INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
The computer animation of legged animals: Simulation, design and control

Girard, Michael J., Ph.D.
The Ohio State University, 1989
THE COMPUTER ANIMATION OF LEGGED ANIMALS: SIMULATION, DESIGN AND CONTROL

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Michael Girard

*****

The Ohio State University

1989

Dissertation Committee
Dr. Bruce Weide
Dr. Thomas Bylander
Dr. Wayne Carlson
Dr. Mohan Ahuja

Adviser’s Approval

Adviser

Computer and Information Science
ACKNOWLEDGEMENTS

My fellow graduate students helped to provide an intellectually stimulating atmosphere for computer animation research, especially Susan Amkraut, Brian Guenter, David Haumann, Tony Maciejewski, John Fujii, George Karl, Robert Lurye and John Chadwick. I am grateful to my advisor, Dr. Bruce Weide, for having faith in my ability and to my readers, Dr. Thomas Bylander, Dr. Wayne Carlson, and Dr. Mohan Ahuja.
VITA

May 19, 1954. . . . . . . . Born - Santa Monica, California
1982 . . . . . . . . . . . . . . B.S. Mathematics, University of California at Santa Cruz, 1980.

1982 . . . . . . . . . . . . . . B.S. Computer and Information Sciences, University of California at Santa Cruz.

PUBLICATIONS


FIELDS OF STUDY

Major Field: Computer and Information Science
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Statement of Problem</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Related Work in Robotics, Biomechanics and Computer Graphics</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Overview of the Dissertation</td>
<td>6</td>
</tr>
<tr>
<td>II</td>
<td>Modelling Limb Motion</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>Representing the Kinematics of Limbs</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Forward and Inverse Kinematics</td>
<td>9</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Controlling Redundant Limbs in PODA</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Dynamics of Limbs</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Recursive Newton-Euler Inverse Dynamics</td>
<td>17</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Simulation of Forward Dynamics</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>Time-Complexity of Dynamics Algorithms</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>Interactive Design of Limb Motion</td>
<td>23</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Interactive Limb Positioning</td>
<td>23</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Composition of Limb Posture Sequences</td>
<td>24</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Speed Control by Arc-length Reparametrization</td>
<td>25</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Interactive Design of Speed</td>
<td>28</td>
</tr>
<tr>
<td>2.5</td>
<td>Modeling Natural Limb Motion</td>
<td>31</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Expressive Movement and Weighted Performance Optimization</td>
<td>32</td>
</tr>
<tr>
<td>2.5.2</td>
<td>The Optimal Velocity Distribution Problem</td>
<td>35</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Solving for Optimal Trajectories with Dynamic Programming</td>
<td>37</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>2.5.4 Optimizing Splined Trajectories with Dynamic Programming</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>III Modelling Coordinated Body Motion</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>3.1 Newtonian mechanics and Sensory-Motor Coordination</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>3.2 Periodic Behavior: Locomotion</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>3.2.1 Leg Support and Stepping Movements</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>3.2.2 Leg Motion Adaptations to Body Motion</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>3.2.3 Periodic Gait Design</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>3.2.4 Gait Shifting</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>3.2.5 Body Dynamics and Support Relations</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>3.3 Non-periodic Behavior: Dance</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>3.3.1 Limb and Body Trajectories</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>3.3.2 Vertical Dynamics of Body Movement</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>3.3.3 Horizontal Dynamics of Body Movement</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>3.3.4 Angular Dynamics: Posture Design vs. Dynamic Stability</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>IV PODA Animation System Design and Implementation</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>4.1 Overview of PODA System Structure</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>4.2 User-Interface and Interactive Control</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>4.2.1 Menu-Driven Control</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>4.2.2 Real-time Playback vs. Real-time</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>4.3 Software Engineering of Animation Systems</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>4.3.1 Utility of Object-Oriented Programming</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>4.3.2 Integration with Other Behavioral Models</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>V Conclusion</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>5.1 Contributions</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>5.2 Future Research</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>98</td>
<td></td>
</tr>
</tbody>
</table>
List of Figures

1.1 3D Computer Animation ........................................ 2

2.1 Link Parameters Associated with Link i .................. 8
2.2 Forward and Inverse Kinematics ........................... 12
2.3 Forward and Inverse Dynamics ............................. 16
2.4 Scheme for Arc-length Reparametrization ............... 27
2.5 Interactive Design of Speed ................................. 29
2.6 Shape of Graph for Linear Movements .................. 32
2.7 Minima in Hand Speed Occur at Maxima in Path Curvature . . . . 33
2.8 m variations about a position at time i ................. 38
2.9 Part of the Dynamic Programming Graph with m=2 and c=1 . . . 39
2.10 Optimal Minimum Cost Path with m=2 and c=1 ........ 40
2.11 Resolution and Range of Acceleration Assuming Initial Constant Speed Distribution ........................................ 41
2.12 Splined Path Segment Variations .......................... 43
2.13 Flow Chart of Path Segment Performance Computation ...... 45
3.1 The Reference Leg Position .......................................................... 52
3.2 Relative Phases for Gaits of Quadrupeds ................................................... 56
3.3 The Stroke Parameter ................................................................. 57
3.4 The Gait Cycle Period .............................................................. 58
3.5 Pitching motion of quadruped ......................................................... 62
3.6 Flow of Control of Locomotion Model ................................................. 64
3.7 Velocity Graph of Support Interval's Vertical Dynamics ......................... 66
3.8 Modelling the degree of stiffness in the knees ....................................... 68
3.9 The Dynamically Stable Banking Angle ............................................. 71
3.10 Sequence of eight body postures. Postures numbered 4 and 5 are in
   a non-supported state. ................................................................. 74
3.11 Flow of Control of Dance Model .................................................... 82
4.1 PODA'S Module Interconnection Graph ............................................ 84
4.2 Menu Hierarchy ................................................................. 87
4.3 Inheritance in Legged Animal Animation Systems ............................. 91
4.4 Integration of Computational Models for Animation ........................... 93
CHAPTER I

Introduction

The most striking and revolutionary application of computer technology to image production lies in the field of three-dimensional image-synthesis. In this approach, the photographic process of capturing an image of a three-dimensional scene is computationally simulated in terms of the perspective, illumination, and the reflectance properties of the objects in the scene. In broad terms, the functionality of three-dimensional computer image-synthesis is to map three-dimensional representations into two-dimensional images.

With the advent of image-synthesis techniques, it became clear that if one transformed the three-dimensional representations used by the display program over time, one could produce three-dimensional computer animation. [see fig. 1.1]. The focus of research in computer animation is, therefore, the study of the representation and procedural organization of way things move and interact in three dimensions.

Given our rich mental and perceptual associations with the real world, com-
computer animation research has become increasingly concerned with developing computational models for the simulation and control of realistic motion phenomena. Building computational models for expressive realistic movement is an exciting interdisciplinary marriage of science, engineering, and art. Engineering must be employed to build models capable of complex natural behavior, science must be used to measure the accuracy of models tested against empirical evidence, and artistic considerations must be given to defining how the animator will interact with the constructed model.

Although all natural phenomena are subject to the rules of Newtonian mechanics, different laws of organization and behavior must be developed to describe the broad diversity of inanimate and living things. Therefore, it is proper that research
in computer animation should be specialized to phenomena which share the same computational characteristics in their simulation, design, and control.

1.1 Statement of Problem

The phenomena addressed by this dissertation is the class of legged animals. Although this modeling problem has been pursued in the computer graphics/animation literature, the great body of research directed toward understanding how legged animals move has been carried out in the biomechanics and robotics fields. Indeed, much of the work presented in this dissertation builds directly on knowledge gained from progress in these fields.

However, the problem of the computer animation of legged animals has its own distinguishing features which set it apart. The first is that the goal of animation is to create the illusion of reality. For this, we need only compute the visual representation of the animal's motion. In other words, since the output is a sequence of images to be interpreted by a human, how we perceive and understand motion is of prime relevance. In contrast to robots and the real animals studied in biomechanics, the computation of physical quantities, such as joint torques, is useful only insofar as it helps us produce more convincing, physically credible visual motion.

Secondly, in computer animation, there is the additional issue of how an animator designs the behavior of legged animals. How do we specify and refine a complex series of movements drawn from our imaginations? How do we make our
mental designs of legged animal motion conform to physical and sensory-motor constraints embedded in a computational model?

1.2 Related Work in Robotics, Biomechanics and Computer Graphics

Robotics research has made great strides toward developing computational models for the control of industrial manipulators and the locomotion of legged machines. Denavit and Hartenberg [Denavit 55], Whitney, [Whitney 69], and Liegeois [Liegeois 77] have been instrumental in developing a theory of kinematic representation and control which may be applied to limbs having any number of joints. Recent developments in the efficient computing of manipulator dynamics (newtonian mechanics), described in section 2.2, are primarily due to the efforts of Luh [Luh 80], Walker [Walker,Orin 82] and Featherstone [Feather 88].

The study of legged locomotion systems, especially the walking machines pioneered by McGhee [McGhee 79] and Orin [Orin 82] has helped to formalize our notions of gait, foothold planning, and issues in static stability. Raibert's research on hopping and running machines has produced some relatively simple control algorithms for maintaining balance and dynamic stability [Raibert 84] [Murphy 84] [Raibert 88].

While the robotics field has been concerned with the engineering of limbed systems, the biomechanics field has sought to understand the nature of biological
limbed systems which already exist. The work of Morasso, [Morasso 83], Ivaldi [Ivaldi 82], Flash [Flash 84], and Hogan [Hogan 86] has helped illuminate the underlying control structure which animals use to perform coordinated limb motions. Kane has developed detailed models of the dynamics of body motion, and his work clarifies the difficulty in controlling body orientation while the body is either falling [Smith, Kane 68] or in zero-gravity [Kane 70].

In the computer graphics field, only the kinematic issues of representing and animating legged animal motion have been addressed. Zeltzer, [Zeltzer 84], Steketee [Steketee 85], and Korein [Kore, Badlr 82] do not attempt to model the dynamics and control needed for realistic motion.

On the other end of the spectrum, the application of robot dynamics simulation to the computer animation of limbs has been investigated by Wilhelms [Wilhelms 85], Armstrong and Green [Armstrong 85] [Armstrong 86], and Isaacs [Isaacs 88]. However, in their work, it is tacitly assumed that simulation of the muscular control of limbs will scale up to handle complex body movements. A key point emphasized in this dissertation is that sensory-motor control of limb motion and higher-level behavior, such as walking, has an intelligent goal-directed component which must be planned before it can be dynamically simulated. For coordinated movements, the locus of the control system resides at a kinematic level before being “parsed” into mechanical actions.
1.3 Overview of the Dissertation

The next chapter describes the problems of animating limb motion, beginning with the kinematics and dynamics of limbs, and then followed by a discussion of the higher level optimal control issues related to naturalistic motion. The chapter on modelling limb motion culminates with a new algorithm, devised by the author, for optimizing limb trajectories using dynamic programming.

The third chapter will document the author's formulation of models for animating coordinated body motion. Two different computational models have been implemented. The first one, described in section 3.2, is specialized for animating periodic locomotion [Girard, Maci 85]. The more general model, which is capable of animating non-periodic movements (such as is found in dance) is described in section 3.3 [Girard, 87].

Finally, the fourth chapter will discuss the user-interface and software development problems confronted in the author's implementation of the PODA computer animation system, which is an interactive, menu-driven program specialized for the animation of legged animals.
CHAPTER II

Modelling Limb Motion

2.1 Representing the Kinematics of Limbs

An arbitrary limb may be specified using the kinematic notation presented by Denavit and Hartenberg [Denavit 55]. In their notation, a coordinate system for every individual degree of freedom present in the limb is parametrically specified. Four parameters are used to define the transformation between adjacent coordinate systems: the length of the link \( a \), the twist of the link \( \alpha \), the distance between links \( d \), and the angle between links \( \theta \) (see fig. 2.1).

Given the above definitions, it can be shown [Paul 81] that the transformation between adjacent coordinate frames \( i - 1 \) and \( i \) denoted by \( i^{-1}T_i \) is given by the homogeneous transformation:

\[
i^{-1}T_i = \begin{bmatrix}
\bar{n} & \vec{\theta} & \vec{d} & \vec{p} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

where

\[
\bar{n} = (\cos \theta_i, \sin \theta_i, 0)^T
\]
Figure 2.1: Link Parameters Associated with Link $i$

$$\vec{d} = (- \cos \alpha_i \sin \theta_i, \cos \alpha_i, \cos \theta_i, \sin \alpha_i)^T$$

$$\vec{a} = (\sin \alpha_i \sin \theta_i, - \sin \alpha_i \cos \theta_i, \cos \alpha_i)^T$$

$$\vec{p} = (a_i \cos \theta_i, a_i \sin \theta_i, d)^T$$

By repeatedly applying successive link transformations the relationship between any two coordinate systems $i$ and $j$ is obtained by

$$^iT_j = ^iT_{i+1}^iT_{i+2} \cdots ^iT_{j}$$

The above equation transforms any link of the limb, defined in its own coordinate frame, to the world coordinate frame. Furthermore, the geometric constraints which determine the degrees-of-freedom at each joint are represented in general form, thus providing for a mathematical mapping between the joint angle
or joint-space and world-space or cartesian-space representations of the limb.

2.1.1 Forward and Inverse Kinematics

Given the above framework, one can easily see that, given the state of the joint angle variables, we can compute the position of all of the links and arrive at the position of the end of the limb. This is called the forward kinematics problem. The reverse situation, that of computing the joint angles from the position of the end of the limb must be considered if we wish to place a foot or hand in some desired place – what Korein and Badler have called “goal directed motion” [Kore,Badlr 82]. This is called the inverse kinematics problem. The legged locomotion models in PODA rely heavily on the need for goal-directed motion: feet must move along trajectories, be placed exactly at desired footholds and held in place as the body passes over them.

In the PODA system, the usual joint-angle specification of a key posture for a limb is augmented by the position/orientation of the end-effector (hand or foot). Limbs (and the spine) may be interactively manipulated into key positions using both forward and inverse kinematics.

Because the animal limbs we wish to model typically have limb geometries which have more than six degrees-of-freedom, we must adopt inverse-kinematic techniques suitable for the class of redundant limbs. Resolved motion rate or pseudo-inverse jacobian control is used for this purpose. This technique linearizes
the kinematic equations of motion about a point and solves for the position of the end-effector in terms of its velocity or “rate” [Whitney 69], [Klein 83], [Liegeois 77], [Maciej 85] A recent publication [Chang 86] presents an alternative solution that yields the end-effector position directly, but which results in a set of non-linear equations that must be solved numerically.

Inverse-kinematic control of redundant limbs takes the following simplified form:

\[ \theta = F(X, \Lambda) \]

where \( X \) is the desired cartesian-space position of the end-effector

\( \Lambda \) is the desired joint-space position

\( \theta \) is the actual joint-space position

and \( F \) is the inverse-kinematic function

The solution that places the end-effector at the desired cartesian position \( X \) is found which, as a secondary goal, minimizes its deviation from the desired joint-space position \( \Lambda \).

More specifically, rate control is based on the fact that a six-dimensional, incremental change in position and orientation for each link varies linearly with an incremental change in the limb’s joint angles. The matrix of linear factors form the Jacobian matrix \( J \) through the equation

\[ \Delta \bar{x} = J(\bar{\theta}) \Delta \bar{\theta} \] (2.1)
It can be shown that that the Jacobian is given by [Waldron 81] [Orin 83]:

\[ J = \begin{bmatrix} p_1 \times a_1 & p_2 \times a_2 & \ldots & p_n \times a_n \end{bmatrix} \] (2.2)

where \( a_i \) and \( p_i \) are the third and fourth columns, respectively, of the transformation matrix \( ^0T_{i-1} \). The first column of the Jacobian is given by

\[ p_1 = [0 \\ 0 \\ 0]^T \quad a_1 = [0 \\ 0 \\ 1]^T. \]

The Jacobian maps linear incremental changes in the limb’s joints into incremental changes in the end-effector’s position. The inverse mapping, however, is needed to solve the inverse-kinematic problem, where we wish to determine the required \( \Delta \bar{q} \) in joint-space to achieve a desired \( \Delta \bar{x} \) of the end-effector in cartesian space.

However, the inverse of the Jacobian will not be square for redundant limbs since they have more than six degrees of freedom. In these cases, the inverse mapping may be found by use of the pseudoinverse [Liegeois 77] [Klein 83] [Maciej 85] given by:

\[ A^+ = \begin{cases} (A^T A)^{-1} A^T & \text{if } m > n = r \\ A^T (A A^T)^{-1} & \text{if } r = m < n \end{cases} \]

2.1.2 Controlling Redundant Limbs in PODA

For redundant limbs, the joint-space is of higher dimension than the six-dimensional cartesian space of the end-effector. Therefore, in the redundant case, the Jacobian transformation has a null space containing an infinite number of joint-space rates
which will produce no end-effector motion (see fig. 2.2). This, of course, implies

that there will also be an infinite number of solutions for non-zero end-effector
movements, since null-space joint rates may be added to any limb movement with­
out influencing the end-effector.

Klein and Huang, in their excellent review of psuedoinverse control [Klein 83],
note that the pseudoinverse mapping produces the minimum norm solution to
equation (2.1), thus moving the joints as little as possible to achieve some new
desired end-effector position. However, they also observed that for the least squares
minimum norm solution, periodic end-effector trajectories will not produce periodic
joint-space solutions. This has the problem that, in the context of animation, links
will appear to be loosely connected objects which drift as if they are being pulled
by the end of the limb.

A more general solution which prevents joint-space drifting is given by [Whitney 69] [Liegeois 77] [Klein 83] [Maciej 85]:

$$\Delta \dot{\theta} = J^+ \Delta \bar{x} + (I - J^+ J) \bar{z}$$  \hspace{1cm} (2.3)

where $I$ is an $n \times n$ identity matrix and $z$ is an arbitrary vector in $\Delta \theta$-space. The matrix operator $(I - J^+ J)$ projects any desired joint-space rate vector $\bar{z}$ into the null space of the Jacobian, thereby insuring that the desired end-effector motion is not altered.

The most natural way of specifying a desired joint-space position is to move toward a rate $\bar{z}$ which minimizes the limb's deviation from some desired joint-space position $\Lambda$. That is, we compute $\bar{z}$ in equation (2.3) as

$$z = \nabla H$$

with

$$H = \sum_{i=1}^{n} \alpha_i (\theta_i - \Lambda_i)^2$$

where $\theta_i$ is the $i$th joint angle, $\Lambda_i$ is the center angle of the $i$th joint angle, and $\alpha_i$ is a center angle gain value between zero and one.

The center angles define the desired joint angle positions and their associated gains define their relative importance of satisfaction. From the animator's point of view, the gains may be thought of as “springs” which define the stiffness of the
joint about some desired center position [Ribble 82]. PODA provides interactive specification of center angles and gains as a means of controlling the redundant degrees of freedom in the legs, arms and spine of the legged animal.
2.2 Dynamics of Limbs

The kinematics of a limb’s structure define its geometric and topological characteristics. By contrast, the dynamics of movement is concerned with its Newtonian mechanical properties. That is, given the mass and moment of inertia of each link in the limb, and the presence of external forces, dynamics defines the relationship between the forces applied at the joints and the acceleration, velocity and position of the limb.

In this section, algorithms for robot dynamics will be presented, drawn in accordance with Featherstone’s notation and description in [Feather 88]. These dynamics formulations provide a robust approach to limb motion, in which the optimization of “physically-based” functions rooted in dynamics may be incorporated into the synthesis of limb motion which I will discuss in Section 2.5.

The computation of accelerations (given current position and velocity) from joint torques is called the forward dynamics problem. Solving this problem allows us to simulate the dynamics of the linked system from given input torques.

However, as noted by Wilhelms [Wilhelms 85], Armstrong [Armstrong 85], and Isaacs [Isaacs 88], the use of forward dynamics must be coupled with control strategies if coordinated goal-directed movement (or animation) is desired. The control of real robot manipulators is typically carried out by planning an ideal reference trajectory, and then calculating the torques required to keep the manipulator on
the prespecified path. The problem of computing torques to achieve desired motion is the \textit{inverse-dynamics} problem (see fig. 2.3).

![Diagram of Forward and Inverse Dynamics](image.png)

\textbf{Figure 2.3: Forward and Inverse Dynamics}

Solutions to the forward and inverse-dynamics problem for linked systems have been well understood in the robotics field for many years. A comprehensive treatment of robot dynamics may be found in [Feather 88] [Hollerb 1982]. It has been found that recursive solutions may be formulated [Luh 80], [Walker,Orin 82], [Armstrong 85] which are dramatically faster than the earlier non-recursive methods. In the following I will review one of these recursive techniques and then present a comparison of its time complexity with the earlier non-recursive approaches.
2.2.1 Recursive Newton-Euler Inverse Dynamics

The non-recursive general equation for the inverse-dynamics of limbs takes the form:

\[ Q_i = \sum_{j=1}^{n} H_{ij} \ddot{\theta}_j + \sum_{j=1}^{n} \sum_{k=1}^{n} C_{ijk} \dot{\theta}_j \dot{\theta}_k + g_i \]  

where

- \( Q_i, \ddot{\theta}_i, \dot{\theta}_i \) are the joint forces, accelerations, and velocities, and

- \( H_{ij}, C_{ijk}, g_i \) are the inertial, Coriolis and gravitational coefficients (which change with the limb state) [Feather 88].

The Newton-Euler method calculates joint torques \( Q \), given joint accelerations \( \ddot{\theta} \) by using recursion to eliminate the execution of repeated calculations. The recursion proceeds in two directions. The forward execution starts from the base of the limb and advances toward its end, recursively accumulating values for the net velocities, accelerations and forces acting on each link. Once these net values are known, a backward procedure, beginning at the end of the limb and proceeding toward its base, computes the recursive relations of the forces transmitted between connecting links and the torques which must be applied at the joints to realize these forces. The precise algorithm [Feather 88], is given below:
• The Forward Calculation: From Link 1 to Link N

1. \( v_i = v_{i-1} + s_i \ddot{q}_i \) (with \( v_0 = 0 \))

2. \( a_i = a_{i-1} + \dot{v}_i \times s_i \ddot{q}_i + s_i \dddot{q}_i \) (with \( a_0 = 0 \))

3. \( F_i = I_i a_i + v_i \times I_i \dot{v}_i \)

• The Backward Calculation: From Link N to Link 1

1. \( f_i = f_{i+1} + F_i + E_i \) (with \( f_n = F_n \))

2. \( Q_i = s_i f_i \)

where:

- \( q_i \) is the position of joint \( i \)
- \( v_i \) is the velocity of link \( i \)
- \( a_i \) is the acceleration of link \( i \)
- \( F_i \) is the net force on link \( i \)
- \( E_i \) is an external force on link \( i \)
- \( f_i \) is the force exerted on link \( i \) by link \( i - 1 \)
- \( s_i \) is the rotation axis vector of joint \( i \)
- \( Q_i \) is the joint torque exerted at joint \( i \)
PODA implements Luh's [Luh 80] more efficient implementation by using a reformulation of the recursive equations cast in link coordinates (rather than world-space coordinates). Here the moment of inertia tensor $I_i$ is constant, so we eliminate the need to transform it to world-space at every new limb position. However, the essential structure of the forward and backward recursive equations in the link coordinate reformulation (and the order of the time-space complexity) are the same as the simpler world-space version presented above [Feather 88].

2.2.2 Simulation of Forward Dynamics

In the real world, limbs move when forces are applied. The computation of forward dynamics is required for simulating the actual behavior of real mechanical systems and robot manipulators. Forward dynamics computations fall into two categories. The first calculates recursion coefficients and propagates motion and force constraints in the spirit of the recursive inverse-dynamics calculations discussed earlier. Armstrong, Green [Armstrong 85] and Featherstone [Feather 88] have devised algorithms which belong to this class.

The second approach is to solve the set of simultaneous equations for the unknown accelerations $\ddot{q}$ in:

$$Q = H(q)\ddot{q} + C(q, \dot{q})$$

This is the most widely used method due to its speed for limbs having less than
Walker and Orin [Walker, Orin 82] have devised and analyzed several methods which solve these simultaneous equations efficiently by utilizing recursive techniques. In the computer graphics literature, Isaacs [Isaacs 88] also uses the simultaneous equation solution method, but his approach is nonrecursive and therefore much more inefficient.

The Walker-Orin approach, as described in [Feather 88], breaks down into three computational steps:

Given joint torques $Q$ in state $(q, \dot{q})$, find joint accelerations $\ddot{q}$

1. Solve for the elements of vector $C$

2. Solve for the elements in matrix $H$

3. Solve the linear equation $H \ddot{q} = (Q - C)$ for $\ddot{q}$

The first step, the calculation of $C$, is accomplished by noting if the acceleration $\ddot{q}$ is set to zero, the dynamics equation becomes: $Q = C(q, \dot{q})$. Hence $C(q, \dot{q})$, which encompasses the Coriolis and gravity terms, may also be conceived as the joint torques which eliminate joint accelerations at a given limb position and velocity. Therefore $C$ may be computed by executing the inverse-dynamics algorithm $InverseDyn(q, \dot{q}, \ddot{q})$ at the limbs current position and velocity with zero

---

1. Its computational complexity is greater, as we shall see in the next section.
acceleration. That is:

\[ C(q, \dot{q}) = \text{InverseDyn}(q, \dot{q}, 0) \]

The second step, computing the inertia coefficients matrix \( H \), may be realized by rearranging the dynamics equation with \( \text{InverseDyn}(q, \dot{q}, \ddot{q}) = Q \) (true by definition) to yield:

\[ \text{InverseDyn}(q, \dot{q}, \ddot{q}) = H\ddot{q} + C \]

\[ \Rightarrow \text{InverseDyn}(q, \dot{q}, \ddot{q}) = H\ddot{q} + \text{InverseDyn}(q, \dot{q}, 0) \]

\[ \Rightarrow H\ddot{q} = \text{InverseDyn}(q, \dot{q}, \ddot{q}) - \text{InverseDyn}(q, \dot{q}, 0) \]

However, since all the non-acceleration terms (for velocity and gravity) on the right hand side of the above equation will cancel, we need only compute the acceleration terms, a procedure I will call \( \text{InverseDyn}^2(q, \ddot{q}) \). With this substitution, we get:

\[ H\ddot{q} = \text{InverseDyn}^2(q, \ddot{q}) - \text{InverseDyn}^2(q, 0) \]

\[ \Rightarrow H\ddot{q} = \text{InverseDyn}^2(q, \ddot{q}) - 0 \]

\[ \Rightarrow H\ddot{q} = \text{InverseDyn}^2(q, \ddot{q}) \]

Given this equality, Walker and Orin calculate the columns of \( H \) by setting \( \ddot{q} \) to \( \gamma_i \) where \( \gamma_i \) is a vector with all zero elements except for the \( i \)th element, which is set to one. That is:

\[ H_i = \text{InverseDyn}^2(q, \gamma_i) \]
Now that $H(q)$ and $C(q, \dot{q})$ are known, The third step is to solve the linear vector equation for the desired accelerations $\ddot{q}$ from the given torques $Q$:

$$ H \ddot{q} = Q - C $$

### 2.3 Time-Complexity of Dynamics Algorithms

The investigation of efficient robot dynamics algorithms which exploit recursion has occupied researchers for the last decade, beginning with Walker's inverse-dynamic formulation in 1979 [Walker, Orin 82] which was refined by Luh, Walker and Paul in 1980 [Luh 80]. Recursive algorithms were the first to have $O(n)$ time complexity [Feather 88]. Non-recursive methods are much slower due to their explicit calculations of the $O(n^2)$ quantities in $H_{ij}$ and $O(n^3)$ quantities in $C_{ijk}$. [Feather 88]. In practice, the difference between the two is dramatic. For example, for a six degree of freedom limb, the recursive Newton-Euler method was over 100 times faster than the non-recursive approaches [Feather 88], [Hollerb 1982], [Luh 80]

Armstrong and Featherstone have proposed $O(n)$ forward dynamics algorithms which propagate values recursively, in the spirit of the inverse-dynamics calculations. However, Featherstone points out that for $n < 9$, the $O(n^3)$ linear equation solvers which use inverse-dynamics to calculate $H$ and $C$ (described above) are more efficient due to the small coefficients associated with their $n^3$ terms [Feather 88].
2.4 Interactive Design of Limb Motion

Given the basic procedures for modelling the kinematics and dynamics of limbs, the animator still needs a means of specifying or designing how an arm or leg should move. In this section, the issue of interactively designing the posture and movement of limbs will be addressed.

2.4.1 Interactive Limb Positioning

With the availability of both forward and inverse-kinematics, the interactive positioning of limbs may be accomplished by either movement of the end-effector, rotation of the joints, or rotation of the joints with the end-effector stationary (when possible). Movement of the end-effector results when the user alters the cartesian coordinates or orientation of the desired position $X$. In this mode, the program solves the inverse-kinematics problem to bring the end-effector to the new desired position. This mode is essential for placing feet and hands at specific places in the environment, e.g., placing the feet on the ground, putting the hands on the hips, or cases involving reaching behavior.

There are times when it is more convenient to move a limb by changing its joint angles rather than positioning its hand or foot. This is particularly true for forming postures which are meant to occur during joint-interpolated movements, such as the swinging of arms or the kicking of legs. The user may select and move a joint forward and backward. In such cases, the program employs forward kinematics to
update the limb and its new end-effector position.

The third mode allows the user to fine-tune the posture of a limb once its end-effector has been constrained to be anchored to some desired position. This is achieved by adjusting the desired joint angles in \( \Lambda \) and then solving the inverse-kinematics problem.

The user may freely alternate among these three modes in order to make the limb's position converge toward some desired configuration.

### 2.4.2 Composition of Limb Posture Sequences

In formal terms, we will define a limb trajectory in terms of the path which the limb negotiates, and a function that controls the speed of the limb along its path. Before embarking on the problem of controlling the speed of movement, I will first describe the interactive specification of the limb's path. The path is defined by a sequence of limb postures, called the *posture sequence*, which the animator desires the limb to pass through. A *posture* is composed of a limb's position in both cartesian-space (the posture's end-effector position) and joint-space (the posture's joint-angles). The actual path taken through these postures may be a function of spline-interpolation of the joint angles or a 3-D interpolating-spline which passes through the end-effector coordinates of each posture in the sequence.

In PODA, each limb has two associated tables which act as posture and path "memories." A *posture table* stores postures which the animator would like to
retrieve at a later time. This is necessary for cyclic motions where we wish the limb to return to previously assumed postures. The animator may append, overwrite, retrieve and delete postures from the limb's posture table by using a menu-driven interface.

The animator assembles postures into a working posture sequence. An interactive menu provides the animator with a means of appending, inserting, replacing, and deleting postures from the working sequence. Once a satisfactory working sequence has been created, it may be stored in the limb's posture sequence table. Sequences of postures which represent different limb paths are remembered and subsequently recalled from this table. As with the posture table, the animator may append, replace, retrieve and delete posture sequences from this table.

2.4.3 Speed Control by Arc-length Reparametrization

A posture sequence roughly describes the path the limb is to follow. The speed at which the limb moves along that path must also be specified. There have been a number of attempts in the computer animation field to deal with this problem. In parametric key-frame animation, interpolating splines are employed to specify position as a function of the spline's parameter: \( \text{spline}(u) = p \). Typically time is assumed to be proportional to the parameter \( u \) of the interpolating spline. That is, \( \text{spline}(u(t)) = p \) where \( u(t) = Kt \). Kochanek has shown that one may vary the tangential speed along Catmull-Rom cubic splines about a control point by
changing the magnitude of the tangent vectors at that point, a parameter she calls "tension" [Kochanek 84]. However, the shape of the path is also altered in the process.

Steketee and Badler [Steketee 85] introduced a "double-interpolant" method which employs a B-spline curve to represent \( u(t) \), thereby allowing variation from the usually assumed linear relationship. Although their technique provides for the variation of parametric speed, \( \|du(t)/dt\| \), this may be dramatically different from the actual speed in the geometric sense, \( \|d(spline(u))/dt\| \).

A central problem in using parametric splines for the specification of motion is that equal lengths in the parameter \( u \) will not, in general, map into equal increments along the curve in Cartesian space. Unevenly spaced control points will result in variations of speed along the curve. What is needed for precise control of speed is a reparametrization by arc-length of the interpolating spline. If the function \( h(u) \) gives the arc-length of the curve, the parameter \( s \), the inverse of \( h \), will yield \( u \) values which map equal increments of \( s \) into equal increments along the curve (see fig. 2.4). The reparametrized curve \( \text{reparSpline}(s) = \text{spline}(h^{-1}(s)) \) has the property that \( \|d(\text{reparSpline})/ds\| = 1 \) [Millman 77].

Although a reparametrization by arc-length for any regular curve theoretically exists, in practice we may not be able to find an analytical solution. Unfortunately, this is the case for spline curves. The norm of the derivative of the spline (the tangent vector) must be integrated and then inverted.
Since this is not practically solvable, we must use numerical techniques to find $h$. A Gaussian quadrature procedure may be used to solve this problem [Mortenson 85]. Of course, this is not sufficient since we still need $h^{-1}$. We approximate $h^{-1}$ by first creating a look-up table with entries for arc-length $h$ for small increments in the spline parameter $u$. We may then get the inverse of $h$ by applying linear interpolation to the table.

With the use of the $h^{-1}$ table, we may now precisely control both the distance and speed of an object (in our case, a limb) which travels along its path.
2.4.4 Interactive Design of Speed

Although the speed may be controlled independently from the path, in the context of limb trajectory motion, one typically wants to design the speed function in relation to postures which define the path. If speed were designed solely over time, we could not easily control how changes in acceleration occurred about specific postures. The animator requires a means of designing speed in relation to specific distances associated with postures alongs the limb’s path.

A solution is to provide the animator with a distance(time) graph control mechanism, a graph which represents distance as a function of time. In such a graph, the vertical axis represents distance. Since the postures in the posture sequence of the limb occur at known distances along the path, we may represent these distances by placing horizontal dotted-lines along the vertical axis (see fig. 2.5). Since the horizontal axis is time, the derivative with respect to time \( \frac{d(dist)}{dt} \) — the tangent vector along the graph — represents speed. Seen in this way, the animator may adjust the speed about specific postures by interactively designing the slope of the graph as it crosses the horizontal lines corresponding to distances at which the postures fall.

The points of inflection on the distance(time) graph are analogous to extrema on a speed(time) graph. We note that if distance(time) graph is scaled in the time dimension, the points of inflection will still occur at the same distances. The net effect is that the speed will change in proportion to the degree of scaling in
Figure 2.5: Interactive Design of Speed
time without changing the relative acceleration in relation to distance. Therefore, the distance(time) graph may be normalized to an arbitrary range. The character of motion is preserved regardless of how the graph is contracted or stretched to accommodate any desired duration.

In order for the animator to have fine control, the distance(time) graph/function must be amenable to modification. In effect, we must be able to design the shape of this graph. The “double interpolant” method introduced by Steketee and Badler [Steketee 85] employed a 2-D spline or “kinetic spline” as a means of specifying a \( u(\text{time}) \) graph. We apply their method to shaping distance(time) graph. Note that their procedures for creating smooth transitions in the “parametric speed” between two motions may be applied equally well in our context for transitions in actual “geometric speed.”

Transforming a 2-D spline into a graph requires that we interpret a point \((x, y)\) on the 2-D curve as \((\text{time}, \text{distance}(\text{time}))\). Once again we are faced with the problem that equal increments in \( \text{spline}(u) \) will not map into equal increments in \( x \). That is, for time \( t_0 \), we must find \( u \) such that \( \text{spline}(u)_x = t_0 \). Then \( \text{spline}(u)_y \) is the distance at \( t_0 \). Numerical methods (such as Newton-Raphson) may be applied to solve for \( u \) with \( \text{spline}(u)_x - t_0 = 0 \).
2.5 Modeling Natural Limb Motion

In the previous section, I described a model for designing motion having any characteristics the animator desires. The trajectories need not conform to any laws of behavior which govern motion in the real world. This is fine if we are not concerned with producing limb motion which appears to look natural. But if we are, we must look for higher-level models of limb movement which embody the dynamics and control strategies employed by real animals. The design of limb motion should be supported by a parametric model which captures the essential patterns of natural limb behavior.

The specification and control of limb trajectories is an active area of research in the robotics field. The problem of trajectory planning is to synthesize a nominal path and speed function which moves a limb (or industrial manipulator) between desired postures. The trajectory planning problem is to solve for the trajectory of the limb in terms of some optimization criteria (such as minimum-time, minimum-jerk or minimum energy) in the presence of dynamic and kinematic constraints (such as maximum achievable joint torques, kinematic joint limits and environmental obstacles).
2.5.1 Expressive Movement and Weighted Performance Optimization

Empirical studies have been conducted to determine how humans and other animals solve the trajectory planning problem. Especially noteworthy are the analyses of unrestrained arm movements in humans. It was discovered that the normalized speed(time) graph of the hand’s motion between two stationary points in space was invariant with respect to the points chosen, the load carried, the speed of motion and the distance traveled. Furthermore, the shape of this graph matched that predicted by an optimization for minimum-jerk about the hand [Atkeson 85], [Atkeson 84], [Morasso 83], [Flash 84], [Hogan 86]. (see fig. 2.6).

![Normalized Speed(time) Graph](image)

Figure 2.6: Shape of Graph for Linear Movements
Stated mathematically, we wish to form a path $\zeta(t)$ such that the mean squared magnitude of the cartesian jerk (the third derivative, the hand's rate of change of acceleration) over that path is minimized. That is we wish to minimize:

$$J = \int_{t_0}^{t_1} \left| \frac{d^3 \zeta(t)}{dt^3} \right|^2 dt$$

(2.6)

The notion that limb trajectory formation is formulated in terms of the hand's coordinate system is also supported by Morasso's study [Morasso 83] of free-form arm movements. He found that the tangential speed of the hand was inversely proportional to the curvature of its path [see fig. 2.7]. In other words, much like a race-car, the hand tends to speed up on the "straightaways" and slow down on the curves. A minimization of jerk found in the point to point movements would also

Figure 2.7: Minima in Hand Speed Occur at Maxima in Path Curvature
produce the maximally "smooth" relationship for curved movements [Hogan 86], [Atkeson 84], [Flash 84].

It is clear, however, that not all limb movement may be characterized by control strategies which optimize end-effector trajectories. In contrast to coordinated arm movements, swinging motion, such as is found in legs during walking, is best understood as a minimization of expended energy or work rather than jerk about the foot [Nagurka 71], [Ghosh 76], [Nelson 83].

In summary, the empirical evidence supports a model of natural trajectory formation in which several performance indices are simultaneously operative [Nelson 83]. The precise weights corresponding to each index depend upon the limb's function and task.

The computer animator is interested in designing trajectories which are not only natural, but also expressive. Aside from the functional behavior, such as reaching or throwing a ball, it seems clear that the path taken during an expressive limb motion, due to its foundation in culture, behavior and language, may not be quantitatively determined. However, the distribution of speed along that path could be optimized, subject to the constraints on the initial and final states of the movement, $\zeta(t_0)$ and $\zeta(t_n)$, and a free-form path designed by the animator.

What optimization criteria should be applied in the determination of the best velocity distribution? How do expressive qualities of movement correspond to specific optimal performance indices? While this mapping remains a topic for in-
vestigation, it seems likely, based on the cited empirical research, that the following correspondences will hold:

- coordinated goal-directed motion $\Leftrightarrow$ minimum jerk about hand
- relaxed swinging motion $\Leftrightarrow$ minimum energy expenditure

The establishment of additional correspondences remains a promising domain for future research.

### 2.5.2 The Optimal Velocity Distribution Problem

In the context of computer animation, the limb trajectory generation problem becomes a matter of finding the *optimal velocity distribution* along the path. Formally, it may be stated as follows: Given the specification of a parameterized joint-space limb path $\zeta(t)$, find the distribution of $n$ positions along that path: $\zeta(u_1), \zeta(u_2), \zeta(u_3), \ldots, \zeta(u_n)$ where $\zeta(u_1)$ is the initial position at time $t_1$, $\zeta(u_n)$ is the final position at time $t_n$, and

$$J(u) = \int_{u_0}^{u_n} F(\tau(u), \zeta(u), p(u))du$$

such that $d$ constraint functions $C_i(\tau(u), \zeta(u), p(u))$, $i = 1 \ldots d$ are satisfied.
The solution of the limb trajectory formation problem is non-trivial. Analytical optimal control techniques have been formulated for simple straight-line trajectories [Sahar 85], [Flash 84], [Nelson 83] but fail to work on curved paths. For example, Kyriakopoulos and Saridis attempted to apply optimal control methods (using the calculus of variations) to the minimization of jerk along a prespecified path. Unfortunately, even in the absence of constraints and dynamics, the method yielded a non-linear two-point boundary problem with no closed-form solution [Kyriakos 85].

Further complications arise because the constraints on robot manipulator movement (as well as natural limb motion) are complex combinations of geometrical, kinematical, and dynamical functions such as the maximum achievable torques of the robot arm, constraints on jerk and joint velocities, and limits on non-colliding joint positions [Lin 83].

Thus, the problem falls into the field of nonlinear constrained optimization [Kirk 70], [Bryson 75]. Dynamic programming has been advocated by many robotics researchers [Sahar 85], [Pfeiffer 87], [Shin 86], [Kircanski 82], [Singh 87], due to its generality and well known capability for solving nonlinearly constrained systems. The strategy taken by dynamic programming is to discretize the state space of the limb's movement, and then find the minimum cost path from the start state to the final state. As the sampling resolution of the discretization increases, the path computed by dynamic programming will converge to the continuous-time
solution [Kirk 70].

Of course, if the dimension of the state space is large (more than three variables), the memory required will also become prohibitive. Therefore, the technique is only feasible for path synthesis if the number of joints is small. Sahar and Hollerbach found the minimum-time paths for a two-degree of freedom manipulator by dynamic programming over a tesselation of the limb's two-dimensional joint-space [Sahar 85]. However, if the limb's movement is constrained to a path $\zeta(u)$, as is the case in our optimal velocity distribution problem, an otherwise n degree of freedom system is reduced to one dimension in $u$, thereby alleviating dynamic programming's "curse of dimensionality."

The problem of synthesizing constrained optimal velocity distributions for robotic manipulators has been addressed by Kircanski and Vukobratovic [Kircanski 82]. The approach I will advance is similar to their technique, but more robust and suited to the requirements of computer animation. In light of this similarity, I will present a description of their approach first.

2.5.3 Solving for Optimal Trajectories with Dynamic Programming

The optimal velocity distribution algorithm developed by Kircanski and Vukobratovic begins with an initial distribution $\zeta(u_i)$, with one position for each time (or frame) $i = 1 \ldots n$. A reasonable first guess would be to distribute the positions at equal intervals (or constant speed). Since the path we are interested in optimizing
is a free-form spline, the methods I described on arclength reparametrization may be used for this purpose.

The next step is to form \( m \) different variations about the initial limb position \( \zeta(u_i) \) for each time \( i \): \( \zeta(u_1^i), \zeta(u_2^i) \ldots \zeta(u_m^i) \). These variations form a set of \( m \) positions, out of which one will be chosen for inclusion in the final optimal distribution (see fig. 2.8).

![Figure 2.8: \( m \) variations about a position at time \( i \)](image)

Depending on the performance index to be optimized, the state of the limb must include not only the position, but higher order time derivatives as well. By computing finite differences between all possible successive positions, we may approximate the velocity at a given time, thereby forming the state \( s_i^j \) at \( t_i = (p, v_i)_j \) where \( v_i = p_{i+1} - p_i \) and \( j = m^2 \). This process of accumulating higher
order terms in the state may be recursively invoked, resulting in $m^{c+1}$ possible states at each time step, where $c$ the order of the derivative, with $m^c$ transitions possible between a states $s_i$ and $s_{i+1}$ (see fig. 2.9).

![Dynamic Programming Graph](image)

Figure 2.9: Part of the Dynamic Programming Graph with $m=2$ and $c=1$

The set of all possible distributions formable by the product-space of each of the variations is exponentially large, consisting of $m^m$ possible trajectories. By using dynamic programming, however, the optimum distribution of this exponential space can be computed in linear time.

Working backward from the final state to the initial state, the dynamic programming solution recursively computes the cumulative cost from state $s_{i-1}^j$ to $s_{i}^k$:

$$COST(s_{i-1}^j) = \min_k J(s_{i-1}^j) + COST(s_{i}^k)$$  \hspace{1cm} (2.7)
where $k = 1 \ldots m^c$. When the algorithm reaches the start node, the minimum cost path in the graph will be found. This path corresponds to the optimal velocity distribution (see fig. 2.10). At each cost evaluation, each of the constraint functions $C_i(\tau(u), \zeta(u), p(u)), i = 1 \ldots d$ may be easily incorporated into the dynamic programming optimization by excluding choices which are in violation. Thus, the presence of constraints which reduce the search space may actually improve the efficiency of the program.\footnote{Search-space reductions must be traded off against the time required to evaluate these constraint functions.}

In spite of the generality of the Kircanski/Vukobratovic algorithm, it becomes impractical for cases in which the speed of the limb changes dramatically. If we assume that our initial distribution is at constant speed, and that the variations are
placed at equal distances on the path $\Delta \|\zeta(u)\|$, then the smallest acceleration possible will be $\Delta \|\zeta(u)\|$. Furthermore, the upper bound for the sum of accelerations (in one direction) taken over the path will be $m(\Delta \|\zeta(u)\|)$ (see fig. 2.11).

![Figure 2.11: Resolution and Range of Acceleration Assuming Initial Constant Speed Distribution](image)

2.5.4 Optimizing Splined Trajectories with Dynamic Programming

The basis for the author's optimization algorithm is the use of spline interpolation in the formation of the velocity distribution. Let $S(u)$ be a spline interpolated trajectory in a limbs joint-space, with $S(i) = v_i$. Rather than constructing variations in placement at every time sample, variations are formed in the placement of the spline's control points $v_i$ at equal time intervals $t_i, i = 1 \ldots n_S$. Then the
trajectory is formed by spline interpolation of optimal control point distribution. The principal advantage here is that both small increments in speed (resolution) and large magnitudes in acceleration (range) may be realized by interpolation of the relatively small number of discretized control point variations.

The algorithm begins with an initial control point distribution \( (v_1, v_2, \ldots, v_{n_s}) \). For each \( v_i \), \( m \) variations \( v_i^1, v_i^2, \ldots, v_i^m \) are formed such that

\[
v_i^j = S(u_i + (j - m/2)(\Delta u))
\]

where \( \Delta u \) = the parametric range in variation \(^3\).

Although the variations are taken from the initial path, changing the location of a control point may slightly alter this path. However, in cases where path preservation is critical, the insertion of new control points (without altering the path) [Mortenson 85] may be used to reduce the path deviation caused by the variation.

Given that the local support for the spline blending functions is over four control points, path segment \( PS_i \) between \( v_i \) and \( v_{i+1} \) will be influenced by variations in \( v_{i-1}, v_i, v_{i+1} \) and \( v_{i+2} \). Therefore, for each path segment having \( m \) variations in control point placement, there are \( m^4 \) path segment variations.

Now, the optimal distribution of control points may be found by minimizing the cost evaluated over the sequence of path segments formed by the distribution.

\(^3\)Using the arc-length reparametrization discussed in the previous section, variations may be selected at equal distance intervals along the spline.
In this context, a node in the dynamic programming graph becomes the \( i \)th path segment associated with one of the \( m^4 \) variations of control points \( v_{i-1}, v_i, v_{i+1} \) and \( v_{i+2} \). Since the first three control points of the \((i + 1)\)th path segment are the same as the last three control points of the \(i\)th path segment, there are only \( m \) possible transitions which must be considered at each path segment node. (see fig. 2.12).

\[
\text{Figure 2.12: Splined Path Segment Variations}
\]

The algorithm for the finding the optimal control point distribution may be computed with the dynamic programming step defined as:

\[
\text{Cost}(PS_i^j) = J(PS_i^j) + \min_k \text{Cost}(PS_{i+1}^k)
\]  

(2.9)

where

\( i = 1 \) to number of path segments,
\[ k = 1 \ldots m \]

\[ j = 1 \text{ to } m^4, \quad m = \text{number of variations}. \]

As with the Kircanski/Vukobratovic approach, constraint equations \( C_i \), are evaluated at each node \( PS_i^j \). If a constraint is violated for a particular path option, that branch is simply culled from consideration as the minimum cost path. In this fashion, only the feasible solution regions of the graph are searched for an optimal path.

Note that the weighted performance index \( J \) is computed over a spline-interpolated path segment. Therefore, higher order path derivatives, such as joint speed and acceleration, may be derived analytically without the costly growth in states associated with forward difference approximations used in the Kircanski/Vukobratovic algorithm.

For cases in which the smoothness of the hand or foot is desired (minimization of jerk), the velocity distribution is formed in terms of the Cartesian-space path swept out by the end of the limb. However, since joint-space functions and constraints must also be considered, inverse-kinematics are employed in the optimization process. Aside from kinematic constraints on joint ranges and speeds, optimizations which involve dynamics, such as the minimization of energy expenditure, are computed in joint-space. The recursive Newton-Euler inverse-dynamics procedure receives the joint-space position, velocity and acceleration from the inverse-kinematics computation at every frame during the evaluation of \( J(PS_i^j) \). The flow
of control between the various computational modules in the calculation of $J(PS_i^j)$ is shown in fig. 2.13.

Figure 2.13: Flow Chart of Path Segment Performance Computation

The space/time complexity of the dynamic programming approach outlined aboved is $O(m^5nk \delta)$ where $m$ is the number of variations, $k$ is the number of joints, $n$ is the number of frames and $d$ is the number of constraints. Although the exponent on the variational term $m$ is quite high — with $m^4$ path segments having $m$ possible transitions — the solution is linear in the number of frames, number of variables and the number of constraints. Due to the range and resolution achieved by interpolating increments in velocity, $m = 5$ is typically sufficient.
An means of increasing the efficiency and the precision of the algorithm is to apply what I shall call gradual refinement. This procedure iterately repeats the dynamic programming over increasingly smaller ranges in variation $\Delta u$, each time starting with an initial distribution equal to the optimal distribution of the previous iteration. With gradual refinement, we may use small values for $m$ and still achieve a fine resolution.

Even without the use of gradual refinement, the computational complexity of dynamic programming in velocity distribution problems compares very favorably with other nonlinear numerical optimization techniques, such as gradient-based mathematical programming strategies [Gill 81] dubbed "spacetime constraints" by Witkin and Kass in the animation literature [Witkin 88]. These methods, which attempt to descend the slope of a single multivariant function of $J$ in order to find local minima, must compute and solve equations involving Jacobian matrices with $knd$ elements and Hessian matrices with $(kn)^2$ elements, resulting in a computational complexity of $O((kn)^2 + knd)$ at every iteration. Furthermore, gradient-based techniques may become hopelessly bogged down or permanently trapped in local minima [Ghosh 76], [Bryson 75], [Kirk 70].
As we move from the problems of modelling and designing movement of limbs toward the broader issue of animating the entire body of a legged animal, the complexity of the task is increased by several orders of magnitude. There are several reasons why this is so. The first, and perhaps most obvious, source of complexity is that the “whole” of body motion is much more than “the sum of the parts.” The limbs, hands, feet, head, and spine of a legged animal all work in concert to achieve a dynamically balanced coordinated motion in the presence of gravity and external reaction forces acting on the feet. In reality, the animal is “solving” the mechanics of an extremely large-scale nonlinear system.

3.1 Newtonian mechanics and Sensory-Motor Coordination

In the last decade, significant research in the robotics field has been directed toward studying and building multi-legged walking and running machines. Due to the difficulty of maintaining balance on two legs, most research has concen-
trated on the problems of walking with four or six legs. [McGhee 79] [Orin 82] [Waldron 81] [Pearson 84] [Lee 84]. To date, walking machines with two legs must move haltingly slow in order avoid falling over [Muir 84]. Raibert's running machines, although much simpler in form than any real animal, cannot easily turn corners or change gaits [Raibert 84] [Murphy 84]. It is safe to say that the control problems associated with tasks as seemingly elementary as locomotion are not well understood.

The problem of animating legged animals is further complicated by the fact that we are interested in simulating the movement of "real" animals, not just legged systems of our own design. Aside from the substantial increase in complexity found in real biological systems, the problem is shifted from the already difficult task of understanding the mechanics of movement to the more empirical question of understanding how animal X solves the mechanics of movement. Hence, the simulation of legged animal motion must encompass both Newtonian mechanical laws and the legged animal's sensory-motor coordination which operates within these laws.

The approach I have taken, which I will present in depth in this chapter, is to achieve the appearance of intelligent sensory-motor coordination by exploiting the fact that the animation program may have knowledge of future states, thereby allowing it to plan or "anticipate" required movements which must be carried out in advance.
In the first section I will describe a model for locomotion which, being a specialized periodic behavior, can be parametrized at a fairly high level. The second section will tackle the more difficult problem of animating non-periodic free-form movement, such as is found in expressive dance.

### 3.2 Periodic Behavior: Locomotion

The notion of planning movement to satisfy goals and constraints in terms of known future states is used extensively by PODA in the execution of coordinating body and limb motion [Girard, Maci 85].

#### 3.2.1 Leg Support and Stepping Movements

A central problem of animating legged animals is that the feet of the animal must remain planted on the ground during the support phase. At the same time, the animator should be able to design the character of the stepping motion while the foot is in the air.

A step is specified in PODA by the desired trajectory of the feet and the center angles and gains on the joints (which may change dynamically). The curve which defines the foot trajectory is defined by a Catmull-Rom (interpolating) spline. Because the desired shape of the curve depends on the geometry of the leg, the control points of the spline are set by moving the foot of the leg. The animator may conceptualize the design of the step as the specification of "key leg positions," in the spirit of a key-framing system. In PODA, a key position records the position
of the foot (as a control point in the spline) and the center angles and gains that are associated with that position. The animator manipulates the foot into each position using PODA's inverse kinematic procedures. Then, once the foot is in place, the joint angles may be adjusted using the center angle and gain parameters.

This approach is distinguished from key-framing or joint angle interpolation systems in that the goal of achieving the desired foot position in Cartesian space is primary – the foot will travel precisely along the smooth Catmull-Rom spline from foothold to foothold. By contrast, if we interpolate the leg positions in joint space, there is no general means of either moving the foot along a curve or placing the foot at a particular place on the ground.

The problem of keeping the legs on the ground as the body translates and rotates is simplified due to PODA's inverse kinematic capability. The problem reduces to solving for the position of the foot in the leg's moving coordinate system so that it is identical to the placement of the foot in the previous frame's world coordinate system (thereby keeping the foot stationary in the world). We solve for this position using:

$$H_{prevFrameFoot_t} = H_{prevFrameFoot_{t-1}}$$

If the animator is to have supervisory control over the legged figure, a means for directing the body's full translational and rotational degree's of freedom must be available. Given that all the legs are on the ground, the problem may be solved
using (equation 3.1). The fundamental problem which remains is to calculate footholds and plan the foot transfer trajectories between them so as to adapt to the desired body motion.

An important concept of foothold planning is the notion of a reference leg position [McGhee 79]. This is the desired position of the leg in mid-stance or half way though the leg's support duration. The posture of an animal when all of its legs are in their reference positions may be regarded as the "standing" position of the animal (see fig. 3.1).

The other key ingredient for the foothold calculation is the ability to predict the body's future positions. In PODA, the body's trajectory may be computed as a function of the desired body trajectory over the ground plane and dynamic constraints due to the timing and force limitations of legs (to be discussed) before the precise footholds are chosen. Since the body's position is known in advance, it is possible to plan ahead in order to step toward the next stable position.

At the beginning of the leg transfer phase of leg \( i \), say at frame \( t \), we may compute the reference leg position in world space at frame \( f_1 = t + \text{transferDuration} + 0.5(\text{supportDuration}) \) as follows:

\[
\text{World}_{T_{\text{Hip}_{f_1}}} = \text{World}_{T_{\text{Body}_{f_1}}} \left( \text{Body}_{T_{\text{Hip}}} \right) \tag{3.2}
\]

\[
\text{World}_{\text{refLegPos}_{f_1}} = \text{World}_{T_{\text{Hip}_{f_1}}} \left( \text{Hip}_{\text{refLegPos}} \right) \tag{3.3}
\]

This foothold will insure that leg \( i \) comes to its "mid-stance" position half way
Figure 3.1: The Reference Leg Position
through its support phase. We must still determine the position of the foot in the body’s coordinate system at the time the foot is placed down. This knowledge is required in order to facilitate moving the foot horizontally with respect to the body during the transfer. This may be accomplished by:

\[ Body_{ref} LegPos_{f2} = Body_{World} \left( World_{ref} LegPos_{f2} \right) \]

where \( f_2 = t + transferDuration \) (3.4)

The generic transfer trajectory designed by the animator may then be adapted to move between the current foot position and the calculated foothold so that the height in the world and proportional distance moved next to the body are preserved.

### 3.2.2 Leg Motion Adaptations to Body Motion

Variations in leg motion occur between gaits, between animals, and between legs. For example, consider human locomotion. During walking the legs swing and touch down on the heel; running is often distinguished by a “kick” of the legs which brings the runner’s foot high above the knee as his foot leaves the ground. The limb trajectory tables, described above, provide the animator with a means of defining a host of trajectories which may be used to distinguish the character of many different types of leg stepping motions. PODA provides an interface which allows the animator to associate a trajectory from a leg’s trajectory table with a
given gait.

Since limb trajectories are designed in the coordinate space of the leg’s hip, they must be adapted during body movement to accommodate variations in footholds and body speed. Another problem is that we must prevent discontinuities in foot velocity during the foot’s transitions between being on and off the ground. Jumps in speed are quickly seen to appear unnaturally jerky. Real physical legs must “follow through” as they lift from the ground due to their backward momentum during the support phase. PODA is able to simulate the appearance of a gradual change in momentum by measuring the leg’s joint-velocities as the foot leaves the ground. A joint-space trajectory may be described by maintaining the measured lift-off joint speed. We sinusoidally interpolate from this constant joint-velocity trajectory to our leg trajectory selected from the table.

When a foot is placed on the ground, real animals form leg trajectories which minimize dramatic changes in horizontal velocity with respect to the ground. This reduces the muscular stress of absorbing sudden accelerations which might occur at the point of contact. PODA alters the selected leg trajectory toward the end of its transfer phase by sinusoidally interpolating its horizontal foot position toward its world-space location in the preceding frame. This, in effect, moves the foot toward the direction and speed of the ground moving under the animal’s body. At the point of contact, the foot will accelerate smoothly to match the velocity of the ground.
3.2.3 Periodic Gait Design

The model of locomotion implemented in PODA utilizes a number of parameters which are convenient for describing the gait of a figure—the terms and relations are derived from robotics research on walking machines [McGhee 79] [Orin 82].

A \textit{gait pattern} describes the sequence of lifting and placing of the feet. The pattern repeats itself as the figure moves: each repetition of the sequence is called the \textit{gait cycle}.

The time (or number of frames) taken to complete a single gait cycle is the \textit{period $P$} of the cycle.

The \textit{relative phase $R_i$} of leg $i$ describes the fraction of the gait cycle period which transpires before leg $i$ is lifted. The relative phases of the legs may be used to classify the well known gaits of quadrupedal animals (see fig. 3.2).

During each gait cycle period any given leg will spend a percentage of that time on the ground; this fraction is called the \textit{duty factor of leg $i$}. For example, the duty factor may be used to distinguish between the walking and running gaits of bipeds. Walking requires that the duty factor of the each of the legs exceed 0.5 since, by definition, the feet must be on the ground simultaneously for a percentage of the gait cycle period. Lower duty factors (less than 0.5) result in ballistic motion identified with running, wherein the entire body leaves the ground for some duration.

We will call the time a leg spends on the ground its \textit{support duration}. The time
spent in the air is the leg’s transfer duration.

The stroke is defined as the distance traveled by the body during a leg’s support duration. If we acknowledge that the foot must traverse the stroke during the transfer phase in order to “keep up” with the body, the stroke may alternatively be regarded as the length of the step taken by the leg over the ground. The body may move over the ground plane in PODA, so the stroke in this context becomes the diameter of a circle in that plane (see fig. 3.3).

The following relationship holds between the legs and the body:

\[
\text{supportDuration} = \frac{\text{stroke}}{\text{bodySpeed}}
\]

The above equation may be used to solve for the time (or number of frames) that
each leg must spend on the ground. By definition we also have

\[ \text{dutyFactor} = \frac{\text{supportDuration}}{P} \]

The amount of time which a leg spends in the air depends on both the leg speed and the arclength of the transfer phase trajectory. That is:

\[ \text{transferDuration} = \frac{\text{arcLength(transferTrajectory)}}{\text{legSpeed}} \]

During the gait cycle period \( P \), a single leg will move through one cycle of support and transfer, hence we have:

\[ P = \text{supportDuration} + \text{transferDuration} \]

for any leg \( k \). In fact, one may imagine the period as a duration subdivided into support and transfer durations (see fig. 3.4).
Figure 3.4: The Gait Cycle Period

The leg state at time $t$ may be determined as

$$legState = (legState_0 + t) \mod P$$

where

$$legState_0 = (R_i) \cdot (P)$$

If the leg state is less than the support duration then the leg is in its support phase, otherwise the leg is in its transfer phase. Moreover, the time of foot placement occurs when the leg state equals zero and the foot liftoff occurs when the leg state is equal to the support duration.

An animator using PODA may design gaits for figures having any number of legs by instantiating the parameters given above. The model makes sure that all
the variables are updated according to functional dependencies, thereby freeing
the animator to experiment with the variables of interest (such as relative phase)
without worrying about the integrity of the other related variables.

We see that each gait may be uniquely defined by a body speed, leg duty
factor, stroke, and the relative phase relationships between the legs. The animator
interactively sets the values of these parameters for each gait in a sequence of gaits
defined using a menu-driven interface.

3.2.4 Gait Shifting

Legged animals frequently change their gaits during locomotion. These changes are
most often a function of body speed. The well-known sequence of gaits executed
by accelerating horses — amble, trot, canter, gallop — is common to many four
legged animals. But rather than limit ourselves to a fixed set of gait shifts, I sought
to develop a model which will animate transitions between arbitrary sequences of
gaits at arbitrary speeds.

To do so, it was necessary to implement a gait shifting algorithm which could
adjust the coordination of legs between any two gaits. The problem is to shift the
phase of the legs, either by reducing or increasing the time the legs spend during
their transfer phase. Applying phase shifts during the support phase was ruled out
due to its possibly unnatural or potentially impossible requirements on the length
of the stride (recalling that the stroke equals the product of the support duration
and the body speed).

Since there is no unique solution for a set of phase shifts between two gaits, an optimization criteria had to be applied. The author chose to minimize the square of the magnitudes of the phase shifts under the constraint that the resulting adjusted leg transfer durations should not be shorter than a given constant. That is, we wish to minimize

$$\text{relativePhaseDist} = \sum_{i=1}^{n} \text{phaseShift}_i^2$$

such that

$$\text{transferDuration}_i + \text{phaseShift}_i \geq k$$

where

1 $\leq i \leq n$

$n = \text{number of legs}$

$k = \text{minimum leg transfer time}$

Minimization of phase-distance has the effect of solving for the smallest degree of shifting which is evenly distributed among the legs. The minimum leg transfer constraint prevents solutions which will require unnaturally quick leg motions.

First, let us consider the problem for cases in which the gait cycle period is the same for both gaits. For our computer-animated gaits, gait cycle periods are adjusted to fall on integer boundaries so that liftoff and touchdown times occur exactly at specific frames. This reduces the number of possible phase shifts to $2 \times \text{numberOfLegs} \times \text{period}$ if we consider all possible differences between the legStates of two gaits. Given the small number of possible candidates, PODA performs a simple exhaustive search.
When the current and next gaits have different periods, a two stage transition is employed. In the first stage, a phase shift is performed to equate the gait-cycle periods in terms of the longest period.

If the current period is shorter,

1. The current gait is shifted to a longer period by shifting all legs by the difference in gait cycle periods.

2. The algorithm for equal gait cycle periods is applied.

If the current period is longer, steps 1 and 2 are reversed.

### 3.2.5 Body Dynamics and Support Relations

A critical factor in the determination of an animal’s body motion is a description of its history of contact with the ground. PODA computes a record of the support and non-support intervals of the animal as a first step in simulating its body motion. The sequence of these intervals forms the **support profile** of the animal.

Currently, animals in PODA must have an even number of legs. Quadrupeds (four legged animals) are supplied with two support profiles — one for each pair of front and hind legs. PODA may then simulate the pitching motions of quadrupeds by isolating the vertical motion of the shoulder and pelvis (see fig. 3.5). PODA records only one support profile for bipeds (two legged animals) since they have only one pelvis. Animals with more than four legs are typically rigid (insects), so one support profile is used for them as well.
Figure 3.5: Pitching motion of quadruped
The building of the support profile is the first stage in the computations for synthesizing animal locomotion. PODA incrementally derives the animal's body trajectory by a four pass process of accumulating constraints on its three-dimensional degrees-of-freedom.

1. The support profile is constructed as a function of the gaits and gait transitions.

2. The vertical and horizontal motion of the body may be computed as a function of the support profiles.

3. The angular motion necessary for keeping the body pointed toward its horizontal direction of movement and the angular banking required to accommodate the curvature of the horizontal path may be calculated once its translational positions at each frame are known.

4. The kinematic motion of the limbs and the calculation of footholds may be computed as a function of the known body positions at each frame (see fig. 3.6).

The record of future positions derived from each pass allows PODA to mimic the body trajectory planning that animals exhibit in the performance of coordinated body movement. In the following, I will discuss the approach taken for each pass of this process.
Flow of control of PODA's locomotion model. The animator's interactively designed gait sequence, body path, and leg trajectories are gradually synthesized into a coherent animation of the animal's body and limbs.

Figure 3.6: Flow of Control of Locomotion Model
Vertical Body Motion

During the non-support intervals of the support profile, acceleration on the animal is restricted to the effect of gravity. Given the downward gravitational acceleration the body must leave the ground at an upward velocity required to keep it in the air for the duration of each non-support interval. PODA solves Newton’s equations of motion for the upward velocity and the resulting ballistic motion of the body during its non-support periods. The final downward vertical speed at the end of each non-support interval is also easily calculated.

The problem of synthesizing the trajectory of the body during its support intervals remains. We must solve for the trajectory under the constraints that:

1. The initial and final vertical speeds must be equal to the values computed for the non-support intervals,

2. The vertical speed must be continuous,

3. The liftoff height must be less than or equal to the length of the leg.

The second constraint prevents the appearance of unnaturally stiff movement in which strong accelerations would have to occur in between frames (such as a bouncing pool ball). The third constraint restrains pelvic movement to remain inside the kinematic boundaries of the leg.

The solution employed by PODA is to describe the velocity curve in terms of
a family of functions of the form \( At^e \) where \( 0 < t \leq (1/2)(supportInterval) \) and \( e \geq 1 \) (see fig. 3.7) In this context, each of the constraints implies, respectively:

1. \( A_1 t_1^e = v_1 \)
   \( A_2 t_2^e = v_2 \)
   where
   \( t_1 = t_2 = (1/2)(supportInterval) \)
   \( v_1 = \) mag. of initial downward speed
   \( v_2 = \) mag. of final upward speed

2. Given \( e_i \geq 1 \), we may join the speed curves piecewise-continuously at speed
\[= 0, \text{ with the first function reversed. (See fig. 3.7)}\]

3. \[\int_0^{\tau_i} A_i t e_i dt \leq D\] where \(D\) is the distance between maximum and minimum vertical heights.

If we use equality in the implication above, and solve for \(e_i\) and \(A_i\),

\[
e_i = \left[\frac{v_i \tau_i}{D}\right] - 1
\]
\[
A_i = \left(\frac{v_i}{\tau_i}\right)
\]

If \(e_i < 1\), then we have a condition where the constraints may be satisfied by setting \(e_i = 1\), since the distance integral (equation 3) will be less than \(D\) under this new assignment.

Another problem is that, especially in a walking gait, the knee joints remain rigid as the legs pass through the midstance position, causing the body to rise along an arc to a maximum (rather than a minimum) halfway through its support phase. The solution PODA adopts is to sinusoidally interpolate between this arc-like trajectory and trajectory resolved using the piecewise speed curve as a function of the ratio of non-support over support intervals during each gait cycle (see fig. 3.8). In this way, walking (having a zero ratio) has a maximum height at midstance and jumping has a minimum height at midstance with running somewhere in between.

Although the above interpolation scheme works reasonably well, the development of a more physically motivated model is a good topic for future research. At the present time, the dynamic control laws which are involved in the establishment
of these vertical body trajectories during the locomotion of real animals are not well understood.

**Horizontal Body Motion**

The desired horizontal path taken by the animal is interactively designed by the animator with a cubic spline (Catmull-Rom or B-spline). Given the desired body speed along different parts of the curve, PODA may calculate the desired positions and velocities along it using the numerical arc-length calculation discussed in section 2.4.3.

The body's ability to turn and speed up is modeled in terms of the number of feet on the ground during the support intervals and the magnitude of the maximum
achievable acceleration each supporting leg may apply to the body as a whole. We call this maximum achievable acceleration the leg’s *impulse*. This parameter, which may be interactively set by the animator, indirectly determines the sense of weight of the animal in relation to the strength of its legs. The maximum achievable acceleration of the body at any given instant is the sum of the impulses of its supporting legs.

PODA determines the desired acceleration at a given frame through velocity error feedback, that is, by subtracting the desired velocity from the current velocity at that frame. The horizontal acceleration of the body is then computed as the minimum of its maximum achievable acceleration and the desired acceleration at each frame. Note that if, at any instant, there are no supporting legs, the maximum achievable acceleration will be zero, leaving a minimum of zero. Therefore, as we expect, the body will move at constant horizontal speed while in the air.

Given the acceleration, Euler integration is applied at each frame to solve for the new horizontal velocity and position.

**Angular Motion: Turning**

The modeling of angular motion during locomotion in PODA is currently restricted to rotations about the yaw and roll axis — that is, turning and banking respectively. The horizontal motion pass outputs the precise positions of the animal, so the desired yaw angle may by found by calculating the direction of movement from
changes in horizontal position. Turning is achieved by solving Newton's equations of motion to bring the body to the measured yaw angle at the beginning of each body support interval. If we assume a constant acceleration during each support period, we have:

$$
\ddot{\theta}_i = \frac{\theta_{i+1} - \theta_i - \dot{\theta}_i(t_s + t_{ns})}{(1/2)t_s^2 + (t_s)(t_{ns})}
$$

$$
\dot{\theta}_i = \ddot{\theta}_{i-1} t_s
$$

where

- $t_s$ = support interval
- $t_{ns}$ = following non-support interval
- $i + 1$ = start of next support interval
- $i$ = start of current support interval
- $i - 1$ = start of previous support interval

Angular Motion: Banking with Dynamic Stability

An animal is dynamically stable when the projections of the gravitational and inertial forces sum to zero about the foot's point of contact with the ground [Vukobra 83]. To maintain dynamic stability, a running animal must adjust its foothold calculation and banking angle to counteract the centrifugal force as it turns.

Let us consider the situation of an animal moving at speed $s$ along a circle of radius $r$. The centrifugal force, by Newton's equations, will be a vector perpendicular to the circle having magnitude $ms^2/r$. If we assume that a planted foot will not slide due to friction, we wish to solve for the banking angle in which there is no net angular torque. This condition is reached when the gravitational
force downward, $mg$, and the centrifugal force outward sum to a force vector which points directly down the banking angle (see fig. 3.9).

Figure 3.9: The Dynamically Stable Banking Angle

Therefore, the banking angle is:

$$\phi = \arctan \left( \frac{s^2}{Rg} \right)$$

and the change in foothold will then be:

$$sideStep = (hipHeight) \tan \phi$$

We generalize this solution to arbitrary curved paths by noting the curvature of the path $K = 1/R$. In order to measure the curvature, PODA fits a spline through the path of the animal's body in the horizontal plane. At the frames in which
legs come to their midstance position, the curvature of the path is measured and the dynamically stable banking angle is computed according to the given equation with $K = 1/R$. 

3.3 Non-periodic Behavior: Dance

In contrast to locomotion, expressive choreography, as well as most other coordinated activities, typically requires the capability for animating the body in a non-periodic fashion. I decided to focus on the problem of dance due to the challenge of creating motion which must be composed with a concern for the details or "micro-structure" of movement. The computer-animation of dance forces the issue of coordination and timing beyond building computational models for specific motor skills — we must identify primitives for the composition of arbitrarily complex forms of movement [Girard, Maci 85] [Girard, 87].

The model for locomotion relies on assumptions we can make about the patterns of movement: the feet move in the periodic rhythm of gaits, the arms (if the animal has arms) swing in opposition to the legs, the feet lift from the toe and land on the heel, the bending of the spine is synchronized to the motion of the pelvis. But expressive movement is much more open-ended in its possibilities. During the support phases, the body height may vary dramatically — during the non-support phases, the entire body may assume several postures before returning to the ground (see fig. 3.10).

3.3.1 Limb and Body Trajectories

In order to define such free-form body movement, PODA supports the composition of body trajectories. A body trajectory, in effect, is nothing more than a sequence
Figure 3.10: Sequence of eight body postures. Postures numbered 4 and 5 are in a non-supported state.
of body postures — it is the union of the spine, head, and various limb trajectories. The pelvis is also given a trajectory. When the feet are planted on the ground, the pelvis may be envisioned as an end-effector moving from its legs. Expressive movement may frequently involve both pelvic translations and rotations.

From the animator’s standpoint, body trajectories are composed, remembered and recalled in the same fashion as limb trajectories: menu-driven interaction is provided for inserting, replacing, deleting, storing and retrieving postures and posture sequences.

3.3.2 Vertical Dynamics of Body Movement

In the PODA system, each posture includes a specification of which legs are supporting the animal from the ground. As the animator “flips through” each posture in a sequence, the number of the posture and the support/non-support state of each leg are displayed in the viewing window (s means “support,” and m denotes “moving, or non-supported”) (see fig. 3.10). Unsupported postures (those having no supporting legs) will be performed while the animal is in the air.

In dance, the height and overall form of the body at the instant of becoming airborne and at the instant of becoming earthbound is not fixed; e.g., one might hop from a crouched position with the knees bent or hop with the legs nearly fully extended. Therefore, the vertical heights of the supporting postures which immediately precede and follow the unsupported postures are used by PODA as a guide
for simulating the vertical motion of the pelvis. We will call these *transition body postures*. For non-periodic motion, the support profile is constructed by recording the non-support periods which occur between these transition body postures.

The same constraints for vertical motion during locomotion hold here, but in this new context, the height of the body at liftoff and touchdown must match the heights at the transition postures which have been specified by the animator. The calculation of the upward velocity at liftoff (and subsequent downward velocity as the body lands) may take into account differences in body height using:

\[
V_{up} = g \left[ (1/2)(t_{ns}) - \frac{h}{g t_{ns}} \right]
\]

where
- \( g \) = gravitational acceleration
- \( h \) = difference in height
- \( t_{ns} \) = duration of non-support interval

All that remains is to find the vertical pelvis trajectories during the support phases. If the support phase is defined by only two body postures, both must be transition postures and PODA will use the given piecewise speed function to find the support-phase pelvis trajectory. If there are more than two, the animator must have intended to modulate the height of the pelvis using the pelvis trajectory defined by the three or more body postures which take place during the support phase.

For this case, PODA sinusoidally interpolates between a trajectory of the form derived from the piecewise speed function and the kinematically specified pelvis trajectory. The distance and interval variables of the piecewise speed function are set to the differences in time and distance between transition postures and
their neighboring support postures (the posture after the touchdown transition posture and the posture before the liftoff transition posture). In this way, the differences in height and time of these neighboring support postures may be used to vary the perceived muscular response of the animal for both counteracting its downward acceleration at the instant of impact and thrusting upward at the instant of becoming airborne.

3.3.3 Horizontal Dynamics of Body Movement

As in the treatment of vertical dynamics, the horizontal motion of the body must take into account whether the animal is in the air or on the ground. Accordingly, the support profile becomes an input parameter to the horizontal dynamics model.

While the legged animal is on the ground, PODA computes the animal's center of mass and measures its velocity as the backward difference between positions at successive frames: \( \overrightarrow{V}_{com} = \overrightarrow{P}_{com} - \overrightarrow{P}_{com-1} \). Newton’s laws of motion tells us that when the animal is in the air, because there can be no horizontal forces brought to bear on the body (assuming the absence of wind resistance), the center of mass of the animal should move at a constant velocity \( \overrightarrow{V}_{com} \) projected on the horizontal ground plane. The magnitude and direction of \( \overrightarrow{V}_{com} \) will be therefore be determined at the initial point of liftoff as the legged animal makes a transition from being on the ground toward being in the air. At each frame \( t \) in the air, the animal is
translated so that:

\[ \vec{P}_{\text{com}} = \vec{P}_{\text{com, lift}} + (t - \text{lift off})\vec{V}_{\text{com, lift}} \]  

(3.5)

The laws which govern an animal's horizontal translational motion when its feet are on the ground may not be so easily characterized. For example, given that the projection of an animal’s center of mass remains within the region bounded by its points of contact with the ground \(^1\), the animal’s center of mass may move in practically arbitrary free-form patterns, restricted only by its muscular capabilities.

The physical feasibility of movements when the projection of the center of mass moves outside of the support region is harder to ascertain, especially if we allow for frictionally induced foot forces which may pull or push the animal in any direction. If we assume these forces are negligible, then leg forces may only accelerate the body's center of mass along a vector directed from the point of support to the center of mass: \( \vec{A}_{\text{com}} = \vec{P}_{\text{support}} - \vec{P}_{\text{com}} \). In cases where the center of mass is outside the support region, the animal should be really be falling in directions in which the velocity has no components directed in opposition to \( \vec{A}_{\text{com}} \).

Due to the difficulty in simulating the complex control problems during the support phase, the compromise “solution” currently adopted by the author is to simply provide free-form spline-interpolation of the body while it is on the ground. The animator must then be responsible for assembling body trajectories that are physically credible. Future research needs to be directed toward developing models

---

\(^1\)The robotics literature refers to this region as the legged animal's support polygon
for coordinated falling motion which may occur when weight is shifted from one foot to another.

3.3.4 Angular Dynamics: Posture Design vs. Dynamic Stability

Ideally, a computer animation system should help us to easily design ballistic motions wherein:

1. The legged animal performs a set of movements in the air, and then

2. Lands in a dynamically stable posture which anticipates the next movement.

During the flight phase, the movement of limbs may induce angular accelerations about all three axes [Smith, Kane 68] [Ghosh 76]. Simulating the angular dynamics from a given set of limb motions is easily computed: the angular motion of a limbed body follows from the application of the conservation of angular momentum about the principal axes of rotation [Kane 70], [Wilhelms 85], [Armstrong 85], [Raibert 88]. However, in the interest of insuring dynamically stable landing positions, we must solve the inverse problem — what body motions will produce some given desired body orientations? For example, consider the problem of finding the limb movements a diver performs in order to achieve some desired posture and orientation at the time of contact with the water.

Formally, the problem could be stated as follows:

Given some initial body angular liftoff position and velocity $S_0$ at time
so level for the set \( \{ M_{s_0}, s_n, t \} \) of feasible body movements during time \( T \) so that the body is in state \( S_n \) at touchdown time \( t_n = t_0 + T \).

If we restrict ourselves to an analysis of specialized movements which are symmetric about all but a single rotation axis, such as Kane's analysis of scissor leg movements to achieve rotations in zero-gravity [Kane 70], Spaepen's study of a backward flip [Spaepen 86], or Ghosh's investigation of a kip-up maneuver on the horizontal bar [Ghosh 76], it is often possible to solve for how specialized movements should be varied in order to achieve desired postural goals.

Marc Raibert, in his paper on "Biped Gymnastics" [Raibert 88], cites Richard Fosbury's introduction of a previously unknown method for executing a high jump (with one's back facing down) as an illustration of the difficulty in modelling complex dynamic skills. The motion problem of finding strategies to maximize a human's jumping height is an isolated case of a more general constrained optimization problem. Given a limbed robot's initial position \( A \) and goal position \( B \), solve for the set of movements which optimize function \( D \) under constraints \( C \). Here \( A \) is the liftoff position, \( B \) is a landing position, \( C \) is the maximum achievable forces generable by a human's musculature, and \( D \) is the jumping height.

One may easily imagine other dynamic coordination problems of this type (aside from the obvious gymnastic and athletic ones). For example, consider an astronaut moving in a zero-gravity environment. In this case, we might wish solve for the
limb movements which turn the body, in minimum time, between two orientations [Smith, Kane 68]. Dancers provide another example of intelligent motion, as they are able to satisfy the formation of several postures in the air within the constraints of the timing and required angular body dynamics dictated by the choreography.

Very little work has been done to address motion tasks with this degree of dynamic complexity. Marc Raibert and his coworkers at MIT’s A.I. lab recently programmed a running biped robot to execute a forward flip. The strategy employed was to maximize flight duration and adjust the pitch rate by inducing a high angular velocity at liftoff. The flip was achieved by applying a sequence of known movements and adjusting their parameters until success ensued.

Noteworthy is that the sequence, or task-level strategy for performing the flip, was formulated by the human researchers, rather than being derived computationally. Understanding the reasoning process which guided the human decisions, based on their knowledge of the mechanics of the problem, is an area which needs more attention. Remarks Raibert, “The need for techniques that bridge the gap between the task level of the motor act and the actuator control level is a deep and important problem in robotics and artificial intelligence” [Raibert 88].

In any case, it is clear that specialized parametric movements performed by the dancer, diver, high jumper and gymnast may not be derived by constrained optimization algorithms because the dimension of the problem space is both astronomically large and highly nonlinear. Therefore, the PODA implementation of
angular dynamics artificially decouples the rotation axes. The current model conserves angular momentum $I\dot{\theta}$ about the yaw axis, where $I$ is the moment of inertia and $\dot{\theta}$ is the angular velocity. Like the horizontal velocity, this can be measured at the point of liftoff. The flowchart for the body dynamics in PODA's non-periodic dance mode is shown in figure 3.11].

---

Figure 3.11: Flow of Control of Dance Model
CHAPTER IV

PODA Animation System Design and Implementation

Thus far, I have focused on the conceptual issues of modelling legged animal behavior. This chapter will discuss the implementation of PODA, beginning with an overview of the system. The second section will describe the user-interface, and the last section will outline the software engineering problems associated with programming for the evolution of legged-animal animation systems.

4.1 Overview of PODA System Structure

An overview of the PODA legged animal animation system, in the form of a module interconnection graph, is shown in figure 4.1. As can be seen in the figure, a clear division exists between the "dance" and locomotion models. This is due to the differences in parameters, where locomotion is governed by periodic gaits, and dance is specified with body trajectories. Modelling transitions between locomotion and dance, or indeed any specialized parametric behaviors such as throwing a
Figure 4.1: PODA'S Module Interconnection Graph
ball and sitting down, is an important subject for future research 1. Although the animator must assign each animal to either "dance mode" or "locomotion mode", the PODA system supports the animation of animals of both types within the same 3D space.

The task of the limb motion module is to provide the animator with a means of designing limb trajectories. After the animator interactively positions a sequence of postures and assigns weights in optimization and constraint functions, the optimal velocity distribution procedure described in Chapter II is used to alter the postures in a trajectory initially specified by the animator, in accordance with the optimality criteria.

The body motion module is cleanly divided according to whether the animal is dancing or is in the locomotion mode. In the dance mode, limb motion will influence the translational and angular momentum of the body, and consequently change its ballistic motion. However, since the locomotion dynamics are based strictly on the animal's support profile (here we assume the legs are massless), the body motion may be calculated independently from the limbs.

Animation is calculated by sending each animal's polygonal structure to the display program for each frame. The perspective transformation is applied according to the position and center of interest of the camera. Finally, animation may be seen by playing back the stored frames at 30 frames/second (applying bitblt

---

1 The parametric modelling of specialized "motor skills" is a promising but difficult task which must also be tackled in the future.
operations at video rates).

4.2 User-Interface and Interactive Control

In Chapter II, I outlined PODA's model for limb motion and its parameters of interactive control. In Chapter III, the interactive specification of locomotion and "dance" were presented. In this section, I will describe how the animator actually positions the legged animal and sets parameter values by a combination of menu selections and interactive keyboard control. No programming is required.

4.2.1 Menu-Driven Control

The fundamental hierarchical structure of the interactive menus is shown in figure 4.2. The locomotion menu facilitates the specification of a series of gaits, a path in the horizontal plane that the animal will follow, and a table of leg trajectories (stored and retrieved by names) for stepping motions. Each gait is specified with a pop-up menu that records its gait parameters and an associated leg trajectory to be used as a stepping motion.

The dance mode interaction is more complex. Body trajectories are assembled by selecting different limbs and, through keyboard interaction, moving the end-effector and joints into place. Keys are mapped to movement in x, y, z, yaw, roll, pitch and each of the joints. Other keys are used to move the camera and center of interest, so that one may easily see the form of the body's posture from different points of view during any manipulation. Postures may be inserted, appended,
Figure 4.2: Menu Hierarchy
deleted and replaced from a trajectory by clicking on the appropriate selection in the menu.

Once a body trajectory is designed, it may be stored in a table by name. Its name will later appear in a pop-up “trajectory menu” for later retrieval, shown in the figure. The trajectories are meant to correspond to primitive movements which may be sequenced into a larger dance or choreography. The “choreography menu” is used to program sequences of trajectories. The animator types a sequence of trajectory names, along with a beginning and ending posture selected from the trajectory (thus providing for assembling subparts of trajectories).

The choreography system supports a few “global” transformations of a trajectory subpart. Body movements may be performed in reverse order by making the starting posture larger than the first. Also, movements may be reflected between the left and right side of the body by placing an “R” at the end of the trajectory one wishes to reflect.

Animation may be calculated for individual selected limbs, individual selected animals, or sets of activated animals chosen from the main menu. This allows the animator to focus on the design of simple elements, gradually constructing complexity in the animation in a “bottom up” fashion.

The animation may be rendered with or without hidden-lines, depending on the animator’s desires. The hidden-line display, based on a z-buffer algorithm, takes significantly longer to calculate, but is necessary for visual clarity as depth
complexity grows when several animals are animated simultaneously.

An option also exists for rendering "bounding boxes" which are parallelepipsds that enclose the animal. This is useful when we are interested in quickly seeing the relative positions of animals as they move through the scene. In addition, camera motion splines may also be checked efficiently with the aid of the bounding box "stand ins" animated by this feature.

4.2.2 Real-time Playback vs. Real-time

The frame rate of the animation may be adjusted interactively, but 30 frames/second (video rate) and 24 frames/second (film rate) are the most useful choices. Calculating frames and playing them back is only necessary when the computer's floating-point performance and rendering speed is not capable of real-time calculation (24 frames/sec). The Symbolics Lisp 3650 Machine, the current platform for the PODA system, performs at only 0.25 MFlops and renders a running quadruped at about 2 frames/second. Recent workstation technology aimed at scientific visualization applications (e.g. Apollo, SGI, Ardent and Stellar) promises speeds nearly 40 times faster.

Realtime calculation of legged animal animation is currently possible with some additional setup time required for the costly constrained optimization calculation described in section 2.5. However, since this calculation is trivially decomposable for parallel processing by assigning different processors to different dynamic pro-
gramming graph segments, implementation on massively parallel multiprocessors is a promising area for future research.

4.3 Software Engineering of Animation Systems

4.3.1 Utility of Object-Oriented Programming

The software engineering of legged animal animation systems is complicated by the fact that these systems are never complete. The behavior of legged animals is simply too complex to be captured by a single computer program (especially the more intelligent animals such as humans). Therefore, one needs a programming methodology which supports the continual evolution of extremely large programs. The key issue is that we augment our software with new behavioral features, constantly building on what came before.

This notion of augmentation of software is central to the inheritance capability of object-oriented programming languages and environments. When class A inherits from class B, A essentially augments the existing procedures and data structures present in B. The PODA system exploits the object-oriented Symbolics Common Lisp "Flavors" system in order to build hierarchies of specialization in the software engineering of both limb modules and entire legged animal modules (see Figure 4.3).

The manipulator flavor contains the data structures and procedures required to simulate the kinematics and dynamics of limbed movements. The spine, leg and
Figure 4.3: Inheritance in Legged Animal Animation Systems
arm flavors each augment these fundamental operations with their own particular extensions. For example, the leg, in contrast to other limbs, must keep track of footholds and include procedures for executing stepping motions. The spine must include operations for bending and twisting in a natural fashion. However, the basic movement primitives in the manipulator flavor are common to all limbs.

A similar hierarchy of specialization may be constructed at the body coordination level as well, since the procedures for periodic locomotion and foothold calculation are practiced by all legged animals. The inheritance capability of object-oriented languages allows us to easily build specific control strategies for any given animal on top of these essential legged animal behaviors.

4.3.2 Integration with Other Behavioral Models

Ideally, an animator should be able to design motion with computational models of many different motion phenomena. I imagine an environment where the animator could create instances from a broad class of moving things (e.g. ocean waves, fire, clouds, birds, etc.), in addition to legged animals. Each model would inherit standard procedures for user-interaction in terms of a set of its own idiosyncratic editable functions (see Figure 4.4). A "browser" module could be written to view and maintain the animator's interaction with each model instance's control parameters. Once the initial conditions for each model instance are set, an "animation supervisor" could be invoked to cycle through each model instance at each frame
Figure 4.4: Integration of Computational Models for Animation
calculation, updating their states, sending messages between them, and finally, rendering their visible structure.

The above scheme was not fully implemented in the design of POMA. However, because of its object-oriented form, POMA could be easily modified to have this structure.
CHAPTER V

Conclusion

In this dissertation, I have discussed the problems associated with simulating, designing, and controlling the computer animation of legged animals. It should be clear to the reader that the simulation and control of natural legged animal motion is difficult. The intractability of the nonlinear constrained optimization problems which arise in the modeling of both limb and body motion is a major obstacle to be crossed.

5.1 Contributions

In chapter II of the dissertation, I have contributed a method for interactively specifying trajectories based on arc-length reparametrization and a viable solution to the optimization of limb trajectories which are constrained to move along a designated splined path. My solution utilizes the kinematic and dynamics algorithms which I have described as background material, applied directly from robotics techniques developed by others.
In Chapter III, I have confronted the problem of designing and animating locomotion and dance by developing schemes which mimic the kinematic and dynamic aspects of the behavior of the animal within a computationally feasible framework. The models of gait and the foothold planning algorithms have been applied from legged locomotion research in robotics. However, the gait shifting algorithm and the modelling required for incorporating the dynamics of body motion into the animation are mine. The dance system organization is based entirely on my original work.

Finally, in Chapter IV, I outlined an object-oriented approach I used in the software engineering of legged-animal animation systems and suggested how one could integrate legged-animal models with systems specialized for the motion of other phenomena.

5.2 Future Research

In the interest of more precise realism, control schemes for the angular dynamics of body movement which are more accurately simulated must be worked out. To this end, specialized models in the spirit of the Kane's scissor kicking in zero-gravity [Kane 70] or Raibert's forward flip [Raibert 88] need to be researched and incorporated into PODA's model so that a repertoire of skilled behaviors may be invoked and controlled by the animator. The question of whether specialized behaviors could be logically inferred or derived from a general purpose knowledge-
base needs to be addressed.

Another broad research domain, not even touched on in this dissertation, is to incorporate motion planning skills into legged animals, so that they may carry out tasks which involve non-trivial geometric relationships with their environment. To do this, the strategies animals use to grasp objects and avoid undesirable collisions with obstacles must be well understood.
Bibliography


[Raibert 88] Raibert, Marc H. "Biped Gymnastics", submitted for publication


