Dynamic Motion Control of an Articulated Figure Using Quaternion Curves

Robert Lake
Mark Green

Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada

August 18, 1995

Dynamic Motion Control of an Articulated Figure Using Quaternion Curves

Robert Lake
Mark Green

Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada

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 to produce a particular motion. The onus should be on the animation system to translate these high-level motion commands into the necessary series of 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?

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.

1. Introduction

One of the more challenging problems in computer graphics is the realistic modeling and animation of a human figure. 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.

The recent emergence of animation systems based on dynamics has shown great promise for improving the quality of human figure animation. Unlike previous systems which produce motion purely as a function of position or velocity, dynamics accounts for the forces and torques producing the motion. Thus, an animated figure can react to its environment in a natural manner that requires minimal effort by the animator.

The animation of human figures produces interesting problems often not found with animation systems handling simpler classes of objects. How can a complicated movement sequence be defined using minimal effort and a language familiar to the animator? How can the simultaneous movement of the various limbs of the human figure be controlled to produce a realistic motion sequence? In the case of dynamics, motion at the lowest level must be specified in terms of forces and torques. Most animators have little intuitive feel for the magnitude and direction required for these terms to produce a desired motion. Ideally, a dynamically-based human figure animation system should translate motion commands in a high level description into a lower level form which produces the forces and torques applied to the figure. The issues of controlling and coordinating the movement of the figure's limbs should be handled by the animation system rather than by the animator.

This paper presents a new dynamically-based motion system designed to produce realistic and controlled human motion using motion sequences found in ballroom dancing. All movement sequences are derived from movement commands specified using an easy to learn ballroom dance notation language. These movement commands are translated by the system and fed as input into a motion control model responsible for applying the forces and torques necessary to produce the specified motion.

2. Articulated Figure Animation Systems

Most articulated figure animation systems produced within the past decade have been based on either kinematics or dynamics. Kinematic systems produce motion through the specification of velocities and accelerations acting on the figure's limbs. Position is then calculated as a function of time.

In 1982, David Zeltzer produced a kinematic description of a walking skeleton by breaking complex motion sequences into a series of primitive movements [Zeltzer82]. These movements were handled by a set of Local Motor Programs (LMPs), each responsible for manipulating a fixed collection of joints. The overall control of the motion generated by the LMPs was handled on a higher level. Zeltzer used a set of 8 LMPs to produce a walking skeleton model.

Dynamic systems produce motion by applying forces and torques to the figure. The equations of motion are then solved to obtain the acceleration, velocity, and position of each joint.

One of the first major systems using dynamic analysis of human motion was *Deva*, presented in 1985 by Jane Wilhelms [Wilhelms85] [Wilhelms86] [Wilhelms87]. This system generated dynamic motion based on a set of torque functions defined by the animator for each degree of freedom using the associated motion editor *Virya*. The primary problem with *Deva* was that the method used for solving the dynamic equations increased quadratically in complexity with respect to the number of degrees of freedom in the human figure. By using a linear recursive formulation for the equations [Armstrong85], Wilhelms and Forsey later produced a dynamic system called *Manikin* which provided interactive manipulation of an articulated figure [Forsey88].

In 1987, Armstrong et al. [Armstrong87] presented a system producing near real time dynamic motion by executing a set of motion processes on each of the figure's limbs. This system was useful for exploring the effects of torques applied to the figure, but could not be easily used for generating complicated animation sequences. Also in 1987, Isaacs and Cohen described their system which produced dynamic motion using behavior functions and kinematic constraints [Isaacs87]. When necessary, this system imposed constraints on the dynamic equations to produce a desired motion. Recently, a hybrid walking model which uses dynamics to obtain the general motion and then applies kinematic cosmetic improvements was discussed by Bruderlin and Calvert [Bruderlin89].

The work presented here is a continuation of the ground work laid by Armstrong et al. All motion generated by the new model is strictly dynamically produced and no kinematic constraints are imposed on the dynamics equations.

3. Quaternions

Most traditional three-dimensional animation systems express orientations in terms of combinations of rotations about a frame's *X*, *Y*, and *Z* axis. These three rotational parameters are known as the *Euler angles* of the rotation. While most people are familiar with Euler angles, some orientations defined by Euler angles can become undetermined. Occasionally a condition called *gimbal lock* results - the loss of one rotational degree of freedom when two rotation axes are superimposed on each other.

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 scalar component and three vector components. These numbers may be expressed in the form $[\lambda, \Lambda]$ with λ representing the scalar and Λ the vector component.

Leonhard Euler proved in 1752 that any three dimensional orientation can be expressed as a single rotation by θ degrees about an axis defined by a unit vector **n** from a reference position. Using spherical trigonometry, a 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 the unit vector **n** multiplied by $\sin(\theta/2)$ [Altmann86]. Note an orientation defined by such a quaternion $[\lambda, \Lambda]$ is identical to the one defined by $[-\lambda, -\Lambda]$.

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

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

The rule for multiplying two quaternions requires several operations involving scalar multiplications, inner dot products, and outer cross products of the vector components:

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

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 $\|[\lambda, \Lambda]\|$. The procedure for calculating the norm of a quaternion can be applied to determine the *distance* between two quaternions:

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

A quaternion whose norm is equal to 1 is called a *normalized quaternion*. The components of such quaternions contain the previously described rotation angle and rotation axis used to obtain an orientation from an inertial frame. Normalized quaternions form a sub-group of the quaternion group and provide a homomorphic mapping to the group of real orthogonal 3×3 (rotation) matrices with determinant $+1$. 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 inverse of a quaternion $[\lambda, \Lambda]$ 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.

Shoemake [Shoemake85] gives simple algorithms for converting between Euler angles and quaternions, and between rotation matrices and quaternions.

4. The Structure of Ballroom Dances

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.

All ballroom dance motions are classified into general categories known as *dances*. The complete set of ballroom dances constitute 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.

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 with identification of the movement sequence. A ballroom dance performance is usually composed of the sequential execution of 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 (commonly referred to as a *step*). Nearly all patterns are composed of the transitions between five fundamental positions. 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.

A divide and conquer approach is applied towards developing a ballroom dance animation model. Rather than generating animation starting at the dance and pattern level, a set of motion processes are developed at the lowest level to produce the transitions from one fundamental position to the next. Once these motion processes have been developed and debugged, algorithms can then be constructed to produce pattern and dance animation by sequentially executing combinations of these transitions. This can result in the generation of a wide and interesting range of human motion.

5. The Ballroom Dance Animation System

The Ballroom Dance Animation System (*BDAS*) is designed to produce dynamically controlled motion from an easy to learn ballroom dance notation language. The current implementation of *BDAS* consists of four components divided into two levels (Figure 5.1). The Top Level allows the animator to switch between three modules composing the Lower Level. Modules in the Lower Level define the components required to generate ballroom dance animation.

Dances and patterns are created and modified using the Dance Library and Pattern Editor. This editor consists of a menu-driven, one-button mouse user interface. The animator creates and modifies dance patterns by selecting symbols representing a subset of a ballroom dance notation language found in manuals published by the National Council of Dance Teacher Organizations, Inc. (NCDTO) [Thornhill-Geiger81]. These symbols define various movements and timings which compose a pattern.

A pattern is represented in a grid. The rows of the grid correspond to body part descriptions, such as the position of the head after each step, and step timings, such as the number of beats required to execute the step.

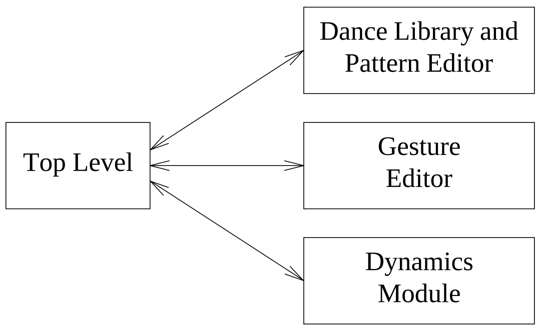

Figure 5.1 Software Architecture

Each column defines a body orientation reached during a specified time interval. The number of columns in the grid correspond to the number of steps in the pattern and all patterns are executed by performing the column-defined steps in a left to right order.

The Gesture Editor is used to assign a positional meaning for the dance notation symbols defined in the Dance Library and Pattern Editor. The human figure goal position associated with a body symbol is set by having the animator interactively manipulate in three dimensions the limbs of a human figure display model. The model consists of 16 limbs (head, neck, upper and lower torso, upper and lower arm, hand, upper and lower leg, and foot) with all limbs covered by four or six polygons. Each polygon is color-coded to aid identification of the side. A goal position for a step is constructed by merging the body orientations associated with the symbols contained in the rows of the pattern defining *Head Position*, *Body Modifier*, *Foot Position*, and *Footwork*. These categories correspond to the position of the head and arms, upper and lower body, legs, and feet.

The figure is represented internally by a tree-like structure with nodes and arcs corresponding to links and joints. The root of the tree is attached to the upper body. Each segment has its own local right-handed coordinate system with the origin set to the proximal hinge of the segment (the point where the segment connects to its parent). A 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). All three rotational parameters may be interactively set for each frame by the animator. In addition to the local frames, an inertial frame is used to define the position of the figure with respect to the environment.

Animation sequences are generated within the Dynamics Module. This module reads and interprets an animator selected dance pattern, executes the motion processes governing the dancer's motion, oversees the dynamics computations generating the actual motion, and updates the new figure positions within a large window on the screen. Parameters such as the downward acceleration of gravity, floor elasticity, limb stiffness, tempo of movement, and motion duration may be set by the animator prior to starting the computations. Throughout the pattern the current step is displayed in a smaller window along with the elapsed time of the animation sequence. Updates to the screen occur every 0.1 seconds of simulated time. At the conclusion of the pattern, all previously displayed frames may be played back either in succession or one frame at a time.

The Dynamics Module uses the equations of motion for articulated rigid bodies described by Armstrong and Green [Armstrong85] and solves them using their recursive linear method. Prior to starting the dynamics computations, physical properties for each segment composing the human figure model are set using data from anthropometric studies of human figures [Hanavan64] and 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. Thus, under certain circumstances (such as when the figure falls onto the floor or interpolates a step incorrectly) limbs can freely pass through each other.

Throughout the dynamics computations, the Dynamics Module makes two important procedure calls. The first procedure models the dance floor and maintains the figure on the floor. This procedure generates upward restorative forces to prevent the figure from falling through the floor and simulates ground-based horizontal frictional forces. The second procedure activates the motor control model, discussed in the next section.

6. Motor Control

Motor control is implemented in *BDAS* using the multi-level structure shown in Figure 6.1. This hierarchical structure is similar to the skeletal control model used by Zeltzer [Zeltzer82] 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.

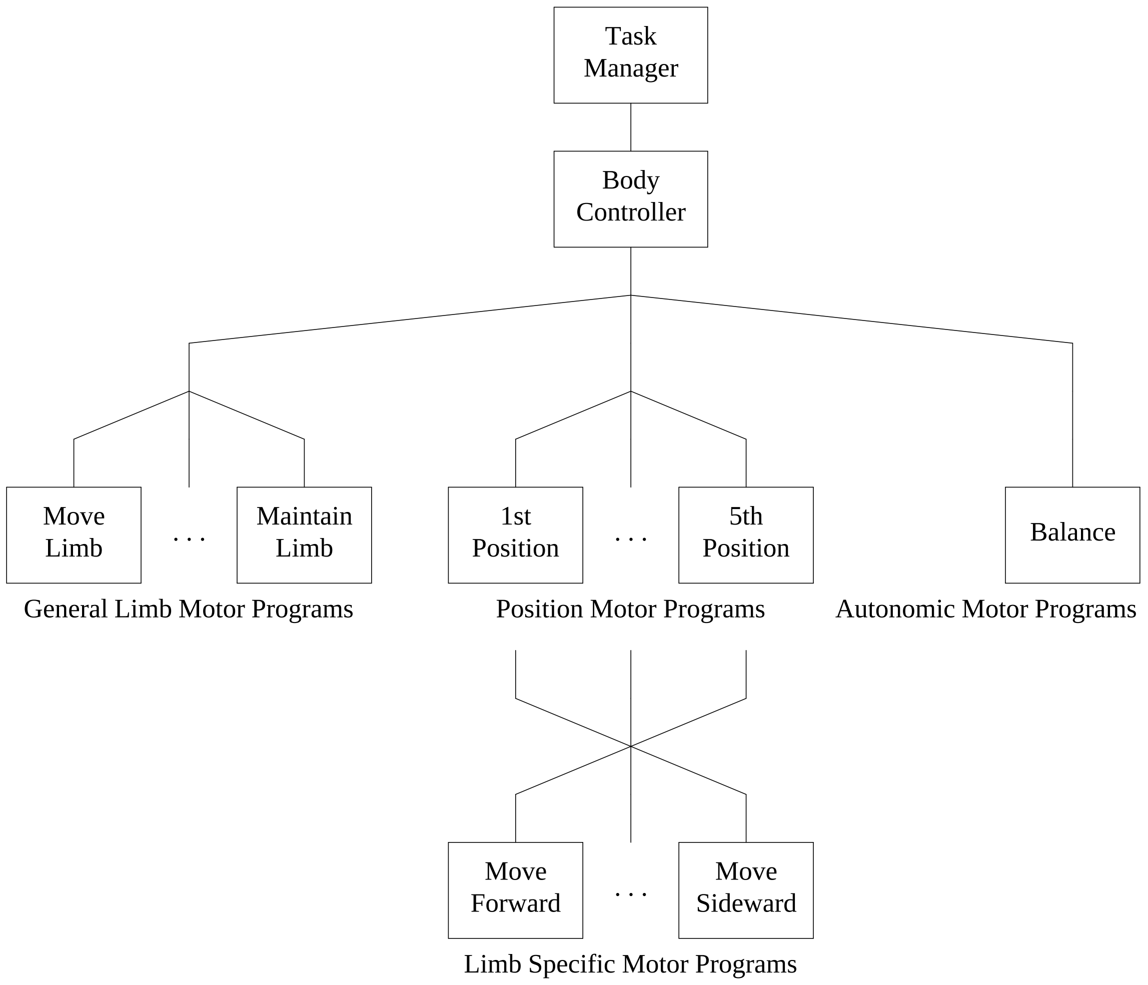

Figure 6.1 Motor Control Structure

The task manager oversees the execution of each dance pattern and is called each time a new step is executed. This virtual process receives a pattern specified by the animator and decomposes it into a series of steps. All steps are placed into a *step queue* for sequential execution by the lower levels of the model. At the completion of each step, the next step is removed from the queue and information is extracted describing the time required to complete the step and the final body position and orientation. Next, the Euler angles defining each segment's goal orientation with respect to its parent are converted into a goal quaternion. Finally, the link's motion state is determined by measuring the distance between the goal quaternion and the quaternion giving the link's current orientation with respect to its parent.

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. Links in this state have relatively strong restorative torques applied whenever the link deviates from its maintained 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 finished setting the states, goal positions, and timers for each link, control is passed to the body controller. The body controller activates and supervises the execution of the motor programs required to perform each step. In contrast to the task manager being executed once 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 leading relative to where it should be since the movement began (the criteria for deciding where a limb should be at a given time is described in Section 8). The timer is then reset to produce an another interrupt after the completion of a fixed time interval. After servicing of all timer interrupts, the body controller initiates human figure movement by executing a series of low-level motor programs.

7. 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 time-dependent 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.

7.1. 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.

7.1.1. 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 any motion constraints.

7.1.2. 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 from the following equation:

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

$[\lambda_t, \Lambda_t]$ 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 of which have 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.

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.

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 may 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 intermediate 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.

7.1.3. 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.

7.2. 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. These motor programs may change the state of one or more limbs during their execution.

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

7.3. 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. Although autonomic motor programs operate at all times while dance patterns are being processed, they can be temporarily deactivated by position motor programs. 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 is 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.

8. Low-Level Motor Control

Low-level motion control is performed during each step by assigning a simple feedback system to each limb marked *move*. 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 a goal position is approached.

Let $[\lambda_s, \Lambda_s]$ and $[\lambda_g, \Lambda_g]$ denote these two quaternions and let $[\lambda_t, \Lambda_t]$ denote an interpolation 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. At the start of execution, a feedback system's error detector is fed two input signals in the form of the current quaternion $[\lambda_c, \Lambda_c]$ and an interpolation quaternion $[\lambda_t, \Lambda_t]$ (representing the desired orientation at time t). 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 Dynamics Module receives the new internal torques 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.

9. Experimental Results

One of the major difficulties encountered with *BDAS* 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 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.

The first experiment tested the general limb motor programs by having the figure perform a simple arm reach to dance position. Figure 9.1 shows the animation sequence generated when the figure is given two seconds to reach this position. All limbs with the exception of the upper arms, lower arms, and hands are set to *maintain*. The remaining limbs are initially marked *move* and allowed to change to *maintain* once they reach their goal position. During the motion sequence, the *balance* process operates to maintain an upright position for the figure. The motion to reach the final position progresses smoothly and appears quite natural. Similar results are obtained when the figure is given 1, 3, and 4 seconds to reach this position.

The next experiment tested a position motion process by having the figure take a single step forward in one second. As much as possible, this motion process applies torques based on biomechanical studies of human gait [McMahon84].

At the beginning of the gait cycle, the motion process temporarily disactivates the figure's balance and applies a small internal torque to the upper body to make the figure fall slightly forward. While the figure is falling forward, added torque is applied to the foot of the support leg to release weight from the swing leg. This torque is applied to make 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 helps anchor the support leg, thus reducing slippage. The swing leg is then swung forward with aid of the *move* motion process. Once the leg is swinging forward, the balance process is reactivated to prevent the figure from falling over. The forward step motor program is then deactivated and the general limb motor programs complete the gait cycle.

The animation sequence generated by this motion process is shown in Figure 9.2. The motion appears fairly natural, although a slight unnatural bend occurs in the support leg while the figure is falling forward. This can be rectified by increasing the stiffness assigned to the figure's joints. Near the end of the cycle, the torque functions draw the legs in slightly faster than is desired. Some horizontal slippage also occurs during the step.

This experiment was extended by having the figure take 12 consecutive forward steps. Throughout the walk, the figure managed to maintain a relatively good position although some steps showed more slippage than others.

The time limit was later increased to 2 seconds per step. Although the figure was capable of taking a step forward within this time interval, more horizontal foot slippage occurred with this slower rate. Attempts to make the figure take 12 steps forward met with only partial success because the figure's increasing momentum tended to make control of its motion more difficult.

Motion processes producing a functional back step and side step have also been developed for the one step per second frequency. These processes, however, are not yet as well debugged as the process for the forward step. Results from limited experiments involving combinations of these steps suggest more control needs to be applied to the figure's motion (directed especially towards its momentum). A better ground model also needs to be implemented because of the large amount of horizontal slippage that occurs during these sequences.

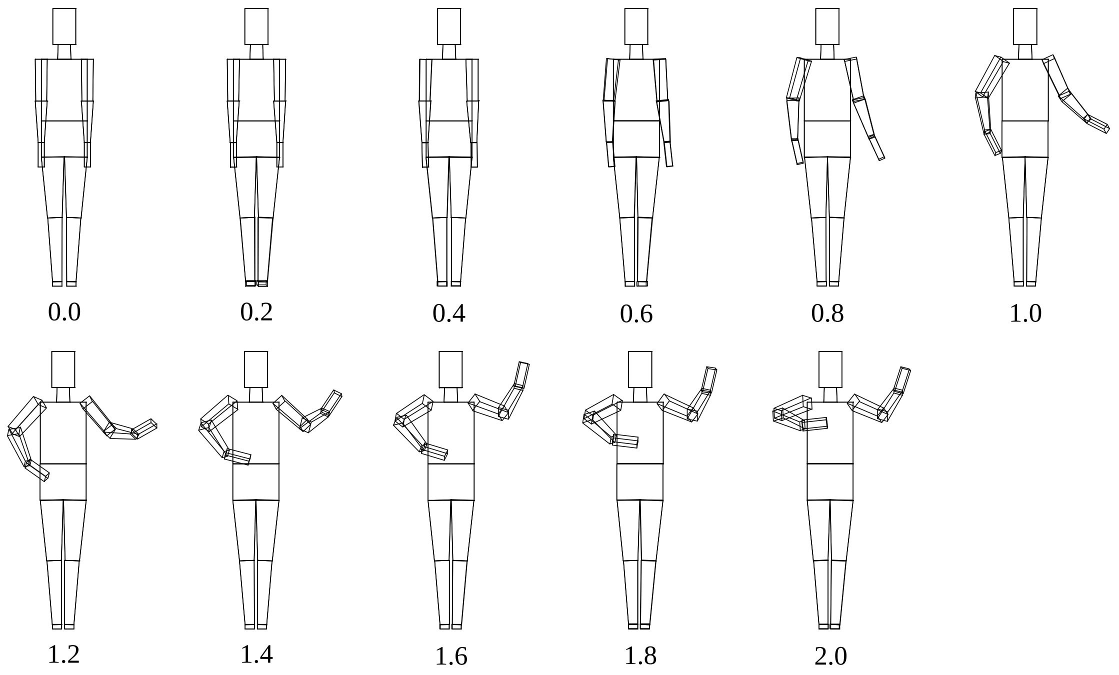

Figure 9.1 Reaching Dance Position in 2 Seconds

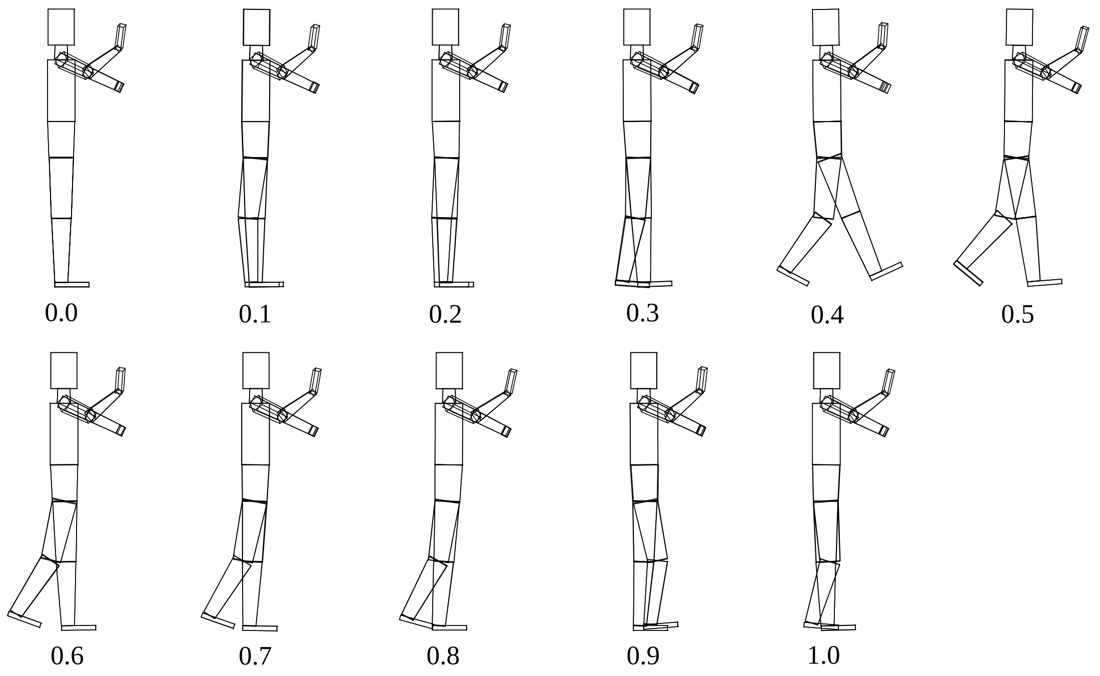

Figure 9.2 Forward Step in 1 Second

10. Conclusions

This paper has presented an animation system designed to dynamically control the motion of an articulated figure by performing interpolations along quaternion curves. All motion sequences are entered using an easy to use, high-level ballroom dance notation language. These motion commands are subsequently translated into forces and torques which are applied to the limbs using by a set of motor programs organized in a hierarchical structure.

Initial results, while encouraging, suggest further control of the figure's motion is required. Improvement in the motion can likely be obtained by applying more biomechanical knowledge as well as additional control theory. While the current implementation uses the same spring and damper constants for each link, this is likely not a realistic configuration. Methods need to be explored for automatically determining acceptable values for each link. A better integration technique needs to be developed for solving the equations so as to reduce the constant by which the moments of inertia for each segment are artificially enlarged.

References

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

Armstrong85.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.

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.

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

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.

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.

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.

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.

McMahon84.T. A. McMahon, Mechanics of Locomotion, *The International Journal of Robotics Reseach 3*, 2 (Summer 1984), 4-28.

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

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

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

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.

Zeltzer82.D. Zeltzer, Motor Control Techniques for Figure Animation, *IEEE Computer Graphics and Applications*, November 1982, 53-59.