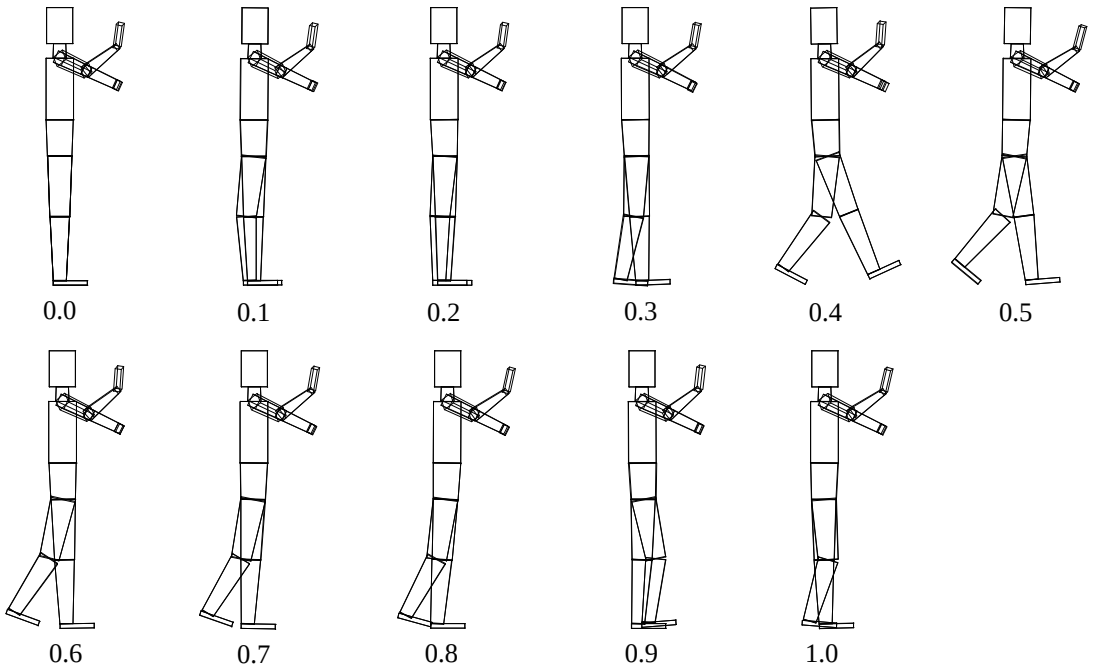


Figure 7.5 Forward Step in 1 Second

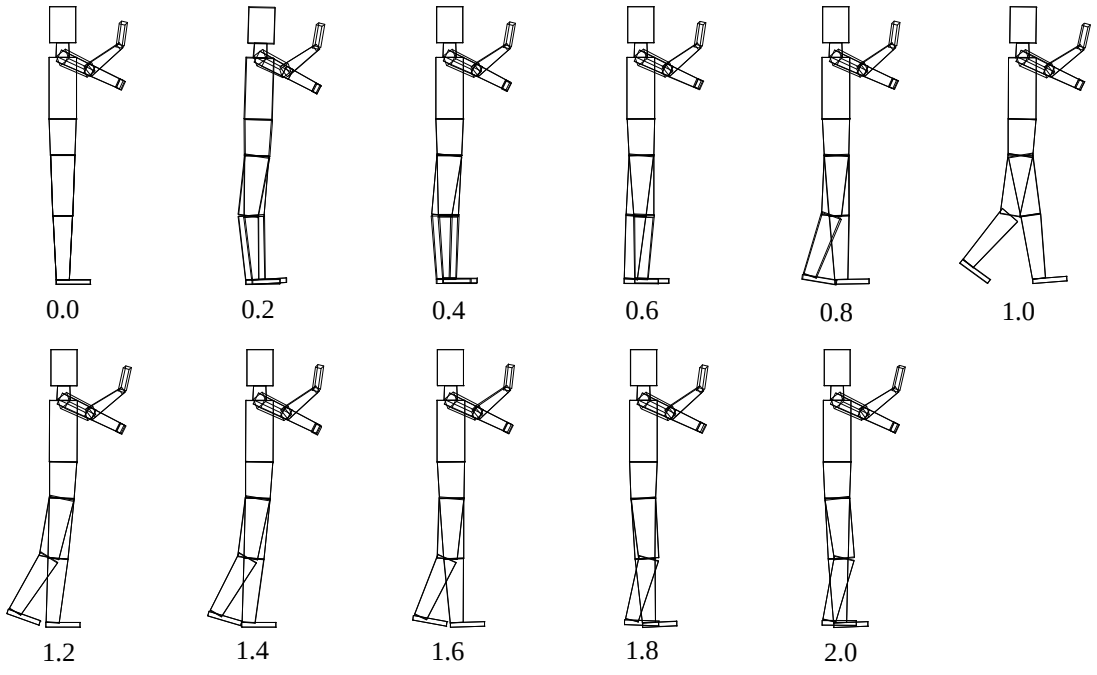


Figure 7.6 Forward Step in 2 Seconds

tinued to upright the figure from positions where it should have fallen over. The sliding problems are caused by the accumulation within the figure of kinetic energy and angular momentum and can be reduced by increasing the figure's stiffness.

7.3. Backward Step into 4th Position

The next position transition experiment tested the motor program responsible for making the figure take a single step backward into 4th position. As with the forward step, this movement was composed of one intermediate goal position representing the midpoint of the backward gait cycle and a final goal position representing the end of the cycle. The final goal position also had one leg in a support stance and the other leg resting slightly above the ground.

Considerable more difficulty was encountered trying to develop the 4th position backward motor program. Most of the human gait discussions found in the available biomechanical literature focused on forward walking steps. As a result, this motor program is not yet as stable as the 4th position forward program. At the beginning of the backward step cycle, the toe of the support foot was pressed into the ground to help anchor the support leg. This helped the upper support leg's *move* limb process push the figure backward. Further anchoring of the leg's position was obtained by countering the torques propagating from the upper leg's *move* process with an extra torque applied to the lower leg. The combination of these two torques enabled the *move* processes to reach the intermediate position while at the same time reducing the amount of horizontal foot slippage.

During the second half of the backward gait cycle, a torque was applied to the foot of the rear leg to anchor the foot to the ground. A second torque was applied to the upper body to assist the *balance* process in maintaining the figure in an upright position. This torque also helped counteract the backward momentum created by the first half of the gait cycle. If this torque was not applied, the figure would complete the gait cycle with its upper body leaning backward. Under this circumstance, a natural transition to the next step using the torque functions became very difficult to perform. All torques applied by the 4th position

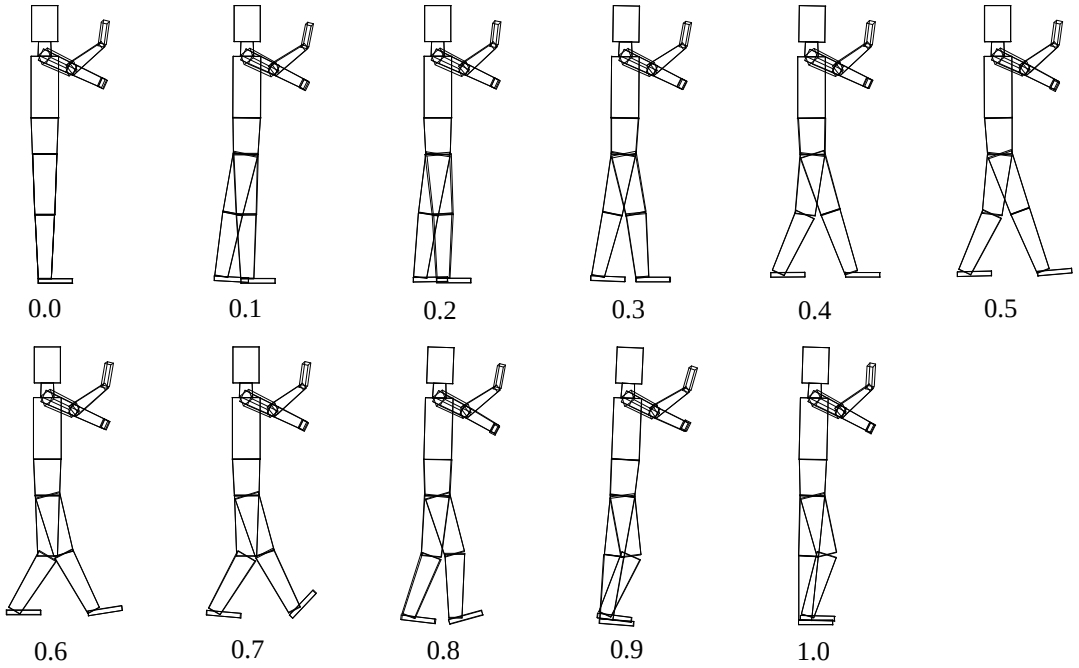


Figure 7.7 Backward Step in 1 Second

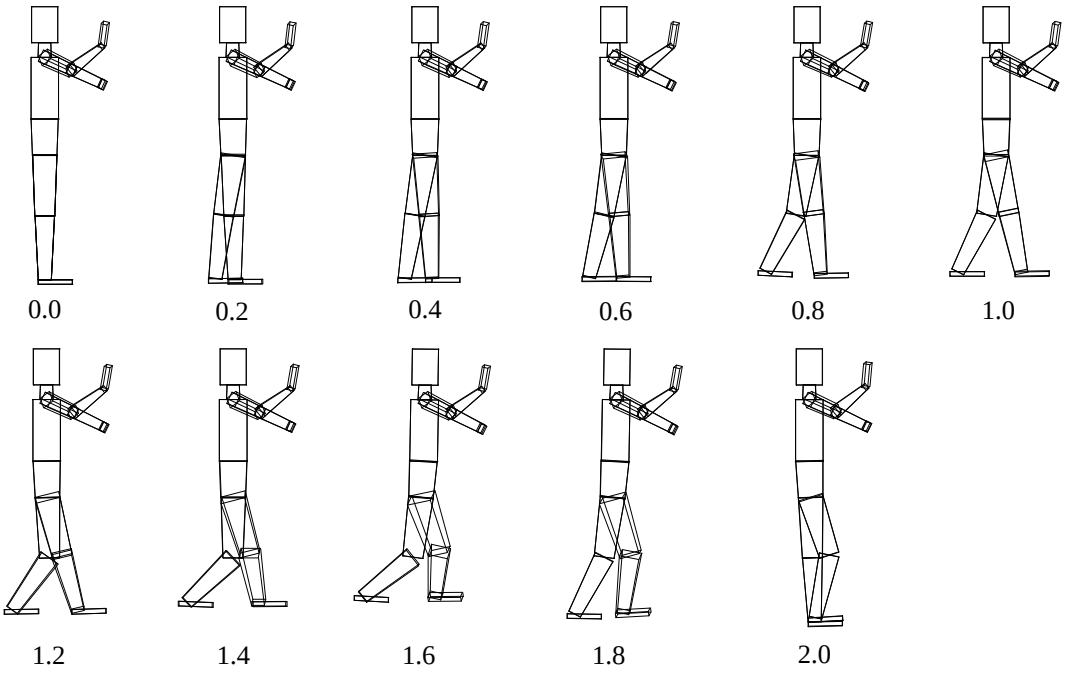


Figure 7.8 Backward Step in 2 Seconds

backward motion process attempted to have the figure complete the step in a vertical position while simultaneously trying to minimize the amount of excess kinetic energy and angular momentum remaining in the figure.

Experimental results for the backward gait cycle are shown in Figures 7.7 and 7.8. During both step rates, the initial support foot slipped forward approximately half a foot length and the overall backward movement was about two foot lengths. The rear foot slipped forward approximately a quarter of a foot length during the last half of the faster gait cycle.

The experiments for the slower rate revealed the torques applied to the swing leg caused the figure to become unstable. During the first half of the gait cycle, excessive sideward foot slippage was caused by the support leg rotating inwards. The body swayed to the outside and was eventually uprighted by the *balance* process. As a result of this foot slide, the entire backward step appeared very unnatural. These problems can be reduced by increasing the spring constants *Mainspring* and *Movespring*. This added stiffness would help prevent the support leg from sliding out from under the figure.

The appearance of the backward gait for both time rates can be improved by rotating the foot of the swing leg so the figure lands on the toe of the foot before coming to rest on the entire foot. The motion process kept this foot off the ground as much as possible to ensure the figure's weight was completely on the support leg before shifting the weight to the swing leg.

As with the forward walk, the backward walk experiment was extended for both cycles by having the figure take 12 consecutive backward steps. The experiment worked reasonably well for the 1 second per step rate, although control of the figure began to slip around the 8th step. The entire long walk for the backward step was characterized mostly by unnatural movement. This result was expected since the motor program was unable to make the figure move a single realistic step backward.

7.4. Side Step into 2nd Position

The final position transition experiment tested the motor program responsible for making the figure

move one step sideward into 2nd position. In all experiments, the side step was completed by activating a (null) process to move the figure to 1st position following termination of the 2nd position motion process.

Both the 2nd position and the 1st position movement transitions were composed of one goal position. The 2nd position goal had one leg extended to the side and the other leg extended in a near vertical position with the knee slightly bent. The 1st position goal was a vertical position with the legs straight and the feet together.

Developing a realistic motor program for a transition into 2nd position was found to be even more challenging than for the other transitions. The final result was a motor program which is still not very stable (nor realistic). However, the process operated as follows.

During the first half of the step, a torque was applied to rotate the upper body in the direction of the support foot. This resulted in a change of weight towards the support foot and released the opposite leg and hip. At the same time, the *move* limb motion process applied torques to rotate the upper support leg, thus causing the weight of the upper body to begin shifting towards the opposite leg. To help prevent slippage, a torque was applied to press the support foot into the ground. At approximately the midpoint of the step, the torque applied to the upper body was released and the balance process was temporarily disabled. This allowed the accumulated torques in the support leg to transfer the weight of the upper body to the other leg without interference from the balance process. After the elapse of a short interval to allow for the weight transfer, balance was restored to the upper body. During the weight transfer, the goal position for the swing leg was changed to remove the bend from the leg. This enabled the figure to land on a firm, straight leg. Following this, the 2nd position process completed execution.

Figures 7.9 and 7.10 display the experimental results for a side step. For side steps executed at the faster step rate, the support foot slipped sideward about one head width. While the legs were being rotated back together, the figure slid about one head width in the direction of the side step. This resulted in a net body movement of about three head widths to the side. The slower step rate also experienced a side slippage of approximately one head width. Near the beginning of the step, the build-up of torques within the

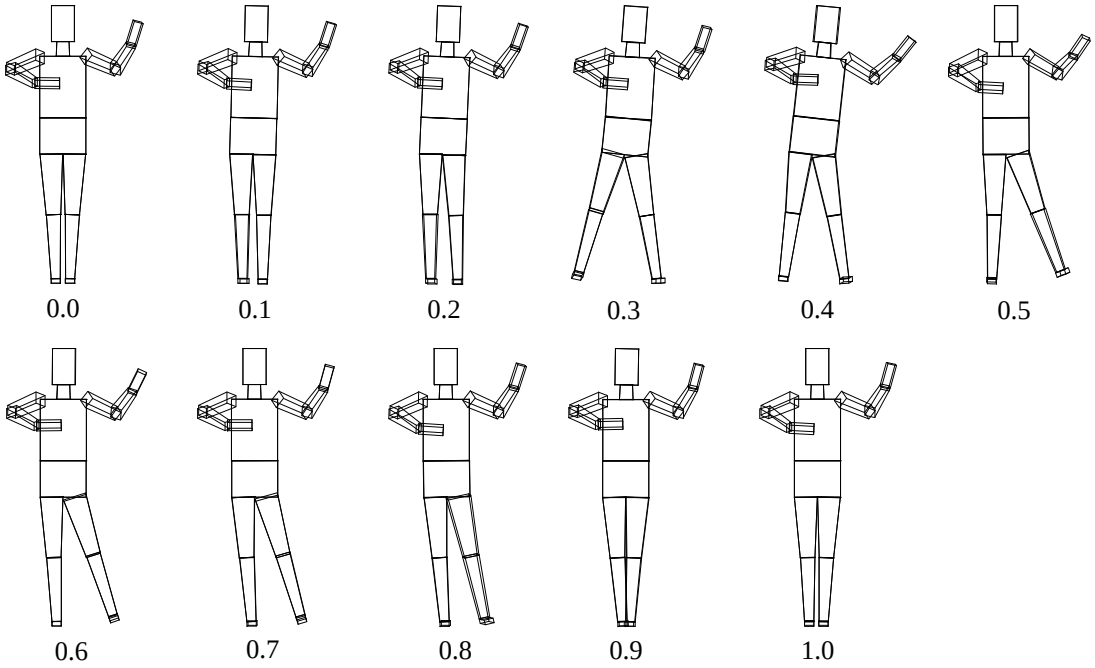


Figure 7.9 Side Step in 1 Second

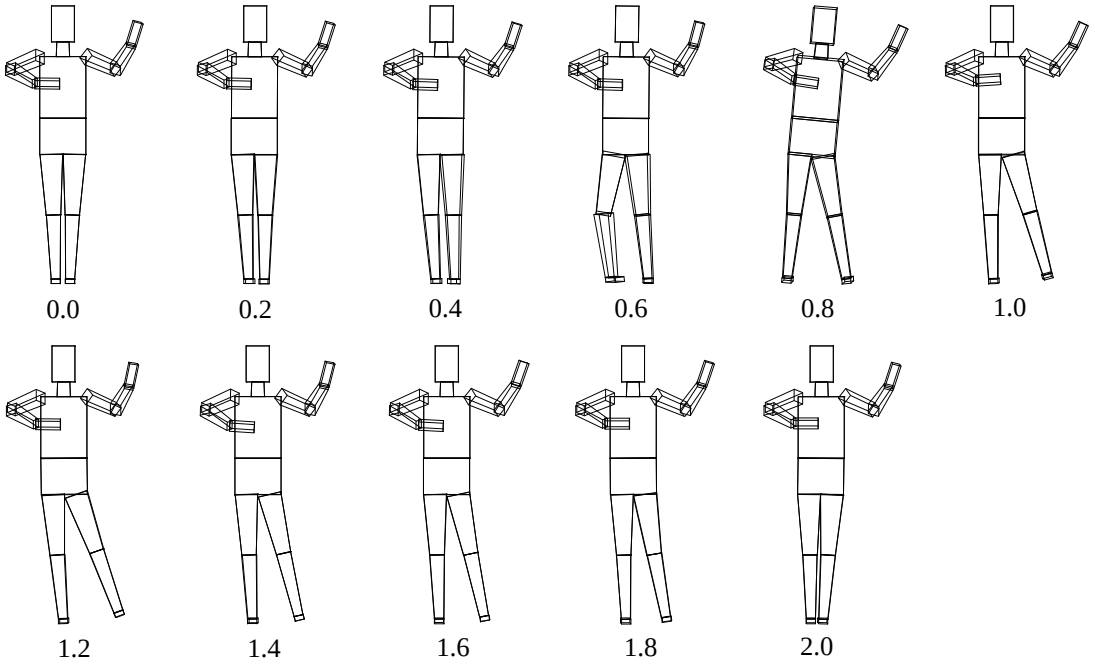


Figure 7.10 Side Step in 2 Seconds

body caused the lower swing leg to rotate out of its natural range. While the feet were being drawn back together, the figure slid along the floor first in the direction of the movement, and then in the opposite direction. This reversal of direction was caused by the foot of swing leg making contact with the ground. During this time, the figure was also sliding forward along the floor. This is likely caused by the combination of excessive kinetic energy within the figure and a faulty ground model. The net movement resulting from this side step was approximately one head width to the side.

The main difficulty with the attempts to make the figure step sidewards was shifting the weight of the body while preventing horizontal slippage with the support leg and foot. Ideally, the movement should be accomplished by minimizing the rotation of the upper body and having the foot of the swing leg lifted slightly off the ground as the rotation of the support leg moves the upper body smoothly to the side. Instead, both Figures 7.9 and 7.10 give the effect of the sudden release of a spring at about the midpoint of the step. This causes the previous support leg to swing into the air while the weight of the body falls onto the other foot.

The second experiment had the figure perform six consecutive side steps. At the faster rate, a reasonable step occurred only for the first step. The remaining steps were characterized by the support foot swinging out to the side and then being drawn back in. More control of the figure's momentum and a better understanding of how humans execute a side step is required before the figure will be able to take more than one reasonably realistic side step.

The results were more discouraging for the slower step rate. Again, only the first step bore any resemblance to a normal step. Unlike the faster rate, the figure fell over while attempting to perform the third step. The duration for which the balance process is disactivated needs to be readjusted along with the magnitude of the torques causing the upper body to tilt to the side if a more realistic sequence is to be achieved.

7.5. Forward and Back Basic in the Foxtrot

With the development of motion processes capable of making the figure take a step forward, backward, and to the side, the final experiment tested the ability of the figure to perform an elementary ballroom pattern. The sequence of moves selected was the *Forward and Back Basic* in the foxtrot. This pattern consists of the *Forward Basic* (explained in detail in Chapter 5) followed by a *Backward Basic* (a pattern similar to the *Forward Basic* except the steps are taken backwards). Thus, the overall pattern contains two steps forward, one step to the side, two steps backward, and one final side step. The figure started with the feet in 1st position and the arms resting at the side of the body. Prior to taking the first forward step, the arms were moved to dance position. Following the last side step, the arms were lowered to the side of the body.

Since the position motor processes operate more consistently at the faster step rate, the pattern was danced with each forward, back, and side step executed in one second. The arms were given two seconds to reach dance position and two seconds to be lowered to the side of the body. The pattern contained none of the symbols indicating if each step should be executed by stepping down on either the heel or the toe of the moving foot.

The results of this dance sequence, while far from perfect, were encouraging. The raising of the arms and the forward steps appeared very similar to the results obtained from the previous experiments. While the figure managed to progress backwards during the two steps backward, more foot slippage occurred than the amount that was present during the step backward experiment. The side steps, however, were a near total failure. During both steps, the support leg swung out and was then drawn back towards the other leg. The net result was very little sideward movement. The pattern concluded with the arms being lowered in a natural manner. While the arms were being lowered, however, the previously accumulated energy and momentum caused the figure to slide along the floor. Overall, the movement, while not graceful, bore some resemblance to a beginner dancer stumbling through this pattern. Again, better results can be obtained if more effort is focused on controlling the kinetic energy and momentum of the figure, as well as increasing the stiffness in the leg joints.

Chapter 8

Summary and Conclusions

This thesis has examined the issues involved with specifying and controlling dynamically produced motion of an articulated figure. An animation system, structured according to a recently proposed hierarchical motion control model, has been introduced to explore methods for efficiently entering complicated movement commands and controlling the generated motion.

The system uses ballroom dancing as an example. Motion commands are specified using a subset of an easy to use, high-level ballroom dance notation language. This notation language enables the animator to specify both simple and complex movements, as well as assign each movement a time limit for completion. The system translates these motion commands into torques which are then applied to the figure's limbs by a set of motor programs organized in a hierarchical manner. These motor programs either move a limb to a new position or maintain a limb at its current position. All torques are generated automatically, thereby relieving the animator from guessing the magnitude and direction of the torques required for each motion. Most torques are produced using muscle torque functions derived from biomechanics. Each limb's movement is controlled by performing interpolations along a quaternion curve defined for the limb. These interpolations allow for the magnitude and direction of the torques to be adjusted while a motion is in progress.

Unlike previous articulated figure animation systems, this new system interpolates limb orientations using quaternion coordinates. Because the system uses dynamics, all limbs are rotated by applying torques along the axis specified by the quaternion transforming the limb's current orientation to its goal orientation. Distance measurements between quaternions allow for the muscle torque function parameters to be adjusted while motion is in progress.

Because quaternions are continuous for all orientations and do not suffer from gimbal lock,

quaternions have been found to be far superior for representing orientation and performing interpolations than Euler angles. However, no efficient and easy method using quaternions is known for determining if a limb's orientation is within its natural range of movement.

Experimental results with simple and complex movements suggest this model is capable of producing realistic time-dependent dynamic motion sequences. Although many of the motion sequences fail to appear very natural, in most cases the limbs either reach or nearly reach their goal position at the end of the allotted time. However, many problems remain with the system.

First, the experimental results indicate further control of the figure's motion is required. Additional biomechanical knowledge needs to be incorporated within the low-level position motor programs to make the generated movement appear more natural. Presently, one set of biomechanical torque functions is being used for all movements. Additional functions need to be implemented to handle different types of movement.

A more sophisticated feedback system needs to be implemented for the limbs whose active state is *move*. These feedback systems should control not only the figure's position, but also the amount of kinetic energy and momentum present in each link. Long range planning algorithms should also be developed so the system can obtain better estimates of the torques required for motion sequences requiring more than one intermediate or final goal position.

Although the current implementation uses the same spring and damper constants for all links, this is likely not a realistic configuration. Methods other than the trial and error technique now employed need to be developed for automatically determining acceptable spring and damper constants for each link, as well as for the floor. Further experiments should also be done with the balance process and floor model to reduce the amount of sliding, especially at the conclusion of long animation sequences. A better integration technique for solving the equations of motion should also be developed so the constant by which the moments of inertia for each segment are artificially enlarged may be either reduced or completely eliminated. This artificial enlargement of each segment's moments of inertia prevents the torque functions from

applying normal torque values to most movements.

While many of the above problems provide challenging topics for future research, other fertile research areas include exploring the dynamic interaction between the figure and a dance partner, and developing behavior models for animation sequences containing several dancers. Despite the problems encountered with *BDAS*, animation systems based on dynamics continue to show great promise for improving the quality of articulated figure animation.

References

Alexander84a.R. M. Alexander, Walking and Running, *American Scientist 72*, (July-August 1984), 348-354.

Alexander84b.R. M. Alexander, The Gaits of Bipedal and Quadrupedal Animals, *The International Journal of Robotics Research 3*, 2 (Summer 1984), 49-59.

Altmann86.S. L. Altmann, *Rotations, Quaternions, and Double Groups*, Oxford University Press, New York City, New York, 1986.

Armstrong85a.W. W. Armstrong and M. W. Green, The Dynamics of Articulated Rigid Bodies for Purposes of Animation, *Proceedings Graphics Interface '85*, May 1985, 407-415.

Armstrong85b.W. W. Armstrong and M. W. Green, The Dynamics of Articulated Rigid Bodies for Purposes of Animation, *The Visual Computer*, December 1985, 231-240.

Armstrong86a.W. W. Armstrong, T. A. Marsland, M. Olafsson and J. Schaeffer, Solving Equations of Motion on a Virtual Tree Machine, Technical Report TR86-11, Department of Computing Science The University of Alberta, June 1986.

Armstrong86b.W. W. Armstrong, M. Green and R. Lake, Near-Real-Time Control of Human Figure Models, *Proceedings Graphics Interface '86*, May 1986, 147-151.

Armstrong87.W. W. Armstrong, M. Green and R. Lake, Near-Real-Time Control of Human Figure Models, *IEEE Computer Graphics and Applications*, June 1987, 52-61.

Badler78.N. I. Badler, J. O'Rourke, S. W. Smoliar and L. Weber, The Simulation of Human Movement by Computer, Movement Project Report No. 14, Department of Computer and Information Science University of Pennsylvania, September 1978.

Badler87.N. I. Badler, K. H. Manoochehri and G. Walters, Articulated Figure Positioning by Multiple Con-

straints, *IEEE Computer Graphics and Applications*, June 1987, 28-38.

Baecker69.R. M. Baecker, *Interactive Computer-Mediated Animation*, Ph.D Thesis, Massachusetts Institute of Technology, April 1969.

Bapu80.P. Bapu, S. Evans, P. Kitka, M. Korna and J. McDaniel, *User's Guide to Combiman Programs*, University of Dayton Research Institute, Dayton, Ohio, February, 1980.

Benesh56.R. Benesh and J. Benesh, *An Introduction to Benesh Dance Notation*, A. & C. Black Ltd., London, England, 1956.

Blakeley80.F. M. Blakeley, *CYBERMAN*, Chrysler Corporation, Detroit, Michigan, June, 1980.

Brown76.M. Brown and S. Smoliar, A Graphics Editor for Labanotation, *Computer Graphics 10*, 2 (Summer 1976), 60-65.

Bruderlin89.A. Bruderlin and T. W. Calvert, Goal-Directed, Dynamic Animation of Human Walking, *Computer Graphics 23*, 3 (July 1989), 233-242.

Cachola86.D. G. Cachola and G. F. Schrack, Modeling and Animating Three-Dimensional Articulated Figures, *Proceedings Graphics Interface '86*, May 1986, 152-157.

Calvert78.T. W. Calvert and J. Chapman, Notation of Movement with Computer Assistance, *ACM 78: Proceedings, 1978 ACM Annual Conference 2*, (December 1978), 731-736.

Calvert82.T. W. Calvert and A. Patla, Aspects of the Kinematic Simulation of Human Movement, *IEEE Computer Graphics and Applications*, November 1982, 41-50.

Catmull78.E. Catmull, The Problems of Computer-Assisted Animation, *Computer Graphics 12*, 3 (August 1978), 348-353.

Chadwick89.J. Chadwick, D. Haumann and R. Parent, Layered Construction for Deformable Animated Characters, *Computer Graphics 23*, 3 (July 1989), 243-252.

DiStefano76.J. J. DiStefano, A. R. Stubberud and I. J. Williams, *Feedback and Control Systems*, McGraw-Hill, Inc., 1976.

Dooley82.M. Dooley, Anthropometric Modeling Programs - A Survey, *IEEE Computer Graphics and Applications*, November 1982, 17-25.

Dransch86.D. O. K. Dransch, J. C. Beatty and R. S. Ryman, ChoreoScribe: A Graphics Editor to Describe Body Position and Movement Using Benesh Movement Notation, Technical Report CS-86-48, University of Waterloo Computer Science Department, October 1986.

Drewery86.K. Drewery and J. Tsotsos, Goal Directed Animation using English Motion Commands, *Proceedings Graphics Interface '86*, May 1986, 131-135.

Fetter82.W. A. Fetter, A Progression of Human Figures Simulated by Computer Graphics, *IEEE Computer Graphics and Applications*, November 1982, 9-13.

Forsey88.D. R. Forsey and J. Wilhelms, Techniques for Interactive Manipulation of Articulated Bodies Using Dynamic Analysis, *Proceedings Graphics Interface '88*, June 1988, 8-15.

Fowles86.G. R. Fowles, *Analytical Mechanics, Fourth Edition*, Holt, Rinehart, and Winston, Inc., New York City, New York, 1986.

Girard85.M. Girard and A. A. Maciejewski, Computational Modeling for the Computer Animation of Legged Figures, *Computer Graphics 19*, 3 (July 1985), 263-270.

Girard87.M. Girard, Interactive Design of 3D Computer-Animated Legged Animal Motion, *IEEE Computer Graphics and Applications*, June 1987, 39-51.

Goldstein59.H. Goldstein, *Classical Mechanics*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1959.

Green90.M. Green, Using Dynamics in Computer Animation: Control and Solution Issues, in *Making Them Move: Mechanics, Control and Animation of Articulated Figures*, Morgan Kaufmann Publishers, 1990.

Grodins63.F. S. Grodins, *Control Theory and Biological Systems*, Columbia University Press, New York City, New York, 1963.

Halliday74.D. Halliday and R. Resnick, *Fundamentals of Physics*, John Wiley and Sons, Inc., New York City, New York, 1974.

Hanavan64.E. P. Hanavan, *A Mathematical Model of the Human Body*, Behavioral Sciences Laboratory, Wright-Paterson Air Force Base, Ohio, 1964.

Hatze81.H. Hatze, *Myocybernetic Control Models of Skeletal Muscle*, University of South Africa, Pretoria, South Africa, 1981.

Herbison-Evans82.D. Herbison-Evans, Computers and the Arts, Technical Report 191, Basser Department of Computer Science, The University of Sydney, October 1982.

Herbison-Evans84a.D. Herbison-Evans, A Dancer Among Us, Technical Report 238, Basser Department of Computer Science, The University of Sydney, May 1984.

Herbison-Evans84b.D. Herbison-Evans, Robots and Dancing, Technical Report 237, Basser Department of Computer Science, The University of Sydney, May 1984.

Herbison-Evans85.D. Herbison-Evans, Dance and Computers, Technical Report CS-85-51, University of Waterloo Computer Science Department, October 1985.

Herbison-Evans86.D. Herbison-Evans, Animation of the Human Figure, Technical Report CS-86-50, University of Waterloo Computer Science Department, November 1986.

Herbison-Evans87.D. Herbison-Evans, Some Poly-Ellipsoid Figures, Technical Report 317, Basser Department of Computer Science, The University of Sydney, December 1987.

Isaacs87.P. M. Isaacs and M. F. Cohen, Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions, and Inverse Dynamics, *Computer Graphics 21*, 4 (July 1987), 215-224.

Kingsley81.E. C. Kingsley, N. A. Schofield and K. Case, SAMMIE - A Computer Aid for Man-Machine Modeling, *Computer Graphics 15*, 3 (August 1981), 163-169.

Laban75.R. Laban, *Principles of Dance and Movement Notation*, MacDonald & Evans, London, England, 1975.

Lasseter87.J. Lasseter, Principles of Traditional Animation Applied to 3D Computer Animation, *Computer Graphics 21*, 4 (July 1987), 35-43.

Lien84.S. Lien and J. T. Kajiya, A Symbolic Method for Calculating the Integral Properties of Arbitrary Nonconvex Polyhedra, *IEEE Computer Graphics and Applications*, November 1984, 35-41.

Magnenat-Thalmann88.N. Magnenat-Thalmann, R. Laperriere and D. Thalmann, Joint-Dependent Local Deformations for Hand Animation and Object Grasping, *Proceedings Graphics Interface '88*, June 1988, 26-33.

McMahon84a.T. A. McMahon, Mechanics of Locomotion, *The International Journal of Robotics Research 3*, 2 (Summer 1984), 4-28.

McMahon84b.T. A. McMahon, *Muscles, Reflexes, and Locomotion*, Princeton University Press, Princeton, New Jersey, 1984.

McNair82.B. G. McNair, D. Herbison-Evans and N. Neilands, Computer Assisted Choreography Teaching, Technical Report 157, Basser Department of Computer Science, The University of Sydney, April 1982.

Milhorn66.H. T. Milhorn, *The Application of Control Theory to Physiological Systems*, W.B. Saunders Company, Philadelphia, Pennsylvania, 1966.

Miura84.H. Miura and I. Shimoyama, Dynamic Walk of a Biped, *The International Journal of Robotics Research 3*, 2 (Summer 1984), 60-74.

Muybridge55.E. Muybridge, *The Human Figure in Motion*, Dover Publications, New York City, New York, 1955.

Parke82.F. I. Parke, Parameterized Models for Facial Animation, *IEEE Computer Graphics and Applications*, November 1982, 61-68.

Pearce86.A. Pearce, B. Wyvill, G. Wyvill and D. Hill, Speech and Expression: A Computer Solution to Face Animation, *Proceedings Graphics Interface '86*, May 1986, 136-140.

Platt81.S. Platt and N. Badler, Animating Facial Expressions, *Computer Graphics 15*, 3 (August 1981), 245-252.

Pletinckx89.D. Pletinckx, Quaternion Calculus as a Basic Tool in Computer Graphics, *The Visual Computer*, January 1989, 2-13.

Politis82.G. Politis and D. Herbison-Evans, Computer Choreology Project at the University of Sydney, Technical Report 204, Basser Department of Computer Science, The University of Sydney, April 1982.

Politis85.G. Politis and D. Herbison-Evans, A Computer Graphics Interpreter for Benesh Movement Notation, Technical Report 267, Basser Department of Computer Science, The University of Sydney, July 1985.

Politis86.G. Politis and D. Herbison-Evans, Computer Animation by Choreography, Technical Report 292, Basser Department of Computer Science, The University of Sydney, September 1986.

Politis87.G. Politis, A Survey of Computers in Dance, Technical Report 311, Basser Department of Computer Science, The University of Sydney, October 1987.

Reeves81.W. T. Reeves, Inbetweening for Computer Animation Utilizing Moving Point Constraints, *Computer Graphics 15*, 3 (August 1981), 263-269.

Reynolds82.C. W. Reynolds, Computer Animation with Scripts and Actors, *Computer Graphics 16*, 3 (July 1982), 289-296.

Ridsdale86.G. Ridsdale, S. Hewitt and T. W. Calvert, The Interactive Specification of Human Animation, *Proceedings Graphics Interface '86*, May 1986, 121-130.

Romain79.E. Romain and F. Colby, *Let's Go Dancing*, Octopus Books Limited, London, England, 1979.

Royce77.A. P. Royce, *The Anthropology of Dance*, Indiana University Press, Bloomington, Indiana, 1977.

Savage78.G. J. Savage and J. M. Officer, CHOREO: An Interactive Computer Model for Dance, *International Journal of Man-Machine Studies 10*, (1978), 1-18.

Sealey80.D. Sealey, NOTATE: Computerized Programs for Labanotation, *Journal for the Anthropological Study of Human Movement 1*, 2 (Autumn 1980), 70-74.

Shoemake85.K. Shoemake, Animating Rotation with Quaternion Curves, *Computer Graphics 19*, 3 (July 1985), 245-254.

Singh83.B. Singh, J. Beatty, K. Booth and R. Ryman, A Graphics Editor for Benesh Movement Notation, *Computer Graphics 17*, 3 (July 1983), 51-62.

Sturman84.D. Sturman, Interactive Keyframe Animation of 3-D Articulated Models, *Proceedings Graphics Interface '84*, May 1984, 35-40.

Thornhill-Geiger81.R. Thornhill-Geiger, *Thirteen Ballroom Dances*, National Council of Dance Teacher Organizations, Inc., Richmond Hill, New York, 1981.

Waters87.K. Waters, A Muscle Model for Animating Three-Dimensional Facial Expression, *Computer Graphics 21*, 4 (July 1987), 17-24.

Wilhelms85.J. Wilhelms, *Graphical Simulation of the Motion of Articulated Bodies Such as Human and Robots with Particular Emphasis on the Use of Dynamic Analysis*, Ph.D Thesis, University of California, Berkeley, July, 1985.

Wilhelms86.J. Wilhelms, Virya - A Motion Control Editor for Kinematic and Dynamic Animation, *Proceedings Graphics Interface '86*, May 1986, 141-146.

Wilhelms87.J. Wilhelms, Using Dynamic Analysis for Realistic Animation of Articulated Bodies, *IEEE Computer Graphics and Applications*, June 1987, 12-27.

Willmert82.K. D. Willmert, Visualizing Human Body Motion Simulations, *IEEE Computer Graphics and Applications*, November 1982, 35-38.

Wolofsky74.Z. Wolofsky, *Computer Interpretation of Selected Labanotation Commands*, M.Sc. Thesis, Simon Fraser University, 1974.

Zeltzer82a.D. Zeltzer, Motor Control Techniques for Figure Animation, *IEEE Computer Graphics and*

Applications, November 1982, 53-59.

Zeltzer82b.D. Zeltzer, Representation of Complex Animated Figures, *Proceedings Graphics Interface '82*, May 1982, 205-211.