A Physically Based Human Figure Model with a Complex Foot and Low Level Behavior Control

by

Michael Allen McKenna

B.S., Massachusetts Institute of Technology
(1987)
S.M., Massachusetts Institute of Technology
(1990)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in Partial Fulfillment of the Requirements for the
Degree of

Doctor of Philosophy
at the
Massachusetts Institute of Technology
June 1994

Signature of Author [illegible]
Program in Media Arts and Sciences
April 29, 1994

Certified by [illegible]
David Zeltzer
Principal Research Scientist, Research Laboratory for Electronics
Thesis Supervisor

Accepted by [illegible]
Stephen A. Benton
Chairperson
Departmental Committee on Graduate Students
Program in Media Arts and Sciences

A Physically Based Human Figure Model with a Complex Foot and Low Level Behavior Control

by

Michael Allen McKenna

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning, on April 29,1994
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

Abstract

Advances in computer hardware and software technology allow the simulation of natural phenomena in increasing levels of complexity. This thesis is concerned with simulating the articulated movements of humans using the laws of physical motion, and contributes to the fields of computer animation and biomechanics. A 90 degree of freedom model of a human figure is developed, and an efficient dynamic simulator is employed to create and analyze physically based, computer generated motions. The foot of the simulated human figure has been modeled with a significant amount of kinematic complexity, with 28 degrees of freedom per foot. A joint-level control layer uses springs and dampers to control postures and movements. A framework for higher level control is implemented, although specific tasks require tailored control strategies. Several example tasks are simulated and described, including maintaining stable standing postures, rising on the toes, and reaching with the arm to designated targets. A simulation of the stepping phase of walking is developed, using passive dynamic effects to generate much of the motion. Together, the simulator and biomechanical model create a framework which can be used to address problems in computer animation and biomechanical research, and eventually, in a clinical setting, to assist doctors in analyzing the problems of specific patients.

Thesis Supervisor: David Zeltzer, Ph.D.
Title: Principal Research Scientist

This work was supported in part by NHK (Japan Broadcasting Company), and equipment gifts from Hewlett Packard Co. and Apple Computer, Inc.

A Physically Based Human Figure Model with a Complex Foot and Low Level Behavior Control

by

Michael Allen McKenna

The following people served on the committee for this Thesis:

Committee Member [illegible]

Marc H. Raibert
Professor of Electrical Engineering and Computer Science
MIT Artificial Intelligence Laboratory

Committee Member [illegible]

Joseph Rosen
Associate Professor in Plastic and Reconstructive Surgery
Dartmouth Medical School

Dedicated with love to my mother, Lynn S. Evans.

Table of Contents

List of Figures

List of Tables

List of Scripts

1 Introduction

1.1 General Problem

The synthesis of "natural" and "realistic" motions of human figures has long been a goal of animation. The design and control of humanoid robots to perform typical human behaviors has also been a goal of our society — the idea of a technologically-created artificial person has been with us for centuries. [Heppenheimer] At the root of these problems is the study of human motions and how they are produced, which has a long history of scholarly pursuit. In recent years, increases in computer performance have allowed the creation of representations of the real world, with which one can interact in real-time. These *virtual environments* (VEs) allow us to unite the pursuits of animation, robotics, and human motor performance, by using computational models of physical phenomena and theories of motion control.

In certain VE systems the representation of natural humanoid movements, generated by computer, is of prime importance. For example, consider systems designed for investigating human factors and performance, for training among multiple persons, and for assisting in clinical (medical) environments. In addition, it is not only the *motions* or *kinematics* of the humans which are of importance, but also the *kinetics* or *dynamics* which should be accounted for. Indeed, simulating the "underlying" dynamics of the system may lead to a more natural representation of the motions.

The simulation of complex, physically based behaviors is still beyond real-time computation and interaction, except with limited examples. Although real-time VEs cannot currently support complex simulations as they are computed, VE frameworks are very well suited to building a dynamic simulation environment. The general purpose, flexible, 3D environment and tools provided by VEs facilitate the development and visualization of dynamics.

1.2 Specific Problem

This thesis is concerned with designing and implementing a computer software system and biomechanical model for simulating an articulated humanoid figure, using dynamic, physically based simulation techniques, in a computer animation framework.

The human figure model developed here is fairly complex in its kinematic structure — 90 degrees of freedom are modeled, with 28 degrees of freedom in each foot. In addition, force generating elements, including dampers, joint limits, and actuators, are present at each joint. The dynamic simulator system, combined with the biomechanical model of the human figure, allows us to generate and analyze complex motions.

Increasing the complexity of the kinematics also increases the problems of control, simulation, and analysis, and can obscure mechanical relationships. However, increased complexity also allows for the examination of functions which do not exist without suitable underlying structural representations. With a complex model subtle effects can be revealed and studied. For the purposes of animation, the problem is obvious and straightforward — in order to show a complex or subtle movement, the foundation for such an action must be present. Although animation has a tradition of "faking it" by presenting images and movements that seem more complex than they really are, without a realistic underlying architecture, animation systems break down as their artificially imposed limits are exceeded. In addition, clinical problems demand that details of structure be examined because they can lead to significant gross effects.

1.3 Approach

A biomechanical model of the articulated human body is developed, based on human kinematic and dynamic parameters. A number of sources of information are employed to design the model: literature regarding cadaver and clinical studies, which measure the properties of limbs and joints; two dimensional anatomical diagrams; and a three dimensional digitized skeleton.

An efficient dynamic simulation computer program, named *corpus*, is developed to create and analyze motions of the biomechanical human figure model. *Corpus* is a general purpose simulation system for articulated, rigid bodies, with a flexible software architecture that allows for the simulation of a variety of jointed figures and mechanisms. Forward and inverse dynamics can be simulated, as well as a hybrid mix of the two, based on the *Articulated Body Method*, developed by Featherstone. [Featherstone 1987]

The biomechanical model incorporates joint actuators, based on non-linear springs, which form a low level control layer and supply the forces needed for postural support and for motion generation. Motions are controlled by manipulating the spring parameters (primarily the spring rest angles) over time. Passive joint forces are supplied by dampers and elastic joint limits.

In order to simulate the figure, forward dynamics is used to compute the motions of the parametrized human figure in response to the internal and external forces. Motions and postures of the human figure can be "calibrated" using inverse dynamics to compute the required joint forces, and inverse control to compute the required actuator parameters. For example, standing, balanced postures and their responses to perturbation forces can be simulated after using such a calibration process.

Through a variety of simulations, the feasibility and utility of the system is demonstrated. Examples include maintaining a balanced, standing posture, reaching with the arm, and rising on the toes. The stepping phase of walking is also simulated, using passive dynamic effects to generate the limb motions and the shape changes of the articulated feet.

1.4 Contributions

This thesis research demonstrates the feasibility of using dynamic simulation to compute the motions of a complex human figure model for animation and other applications. The difficulty in computation arises not only because there are many degrees of freedom in the model, but also because there are very small links in the figure's feet, placed under large stress forces. This creates a very "stiff" numerical system, which must be sampled with very small time steps in order to accurately compute its motion. The rate of simulation ranges from near real-time to one half hour per frame (where a frame represents 1/30 second of real time), depending on the number of degrees of freedom included in the model, and the type of motion being simulated.

This research contributes a new, complex biomechanical model of the human, with 90 degrees of freedom. The foot of the simulated human figure has been modeled with a sig-

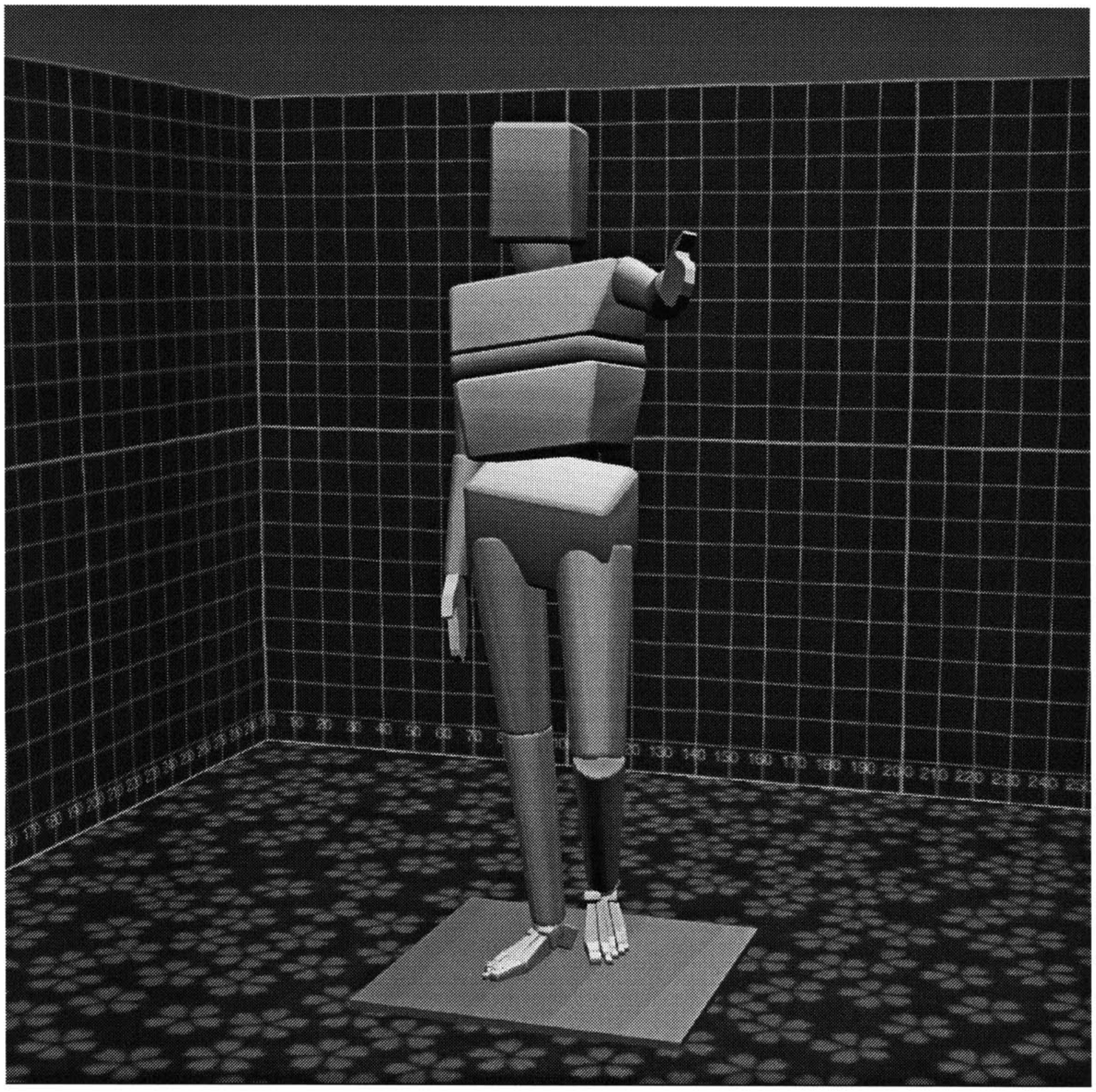


Figure 1: The human figure model.

nificant amount of kinematic complexity, with 28 degrees of freedom per foot, representing the most complex kinematic model known to the author. The human figure model is shown in Figure 1.

A method is presented for controlling postures and movements, based on spring actuators and dampers. The actuators can be adjusted manually by the user, in order to examine "what would happen" under different conditions, or they can be automatically calibrated to achieve specified postures. Motion is controlled by using motor programs, which vary

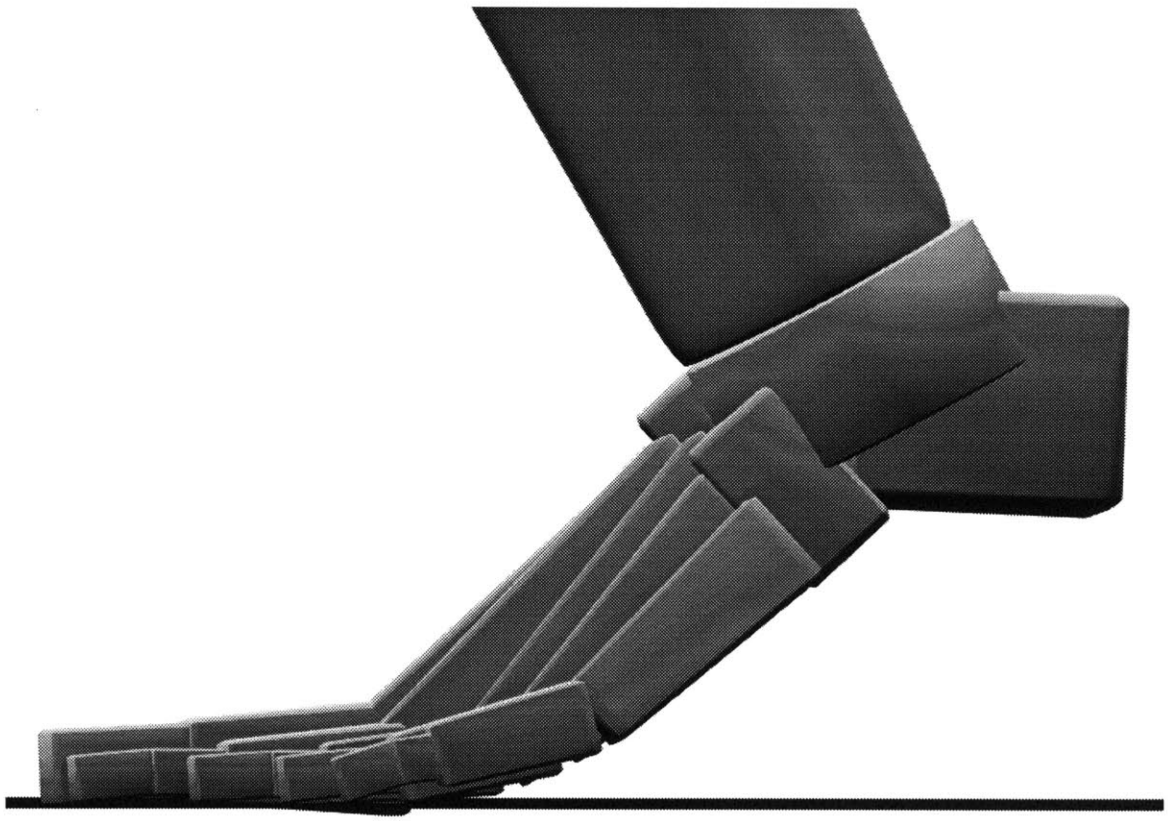


Figure 2: The shape of the simulated foot model during walking, before toe-off of the stance leg.

the actuator parameters over time. The control system architecture borrows from robotic control techniques, tailored to use biomechanical elements.

An efficient, general purpose dynamic simulator has been implemented. A recursive formulation is used, which has a computation time linear with the number of joints, allowing complexity to be explored. Together, the dynamic simulator and biomechanical model, with its low level actuator control, form a powerful system for the generation and analysis of human motions. Through a number of example simulations, the utility of the system is verified. Figure 2 shows a still frame from one such simulation.

2 Background

This chapter will survey background material and related work pertaining to the field of human motion simulation, and will cover the general topics of:

Motion Simulation
Motion Control
Human Biomechanical Parameters
Analyses of Human Movement and Gait
Machine Locomotion
Computer Animated Simulations of Human Movement
Previous Work by the Author: Hexapod Locomotion

The reader is directed to the **Glossary** near the close of this document for clarification of unfamiliar terminology.

2.1 Motion Simulation

This thesis is concerned with the motion of *articulated figures*, comprised of rigid bodies or links, connected by joints. A joint allows the two bodies which it connects to move relative to each other in some manner, such as by translating or rotating. A joint which allows motion in a single direction only, e.g. in the positive and negative X direction, provides a single *degree of freedom* (DOF). Joints can allow for motions in multiple directions, providing multiple DOFs. A single body, or an articulated figure, free to move in space has 6 intrinsic DOFs — three translating DOFs and three rotary DOFs.

Within an articulated figure, *proximal* bodies are ones which lie closer to the center of the figure, compared to *distal* bodies, which lie further from the center. For example, the fingers are more distal than the forearm, and the upper and lower ends of the forearm are labelled proximal and distal, respectively. An *end effector* is a body which lies at the end ("peripheral terminus") of a kinematic chain.

A rigid body has a center of mass (COM), which locates the "average" point of matter in the body, towards which any external body is attracted by the force of gravity. A linear force applied at any location other than the COM of a body creates a rotational acceleration in the body (in the absence of a counteracting force or torque). Just as a single body has a COM, an articulated figure also has an overall COM, which plays an important role in the dynamics of the figure. Winter points out that the terms "center of mass" and "center of gravity" (COG) are often used interchangeably, but that COM is the proper term to use when referring to three dimensions. [Winter 1990] The center of gravity refers to the center of mass in the vertical, gravity defined direction only.

The motion of articulated figures can be described using *kinematics* or *dynamics*. A kinematic description of motion uses only geometry to analyze or control a figure's movements. A dynamic description also models the forces which lead to the figure's motions. A dynamic model essentially includes a kinematic model, as well.

There are two major types of kinematic computations-*forward* (or *direct*) and *inverse* (or *reverse*). Forward kinematics describes motion in terms of joint angles and positions. (Note: the term *position* will be used in a general sense, to describe either rotational orientations or linear displacements. Similarly, the term *force* will be used to generally describe both rotary torques and linear forces. Later in this document, we will discuss *spatial notation*, which unifies the rotational and translational aspects of motion.) Using forward kinematics, joint positions are specified, and the position of an end effector is then determined by the positions of the joints which precede it. Inverse kinematics defines motion in terms of Cartesian coordinates and other kinematic goals. Thus, the position and/or orientation of an end-effector is specified in space, and the joint positions of the articulated chain are then computed.

There are also two major types of dynamic simulation: forward (or direct) and inverse (or reverse). Forward dynamics refers to computing the motion of a figure from applied forces. Inverse dynamics refers to computing the forces which must be applied to a figure, to achieve a specified motion. Consider Newton's second law of motion:

$$f = m\,a \qquad \text{Eq. 1}$$

which reads that *force* equals *mass* times *acceleration*. Forward dynamics solves for acceleration from the specified mass and force, as in:

$$a = \frac{f}{m} \qquad \text{Eq. 2}$$

Inverse dynamics solves for the force, from the known mass and acceleration, as in Eq. 1. *Hybrid* methods for dynamics allow a mixture of forward and inverse techniques, so that, within an articulated figure, the force can be specified for some joints, and the motion specified for others, and the system then solves for the unknown values. *Physically based simulation* or *physically based modeling* is the use of dynamics, or the laws of physics, to generate and analyze a motion process.

Forward dynamics is more analogous to the real world, in which forces result in the accelerations of bodies. Inverse dynamics allows us to analyze how motions in the real world are generated, by computing the forces needed to create the observed accelerations.
To further explain forward dynamics, some assumptions and constraints used for forward simulation are paraphrased from [Winter 1990]:

- There should be no kinematic constraints whatsoever — bodies are free to fall, joints are free to move under the influence of forces, etc.
- The initial conditions of the simulation include the positions and velocities of all bodies.
- The only inputs are externally applied forces and internally generated forces and moments.
- The model must incorporate all important degrees of freedom, including joint limits, which are modeled as passive internal forces and moments.
- The external reaction forces which occur between the bodies and the ground must be calculated.

The forward dynamic simulations performed in this research obey these assumptions. Other simulations use inverse dynamics, and some use a hybrid mix, to kinematically control some DOFs, while forward simulating the remainder.

For a good introduction to dynamics, Brady, *et al.*, develops the kinematic and dynamic equations for a two-link robotic arm. [Brady] In **Appendix C Dynamics Verification** in this document, the two-link model is briefly presented, and is developed to work with *corpus*, the dynamic simulation program employed for this thesis.

The algorithms which incorporate the equations of motion into a dynamic simulator can be formulated in a number of ways. One of the most direct ways to develop a dynamics algorithm is to use a matrix solution approach. The equations of motion for each body in an articulated figure or dynamic system are established, taking into consideration the constraint forces need to maintain joint relationships between bodies (and other possible constraints). A large array is then created from the undetermined variables (such as accelerations, for forward dynamics). A matrix inversion is required as part of the solution, which generates a fairly expensive computational cost: $O(n^3)$ where n is the number of joints or other constraints. Such an approach is well developed in an advanced dynamics system by Isaacs and Cohen, which allows for mixed forward and inverse dynamics, and forward and inverse kinematics, but at a high computational expense. [Isaacs 1987; Isaacs 1988]

In many cases, it is possible to take advantage of the constraint relationships, in particular the joint constraints, to simplify the computation. By carefully examining the equations of motion, one can discover recursive relationships which relate the motions between the parent and child links. The solution for the bodies' motions do not have to be computed simultaneously, but rather, it is computed a piece a time, using local relationships (although each body will correctly influence every other body). Recursive solutions typically have a order of complexity, $O(n)$, that is linear with the number of joints, n, in the articulated figure. The efficiency provided by the recursive formulations is critical when complex systems with many degrees of freedom are simulated.

Armstrong developed a recursive algorithm for the forward dynamics of articulated figures, originally developed for simulation of the US space shuttle's robotic arm, the Shuttle Remote Manipulator System. [Armstrong 1979] The algorithm's computational expense is linear with the number of joints: $O(n)$. All joints in the figures are required to be 3 DOF, although an algorithm is outlined to remove this restriction, but with a computational cost penalty. Armstrong and Green employed the system to investigate the use of dynamics for animation, [Armstrong 1985] including the simulation of a human figure (discussed later). [Armstrong 1987]

Featherstone describes an efficient technique, which he terms the *Articulated Body Method* (ABM), for the forward and inverse dynamics simulation of branching, articulated figures. [Featherstone 1983; Featherstone 1987] The method is linear with the number of joints, *O(n)*, for forward and inverse computations. In a comparison of different algorithm complexities, Featherstone claims that his method is the most efficient when the number of joints exceeds nine. Featherstone introduces "spatial notation," which is used to combine the translational and rotational aspects of motion into unified 6 dimensional quantities. This simplifies the complexity of the equations of motion, and can allow for more intuitive manipulation of motion and force terms. Featherstone's ABM is the simulation method used in this research, and it forms the mathematical foundation for the dynamic simulator.

Another efficient simulation method for forward dynamics is presented by Lathrop, using the spatial notation introduced by Featherstone. [Lathrop] Lathrop's algorithm is also linear with the number of joints, but it allows for motion constraints at the end-effectors, and for the inclusion of kinematic loops in the articulate figure's structure, although at a moderate computational expense (no worse than $O(n) + O(l^2)$, where n is the number of joints, and l is the number of internal loops). Featherstone's algorithms can be extended to handle loops, but at a significant reduction in efficiency. [Featherstone 1987] Lathrop's algorithm was used to build a dynamic simulation system, in a computer graphics framework by Schröder. [Schröder]

An efficient recursive solution to the inverse dynamics problem, using a Lagrangian formulation, is presented by Hollerbach. The algorithm is linear with the number of joints, $O(n)$, whereas previous Lagrangian dynamics were of $O(n^4)$. [Hollerbach] Through an analysis of the number of required multiplication and addition steps, Hollerbach shows that his method is comparable in complexity, although still somewhat more expensive, than recursive newton-euler algorithms for inverse dynamics.

We will conclude this sub-section with a discussion of some of the limits in the rigid-body, articulated figure simulation systems that we have reviewed thus far. Most articulated figure dynamic simulators, including the system employed in this thesis, treat the links and joints as idealized, non-flexible entities. In real mechanisms, joints cannot be created perfectly, and usually the link can move small amounts in directions which would not be permitted by an ideal joint. In addition, the link and joint structures can flex by some amount, determined by their material properties, which may or may not have an important influence over their overall behavior. In a similar manner, vertebrates have flexible skeletons, which can play a significant role in the energetics of movements such as galloping. [Alexander 1985] Biological joint structures are quite complex, being formed by the bone surface geometry, connective cartilage and ligaments, and the surrounding tissues. These joints generally allow for very complex motions, in many degrees of freedom, even if the overall motion is almost entirely limited to one degree of freedom. Dynamic simulators can be designed to model the flexible properties of joints and linkages, but such systems are complex and computationally expensive. [Pfeiffer; Yang] Nonetheless, taken in the larger view, these effects ultimately should be present in a complete human model, so that effects such as energy storage through bone bending, and critical stresses in bones can be examined.

We have not reviewed the dynamics of highly flexible and plastic objects, as we concern ourselves in this phase of the research with rigid objects only, to simplify the simulation problem. Computer animated simulation of deformable objects has been demonstrated with significant success by a number of researchers. These simulation systems employ the physical laws of motion for continuous, deformable bodies, which have various mechanical properties. The resulting dynamic differential equations are numerically solved to generate and analyze motion. [Terzopoulos] With respect to this research, simulations of human tissues are of particular interest. Pieper developed a model of human skin tissue, with an interactive graphical interface to plan and simulate plastic and reconstructive surgery operations. [Pieper 1992] Chen has demonstrated a deformable model of human skeletal muscle, with active and passive internal forces. [Chen] These shape models are all highly related to the biomechanical and functional models of human organ systems. Future work would see the unification of these and other functional models and simulation techniques, which would mutually interact to create a highly complex human body model. The problem is a difficult one, but the rewards will be similarly considerable. These issues will be discussed further in the Future Work sub-section in **Conclusions** (8).

2.2 Motion Control

The control of motion is a complex problem. In the real world, and in dynamic simulations, motions and postures must be controlled through the application of forces and interactions with the environments. The physics involved in the system can complicate the problem; links have momentum and inertia, there are powerful interaction forces between the limbs, and disturbances can impinge from a variety of sources.

In biological systems, motions are the result of complex interactions between the central nervous system, the skeletal system, and the neuromuscular system, as well as the physics of motion and the environment's properties. [McMahon] The ways in which organisms control motion is highly related to their biomechanical properties, which are discussed in the following sub-section. The roles of muscle properties, reflexes and other peripheral feedback, central motor patters, learning and adaptation mechanisms, and higher order planning and coordination are all critical to real world motor tasks. These systems are not fully understood, especially taken in their entirety. High level learning and behavior selection remains an open question, with active research to develop hypotheses for biological systems, [Minsky] and computer algorithms. [Maes; Zeltzer 1991]

Bizzi and his colleagues describe an *equilibrium position* hypothesis, based on experimental evidence, which forms a theoretical foundation for posture and movement control. [Bizzi 1982; Bizzi 1984] In many ways muscle acts mechanically like a spring. Muscle stiffness is a function of its activation level, a signal from the central nervous system (CNS). The muscle exhibits a length/force relationship like a spring. Given a level of muscle activation in muscle agonist-antagonist pairs, there is a certain limb posture at which the muscle forces balance out to zero. This is an equilibrium point for the limb, at which it will remain at rest. Different limb postures form different equilibrium positions, each with its corresponding set of muscle activation levels. To move from one posture to another the CNS changes the muscle activations to the appropriate levels for the new equilibrium position. Motion to the equilibrium position is then generated by the mechanical properties of the innervated muscle. Their experimental evidence, based on studies of monkey arm movements, indicate that the CNS does not instantly switch the neural signals to specify the target equilibrium position when motion is initiated, but rather that a trajectory of equilibrium positions is specified over time by the CNS. The details of such a trajectory formulation remains an open issue. Features of the equilibrium position hypothesis are employed in this thesis for motion and posture control.

The control of robotic motion is, not surprisingly, quite rooted in its engineering foundations. Brady, *et al.*, present the general principals of robotic motion. [Brady] The kinematics and dynamics of a given robotic system, including its control system, are formulated. Inverse dynamics can be used to compute the required forces for a specified kinematic goal. The computed control can be applied using *open loop control*, in which the control signals are completely pre-computed and are applied over time to generate the specified motion. Any inaccuracies in the computation or modeling of the robot, or any disturbances will cause the robot to perform the motion incorrectly to some degree. Using *feedback control,* the performance of the robot is measured during the motion, and deviations from the intended path are used by the control system, in real-time, to attempt to compensate. The feedback can be mapped in a linear fashion, e.g. proportional to the error, yielding *linear control.* Non-linear feedback mappings can also be employed, yielding *non-linear control.* A complete model of a robot system includes not only the dynamics and kinematics of the device, but also its control and actuation systems. Model based control takes into consideration the entire system, or *plant.* [An]

The planning of trajectories for robot limbs to follow can be approached in several different ways. A *joint space* controller plans a trajectory in terms of the joints angles of the system. A trajectory can also be planned in *Cartesian space*, for example, specifying a path through 3D space for the end effector of a robot to follow. Inverse kinematics computes the joint angles associated with that Cartesian trajectory. *Compliant motion* is used when part of a robot is in continuous contact with another surface. Rather than controlling position in such a situation, it may be more appropriate to control force at the robot manipulator. [Brady] It is also possible to develop hybrid control systems which control combinations of force and position, or force and acceleration. [An]

The control of motion in computer animation systems began with kinematic techniques. Using *keyframe animation*, an animator creates different postures of an articulated figure, which are stored as "keyframes." To create motion, the computer interpolates the pose of the figure from one key frame to another, over time. [Sturman] Different interpolation techniques can be used to make the motions appear more smooth, and to impart certain qualities to the motion. [Kochanek; Steketee] Talented animation artists can make much of this technique, creating realistic motions, and characters that seems "full of life," often incorporating traditional animation techniques into their work. [Lasseter] However, creating quality animation usually requires many keyframes, and it remains a time-consuming process, restricted to those with the talent.

Kinematic control can be provided through either forward or inverse kinematics, or a hybrid mix. Using forward kinematics, the joints angles of the figure must be specified by the animator. Inverse kinematics allows the animator to specify the positions and motion trajectories of the end effectors in Cartesian space, a very difficult task using forward kinematics. [Girard 1987]

Motions for articulated figures can also be input by performance. "Scripting by enactment" is described by Ginsberg and Maxwell, in which a human performer, wearing three dimensional trackers, acts out movements which are digitized by computer. The recorded movements can be "replayed" in an articulated figure, which they term a "graphical marionette." [Ginsberg] This technique is attractive because complex, realistic movements can easily be obtained. It is currently used with mechanical "puppets" to produce character animation. [Walters] Combining such input with real-time graphics output provides powerful feedback for the puppeteers, and allows for real-time performance to audiences.

Animation by enactment can also be used to generate facial animation. Williams presents a system for animating parametrized three dimensional faces driven by video input of human actors' faces, which have tracking spots affixed to the skin. [Williams 1990] Recent work by Pentand, *et al.*, uses video-based input of human actors, which do not require any special tracking markers. The video input is analyzed, and the actors facial movements trigger simulated muscles in the facial model to generate expressions. [Essa] The overall body motion and approximate limb motions of performers can also be also be tracked in real-time using video acquisition and image analysis. [Darrell]

Motions of articulated figures can also be specified using different forms of motion notation, such as methods used to "transcribe" the major features of dance compositions. Singh, *et al.*, describe an interactive graphical system used to edit Benesh Movement notation, a method for describing human body postures and movements on a 2D "score," analogous to a musical score. [Singh] The system was used primarily as a tool to assist with the task of transcribing the score, but did not allow for animation between the different postures. Keyframe techniques can be used to generate motions from the score data, however. Such animation is problematic because there is a great deal of ambiguity as to how exactly the movements are performed. Especially in the case of dance, in which the motions are highly dynamic and graceful, an underlying model of realistic motion generation is critical.

Instead of using animator or actor input (*guiding* control), a *programming* approach can be used to control animated movements. [Zeltzer 1990] Reynolds describes a computer animation system which is based on an animation/graphics programming language. [Reynolds 1982] Using an object-oriented approach, independent control structures, called actors, perform graphical operations on objects over time to generate animation. Programming methods allow for *adaptive* animation, in which motion behavior is modified in response to input, environmental, or internal changes. Animations of bird flocking and fish schooling behavior have been created by several computer graphics researchers. [Reynolds 1987; Amkraut] The group behavior of the animals is determined by the aggregate behavior of the individual animals.

Using a guiding approach, motions are defined explicitly. Using programming control, animation is described procedurally. Using a *task level* approach, behavior is implicitly defined, in terms of events, relationships and goals. [Zeltzer 1990]. Task level systems are created by appropriately combining guiding and programming control at high levels of abstraction. Zeltzer's work with a goal-directed, walking human skeleton is discussed below in the "Computer Animated Simulations of Human Movement" sub-section.

Badler and his colleagues have worked for a number of years on high-level control of human figures, using inverse kinematics and other techniques. [Badler 1985; Badler 1987; Phillips; Lee] By using multiple kinematic constraints, or goals, the posing of human figures can be simplified, while giving more natural appearing results. Complex goals can be specified, including a "stability" constraint which keeps the figure's center of mass within the support region formed by the contact between the figure's feet and the ground. [Phillips] They have worked with "strength-guided motion" as well, using a kinematic model with some dynamic elements to simulate a human executing a lifting task. [Lee] The motion is based in part on an analysis of the forces involved and a "comfort" and "strength" model of the human.

There has been a more recent move in computer animation to incorporate a dynamic basis for movements, using physics as a means of generating the motion. The use of dynamic simulation can be considered a form of motion control, even if there are no controlling forces or constraints. The "passive" motions generated by the physics alone can be very complex, and, in general, cannot be duplicated by kinematic techniques alone. Dynamic motor control can be applied atop of physically based simulation to control the behavior of animated dynamic objects and figures. The previously discussed issues remain important topics for physically based animation: kinematics remains intrinsic to the motion, and is inseparable from the dynamics; guiding and programming control are both applicable; and task level control can be constructed, so that the behavior of a simulated figure is driven by high level goals.

A number of researcher have investigated physically based animation as a means of motion production. Barzel and Barr present a system which kinematic constraints are used with dynamic simulation to construct and simulate articulated structures. [Barzel] The structures "self-assemble" as constraint forces pull the bodies together, over time, to form the joints. Control is provided through use of the kinematic constraints, while the dynamics responds passively.

Isaacs and Cohen developed a powerful dynamic simulation system, named DYNAMO (for DYNAmic MOtion), for animation. [Isaacs 1987; Isaacs 1988] The system allows for a very general-purpose mix of kinematic constraints in a dynamic framework. Motion can be controlled by force functions and kinematic specifications. Inverse dynamics can be applied as well to determine the forces required to accomplish a specified kinematic motion.

Witkin and Kass describe *spacetime constraints* as a means of controlling and producing physically based animations. [Witkin 1988] Spacetime constraints are a form of optimal control, in which the goals of the motion are specified, and the system solves for the time-varying actuator forces which will accomplish the goal, using the minimum energy, or some other measurable parameter to be optimized. The solution of such a system is typically very complex, because the solution is obtained for the entire time sequence of the motion. Other optimization control strategies have been used for animation as well. [Brotman]

Systems presented by Wilhelms [Wilhelms 1985; Wilhelms 1987] and Armstrong and Green [Armstrong 1985; Armstrong 1987] use similar means of motion control for the dynamic simulation of articulated figures. Using forward dynamics, forces are applied at joints to generate movements. A variety of joint forces can be employed, including dampers, springs, joint limit springs, and direct specification of forces. Similar methods are used in this research to apply dynamic motion control, using forward dynamics.

2.3 Human Biomechanical Parameters

This sub-section begins with some basic terminology used to refer to human anatomy. Three orthogonal planes are used to describe the dimensions of the human body: *frontal, sagittal,* and *horizontal* (refer to Figure 3) .

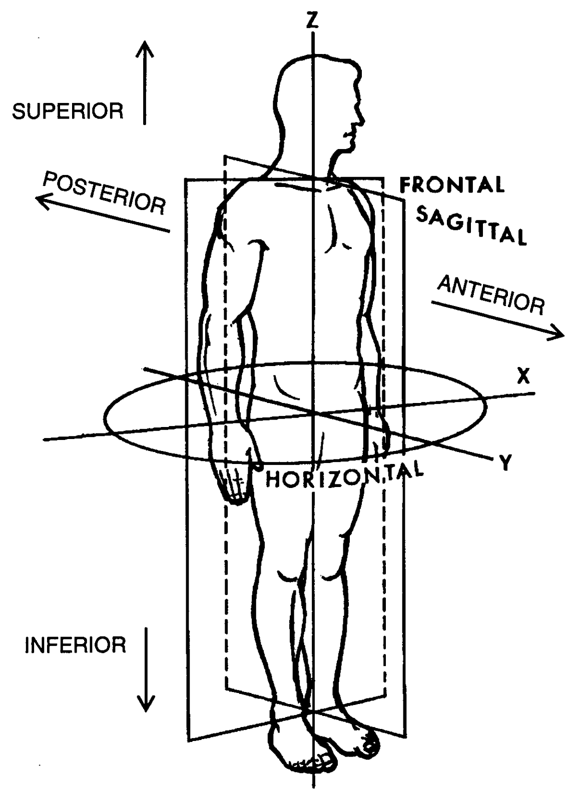


Figure 3: The cardinal planes of the human body.

Adapted slightly from Williams and Lissner. [Williams 1977]

Lateral refers to structures which are situated farther from the midline of the body. *Medial* refers to structures situated towards the midline of the body.

The *horizontal* plane is also referred to as the *transverse* plane. The *frontal* plane is also known as the *Coronal* plane. The *sagittal* plane is also referred to as the *midsagittal* or *medial* plane.

The terminology for different types of joint motions is shown in Figure 4, page 31. A *rotation* at the hip causes the leg to pivot about its long axis. *Flexion* (or *flection*) of the hip brings the leg forward, while flexion of the knee brings the lower leg backwards. Flexion generally diminishes the angle formed by the joint, whereas, *extension* increases the angle and straightens the limb. *Abduction* at the hip moves the leg outwards, to the side of the body, while *adduction* moves the limb towards the center.

In order to simulate the human body, accurate *biomechanical* data is required— i.e. kinematic and dynamic parameters which capture the pertinent information needed to describe a humanoid figure. The kinematic parameters include the geometry of the limbs, and the joint DOFs. The limb geometry primarily describes the length of the limb from joint to joint, often termed the "link" length. The kinematics of the exterior geometric shapes of the limbs can also be an important biomechanical parameter, when the shapes are used for geometric collision detection between different objects and between objects and the ground. The geometric object data can also be used to define the volume of the limb seg-

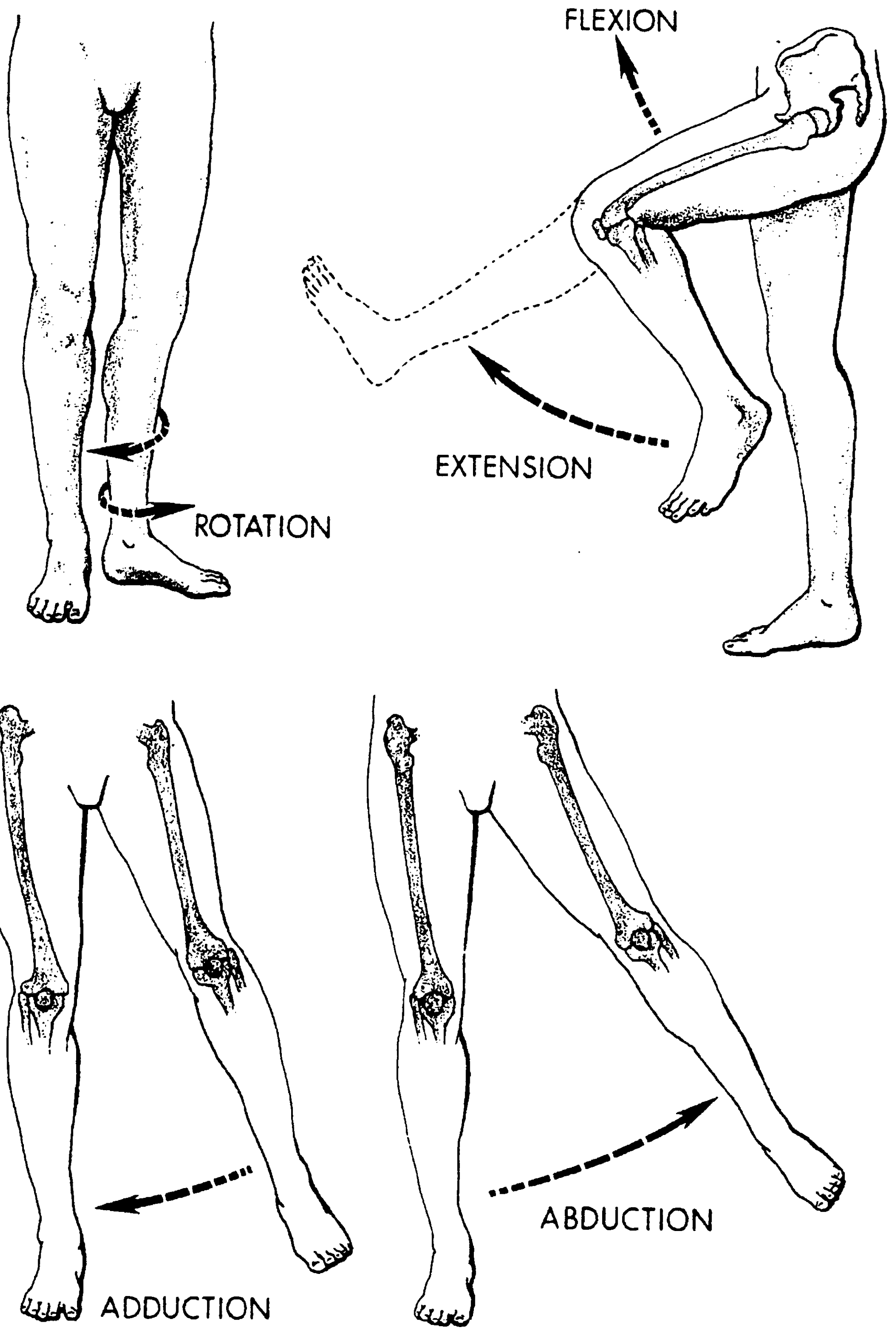


Figure 4: Joint motion terminology illustrated on the lower limbs.

From [Huelke].

ment, in order to automatically calculate the mass and inertia of the segment (dynamic biomechanical parameters). These shapes can be used for graphical display as well.

The other primary kinematic parameters are the joint DOFs. These includes the number of DOFs at each joint, and the directions in which the DOFs allow the limbs to move. Because the kinematic biomechanical parameters primarily describe the potential rigid-body motions of the figure, they can be termed the *skeletal* parameters.

The dynamic biomechanical parameters describe the data needed for dynamic simulations and include the inertia of the limbs and sources of forces within the figure. The total mass, the mass distribution, and the center of mass of a limb all influence the motion of that limb. There are several different sources of data to approximate the limb inertias: cadaver studies, machine measurement *in vivo*, and computer approximations based on integrating volumes of different densities for different tissue types. Biomechanical models of internal forces are divided into passive and active components. Passive components arise from the structures in the bones, joints, and surrounding tissues which are not under active neurological control. These forces include velocity-dependent damping, and position-dependent joint limits. Active forces are supplied by muscles, which have both active and passive elements, as well.

In classic cadaver studies by Braune and Fisher in the late 1800's, biomechanical parameters for the major human body parts were measured. [Braune 1988] The kinematic link length parameters of dissected limbs were measured. The limbs were weighed, and in addition, the moments of inertia were determined by swinging the limb segments as a pendulum about different axes, and measuring the oscillations.

Dempster directed similar work in more detail in the 1950's, using human cadavers, augmented with studies of living subjects to measure biomechanical information. [Dempster] Limb joints were studied to measure types and ranges of motion. Link kinematics were measured, and limb masses, centers of mass, densities and inertias were measured as well. The data was used to analyze work space requirements for seated operators, to assist in the design of aircraft cockpits. Drillis and Contini designed a model which parametrizes body segment lengths as a function of overall body height, as depicted in Figure 5. [Drillis; Winter 1990]

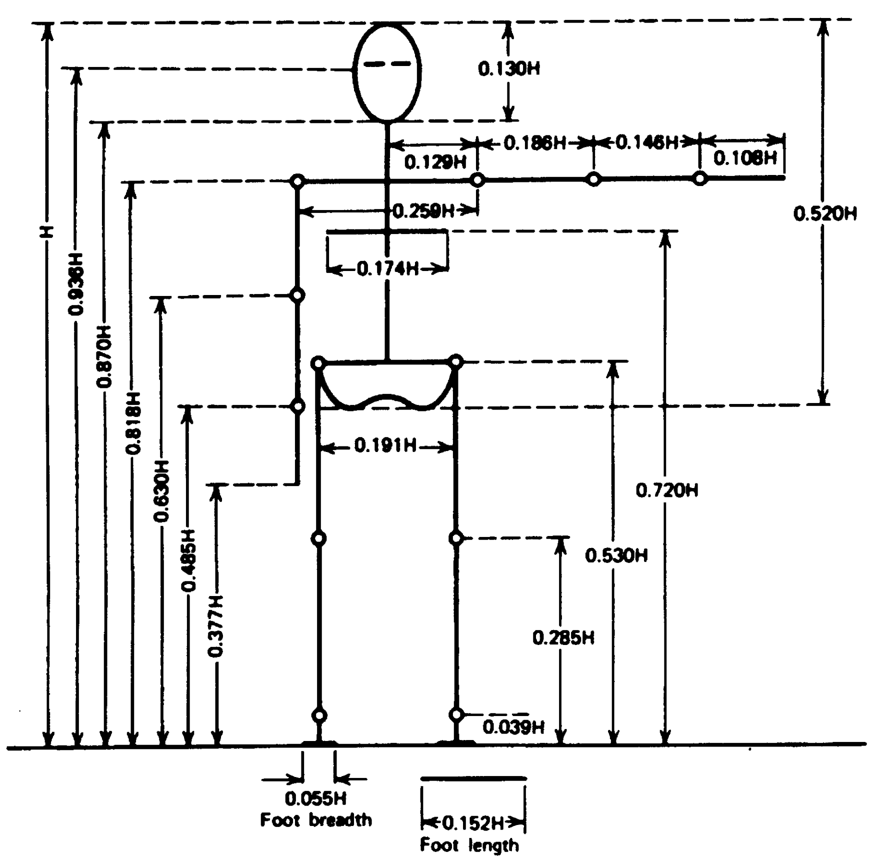


Figure 5: Body segment lengths as a function of overall body height, *H*.

This model was prepared by Drillis and Contini. [Drillis] Image from [Winter 1990].

A three dimensional geometric model of the human skeleton was designed by Stredney to facilitate animation and to provide a tool for anatomical education. [Stredney] Using three dimensional modeling techniques such as lofting and solids of revolution, Stredney manually created polygonal models of skeleton bones using real bones and diagrams as references. The skeleton model (also known as "George") is shown in Figure 6. This model was used by Zeltzer in his *skeletal animation system* (discussed below). [Zeltzer 1984] This model was also used as a preliminary guide in constructing the human figure model designed in this research (discussed in the section **Biomechanical Model** (6)).

Medical illustration literature is also a useful source of information regarding human kinematics. In particular, the structure of the foot has not been examined to the same extent as the major limbs of the body. A detailed illustration of the human foot skeleton, by Goldfinger, is presented in Figure 7. [Goldfinger] This diagram, as well as others by Goldfinger and other medical illustrators, [Gray] were used in this research as another source of kinematic data.

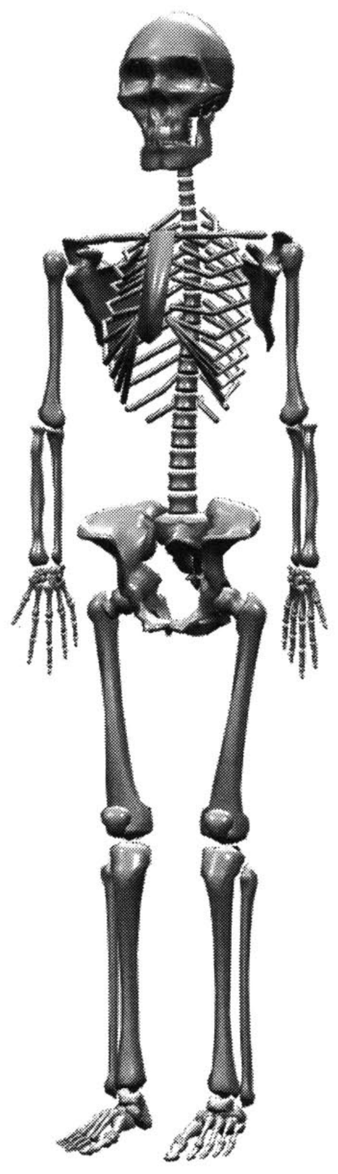


Figure 6: The human skeleton model created by Stredney. [Stredney]

An important aspect of biomechanics is the models that have been developed of the internal forces in humans. The passive elastic forces, which create joint limits, can be measured from human subjects. Yoon and Mansour measured the passive elastic moments at the hip, using an apparatus to measure force (from a load cell) as a function of angle (measured by a goniometer). [Yoon] They determined that the force could be approximated by a sum of exponentials. Hof and Van den Berg measured the passive elastic force at the ankle, and observed an exponential relationship between the ankle angle and the resulting force. [Hof] Similar testing was performed by Hatze, who also modeled the passive elastic forces in the leg as sums of exponentials. [Hatze] Audu and Davy use a sum of two expo-

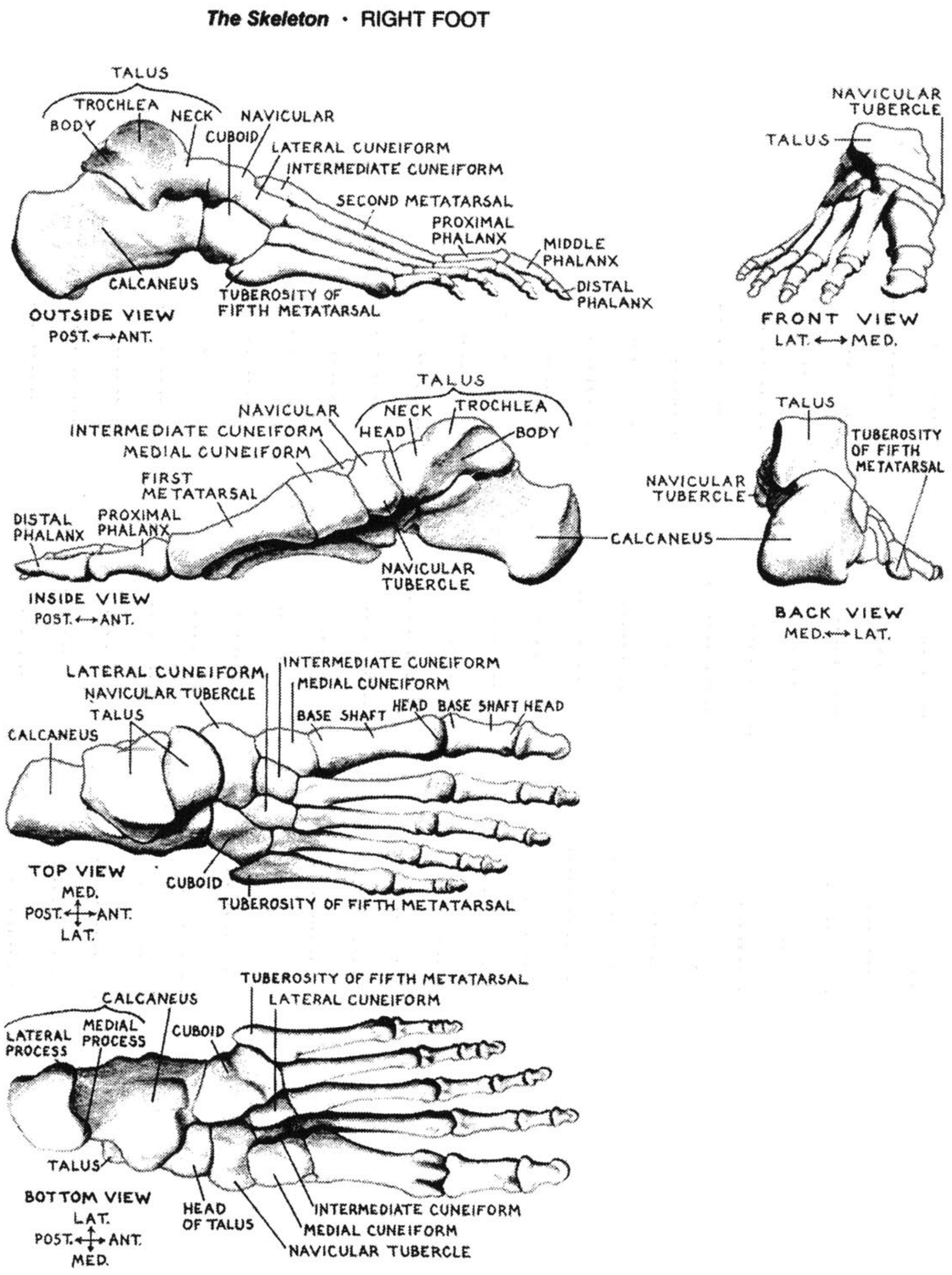


Figure 7: The skeleton of the human foot. [Goldfinger]

nentials to model the passive joint limit forces in the leg, fitted to the more complex model of Hatze, which resulted in severe numerical problems in their system in its original form. [Audu] Passive damping, due to viscous drag and friction in the joint and surrounding tissues is usually modeled as a linear element, where the resistive force is directly proportional to the velocity. [Audu; Hatze]

The final biomechanical element to examine is muscle. Human skeletal muscle can be modeled as combinations of springs, dampers, and force generators. Hill developed a model of muscle, based on observations of muscle properties. The model included a force generating contractile element, in parallel with a linear damper and a linear spring, in series with another linear spring. [McMahon]

As computational models of muscle came into more widespread use, a need developed for a general purpose skeletal muscle model which could be tailored to match a given muscle's function. Zajac and his colleagues developed a dimensionless, second order dynamic model of the skeletal musculotendon actuator. [Zajac 1986; Zajac 1989] The model has four parameters which are specified to scale it to a specific actuator. These parameters are tendon slack length, optimal muscle fiber length, pinnation angle, and muscle strength. This model has been incorporated into several biomechanical simulation systems. [Chen; Delp]

To analyze running motion, McMahon takes a simplified approach to modeling the properties of muscle, in the context of the human body. [McMahon] The model incorporates the mechanical properties of isolated muscle, along with the properties that arise from the muscle's feedback and the central nervous system descending commands. His model is depicted in Figure 8. The stretch reflex in muscle can be shown to act as a stiffness regulator, yielding a simple spring. The spindle organs feed back information regarding the length and velocity of the muscle, and effectively create a dampening property in muscle. A similar model is used in this research to lump together the muscle groups at a limb, with their feedback properties, yielding "tunable" damped spring actuators at the figure's joints.

2.4 Analyses of Human Movement and Gait

In the late 19th century, Edweard Muybridge conducted some of the first quantitative studies of gait and other human movements, using time sequence photography [Muybridge]. Figure 9 shows an example photographic sequence by Muybridge.

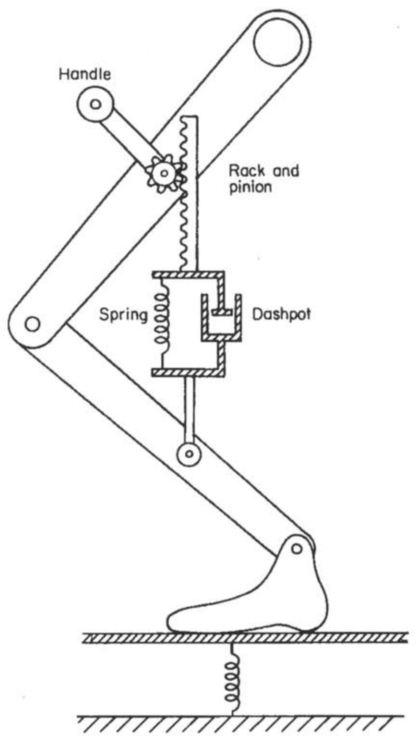
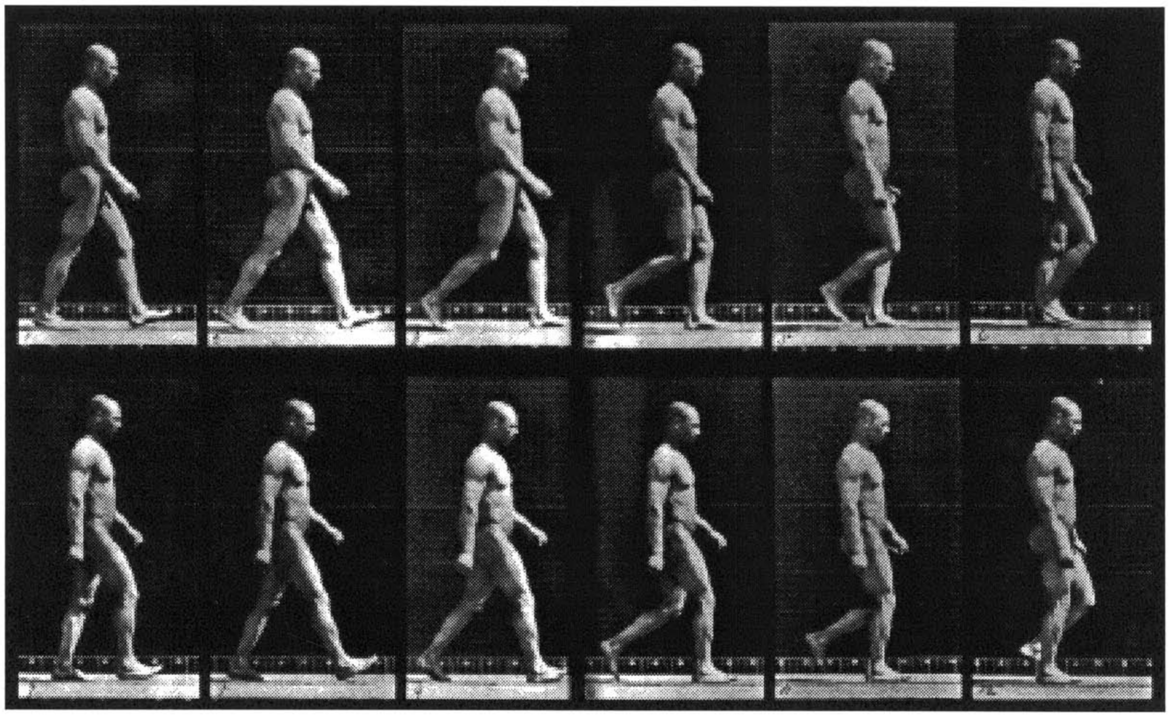


Figure 8: A mechanical model of the lower leg, using a simple spring and damper actuator.

The central nervous system specifies the rest length for the system, adjusting the rack and pinion in the illustration. The muscles, with their feedback systems are modeled by the spring and dashpot (damper).

Image and model from [McMahon].

Figure 9: A photographic time sequence of a male subject walking by Muybridge. [Muybridge]



Also in the late 19th century, Braune and Fischer performed pioneering work in human biomechanics, with a focus on a better understanding of walking. [Braune 1987] Through photographic studies of human walking, they measured approximate joint motions, and by using manual calculations, computed joint torques and the motion of the body's center of mass. Two cameras were used to photographically record the motions of luminous markers ("Geissler tubes," employing incandescent nitrogen) attached to the limbs. A subject walked by the cameras, with illumination only from the markers. Using triangulation on the recorded motion traces, the 3D displacements of the markers were extracted, and from that they projected back along the limb to determine joint motions. Combining this information with their measurements of the inertias of different body segments, [Braune 1988] they computed the displacement of the body's COM over time, and performed biomechanical analyses of the forces involved in the swinging leg during walking. Their conclusions, however, were that the muscle forces must be acting more strongly than gravity and inertia in the swinging leg. Although muscles are active, they do not dominate, [Winter 1978], and it can be shown that successful stepping motions can be generated without any contributing muscle force, a subject that we will return to. [Mochon 1980-A; Mochon 1980-B].

With the advent of digital computers, the process of acquiring and processing gait data became much more feasible. Joint motions can be tracked using goniometers, which are potentiometers connected to a frame which attaches to the two limbs which span a joint. [Winter 1990] In addition, optical tracking can be used to triangulate the locations of markers attached to limbs, and automatically extract their three dimensional location. Mann, *et al.*, describe their TRACK system, which uses two optical cameras and an LED array attached to each tracked limb. [Mann] The system has an overall accuracy to within one millimeter in position and 20 milliradians in orientation.

Another important tool in the study of gait is the force plate. As the subject walks across the plate, it measures the forces being exerted on it by the feet. These forces are known as the ground reaction forces (GRF). The vertical force, normal to the ground, reveals how the feet are pressing down on the ground, and the horizontal, tangential forces reveal the ways that the feet push forwards, backwards and to the sides, against the forces of friction.

By pairing kinematic acquisition with force plate data and a biomechanical model of a human figure, we can use inverse dynamics to determine the torques which are applied at the joints. [Winter 1978] In order to determine how the muscles are activated to generate the computed forces, an optimization process must be used. [Patriarco] Because more than one muscle spans each joint, and agonist-antagonist pairs can create the same net torque using different stiffnesses, there are many more muscle activation parameters than their are input joint torques. An optimization criteria (such as minimum expended energy) is specified and knowledge about which muscle groups typically work together, etc. is employed to determine possible muscle activations and forces. Mann, *et al.*, found that accurate measurements of the joint motions are important when determining muscle force — more important than the type of optimization method, or criteria used.

The *determinants of gait* are a way to describe the motions of the limbs during human walking, put forth by Saunders, Inman, and Eberhart in 1953 [Saunders; Inman]. There are six determinants of 'normal' gait. In general, each new determinant of gait adds the requirement for a new degree of freedom in the walking system. [McMahon] It is valuable to review this work to demonstrate the degree of freedom complexity which should be included in a system to reproduce the major aspects of human walking.

The first determinant of gait allows for one DOF per leg, at the hip, and results in a "compass gait," a kind of stiff-legged walk, with no motion of any joints except for flexion at the hip. During the swing phase the hip flexes, swinging the leg forward, and during stance the hip extends, moving the leg backwards. Such a gait would not be physically realizable since there is nothing to prevent the body from falling toward the swing-leg side of the body. In addition, because there is no flexion and extension of the knee for the swing leg, the swing foot misses the ground with an infinitesimal clearance. The compass gait results in a "bouncing" motion of the pelvis and COM. The COM sweeps out a series of arcs, connected by sharp cusps, due to the inverted pendulum motion of the body and stance leg.

The second determinant of gait adds rotation of the pelvis, about the vertical axis, to the compass gait. The magnitude of this rotation is approximately $\pm 3^\circ$ for normal walking, with a greater magnitude for faster walking speeds. Pelvic rotation serves to extend the effective length of the legs, increasing the step length, and thus walking speed, and also somewhat flattening the arcs formed by the motion of the COM. The addition of pelvis rotation necessitates the addition of rotation at the hip, adding another DOF to the leg.

The third determinant of gait adds pelvic tilt to the two previous determinants. Just before toe-off of the swing leg, the pelvis dips down approximately 3° towards the swing leg side, then rises more slowly, until it is again level, when the swing leg makes heel-strike with the ground. Pelvic tilt serves to flatten the arcs formed by the motion of the COM, since the center of the pelvis no longer rises as far as the stance leg side. Pelvic tilt adds the requirement of hip abduction/adduction, so that the hip now has three DOFs (as is anatomically correct). Pelvis tilt also necessitates the addition of flexion of the swing leg's knee, so that the foot clears the ground.

The fourth determinant of gait is flexion of the stance leg knee. The knee bends slightly during stance, with maximal flexion near the middle of support. This further flattens the path of the COM, since the pelvis does not rise as far in mid-stance, when knee flexion is maximal. A new DOF does not need to be added for this determinant, since flexion and extension of the knee (during swing) was added with the third determinant.

The fifth determinant is plantar flexion of the stance ankle and foot— i.e. bending of the ankle and sole. During touchdown of the heel, the ankle plantar flexes (while the knee flexes), absorbing impact energy. During double support, before toe-off of the swing leg, the foot plantar flexes, smoothing the transition to the swing phase. Plantar flexion of the foot plays a role in establishing the initial conditions for the swing. The cusps between the arcs formed by the motion of the COM are smoothed by this determinant. Two DOFs are added— one flexion/extension DOF at the ankle, and one in the foot to represent the joints between the metatarsals and phalanges as a group (although using only one DOF in the foot is a gross simplification of the actual structure).

Lateral displacement of the pelvis is the sixth determinant of gait. The center of mass of the body moves laterally, moving the COM closer to the supporting foot in mid-stance, such that the body rocks from side to side somewhat as locomotion progresses.

An additional important factor of gait is the inversion-eversion-inversion sequence at the subtalar (lower ankle) joint, during the support phase of walking. There is a slight inversion at heel strike, followed by eversion during most of the stance, followed by an inversion at heel-off. The sub-talar joint acts in large part to allow the leg to rotate during stance, while keeping the foot in non-slipping contact with the ground. [Inman]

The gait determinants serve in part to flatten the motion of the COM of the whole body. The motion is not completely flat, but rather, it oscillates in a sinusoidal pattern, vertically and horizontally. However, the total energy of the entire system is maintained at a nearly constant level. There is a trade-off between the kinetic and potential energies; the two are out of phase, and so energy is exchanged between the two. As the COM loses potential energy— moving to a lower height— the COM accelerates, gaining kinetic energy. The opposite occurs as the COM rises.

Mochon and McMahon studied some of the passive aspects of walking in work they dubbed "ballistic walking," since the limb motions they studied were produced by the forces of inertia and gravity. [Mochon 1980-A; Mochon 1980-B] Using computer simulations of simplified biomechanical models of the legs and pelvis, in the sagittal plane, they demonstrated that the swing leg successfully steps without any muscle force, if the correct initial conditions (joint angles and velocities) are satisfied. A number of simulation were investigated, with an increasing biomechanical complexity (introducing more of the gait determinants). As the model's complexity increased, a closer correspondence was found between the computed ground reaction force, and the ground reaction force measured for humans. Their initial work covered basic limb motions only. [Mochon 1980-A] Later work added stance leg knee flexion and foot flexion [Mochon 1980-B]. McMahon presents further material on ballistic walking, including the addition of pelvic tilt, as well as an excellent review of muscle and reflex properties. [McMahon]

Alexander examines the mechanics and energetics of locomotion. [Alexander 1976; Alexander 1985; Alexander 1990] He has analyzed different types of bipedal gaits, and computed the energy required to achieve them. The role of springs in locomotion has figured heavily in his work. Elastic components are present in flexible bone, in muscle and ligament tissues, and in feet or animal paw pads. These elements can store energy from one phase of walking and return it in another. Flexible interactions between the foot and ground create a more stable foothold by creating a compliant interface and by reducing "chatter," in which the foot vibrates off of the ground and slips.

Frank analyzes the stability of an "algorithmic" walking biped at low speeds. [Frank] The biped's motions are algorithmic, in that the feet and legs of the system are algebraically related, like a clockwork system. Analyzing slow walking speeds, in which both feet remain on the ground for most of the time, Frank shows that such a biped is inherently stable, and is able to recover from disturbances within a given range. He describes a potential energy surface which provides an estimate of the energy required to destabilize the biped.

Gubina, *et al.,* have simulated biped locomotion, including an active control system. [Gubina] The simulation is greatly simplified compared to a real robotic or biological system. Motion is restricted to the sagittal plane, and the legs are treated as massless entities. The non-linear equations of motion for the system are developed, and a control system is then designed based on a linearization of the dynamics equations with feedback control. Body attitude and altitude are controlled, as well as step length and frequency. The system produces stable results, even in the presence of large disturbances.

Siegler, *et al.*, describe a simple model of the lower body, which they use to simulate the stance phase of walking. [Siegler] Using straight, telescoping legs, the initial conditions of the limbs are established just before heel strike, and the system then responds passively, using gravity and inertia. Because passive dynamics are used, this work is similar to Mochon and McMahon's "ballistic walking." The upper body is modeled as a point mass at the hips, and the legs have 1 DOF at the hip, and 1 telescoping "knee" joint. In some simulations, they allowed the body to move laterally, creating a three dimensional simulation for the body movements.

Onyshko and Winter use forward dynamics to generate a simulation of human walking. [Onyshko] A seven segment, six joint model of the body, limited to the sagittal plane, is used. Initial conditions for the positions and velocities of the limbs are established, and a set of joint torque profiles are applied at the joints, in an open-loop manner. The joint torques were developed through a trial and error process, until one full cycle of walking was successfully generated.

Amirouche, *et al.*, present a system, DYAMUS, for the simulation of human movements. [Amirouche] The system is capable of forward and inverse dynamics, with constraints. Internal joint springs and dampers can be included in the human figure model. A simulation of walking is discussed, based on a five segment, five DOF model, which uses kinematic constrains that prescribe the motions of the swing leg's foot and the body's COM.

Morlock describes a biomechanical analysis of the foot and ankle, using a six-segment model. [Morlock] The following segments are represented in the system: the talus; the calcaneus; a link which lumps together the three medial metatarsals and the cuneiforms; a link for the two lateral metatarsals, the cuboid, and the navicular; a link for the phalanges in the three medial rays; and a link for the phalanges in the two lateral rays. Ten muscles from the leg, and five ligament structures were also included in the model. The model was used to analyze a "lateral side shuffle movement," which was recorded from a subject using gait acquisition equipment. Inverse dynamics was employed to determine the joint forces, from the subjects's motions. An optimization analysis was then performed to estimate muscle forces.

Simkin created a foot model used to analyze standing posture. [Simkin] The model incorporated 17 joints, and 14 ligaments, with 6 points for support. The foot-ground pressure distribution forces and internal forces and torques were computed as the foot model was incrementally loaded from above, at the ankle joint. The ground reaction force was compared to measured pressure distributions in standing adults and was found to be similar to, but less uniform than the measured values.

Meglan developed a passive mechanical model of the foot based on a set of viscoelastic spheres jointed together by a rigid frame. [Meglan] The foot model was included in a full body model, and was used to analyze the ground reaction forces during walking, and to simulate simple human motions. The system was capable of reproducing an approximation of the ground reaction forces recorded during human walking. Forward simulation was more successful than inverse dynamics at generating realistic ground reaction forces in his system, due to problems of accurately measuring foot kinematics from human subjects.

2.5 Machine Locomotion

Robots that walk and run will be covered briefly, here. The control of robots presents a similar problem to the dynamic control of a physically-based simulation of an articulated figure — forces are used in both types of systems to control motion.

There have been a number of statically-stable machines designed with four or more legs. Liston and Moser describe a quadruped "truck" which is controlled by a human pilot. [Liston] Pugh, *et al.*, present the adaptive suspension vehicle, a large (over 19 feet long, 7 feet wide, and 10 feet high), six-legged, human piloted walking machine. [Pugh]

Autonomous walking robots include the OSU (Ohio State University) hexapod, [McGhee] the CMU (Carnegie Melon University) hexapod, [Sutherland], and the ODEX hexapod, intended as a telerobot for nuclear applications. [Russel] All of these statically stable robots are designed to keep enough legs on the ground at a time in order maintain stability. The COM of the machines must always lie within the support region formed by the supporting legs in order to remain statically stable, and forward speed must be low enough that momentum effects do not dominate.

Bipedal robots complicate the problem of maintaining balance, especially then the gait is *dynamic*, i.e. there are phases of activity during which the system is not statically stable. In general, bipedal gaits are dynamic, with the exception of very slow, careful gaits. During walking at normal speeds in humans, the gait is quite dynamic, with few, if any periods during which the COM of the body falls within the region of the supporting foot. Running is a highly dynamic activity.

Kato, *et al.*, describe a *quasi-dynamic* walking biped. [Kato] The robot had a wide foot base, and, in general, maintained its COM within the region of the supporting foot. However, when the robot was ready to transfer weight to the other foot, it executed a short, dynamic "fall" to that foot, using a model of an inverted pendulum to plan the motion. The robot had 10 DOFs and was hydraulically powered. Later work employed a robot with a large, 3 DOF upper body, which acted as an inverted pendulum used to stabilize walking. [Takanishi]

One of the earliest, fully dynamic gaits for a biped robot was demonstrated by Miura and Shimoyama. [Miura] The robot employed active balance, using straight stilt-like legs without extra foot bases, so that statically stable support could never be provided. The robot constantly and rapidly stepped from foot to foot, yielding a stable gait, but with a somewhat "twitchy" look. The motion control was based on utilizing the motion of an inverted pendulum, which was decomposed into two components: one in the sagittal plane, and one in the frontal plane. Stepping was planned to retain stability and achieve motion goals, using a simplified model of the inverted pendulum system, with linear feedback control.

Furusho and Masubuchi present a biped robot capable of steady state walking. [Furusho] The robot employs a stepping cycle speed of 0.45 sec, with a walking speed of 0.8 m/sec. A hierarchical control system is used, with lower-level local feedback. A simplified model of the robot is employed by the control system to maintain stability. Wide "feet" structures are used to provide lateral stability, and the motion is thus limited to the sagittal plane.

The work of Raibert has focused on the design and control of running robots. [Raibert 1986] Raibert has developed a variety of robots, including monoped, biped and quadruped models, which use a uniform approach to control— de-coupling the control of a set of sub-goals, such as "hopping" height and running speed. Recently, Raibert has employed his models of robotic control to simulate dynamic locomotion for the purposes of computer animation, including non-humanoid bipedal running. [Raibert 1991]

To conclude the discussion of walking mechanisms, we review the work of McGeer, who examined *unpowered,* passive bipedal walking machines. He created and analyzed simple mechanisms which were capable of walking down a shallow incline, without any form of internal power. Gravity provided the power for the machines, and the mechanical design created stable walking patterns. One mechanism used a stiff-legged bipedal walk, using small motors to rotate the foot to the side as it swung forward, so that it would clear the ground. [McGeer 1990-A] Another mechanism used a mechanical design with bending "knees," which allowed the foot to clear the ground as it swung forward, without the use of any internal power. [McGeer 1990-B] McGeer's mechanisms were bipedal in form, and created natural appearing humanoid movements, although they actually employed two pairs of legs, which provided side to side stability, essentially by-passing the problem of lateral balance. McGeer has also analyzed bipedal running as a passive activity. [McGeer 1990-C]

2.6 Computer Animated Simulations of Human Movement

In the early 1980's Zeltzer developed the *skeleton animation system* which was used to animate the motions of articulated figures. His primary research concerned task level interaction with a simulated human capable of walking over moderately uneven terrain, using a skeletal model. The simulation employed a hierarchy of finite state machines and kinematic motor programs to control joint motions. [Zeltzer 1982; Zeltzer 1984]

The simulations were kinematic in nature, using geometric rules to create motion. Because the simulated motions were based on clinical data describing normal human gait, [Inman; Saunders] the motions appear quite realistic for slow walking over planar terrain. However, rapid movements, or motion over uneven terrain, cannot be accurately simulated without at least accounting for the rigid-body dynamics of jointed figure motion. Thus, the strictly kinematic nature of the skeleton animation system limits its adaptability and restricts what can be learned from the simulation.

Sims employed inverse kinematics and dynamic elements to simulate various adaptive gaits, over uneven terrain [Sims]. Sims' system allowed for the rapid, interactive creation of animal (non-human) forms with varying numbers of legs and limb configurations. The animals could then be automatically controlled to walk, trot, run, hop, etc. Sims designed and implemented a *figure editor*, a visual tool for designing jointed figures. The editor provides a MacDraw-style interface which allows the drawing of 2D schematics of 3D, jointed figures. These 2D representations are automatically transformed into three dimensional, articulated figures — which can then be made to walk over uneven terrain using inverse kinematics and a gait sequencer. Once a jointed figure has been created, the user simply selects the desired gait for the figure to use, and the system automatically generates the animated motions required to negotiate a given terrain. Different gaits are generated by specific functions which sequence the stepping of the legs, relative to each other. The leg motions are controlled using inverse kinematics, such that the target positions for the "feet" are specified, relative to the body, and the system computes the joint angles needed to reach that target.

Girard developed interactive methods for the specification of animal and human gaits for computer animation. [Girard 1985; Girard 1987] Limb motions and stepping patterns could be interactively specified by the user, and the system automatically sequenced the motions to generate walking and running actions. Girard used kinematic control, but added certain elements of dynamics, in order to create motions which appeared more realistic. For example, when all of the legs of a figure left the ground, a ballistic trajectory for the body would be used, and during turns the body would "bank" to the side.

Bruderlin, *et al.*, developed a model of human walking which used a mixed (or hybrid) method of kinematic and dynamic control. [Bruderlin 1988; Bruderlin 1989] Simplified models of the legs were dynamically simulated, and kinematic methods were then used to complete the simulation, adding motions of the foot, leg, pelvis, and upper body. More than 20 locomotion parameters could be specified to tailor the walking characteristics.

Wilhelms and her colleagues have experimented with animated simulation of human figures. [Wilhelms 1985; Wilhelms 1987] Using forward dynamics, based on the Gibbs-Appell formulation ($O(n^4)$ where n is the number of DOFs) motions are simulated, based on the application of forces. Modeled forces include gravity, ground and collision springs, and joint springs, dampers, joint limit springs, and direct open-loop torques. Few examples are presented of simulations of human movement, however. Typical simulations include falling passively to the ground, or applying a joint torque in zero gravity.

Armstrong, *et al.*, present a near-real time forward dynamic simulation of a human figure model. [Armstrong 1985; Armstrong 1987] Using their recursive dynamics formulation, of $O(n)$, the motions of a human figure, with approximately 9 spherical joints, is computed in response to forces. Joint springs and dampers are simulated, and limb motions are controlled by switching the spring rest angles to new positions, and imposing limits on the maximum forces and the rate of change of the forces. The development of the human figure model is not presented, and few examples of simulated motions are presented.

2.7 Previous Work by the Author: Hexapod Locomotion

The Background Section ends with the author's previous work in the field. The *roach*, a real-time, kinematic simulation of a hexapod was developed by McKenna, *et al.* [McKenna 1990-A] The user interacts with the hexapod in the virtual environment system *bolio.* [Zeltzer 1989] (See Figure 10). Using a gestural interface, the user guides the walking behavior of the hexapod, issuing commands, specifying walking directions or positions, etc. The hexapod controls its own low-level behavior, and can act autonomously in response to environmental "stimuli."

The coordination mechanism of the roach, which generates the walking gait is based on neurological features found in the cockroach and other insects. [Wilson; Pearson] *Oscillators*, or pacemakers, trigger stepping at each leg. Coupling between the oscillators generates the coordinated stepping pattern, dependent on the frequency of the oscillators. *Reflexes* serve to reinforce the basic stepping pattern, and to provide enhanced stability. Kinematic motor programs move the positions of the feet and the body, and the leg angles are calculated by inverse kinematics. The stepping patterns generated by the coupled oscillators are virtually identical to the real patterns displayed by insects.

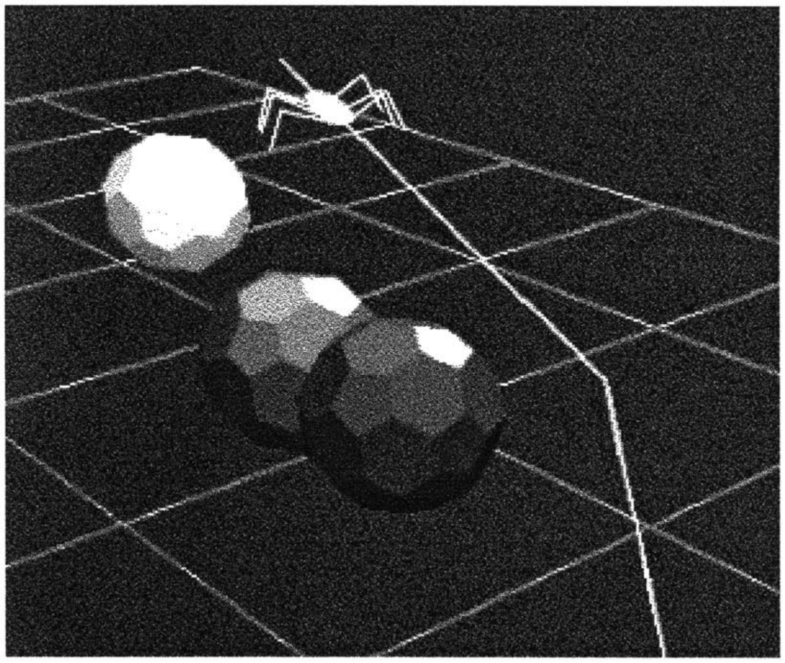


Figure 10: The kinematic roach follows a collision-free path in the *bolio* virtual environment system. [McKenna 1990-A; Zeltzer 1989]

Different oscillator frequencies create different stepping patterns, and different walking speeds. The slow gaits are "wave" gaits, in which a wave of steps travels up each side of the body, from back to front (see Figure 11).The fastest gait generated by the coupled oscillators is the tripod gait, in which a stable tripod of legs supports the body, while the

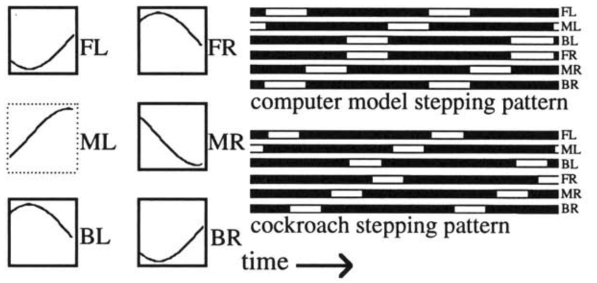


Figure 11: A wave gait stepping pattern generated by the coupled oscillator mechanism, compared to a slightly different wave gait stepping pattern, exhibited by the cockroach. The white regions in the patterns denote stepping activity. The boxes and sine wave segments at the left depict the computational model of the oscillators over a short period of time. An oscillator will trigger stepping activity in its leg when the oscillation reaches its peak. The phase differences between oscillators can be seen in the diagram.

other three legs step (see Figure 12). The coupled oscillator mechanism generates smooth gait changes, as the oscillator frequency is smoothly varied.

We have also experimented with *reflexive* feedback from the environment. A *step reflex* triggers stepping when the leg angle, relative to the body, exceeds a specified value. A simulated *"load-bearing" reflex* prevents a leg from stepping when an unstable leg configuration would result. These reflexes serve to reinforce the basic stepping pattern generated by the coupled oscillators, while making the system more robust.

The gait controller triggers "step" and "stance" motor programs in the legs. The "step" program moves the leg up and forward, then down and forward, relative to the body. The "stance" program keeps the "foot" in place on the ground, as the body moves forward. Inverse kinematic is used to compute the leg joint angles from the specified foot location.

The *roach* software was used to create an animated character in the short animation *Cootie Gets Scared*, [McKenna 1988] (see Figure 13). The control of the Cootie was "scripted" using high-level commands to control the walking speed and direction, as well as other character properties, such as head turning, etc. The leg and body motions were kinematically-controlled. However, the head and antennae of the Cootie were dynamically-simulated, to realistically respond to the force of gravity, and the accelerations of the body.

The hexapod locomotion research was extended to a incorporate a dynamic model for motion simulation and control. [McKenna 1990-B; McKenna 1990-C] The kinematic structure of the dynamic roach, shown in Figure 14, was based on the general biomechanical properties of insects, in particular, the cockroach. The motions of the roach are forward simulated; acceleration is the result of applied force. Coordinated locomotion is generated by the gait controller, as described above for the kinematic roach. Motor control for the legs is provided by dynamic motor programs and spring actuators, which deliver forces to the

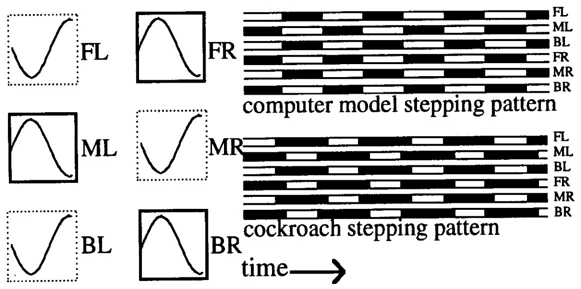


Figure 12: The tripod gait.
The primary difference between the computer model and the cockroach stepping pattern is that the computer model produces longer step and shorter stance phases. This is due to the idealization of the kinematic model— a leg can change from step to stance and take up the load instantaneously.

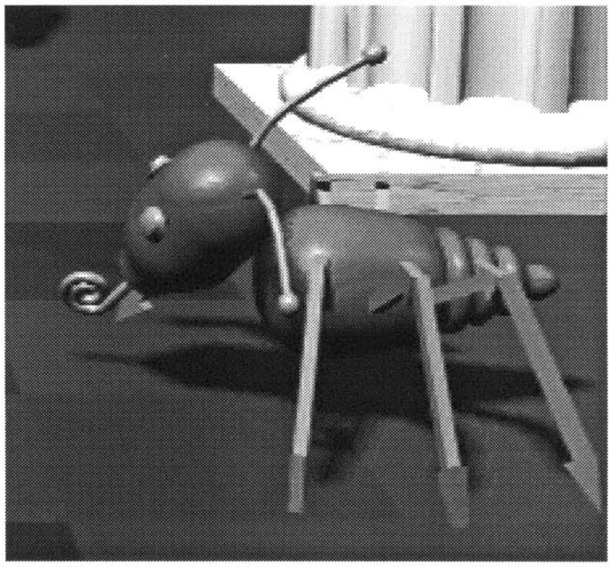


Figure 13: The hexapod, "Cootie," from the computer animation *Cootie Gets Scared.* [McKenna 1988]

The Cootie's high level actions were 'scripted' by the animator. The motions of the legs, body and antennae were automatically generated by the software. The Cootie's movements were kinematically controlled, in general, with the exception of the head and antennae, which were dynamically simulated.

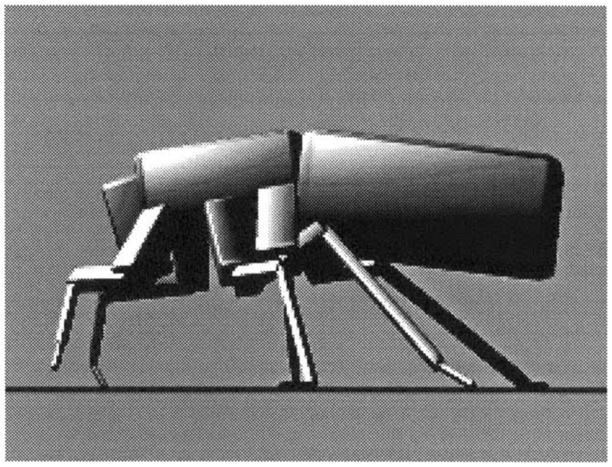


Figure 14: The parametrized dynamic roach. [McKenna 1990-B]

The figure begins to step with three legs using the tripod gait in this illustration. It has 5 degrees of freedom (DOFs) per leg, and a head and abdomen joint. The entire system has 3 translating and 3 rotating DOFs, for a total of 38.

leg joints. Forward body motion is the result of traction at the ground, as the legs "push" backwards. The simulated roach displays stepping patterns very similar to those of real insects, as well as realistic walking behaviors.

The program *corpus* was developed by the author for the simulation and control of dynamic locomotion. The flow of control in *corpus* to simulate dynamic hexapod locomotion proceeds basically as follows (see Figure 15). At the highest level of control the animator, or another program, sets the desired speed and other parameters. The gait controller coordinates the stepping pattern based on the specified speed, and sends appropriate step and stance commands for each leg to the motor programs. The motor programs compute leg joint forces which are sent to the dynamic simulator. The simulator incorporates these forces with externally applied forces, such as gravity and contact forces, and computes the motion of the figure. The graphics system then renders the figure and its environment, based on the computed positions. Each module has several parameters which can be set in a scripting language. These parameters include such factors as stepping speed, spring stiffnesses, the kinematic structure of the figure, and link size, shape and density. *Corpus* has been extended and remains the platform for the current work, and it is discussed in more detail in Section 4, **The Program Corpus**.

The dynamic simulator employs the *Articulated Body Method* (ABM) for dynamic simulation, based on the work of Roy Featherstone. [Featherstone 1983; Featherstone 1987] See Section 5, **Dynamic Simulator** for further discussion of the ABM and dynamic simulation in the *corpus* environment.

In order to derive the articulated figure we used in our locomotion experiments, we referred to diagrams of the insect Blatta and Periplaneta Americana, and descriptions of insect physiology written by entomologists. [Hughes] We then derived the hexapod shown in Figure 14. This articulated figure is modeled as being 2.9 cm long, and has a mass of 2.1 gm. There are 38 unconstrained degrees of freedom in the figure.

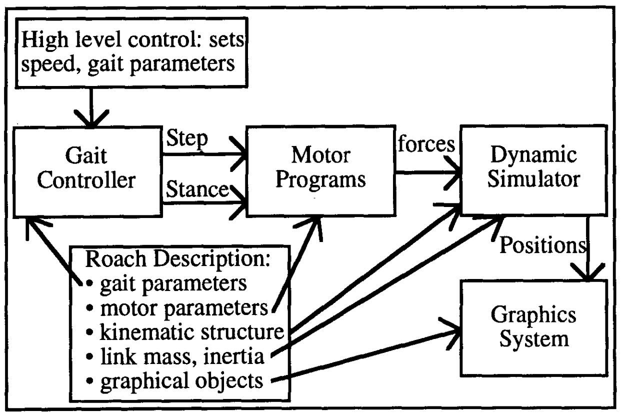


Figure 15: Block diagram of the dynamic hexapod control and simulation system, implemented in the program *corpus*.

The stepping pattern is generated using the coupled oscillator mechanism, described previously for the kinematic hexapod. The gait controller triggers motor activity, but the actual motion is controlled in *corpus* using dynamic motor programs. These functions operate at each joint at which the motion must be controlled, and can be grouped together to create more complex motor programs, such as "step" or "stance" for an entire leg.

The motor programs generate forces by modeling the force response of exponential springs, in combination with linear velocity dampers. As their name implies, exponential springs have an exponential relationship between the force they generate and the displacement of the joint from the spring rest angle, as in:

$$f = \mathrm{sign}\,(x_s - x)\ \alpha\, \mathrm{e}^{\beta\,(|x_s - x|)} - b\,\dot{x} \qquad \text{Eq. 3}$$

where f is the generated force, x is the joint angle, $\dot{x}$ is the joint velocity, x_s is the spring rest angle, α is the linear spring constant, β is the exponential spring constant, and b is the damping constant. The "sign()" function returns -1, 0, or 1, depending on the sign of the argument. In a sense, the exponential spring forces create a steep potential well, such that the controlled joint will likely stay near the rest angle of the spring.

In order to generate motion, the motor programs move the spring rest angles from their current angle to a target angle. This moves the potential well and in effect, "drags" the joint along with spring. In some sense, this method "keyframes" the controlling space of the springs.

The motor programs were progressively "tuned" by the author, over the course of several simulations, using a trial and error method. For example, initially the posture was too low to avoid dragging the abdomen, so the joints were extended further, raising the posture. Also, the legs did not step high enough initially, so the motor programs were modified to retract the legs further during stepping.

The "springy" method of motor control, with trial and error calibration, may not be suitable for actions which require exact movements, such as grasping and manipulating objects. However, the fact that the springs create a compliant system can be of great benefit. For example, we have experimented with simulations over uneven terrain, which employed the level terrain motor programs, and the hexapod conformed to the uneven surface due to the springy compliance at the joints.

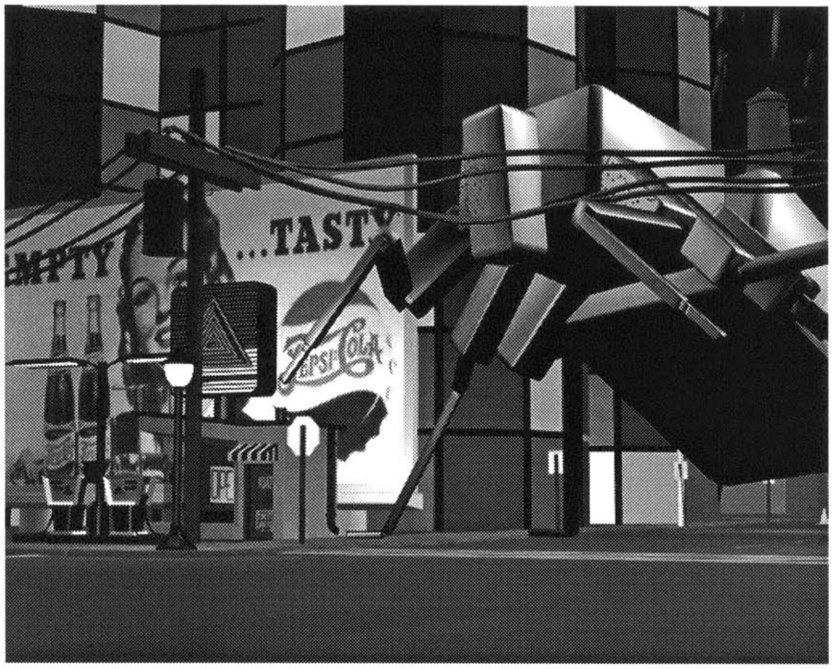


Figure 16: Scene from the animation *Grinning Evil Death.* [McKenna 1990-D]
As the Roach walks forward, it generates collision forces against the wires.

The dynamic roach and other simulated figures played key roles in the award-winning computer animation, *Grinning Evil Death*, by McKenna and Sabiston [McKenna 1990-D] (see Figure 16).

3 Approach

This thesis is concerned with the creation of a physically based model of a human figure, which can be simulated to perform and analyze motion based tasks. One major component of this work is an efficient simulation system which incorporates the physics of motion. Another major component is the biomechanical model of the human figure. The final component is a dynamic control system used to influence the motions of the figure, resulting in animated simulations of movement.

In this section we cover these major areas, and in the process, we will develop a basic human figure model to demonstrate some of the concepts. The following sections present the complex model development and the results in detail.

3.1 The Program *Corpus*

Corpus is a computer program, developed by the author, for the simulation and animation of articulated figures. It is the primary research tool used to investigate this thesis topic. *Corpus* includes a 3D computer graphics system integrated with a dynamics simulation system. Biomechanical models are incorporated in *corpus*, partly as supporting simulation software, and partly as the structural definition of the articulated figures and dynamic environment.

Corpus is a flexible system, with an interactive, programming-type of interface. This allows the user to explore different configurations of articulated figures, different simulation parameters, various graphical properties, etc., all in the same environment.

The section **The Program Corpus** (4) provides an overview of the system.

3.2 Dynamic Simulator

The use of the dynamic simulator is central to this thesis. This works focuses primarily on the forward dynamics of the human figure model — the animated motions (or stationary postures) of the figure are computed in response to the applied forces. In addition, inverse dynamics is used to compute what forces are exerted, and what control parameters are used for different postures or motions.

Dynamic simulation is an excellent tool for the generation of "realistic" movements, because the motion is physically based. Although dynamic simulation is complex and compute-intensive, and the dynamic control of multiple DOFs can be a difficult task, dynamics simplifies other aspects of the control problem. In comparison to manual animation, in which the animator must specify the motions or key poses for every DOF, the advantage is significant; dynamics generates the motions of all DOFs automatically, although the animator loses much of the direct control over those motions. In comparison to kinematic techniques, dynamics generates realistic motions, without artificial means to compensate for the lack of physics in the model. With passive dynamics motions in particular, the dynamic control problem is minimized, yet complex motions are generated in the figure. Kinematic methods would require very complex control methods to emulate such passive dynamic movements.

Because the dynamics equations are based on the laws of motion that are experienced in the real world, a dynamics simulator forms a platform from which we can explore hypotheses of motor control, including biological or robotic control strategies. Just as real animals and people must supply forces through their muscles, and robots through their motors, the simulated figures use forces to control motion.

The dynamics simulator also serves as a tool for motion analysis. Using inverse dynamics, the forces exerted during movements are computed. Using additional techniques, we can compute the control parameters which are used to drive those motions. When complex muscles models are incorporated into the simulator and biomechanical model, we can compute how the muscles might be activated to perform a given movement.

Featherstone's *Articulated Body Method* (ABM) [Featherstone 1983; Featherstone 1987] is the algorithm employed for dynamic simulation. The ABM is an efficient method for forward and inverse dynamics simulation, due to its recursive nature. The computation time grows linearly with the number of joints: O(*n*). The mathematics of the ABM are given in Featherstone [Featherstone 1983; Featherstone 1987] including descriptions of *spatial algebra* and *spatial notation*, in which the ABM is written. Spatial notation allows the translational and rotational aspects of motion to be treated together, uniformly, in six dimensional vector and matrix quantities.

The ABM, spatial algebra, and other aspects of the simulator in *corpus* are covered in the section **Dynamic Simulator** (5).

3.3 Biomechanical Model

In order to simulate the human body, accurate *biomechanical* data are required— i.e. kinematic and dynamic parameters which capture the pertinent information needed to describe a humanoid figure. The kinematic parameters include the geometry of the limbs and the joint degrees of freedom (DOFs). The dynamic biomechanical parameters describe the data needed for dynamic simulations and include the inertia of the limbs and sources of forces within the figure.

As it operates within the dynamic simulator, the biomechanical description allows us to examine *function*, which arises from the kinematic and dynamic structure. With biomechanical data that is tailored to match a given person, we can examine the specific function of that person's structure. If "normal" function is disrupted, *patient specific* data allows a doctor to examine the problems particular to that patient. Using a simulation system, the doctor can attempt to determine what can be done to best repair the problems, and restore maximal function to the patient. Such analyses can be performed today, using systems such as *SIMM*, which includes a detailed model of the human legs, with representations for all of the major muscles. [Delp] This thesis does not deal directly with patient specific data, but it is an important issue for future research. This thesis lays the groundwork for such investigations, and advances the complexity of simulated foot models, allowing new function to be examined.

The mechanical data and models used in this thesis are based on the biomechanics of humans. A fairly complex 3D model is developed for the kinematic structure of the figure, based on the sizes and inertias of "typical" human limb segments, which are determined by cadaver and clinical studies. [Braune 1988; Dempster] The major degrees of freedom in the human body (arms, legs, neck) are defined. A complex spine is not modeled, but a 3 DOF "waist" is included. The hand is a single object with a 3 DOF connection to the forearm, but the foot is modeled with a significant complexity — 28 DOFs per foot. The degrees of freedom used in normal walking, as defined by the determinant of gait [Inman; Saunders], are all supported. The joint axes are based on anatomical studies of joint motions. [Murphy; Procter]

For the purposes of illustration, we examine a basic biomechanical model of a human figure here. The model is very fundamental; simple geometric block objects are used to form an armless, simplified human figure model. Two very similar basic models were created, one based on an informal measurement of the author's limb lengths, and another based on clinical studies of human anthropometrics. [Dempster] The differences between the two models are not important for the examples which follow, and in general we will not bother to make a distinction between the two, referring to both as the "basic human figure."

The entire articulated figure is free to move and rotate in any direction, with 3 DOFs for translating motions, and 3 DOFs for rotating motions. There are two legs, each leg being composed of three links — one object for each thigh, shank, and foot. There is 1 DOF at the hip, one at the knee, and one at the ankle. All of these DOFs work in the sagittal plane, providing flexion and extension at those joints. There is no waist or spine model, nor are there any arms. A head with one neck joint was added, yielding a 13 DOF model. The masses of the links were not matched directly to values measured from humans, as was done with the complex model. The masses were determined by the geometric volume of the graphical link objects, using a density of 800 kg/m^3 (which is somewhat low, 1100 would be more appropriate). The values of the link lengths, masses and densities are presented in Table 1.

The kinematic structure of the basic figure model is shown in Figure 17. This human figure model is similar in complexity to many dynamic models used in computer graphics and animation, and in walking biomechanical research. [Mochon 1980-A; McGeer 1990-A; Wilhelms 1987; Bruderlin 1989; Pai]

The biomechanical model also incorporates a model of passive joint forces. *Joint damping* results from the frictional and viscous properties of the joints, muscle, and other surrounding tissues. A linear damping model based on joint angular velocity is used to approximate

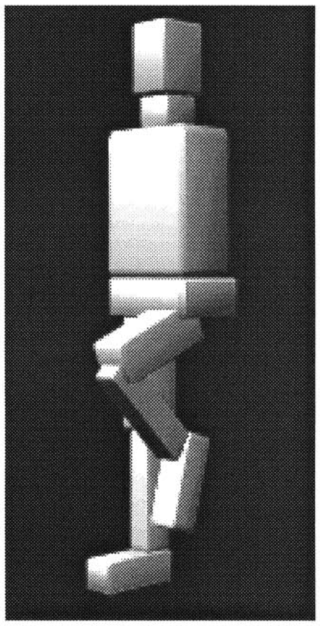


Figure 17: The basic model of the humanoid biped figure.

There are three DOFs per leg— flexion and extension at the hip, knee, and ankle. There are 3 translating and 3 rotary DOFs for the overall body motion, and a rotary DOF for the head, for a total of 13 DOFs.

Table 1: Link parameters of the basic human figure model. Length values for the thigh, shank and foot were rounded from measurements by [Dempster].

link	*length (m)*	*mass (kg)*	*density (kg/m^3)*
thigh	0.4	5.3	800
shank	0.4	2.8	800
foot	0.25	0.8	800
head	0.22	6.0	800
total body	1.71	61.7	800

these forces in the simulator. *Joint limit* forces are due to the passive elastic elements of the musculotendon organs and surrounding tissue, as well as contact forces between joint and bone structures. Joint limit forces can be modeled as sums of exponentials as a function of joint angle. [Audu; Hatze; Yoon]

Controlled, active joint forces are provided by an actuator/muscle model, which is based on the general features of muscle, modeling its viscoelastic properties. Muscle is considered as a "tunable" spring, [Bizzi 1982; McMahon] and the force response of the simulated actuator is modeled as an exponential spring, which provides control by moving the rest position and modifying the stiffness parameters. This approach is based in part on the general features of the equilibrium position hypothesis, an attempt to explain biological motion control. [Bizzi 1982; Bizzi 1984] This hypothesis states that different postures are formed by different state configurations in the control space of the neuromuscular system. Motions are controlled by moving the control state from one configuration to another. In this system, *motor programs* are used to modify the actuator parameters from one configuration to another, in order to control motion.

The biomechanical properties are very important to the simulation, because they determine, in part, what motions result. They are especially important to passive motions, because they are highly governed by the mechanical design. As an example, Figure 18 shows the basic biped figure falling to the ground, with and without joint limit forces.

To create the complex human figure model used in this research, several sources of information were employed. To define the kinematic structure, a digitized skeleton model [Stredney] was used as a 3D geometric reference, the link lengths were established to match published anthropometrics data [Drillis], and the model was refined using anatomical diagrams. [Goldfinger]. Masses of the links were based on human body studies. [Winter 1990] The complex biomechanical human figure model is described in the section **Biomechanical Model** (6).

3.4 Simulations of Human Posture and Movement

Using the biomechanical figure model in the dynamic simulator, a number of human motion simulations were investigated, including simulations of standing posture, a reaching task, and the swing phase of human walking.

The different simulations use the same low-level behavior control provided through the joint actuators, dampers and joint limits. Higher level control is achieved by manipulating the parameters of these low-level mechanisms. There is no single high-level control layer incorporated into the system, to control all actions. Given a particular task, high-level control is developed, based on the requirements of the action. The high-level control strategies that have been employed in this research share some common aspects, such as: using joint spring actuators to maintain posture or generate movement, using motor programs to vary the control state of the actuators over time, using inverse dynamics to calibrate different control states for the actuators, and using passive dynamic effects to generate motion.

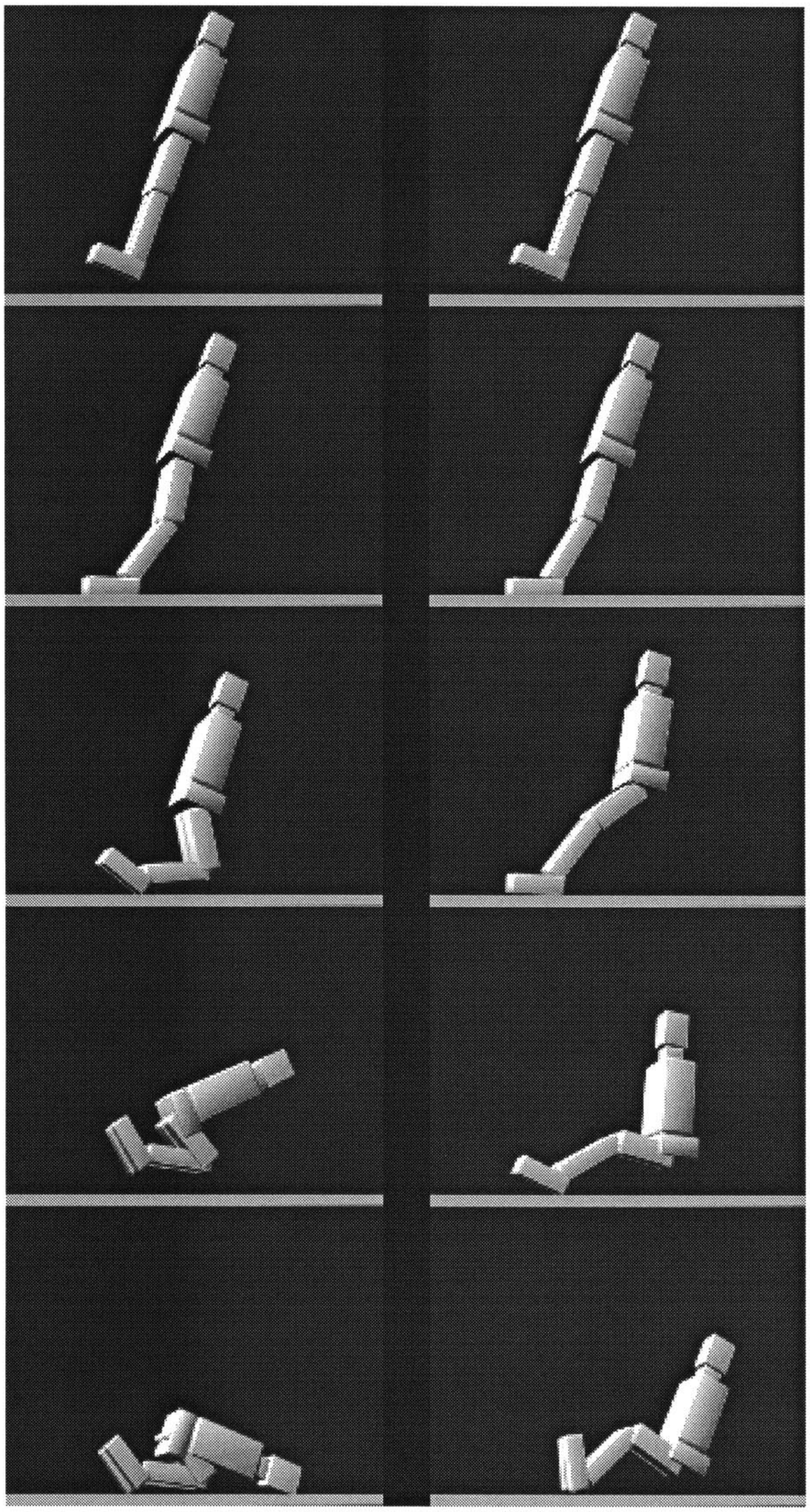


Figure 18: The basic biped figure falls with and without joint limits.

Left: Without limits, the figure takes on a very unnatural limb configuration, as the knees hyperextend "backwards."

Right: With the addition of passive joint limits, the figure responds in a much more "natural" manner, without active control.

First, we will examine the simulation of standing. Maintaining a standing posture with a simulated figure requires that the figure deliver the appropriate forces (torques) at the joints to resist the forces created by gravity and contact with the ground. The figure must be posed such that its overall center of mass (COM) lies within the region of support, formed by the points of contact between the figure and the ground.

The use of inverse kinematics would be an appropriate way to generate such a posture. This is demonstrated well by the system *Jack*, by Badler, *et al.* [Phillips] Because kinematics were of a lesser importance in the current phase of this research, *corpus* was not given facilities for inverse kinematics. In this research, the standing postures were generated "manually," by the author using scripts to control the joint angles of the figure. In some instances, poses were mimicked from drawing and photographs of humans. In other simulations, dynamic methods were used to generate non-standing postures.

Once a given posture has been defined, it must be maintained by the figure. There are a number of approaches that can be used to achieve this. The most basic (and limited) way to maintain a posture is to make the figure one rigid object, creating a figure similar to a stone statue. Because the posture is already stable, and there are no joints to move, the figure "stands" on its own, but the figure cannot move in any way, except to topple and fall if the support surface moves or the figure is disturbed by sufficient external forces.

In a similar manner, without transforming the figure into one rigid object, the joints of the figure can be kinematically locked, so that they cannot move. Now, as the figure stands, inverse dynamics can be used to compute the torques which are active at the locked joints. These forces are the ones needed to satisfy the condition that the joints remain motionless, with no velocities or accelerations introduced, against the externally applied forces of gravity and ground contact. The figure still behaves as a stone statue, albeit one that incorporates accurate strain gauges at its joints.

Now that we know what joint forces are need to maintain the posture, we can employ that information to simulate the human figure using forward dynamics. The joints of the figure are "unlocked" and are free to move under the influence of applied forces. To maintain the posture, the forces computed by inverse dynamics are applied, analogous to the forces which are generated by the muscles in humans. One method of applying these forces is through "open loop control," in which the computed torques are simply applied directly at the joints. The same force is continually applied over time; there is no feedback from the system to change this force, thus the "open-loop" nature of the control. Because the posture is intended to be stable and non-moving, it would seem that this approach would be successful. However, any error or inaccuracy in the computations of the inverse dynamics or of the forward dynamics over time will allow small amounts of motion to be generated at the joints, creating a small change in the posture. There is no mechanism to counteract the motion, and the computed forces are valid for the initial posture only, and so the errors and motion then build rapidly. In practice, this breakdown occurs almost immediately in the simulator, resulting in a falling figure. Even without errors, any externally applied perturbation to the figure would not be counteracted. In order for this approach to work, the inverse dynamics analysis would have to continually re-applied to recompute the joint forces to account for any error or perturbations.

Instead of directly applying the computed forces at the joints, we can use the spring-based actuators to deliver the forces. After performing inverse dynamics, we need to determine the control parameters required to deliver the given forces. We can term this process *inverse control*, which is analogous to inverse dynamics. We can also define *forward control* as the process of specifying control parameters for the actuators, which then compute the applied forces. Using inverse control, the control parameters of the springs are "calibrated" to match their force goal, defining an equilibrium position of the control state. In a sense, this "pre-loads" the springs so that they are prepared to support the body and counteract gravity. Using inverse control to calibrate forward control is the approach used in this thesis to control standing and other postural goals.

The control parameters of the springs are their rest angles and stiffnesses. Inverse control could be used to adjust any combination of these parameters, although when more than one parameter per joint must be computed, the solution is underconstrained, with multiple possible solutions from which one solution must be selected. To simplify, inverse control is used to solve for the rest angles only, with specified stiffnesses.

After performing the inverse dynamics and inverse control analyses, we can successfully simulate the standing posture of the figure using forward dynamics. The standing figure is not rigid, and the joints will deviate from their initial positions in response to perturbation forces. Applied forces which are not too great in magnitude will cause the figure to change its posture, away from the force, but a stable posture will be maintained as long as the fig-

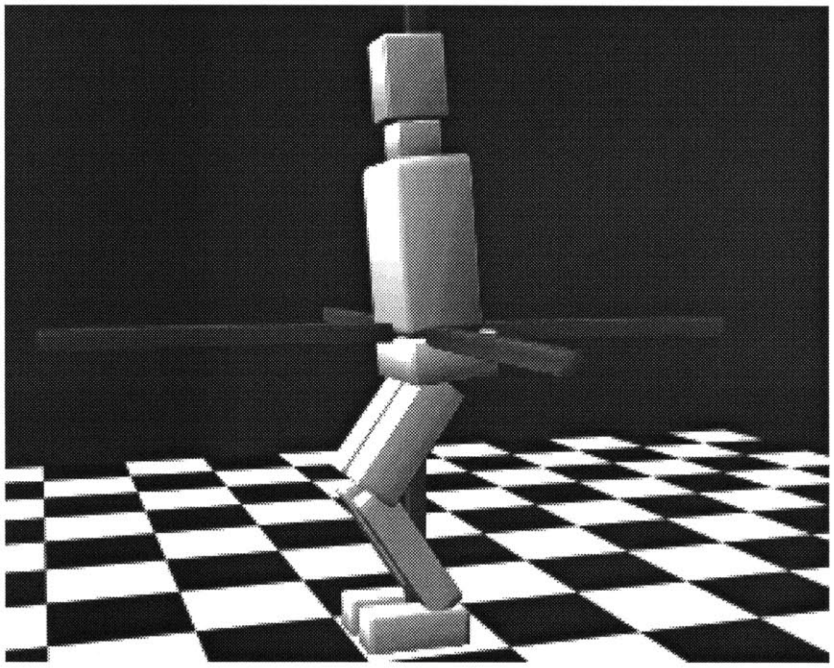


Figure 19: A balanced, stable posture, using the basic biped model.

The COM of the body is indicated by the 3D cross-bars, which intersect within the torso.

This posture is maintained against the force of gravity and the reaction forces at the ground by the actuators. The dampers and exponential springs employ feedback to reject a range of perturbations and errors.

ure's COM remains within the region of support formed by the foot-ground contact. Larger forces will cause the COM to move too far, and stability will be lost and the figure will topple. These external forces can also be taken into account by the inverse dynamics and control analyses to counteract the forces (within limits) with the actuator springs. Figure 19 depicts a stable posture of the basic human figure, maintained over time by the joint actuators.

Human walking, using the complex model, is also explored through simulation, although the model of locomotion is not a complete control system. In the walking simulations, we attempt to take advantage of passive dynamic effects, where the natural system dynamics, including the biomechanical model, create the motions. Walking is a complex control problem — especially in the case of human walking, which is highly optimized. By simulating the passive effects, we can examine the system behavior which arises not from high level control, but rather from the inherent mechanical design. In some sense, passive dynamics is a very "pure" form of forward simulation, because the motions which are generated are based purely on the mechanical properties, without the addition of any potentially "arbitrary" control changes. Under the right conditions, the passive system behavior can generate very realistic motions when compared to the real motions of humans. Human walking is, of course, not a purely passive activity. Muscular forces are required to support

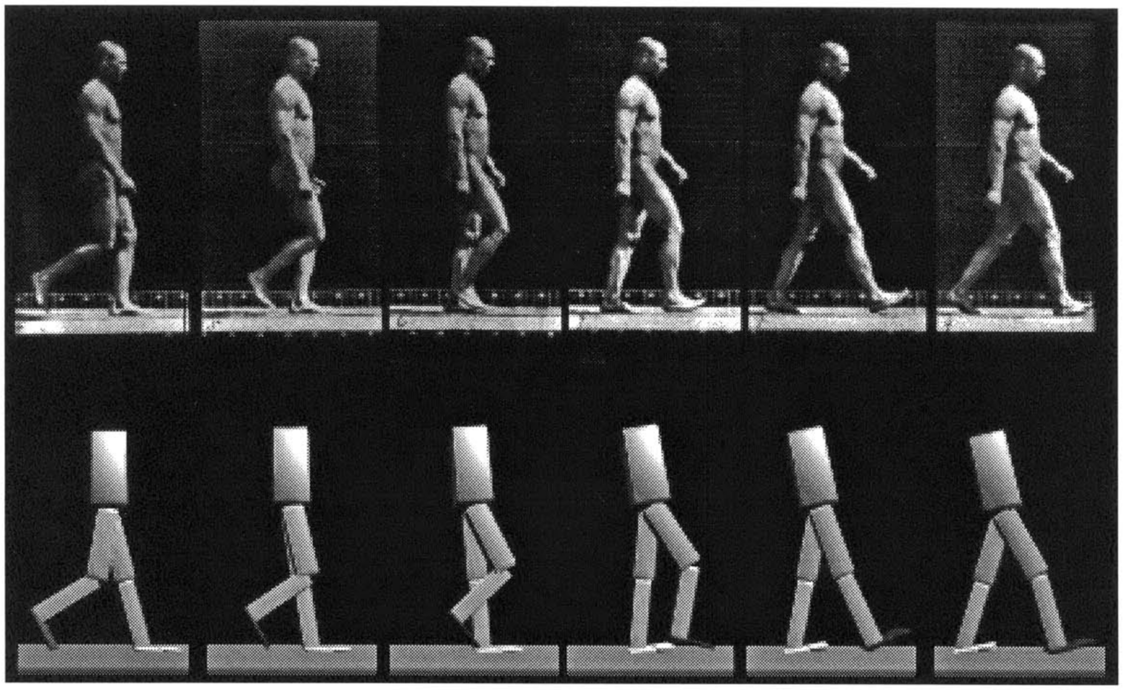


Figure 20: Passive stepping motion, using the basic biped model, compared to a Muybridge sequence.

the stance leg and upper body. In addition, electromyographic studies and inverse dynamic simulations [Winter 1978] show that muscles are not completely inactive in the swing leg, although they are mostly inactive. [McMahon]

The walking simulations investigated in this research are not purely passive either, but passive effects are employed where possible, and the walking simulation is limited to the phase of walking during which the passive forces are more significant — during the swing phase, from the time of toe-off of the swing leg to its heel-strike.

As an example, we will examine the passive stepping activity of the basic human figure model (see Figure 20). This simulation reproduces some of the results from Mochon and McMahon's passive "ballistic walking" experiments. [Mochon 1980-A; Mochon 1980-B] A hybrid dynamics approach was used for the motion simulation in order to simplify the problem. The stance knee was kinematically locked, and the stance hip and ankle were kinematically controlled to undergo a constant angular velocity rotation. Initial conditions were established for the joint positions and velocities of the step leg and for the initial overall body velocity. The system was then allowed to simulate forward, and a successful stepping motion resulted. The step leg acts as a passively-swinging double pendulum, and the stance leg acts as a passive inverted pendulum. The coupling of these two types of motions results in a natural appearing human step. The feet were made very wide to temporarily bypass the problem of maintaining lateral stability.

The passive effects during stepping are explored further with the complex body model. Passive effects are added to generate arm swinging motions, lateral body motion, and most notably, the stance leg foot shape. The results are compared to other simulations and to human walking.

Other simulations are investigated as well, including a near real-time simulation of a reaching task using the figure's arms. The simulation results are presented in the section **Simulations** (7).

3.5 Computer Graphics and Animation

The dynamic simulation and control system within *corpus* is embedded in a graphics and animation system also part of *corpus*. Animation aids in the study of the motions, and is certainly a tool for creative expression and communication. Using interactive graphics, we can directly observe the results of the simulations and the user inputs. The simulation results (joint accelerations, velocities, positions, etc.) are shown in *context*; we do not have to examine plots of joint angles over time, but instead (or in addition) we can observe the movements of the limbs of the figure.

The geometric data used to display the body segments shares relationships with the dynamic properties of those bodies. The graphical objects can be used in the geometric collision detection between bodies and the ground, and between different bodies. The mass and inertia can be automatically calculated from the geometric volume, with a specified density. Each body segment is composed of any number of graphical objects. Each object is specified as to whether or not it contributes to the body's inertia, and whether or not it is a "colliding" object. In this manner, complex body inertias can be created from multiple objects of different densities, and collision detection can be performed on simplified objects for increased speed performance.

The integration of a rendering and animation system with the dynamic simulator adds many visual parameters which have no direct effect on the dynamics, such as color, shading, lighting, etc. These parameters may not influence the realism of the motions, but it can be used to enhance the realism of the images. Facilities are also supported in *corpus* for manipulating 2D elements, allowing for the creation of animated 'diagrams.' These 2D elements include antialiased text and lines, and bitmaps, with matting functions.

4 The Program *Corpus*

4.1 Overview

Corpus is a computer program, written by the author in the **C** language. *Corpus* provides functionality for creating dynamic simulations of articulated bodies, which can be animated over time using computer graphics. The program *corpus* derives its name from the Latin word. The definition of "corpus" reads: "the body of a man or animal..." [Webster's] *Corpus* is a system for simulating mechanical, animal and human structures.

Corpus provides a flexible system for generating graphics, animation and dynamic simulation. *Corpus* is controlled by giving it text commands, which it interprets in real-time. Thus, the system behaves like an interpreted computer language. This is accomplished by the *parser* sub-system in *corpus*, which converts the input text commands to executed commands. The *graphics* sub-system generates rendered images, based on the 3D database. The *dynamic simulator* subsystem creates and analyzes motion, based on its dynamic object database (see Figure 21).

These three sub-systems are the primary *procedural* elements in *corpus*, providing its computational functionality. [Zeltzer 1990] The procedural elements perform the simulation, graphics and other functions, but they require data and instructions in order to implement a given simulation. The *structural* elements in *corpus* define the objects, data, and simulation forms, such as: the articulated body kinematic definitions; body sizes, shapes, and masses; gravity, ground stiffness, and other environmental parameters; internal actuator parameters; motor programs; etc.

Because *corpus* uses an interactive interpreter, command sequences do not have to be established beforehand. Therefore, it is not necessary to re-compile *corpus* in order to generate different simulations. The user can construct a simulation or three dimensional envi-

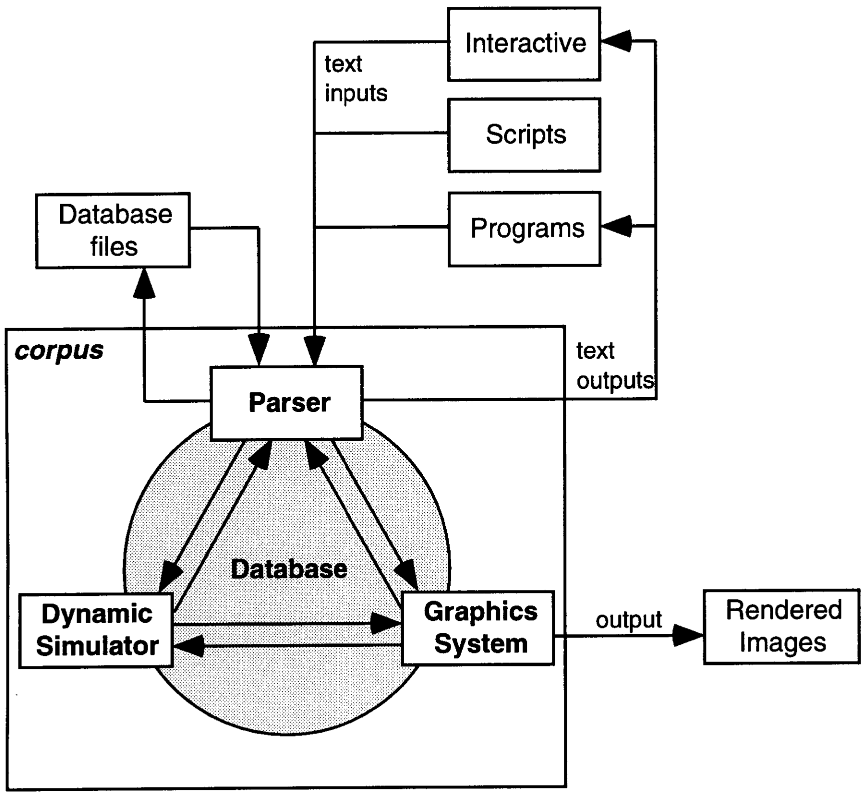


Figure 21: Block diagram of the program *corpus*.

ronment and vary parameters as events progress, using input commands. This allows for exploration and experimentation, since different commands can be "tried out" very rapidly. In addition, external programs can connect to *corpus* in real-time, either as a controlling or controlled process, without "linking" to *corpus*. Because it is interpreted in real time, *corpus* provides some of the advantages of a true interpreted language, such as **lisp**, but with the added advantages of higher execution speed and a larger hardware and software support base (because it is implemented in **C**). The disadvantage of a parser approach, as opposed to an interpreted language, is that internal commands cannot be constructed or re-defined in real-time (re-compiling is needed), and the language structure is less sophisticated.

4.2 Parser

The parser accepts text input, and generates text output in *corpus*. A number of input sources can be given to *corpus*: interactive (typed) text commands, script files, and other computer programs. The input text is commands and data, which the parser interprets. Some command examples are "`render`", to generate an image of the 3d scene, and "`eye 10.2 5.5 1`" to set the rendering eyepoint (or viewpoint) to the {x,y,z} coordinate:

{10.2, 5.5, 1.0}. The text output consists of command feedback, results and requested data.

This interface to *corpus* can be considered a *programming* interface, because *corpus* acts as a language, which is programmed via its input. [Zeltzer 1990] However, *guiding* interfaces can be created in *corpus*, as well, by combining *corpus* with other I/O programs which map user inputs to *corpus* commands and data. For example, *corpus* has been used to experiment with *viewpoint dependent imaging*, in which the graphical display of a three dimensional environment is updated in real time, based on the position of the viewer's head. [McKenna 1992] Tracking devices were used to locate the user's head and hand, so that the graphical viewpoint could be adjusted and the user could manipulate objects.

In addition, *task level* interfaces can be created in *corpus*, by establishing high-level simulations, which are task, or goal, driven. [Zeltzer 1990] These task level interfaces are created by using the programming interface to create a simulation environment, possibly in combination with guiding interfaces to manipulate data and controls. For example, the simulated roach, discussed in the **Background** section, uses high-level parameters, such as walking speed and direction, to control the task of walking. [McKenna 1990-A; McKenna 1990-B; McKenna 1990-C]

The basic use of *corpus* is from a keyboard with a text display. The interactive text input is typed by the user from the keyboard. *Corpus* displays an input prompt, (such as "`corpus>`") after which the user types in command text. *Corpus* may then display text output, providing command feedback and results.

Scripts are text files which contain commands. They are read into *corpus* and executed, essentially as if a user had typed them. Scripts are useful for storing command sequences which are used repeatedly, such as a script to construct and initialize an articulated figure, or a script to vary a set of rendering parameters. Scripts are a key component for *corpus*, because they are typically the means of storing articulated body definitions, simulations and parameters. *Corpus* alone has no "built-in" simulations or objects, so we use commands and scripts to define them. Thus, scripts store both structural descriptions which are loaded into the *corpus* database, as well as commands which control procedural execution. Scripts do not employ the text output from *corpus*, although their execution may be conditionally based on parameters within *corpus*.

Other programs can function in cooperation with *corpus*. The output of an external program can be used as a substitution for, or an augmentation of, the interactive text input to *corpus*, sending commands to *corpus* for it to execute. The text output from *corpus* can then be used as feedback to the external program. In such as situation, the external process is the "parent" process in relation to *corpus*, and it acts as the controller, or "master," while *corpus* serves as the "slave" process. For example, a simpler version of *corpus* was used as a "child" of the virtual environment program *bolio*. [Zeltzer 1989] In this example, *corpus* computed the motions of a kinematic hexapod, and *bolio* handled the device I/O, the graphics rendering, other simulations, and the overall virtual environment coordination. [McKenna 1990-A]

Alternately, an external process can be started as a slave to *corpus*, with *corpus* functioning as the master. *Corpus* issues commands to its "child" process and receives feedback commands and data from it. A simple example of this is the *spline* program, which performs interpolation computations and returns data for *corpus* to use. A more complex example is provided by the following situation: the programs *corpus* and *vestool* (another dynamics simulation program [Schröder]) have been used to cooperate in the simulation of multiple, interacting dynamic objects. *Vestool* computed the free motion of the objects, and *corpus* computed the impact responses during collisions.

The *corpus* parser includes some basic language-related commands, including integer variable manipulation, arbitrary data lists, conditional execution, and simple function definitions. A more complex language could be used within the *corpus* parser. The author created the *corpus* parser as a means to rapidly prototype different situations, with the ability to add new commands very easily. In recent years, however, standard packages have become available which provide a similar type of parsing function, but with more sophisticated language capabilities. Notably the *Tcl* system provides a flexible command interpretation language. [Ousterhout]

4.3 Dynamic Simulator

The dynamic simulator performs the computations required to simulate the dynamic motion of the articulated figures described within its database. The parser is first used to define the simulations, setting the parameters and defining the dynamic objects for the simulator to operate with. The dynamic simulator sub-system is linked to the graphics sub-system, and they share some common data. There are also significant regions of the database which are not shared, but which are unique to either the simulator or graphics system.

Parser commands are used to control the dynamic simulator, and to query for simulation results. After defining a simulation environment, the parser command "`go`" instructs the simulator to perform the computations required to simulate forward in time. The simulator returns, after computing the motions and/or forces of the articulated figure(s). The simulator "moves" forward in discreet steps in simulation time, returning control to the parser between each "step." Results can be saved to or loaded from files, via parser commands.

The dynamic simulator is covered in detail in the following section, **Dynamic Simulator**.

4.4 Graphics

The graphics sub-system in *corpus* generates the rendered image output. The output can be in the form of a "bitmap" file, using software rendering, or a screen image using special-purpose rendering hardware which greatly accelerates the process. Software rendering offers enhanced anti-aliasing and lighting effects, and is compatible with all hardware platforms which support standard **C**. Hardware rendering is rapid enough to offer real-time rendering (depending on the scene complexity). Hardware support for rendering in *corpus* is provided on Hewlett Packard and Silicon Graphics workstations.

The graphics system can also optionally output additional data with it's rendered image files. "Depth map" files contain a floating point value for each image pixel, which specifies how far (or deep) that image element was from the rendering viewpoint. "Alpha map" files contain opacity/masking values for each image pixel, and are especially useful for digital matting.

Commands are provided to load and manipulate graphical objects. A number of commands are used to transform (scale, rotate, translate, and shear) the objects in space. Additional commands affect the rendering properties of the objects, such as color, transparency, surface specularity, etc. A set of commands controls the environment lighting. Another controls the rendering camera parameters (eye position, lookat point, field of view, etc.). The rendering sub-system is activated to generate the current image with the parsing command "`render`."

There are a set of additional graphics functions which are not an intrinsic part of the major graphics sub-system. The functions are provided by smaller-scale sub-systems, which operate largely independently from the primary graphics system. However, these smaller systems cooperate with the graphics system, sharing framebuffer resources. One set of functions perform anti-aliased line drawing. Another renders anti-aliased fonts. Another set handles bitmap manipulation, including digital matting, and frame-buffer and file I/O. Although they are not strictly graphical, a video-deck control set is also included.

4.5 More *Corpus* Information

Corpus is not the sole creation of the author. A number of object libraries by others are linked with *corpus* to provide functionality. However, the overall structure of *corpus*, and its development are due to the author. The parser and dynamics simulator were written nearly entirely by the author. The primary graphics system (without its parser interface) is provided by *rendermatic*, a rendering package written primarily by David Chen. Please see the **Acknowledgments** for other contributors to *corpus*.

A list of the *corpus* commands is given in **Appendix A, Corpus Help**. More detail on the usage of *corpus* in provided in **Appendix B, Corpus Tutorial**. Examples of using *corpus* for dynamic simulation are presented in **Appendix C, Dynamics Verification**.

A simple example script for controlling *corpus* is shown in Script 1. This sort of instruction list could be typed in by the user directly to *corpus*, or, it could be stored in a script file which is read into corpus, using the "do" command. The image generated by the example script is shown in Figure 22.

An example session with *corpus* is shown in Script 2. Both the text inputs to and outputs from *corpus* are shown in the script, as a user would interact with the *corpus* parser.

Script 1: An example *corpus* script.
This script loads an object, transforms it, sets its properties, and renders an image.

```
# comment lines begins with '#'

# load the object from a file
get block from data/unit_cube

# transform the object
# use post-multiplication order
postmult
scale block 2 2 2
rotate block z 45
move block 10.2 2.5 0

# set object graphics properties
color block .5 .2 .2
shadeparam block .8 .4 30 .5
facet block
```

```
# set camera position, etc.
lookat block
eye 10 10 10
fov 30

# set the color of the "background"
backgroundcolor 1 1 1

# add a new light
lightmake light.2
lightpos light.2 100 200 100
lightcolor light.2 1 .9 .8
lightdimmer light.2 .7

# render the image
render
```

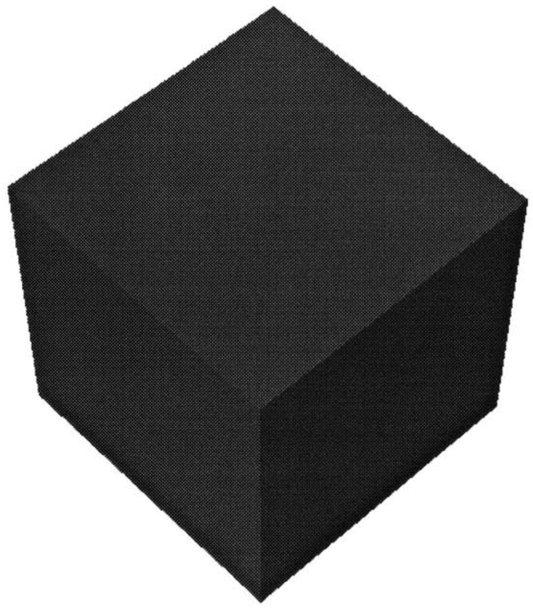


Figure 22: The output image generated by *corpus*, using Script 1.

Script 2: An example interactive session with *corpus*.
Both the inputs from the user, and the output from *corpus* are shown. The output is shown in italics, for illustration purposes.

```
corpus> get b from ../data/unit_cube
corpus> eye 3 2.5 1
corpus> lookat b
corpus> eye
eye: 1 2 1
corpus> lookat
lookat: 0 0 0
corpus> render
rendering...done
corpus> move b .1 .1 0
corpus> render
rendering...done
corpus> move b .2 0 .1
corpus> render
rendering...done
corpus> whereis b
b: .3 .1 .1
corpus> lookat b
corpus> render
rendering...done
corpus> quit
UNIX>
```

5 Dynamic Simulator

5.1 Introduction

The dynamic simulator is a core element of this research, as it forms the underlying basis for all movements. Featherstone's *Articulated Body Method* (ABM), as introduced in the- **Background** (2) and **Approach** (3) sections, has been implemented and extended as part of *corpus*. This section describes the equations which define the simulator system, provides information on the implementation of the simulator, and explains the ways in which the simulator system in used to generate and analyze physically-based motion.

The *corpus* dynamic simulator operates on rigid bodies, and articulated structures comprised of rigid bodies connected by joints. The articulated structures must be branching in nature, i.e. without closed kinematic loops. Loops can be approximated using closure spring forces, and the algorithm can be extended to handle loops exactly, but with a significant loss of efficiency. Contact between an articulated figure and the ground can create closed loops — when more than one part of an articulated figure touches the ground, it can be said that a closed loop is formed, with the ground itself forming part of the kinematic loop. These kinds of loops are not treated explicitly as closed loops in *corpus*, but are handled through contact forces, which simulate the supporting and friction forces at the contact between the figure and the ground.

Corpus simulates forward, inverse and hybrid dynamics on articulated figures. In addition, *corpus* provides first order (or "aristotelian") dynamic simulation, in which velocity is proportional to force. First order dynamics allows for rapid convergence to final resting states.

The simulation of articulated figure is performed in the context of applied forces, both internal, joint forces, and external, environmental forces. Internal forces are those forces generated "within" the figure, and are applied at the joints as torques or forces. These include elements such as joint dampers and springs. External forces comprise the forces applied to the figure from sources outside of its own body, from the environment. These include gravity and contact forces.

The central element of the simulator algorithm is the *Articulated Body Method* (ABM) for forward, inverse and hybrid dynamics as described by Featherstone, [Featherstone 1983; Featherstone 1987] and his original texts should be consulted concerning the derivation of the algorithm and for additional details. This section provides the final equations required for implementation of the algorithm, as well as the required background in spatial notation. The material here is presented from a somewhat different perspective on the issue, focusing more on a complete dynamic simulation environment, which should be complimentary to Featherstone's texts. In addition, this section expands on Featherstone's algorithms, and covers additional topics required for a complete simulation system, such as internal and external forces.

The dynamic simulator in *corpus* is a general purpose system, capable of simulating a wide variety of articulated structures, over a wide range of conditions. Different figures and simulations are constructed using different command sets or scripts in *corpus*. As described in the previous section, **The Program Corpus**, the dynamic simulator is linked to a parsing sub-system, which allows dynamic parameters to be set and queried, and provides the commands for constructing and controlling articulated figures and simulations.

The simulator in *corpus* was tested for correct implementation and numerical accuracy using a variety of test cases. These are discussed in **Appendix C, Dynamics Verification**. The tests in the **Dynamics Verification** appendix are also useful examples of how simulations are constructed using scripts in *corpus*.

Also, see the **List of Terms**, page 264, for reminders of the various terms used, and the **Glossary**, page 266, for unfamiliar terminology.

5.2 Spatial Notation

Before the dynamics algorithm is presented, we must cover "spatial notation," the mathematical notation in which the ABM is written. Spatial notation was developed by Featherstone as a way of combining the linear and angular components involved in rigid body motion. Spatial vectors are 6 dimensional quantities, which contain both the 3 dimensional linear, or translating, degrees of freedom (DOFs), and the 3 dimensional angular, or rotary DOFs. The equations of motion can be expressed in a more compact form using spatial algebra, than by using traditional 3-dimensional vector math.

Featherstone describes a spatial vector as "a 6-dimensional vector which can be used to represent the combined linear and angular components of the physical quantities involved in rigid-body dynamics." [Featherstone 1987]

Spatial quantities are denoted by a carat ("$\hat{}$") above them, as in $\hat{a}$. Spatial vectors are 6×1 column vectors, and spatial matrixes are 6×6 matrixes. Column vectors (such as spatial vectors) are commonly used in robotics research, whereas row vectors have historically been used in computer graphics literature and software implementations (so that vectors and matrixes must be transposed to convert from one system to another). It is important to keep this difference in mind when reviewing simulation and computer graphics publications. Spatial algebra is based on standard matrix arithmetic; however a different transpose operation is used (the "spatial transpose").

We will now discuss some different spatial quantities. Spatial velocity, $\hat{v}$, defines the 6 dimensional velocity of a body, from the point of reference of the coordinate frame origin. It is given as:

$$\hat{v} = [\omega^T v_O^T]^T \text{, or } \hat{v} = \begin{bmatrix} \omega \\ v_O \end{bmatrix} \text{, as in } \hat{v} = \begin{bmatrix} w_x \\ w_y \\ w_z \\ v_{Ox} \\ v_{Oy} \\ v_{Oz} \end{bmatrix} . \qquad \text{Eq. 4}$$

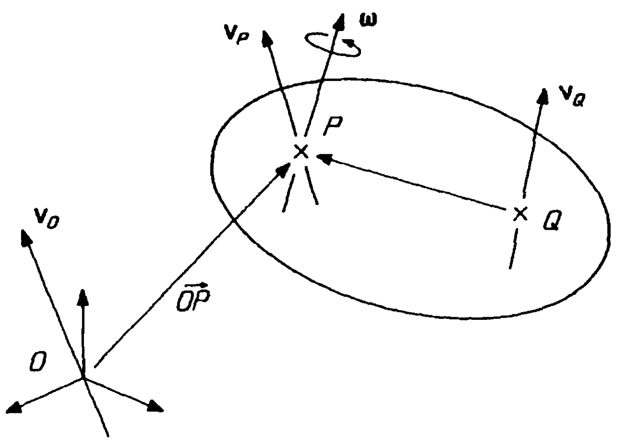


Figure 23: The velocity of a rigid body. From [Featherstone 1987].

It is composed of the angular velocity, ω, and the linear velocity of the rigid body at the origin of the coordinate frame, v_O. See Figure 23. If the linear velocity, v_P, of another point P within the body is known, it is transformed to the origin using the following rule:

$$v_O = v_P + \overrightarrow{OP} \times \omega \ , \qquad \text{Eq. 5}$$

where $\overrightarrow{OP}$ is the vector from the origin to the point P. There is a similar transformation for finding the linear velocity at any other point, as in:

$$v_Q = v_P + \overrightarrow{QP} \times \omega \ . \qquad \text{Eq. 6}$$

The angular velocity, ω, remains the same at all points.

The spatial acceleration, $\hat{a}$, of a rigid body is given as:

$$\hat{a} = \begin{bmatrix} \dot{\omega} \\ a_O \end{bmatrix} , \qquad \text{Eq. 7}$$

where a_O is the linear acceleration of the point in the rigid body which is instantaneously passing through the origin, and $\dot{\omega}$ is the angular acceleration. If the linear acceleration and velocity of the body is known at a point P, the linear acceleration at the origin is given as:

$$a_O = a_P + v_P \times \omega + \overrightarrow{OP} \times \dot{\omega} \ . \qquad \text{Eq. 8}$$

Spatial force, $\hat{f}$, is given as:

$$\hat{f} = \begin{bmatrix} f \\ \tau_O \end{bmatrix}, \qquad \text{Eq. 9}$$

where f is the linear component of the force, and τ_O is the torque at the origin. If the linear force is applied to the body at a point P, then the torque at the origin is computed as follows:

$$t_O = \tau_P + \overrightarrow{OP} \times f \ , \qquad \text{Eq. 10}$$

where τ_P is the torque, if any, at point P. The linear component of the force, f, does not change with respect to different locations.

Before introducing the next spatial quantity, the spatial inertia tensor, we will introduce a few operators. The cross operator takes a "standard" (non spatial) 3 dimensional vector and creates a 3×3 matrix, which "encodes" the multiplication products from a standard cross product operation. The operation is given as follows:

$$a \times \ = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \times \ = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} . \qquad \text{Eq. 11}$$

Thus, the operation $(a \times)\ b$ (matrix times vector) is equivalent to $a \times b$ (cross product of two vectors). This operator is also known as the anti-symmetric matrix, [Featherstone 1987] or the skew-symmetric matrix, which is often designated with a tilde ("~"), as in $\tilde{a}$, in other notations. [Armstrong 1985]

The spatial cross operator, ($\hat{a}\,\hat{\times}$), is the spatial algebra equivalent of a three dimensional cross operator, and it is defined as:

$$\hat{a}\,\hat{\times} \ = \begin{bmatrix} a \\ b \end{bmatrix} \hat{\times} \ = \begin{bmatrix} a\times & 0 \\ b\times & a\times \end{bmatrix} . \qquad \text{Eq. 12}$$

The spatial transpose operator (denoted by a superscript S, as in a^S) is used in place of a standard transpose operator in spatial algebra. For a spatial vector:

$$\hat{a}^S = \begin{bmatrix} a \\ b \end{bmatrix}^S = \begin{bmatrix} b^T & a^T \end{bmatrix} . \qquad \text{Eq. 13}$$

For a spatial matrix:

$$\hat{A}^S = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^S = \begin{bmatrix} D^T & B^T \\ C^T & A^T \end{bmatrix} . \qquad \text{Eq. 14}$$

The spatial rigid body inertia tensor, $\hat{I}$, can be constructed from 3×3 sub-components, as follows:

$$\hat{I} = \begin{bmatrix} H^T & M \\ I & H \end{bmatrix} , \qquad \text{Eq. 15}$$

where:

$$H = m\, \overrightarrow{OP} \times \quad , \qquad \text{Eq. 16}$$

where *m* is the scalar value of the rigid body's mass, and the center of mass is located at point *P*, with respect to the origin, *O*. The mass matrix, M, is given as:

$$M = m \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \qquad \text{Eq. 17}$$

The rotational inertia of the rigid body at the coordinate frame origin, *I*, is given as:

$$I = I^* + OP \times m\, \overrightarrow{PO} \times \quad , \qquad \text{Eq. 18}$$

where I^* is the 3×3 rigid body inertia tensor, as in:

$$I^* = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} . \qquad \text{Eq. 19}$$

[Wilhelms 1985; Winter 1990] The moments of inertia are defined as:

$$I_{xx} = \int (y^2 + z^2)\, dm \text{ , and etc.,} \qquad \text{Eq. 20}$$

where *dm* is the differential mass element. [Marion] The products of inertia are given as:

$$I_{xy} = \int xy\, dm \text{ , and etc.} \qquad \text{Eq. 21}$$

Spatial transformation matrixes convert spatial values from one coordinate frame to another. The matrix ${}_P\hat{X}_O$ is the spatial transform which transforms values from the coordinate system "*O*", to the coordinate system "*P*". This spatial matrix can be constructed from 3×3 sub-components:

$${}_P\hat{X}_O = \begin{bmatrix} E & 0 \\ 0 & E \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \overrightarrow{OP}\times^T & 1 \end{bmatrix} = \begin{bmatrix} E & 0 \\ E\,\overrightarrow{OP}\times^T & E \end{bmatrix}, \qquad \text{Eq. 22}$$

which corresponds to a shift of origin from *O* to *P*, followed by a rotation about the point P. The *E* matrix is a 3×3 rotation matrix. Note the order of multiplication where the rotation and translation spatial matrixes are concatenated. The inverse transformation, represented by ${}_O\hat{X}_P$, is the spatial transpose of ${}_P\hat{X}_O$ (i.e. ${}_O\hat{X}_P = {}_P\hat{X}_O^S$).

Multiple spatial transformation matrixes are combined as follows:

$${}_Q\hat{X}_O = {}_Q\hat{X}_P\, {}_P\hat{X}_O\ . \qquad \text{Eq. 23}$$

A spatial vector is transformed in the following manner:

$$\hat{a}_P = {}_P\hat{X}_O\, \hat{a}_O\ . \qquad \text{Eq. 24}$$

A spatial tensor is transformed with the following multiplications:

$$\hat{I}_P = {}_P\hat{X}_O\, \hat{I}_O\, {}_O\hat{X}_P\ . \qquad \text{Eq. 25}$$

5.3 Single Body Dynamics

We will now discuss the dynamics of a single rigid body, before proceeding to the algorithms for articulated body motion. In spatial notation, the equations of motion are quite compact. The equations of motion for a rigid body, free to move in space are given as:

$$\hat{f} = \hat{I}\,\hat{a} + \hat{p}^v\ , \qquad \text{Eq. 26}$$

and

$$\hat{p}^{v} = \hat{v} \hat{\times} \hat{I} \hat{v} \ , \qquad \text{Eq. 27}$$

where $\hat{f}$ is the spatial force applied to the body, $\hat{I}$ is the body's spatial inertia tensor, $\hat{a}$ is the spatial acceleration, $\hat{v}$ is the body's spatial velocity, and $\hat{p}^{v}$ is the bias force, which accounts for velocity-dependent spatial accelerations. The bias force is required due to coupling between rotation and linear velocity. Even in the absence of any applied force, a spatial acceleration can be generated when the body has a non-zero spatial velocity, and the bias force accounts for this. To paraphrase Featherstone, the bias force is equal to the force which must be applied to a body in order to give it zero spatial acceleration, in the absence of any other applied forces.

To solve the forward dynamics problem, we solve for the acceleration, as in:

$$\hat{a} = (\hat{I})^{-1} (\hat{f} - \hat{p}^{v}) \ . \qquad \text{Eq. 28}$$

To solve the inverse dynamics problem, we solve for the net applied force, from a specified acceleration, as in Eq. 26.

The dynamics computations give us an instantaneous solution to the problem. The core of the simulator is formed by these equations, and thus it provides an instantaneous solution. In order to move forward in time, we must integrate the time-dependent variables. Numerical integration uses discrete, instantaneous samples, such as those computed by the simulator, to approximate the actual integration.

For example, when we solve the forward dynamics problem, the body's acceleration is computed, based on the applied forces and current velocity, at a given instance in time (t_1). In order to determine the velocity at a future time (t_2) we need to integrate the acceleration over that period, as in:

$$\hat{v}_{t_2} = \hat{v}_{t_1} + \int_{t_1}^{t_2} \hat{a} \, dt \ . \qquad \text{Eq. 29}$$

Numerical integration techniques approximate the above, using discrete values of the acceleration. There are three integration techniques available in *corpus*: euler, fixed-step runge-kutta, and adaptive step-size runge-kutta. The euler algorithm is the simplest of the integrators, but it provides the basic concept of the numerical integrator. To update the value of the spatial velocity:

$$\hat{v}_{new} = \hat{v}_{old} + \hat{a}\ dt \ , \qquad \text{Eq. 30}$$

where *dt* is the amount of time over which to integrate (t_2 - t_1). We refer to *dt* as the *timestep*, or the amount of simulation time which passes between each simulation step.

The fixed-step runge-kutta algorithm takes four sub-steps for each timestep. [Press] In other words, the dynamics equations are evaluated four times, at different time points between t_1 and t_2, each point being a different sub-step. The results from the sub-steps are combined, using weighting factors, in order to obtain a 4th order solution.

The adaptive runge-kutta algorithm takes six sub-steps, and uses two different sets of weighting coefficients to combine the results into both a 4th order and 5th order solution. [Forsythe] The difference between the two results is used as an error measure. If the error is greater than the specified error tolerance, the algorithm discards the results, and sets a smaller step size, based on the error. The previous state is restored, and the integrator begins again with the smaller step size. This allows the integrator to return a result with a certain accuracy, regardless of the complexity, or changing conditions within the simulation.

For most simulations, the best results are obtained with the adaptive integrator, because it is designed to adjust itself to the “stiffness” of the problem. At times it will take a few large steps, at other times it will take many small steps, adapting to variations in the system which mandate a change in step-size in order to maintain a certain level of accuracy. The non-adaptive algorithms are appropriate if the computation must be completed at regular, or very short intervals. However, adaptive algorithms can be instructed to halt further sub-division after a specified limit. In general, the 4th order, non-adaptive integrator will compute a more accurate solution than the euler integrator, for the same amount of computation time. The euler integrator’s primary usefulness lies in its simplicity, which can be of value during the debugging process.

Comparisons of the different integrators are provided in **Appendix C, Dynamics Verification**, along with some examples of single body simulations.

5.4 Articulated Body Forward Dynamics

Articulated figures in *corpus* are collections of rigid bodies, connected together by idealized joints. The joints constrain the relative motion of the two connected bodies, allowing the bodies to rotate or translate with respect to each other. Articulated figures in *corpus* are defined as branching structures, without closed kinematic loops.

An articulated figure is composed of n bodies. Different bodies within an articulated figure are referred to by an index, i, which ranges from 1 to n. Quantities which are associated with a given body carry its index as a subscript after the term. For example, the spatial velocity for body i is $\hat{v}_i$.

One body within the articulated, branching structure, is designated as the root or base body, and it is defined as body 1. The root is proximal to all other bodies. Any body could be chosen as root, however, it is often logical to choose a central object as the root. "Leaf" bodies are bodies which lie at the distal end of a kinematic chain.

The joints between connected bodies are defined by the spatial joint axis, $\hat{s}_i$. In *corpus*, every joint has one degree of freedom (DOF). The equations of the ABM can be extended to handle multiple DOF joints, [Featherstone 1987] and *corpus* could be extended to handle these cases as well. The joints can be rotary (1 degree of freedom "hinge" joints), or translating ("prismatic" or "sliding" joints). Multiple degree of freedom joints are created in *corpus* by concatenating bodies, with their joint axes aligned in different directions.

A rotary joint axis is given as follows:

$$\hat{s} = \begin{bmatrix} x & y & z & 0 & 0 & 0 \end{bmatrix}^T , \qquad \text{Eq. 31}$$

where {x y z} is the normalized vector which defines the axis about which rotations occur. A translating joint axis is defined as:

$$\hat{s} = \begin{bmatrix} 0 & 0 & 0 & x & y & z \end{bmatrix}^T , \qquad \text{Eq. 32}$$

where {x y z} is the normalized vector which defines the axis along which translations are allowed. As defined above, these joint axis are positioned to lie at the coordinate frame origin. The spatial joint axis can be transformed using a spatial transformation matrix.

The joint position of body i is given by the scalar q_i, in meters or radians. Joint velocity is $\dot{q}_i$, and joint acceleration is $\ddot{q}_i$. The joint force of body i is given as Q_i.

The joint constraint maintains that there can be no transmission of force through the joint, from one body to another along the direction of the joint, when the joint is un-powered:

$$\hat{s}_i^S \hat{f}_i = 0 \,, \qquad \text{Eq. 33}$$

where $\hat{f}_i$,is the net force transmitted to body i from its parent, body i-1, through joint $\hat{s}_i$ (recall that the superscript S denotes a spatial transpose). If there is an active force at the joint, then it is included in the net force applied from one body to another through their connecting joint. The component of the net force on the joint axis is:

$$Q_i = \hat{s}_i^S \hat{f}_i \,, \qquad \text{Eq. 34}$$

Due to the hierarchical structure of the articulated figure and the constraints of the joints, many relationships exist between parent and child bodies. The spatial velocity of body i is the sum of its parent body's spatial velocity and the spatial velocity created by its own joint velocity:

$$\hat{v}_i = \hat{v}_{i-1} + \hat{s}_i \dot{q}_i \,. \qquad \text{Eq. 35}$$

The acceleration of body i is the sum of its parent's spatial acceleration, and the spatial acceleration generated by the joint velocity and acceleration:

$$\hat{a}_i = \hat{a}_{i-1} + \hat{v}_i \hat{\times} \hat{s}_i \dot{q}_i + \hat{s}_i \ddot{q}_i \,. \qquad \text{Eq. 36}$$

The central equation of motion in the ABM is:

$$\hat{f}_i = \hat{I}_i^A \hat{a}_i + \hat{p}_i \qquad \text{Eq. 37}$$

where $\hat{f}_i$ is the net spatial force applied to body i, $\hat{I}_i^A$ is the *articulated body inertia*, $\hat{a}_i$ is the spatial acceleration, and $\hat{p}_i$ is the bias force, all of the ith body in the articulated figure. The articulated body inertia allows us to establish a linear relationship between the applied force and the acceleration of a rigid body, even though the motion of the body is constrained by its connection to other bodies through joints. The bias force incorporates the velocity dependent bias force, $\hat{p}^v{}_i$, and forces transmitted through the joints.

The recursive relationship for the articulated body inertia is derived by Featherstone, and determined to be:

$$\hat{I}_i^A = \hat{I}_i + \hat{I}_{i+1}^A - \frac{\hat{I}_{i+1}^A \hat{s}_{i+1} \hat{s}_{i+1}^S \hat{I}_{i+1}^A}{\hat{s}_{i+1}^S \hat{I}_{i+1}^A \hat{s}_{i+1}} , \qquad \text{Eq. 38}$$

and

$$\hat{I}_n^A = \hat{I}_n , \qquad \text{Eq. 39}$$

where n = the index of any body which lies at the end of an articulated chain (a leaf link).

The recursive formulation for the bias force is given as:

$$\hat{p}_i = p_i^v - \hat{f}_i^{ext} + \hat{p}_{i+1} + \hat{I}_{i+1}^A \hat{v}_{i+1} \hat{\times} \hat{s}_{i+1} \dot{q}_{i+1} \qquad \text{Eq. 40}$$

$$+ \frac{Q_{i+1} - \hat{s}_{i+1}^S \hat{I}_{i+1}^A \hat{v}_{i+1} \hat{\times} \hat{s}_{i+1} \dot{q}_{i+1} - \hat{s}_{i+1}^S \hat{p}_{i+1}}{\hat{s}_{i+1}^S \hat{I}_i^A \hat{s}_{i+1}} \hat{I}_{i+1}^A \hat{s}_{i+1} ,$$

$$p_i^v = \hat{v}_i \hat{\times} \hat{I}_i \hat{v}_i , \qquad \text{Eq. 41}$$

and

$$\hat{p}_n = \hat{p}_n^v - \hat{f}_n^{ext} . \qquad \text{Eq. 42}$$

I have included the external force, $\hat{f}_i^{ext}$, explicitly in the bias force, in a slightly different form than Featherstone.

Starting with the outermost, or "leaf," bodies in the articulated figure, the articulated body inertias and bias forces are computed inwards to the base body. The root acceleration is then computed, using:

$$\hat{a}_1 = (\hat{I}_1)^{-1} (-\hat{p}_1^v) . \qquad \text{Eq. 43}$$

The joint accelerations are then computed outwards, from the base to the leaf bodies, as follows:

$$\ddot{q}_i = \frac{Q_i - \hat{s}_i^S \hat{I}_i^A (\hat{v}_i \hat{\times} \hat{s}_i \dot{q}_i) - \hat{s}_i^S \hat{p}_i - \hat{s}_i^S \hat{I}_i^A \hat{a}_{i-1}}{\hat{s}_i^S \hat{I}_i^A \hat{s}_i} . \qquad \text{Eq. 44}$$

The above equations were derived in a uniform, unchanging coordinate frame. However, there are advantages to using body-local coordinate frames, in which each body has its own local coordinate frame, which travels along with the body. Using body-local coordinates, the COM of the body, and thus the spatial inertia tensor, are unchanging. Integration in a local frame is more efficient, as well.

When using body-local coordinate frames, transformation matrixes are used to translate the values from one body into the coordinate system of a second body, so that they can be combined in the equations of motion. The matrix ${}_i\hat{X}_j$ is the spatial transform which translates values from the coordinate system of body *j*, to the coordinate system of body *i*.

The following is the set of equations for the ABM, written using local coordinate frames:

$$\hat{v}_i = {}_i\hat{X}_{i-1}\,\hat{v}_{i-1} + \hat{s}_i\,\dot{q}_i\ , \qquad \text{Eq. 45}$$

$$\hat{a}_i = {}_i\hat{X}_{i-1}\,\hat{a}_{i-1} + \hat{v}_i \hat{\times} \hat{s}_i\,\dot{q}_i + \hat{s}_i\,\ddot{q}_i\ , \qquad \text{Eq. 46}$$

$$p_i^v = \hat{v}_i \hat{\times} \hat{I}_i\,\hat{v}_i\ , \qquad \text{Eq. 47}$$

$$\hat{I}_i^A = \hat{I}_i + {}_i\hat{X}_{i+1}\hat{I}_{i+1}^A - \frac{\hat{I}_{i+1}^A\hat{s}_{i+1}\hat{s}_{i+1}^S\hat{I}_{i+1}^A}{\hat{s}_{i+1}^S\hat{I}_{i+1}^A\hat{s}_{i+1}}\,{}_{i+1}\hat{X}_i\ , \qquad \text{Eq. 48}$$

$$\hat{I}_n^A = \hat{I}_n\ , \qquad \text{Eq. 49}$$

$$\hat{p}_i = p_i^v - \hat{f}_i^{ext} + {}_i\hat{X}_{i+1}(\hat{p}_{i+1} + \hat{I}_{i+1}^A\hat{v}_{i+1} \hat{\times} \hat{s}_{i+1}\,\dot{q}_{i+1} \qquad \text{Eq. 50}$$

$$+ \frac{Q_{i+1} - \hat{s}_{i+1}^S\,\hat{I}_{i+1}^A\hat{v}_{i+1} \hat{\times} \hat{s}_{i+1}\,\dot{q}_{i+1} - \hat{s}_{i+1}^S\hat{p}_{i+1}}{\hat{s}_{i+1}^S\,\hat{I}_{i+1}^A\hat{s}_{i+1}}\,\hat{I}_{i+1}^A\,\hat{s}_{i+1})\ ,$$

$$\hat{p}_n = \hat{p}_n^v - \hat{f}_n^{ext}\ , \qquad \text{Eq. 51}$$

and

$$\ddot{q}_i = \frac{Q_i - \hat{s}_i^S\,\hat{I}_i^A\,(\hat{v}_i \hat{\times} \hat{s}_i\,\dot{q}_i) - \hat{s}_i^S\,\hat{p}_i - \hat{s}_i^S\,\hat{I}_i^A\,{}_i\hat{X}_{i-1}\,\hat{a}_{i-1}}{\hat{s}_i^S\,\hat{I}_i^A\hat{s}_i} \qquad \text{Eq. 52}$$

Note that there are common sub-expressions in the global and local forms of the ABM equations (such as $\hat{v}_i \hat{\times} \hat{s}_i \dot{q}_i$). These can be extracted, computed first, and substituted into the equations to increase computational efficiency. [Featherstone 1987] This is the way that *corpus* is implemented.

5.5 Hybrid Dynamics

The forward dynamics computations can be augmented to perform inverse and hybrid dynamics. [Featherstone 1987] Inverse dynamics is used to compute the force (unknown) required to accomplish a specified motion (known). Hybrid dynamics allows a mix of forward and inverse dynamics; at every joint, either the applied force or the acceleration is specified, and the unknown value is then computed during the simulation recursion. Using hybrid dynamics, part of a figure can be "driven" through a kinematic trajectory, while the remainder of the body is dynamically simulated, so that it responds to the applied forces, as well as the motions of the kinematically controlled joints. Hybrid dynamics are implemented in *corpus*, and will be used in this thesis to perform inverse dynamics on the biped, as part of the control system computations to automatically derive the control parameters necessary for standing and other postures.

When the joint between body *i* and *i*+1 is kinematically controlled (joint $\hat{s}_{i+1}$), the following relationship is used (see [Featherstone 1987]):

$$\hat{f}_i = \hat{I}_i \hat{a}_i + \hat{p}_i^v + \hat{I}_{i+1}^A (\hat{a}_i + \hat{v}_{i+1} \hat{\times} \hat{s}_{i+1} \dot{q}_{i+1} + \hat{s}_{i+1} \ddot{q}_{i+1}) + \hat{p}_{i+1} \,, \qquad \text{Eq. 53}$$

where $\hat{v}_{i+1}$ is the spatial velocity, $\hat{s}_{i+1}$ is the spatial vector which represents the joint axis, $\dot{q}_{i+1}$ is the scalar joint velocity, and $\ddot{q}_{i+1}$ is the joint acceleration, all of body *i*+1, the distal "child" of body *i*.

From Eq. 53, we can extract the recursive relationships for $\hat{I}_i^A$ and $\hat{p}_i$:

$$\hat{I}_i^A = \hat{I}_i + \hat{I}_{i+1}^A \,, \qquad \text{Eq. 54}$$

$$\hat{p}_i = \hat{p}_i^v - \hat{f}_i^{ext} + \hat{p}_{i+1} + \hat{I}_{i+1}^A (\hat{v}_{i+1} \hat{\times} \hat{s}_{i+1} \dot{q}_{i+1} + \hat{s}_{i+1} \ddot{q}_{i+1}) \,. \qquad \text{Eq. 55}$$

(This is in slightly different form than Featherstone, to incorporate the external force). It can be seen from Eq. 54 that articulated body inertias essentially sum when the joint between two bodies is kinematically controlled.

When joint *i* is kinematically controlled, the joint force can computed as:

$$Q_i = \hat{s}_i^S (\hat{I}_i^A \hat{a}_i + \hat{p}_i) \ . \qquad \text{Eq. 56}$$

These inverse calculations (Eq. 54 and Eq. 55) were extended to local frames, by the author:

$$\hat{I}_i^A = \hat{I}_i + {}_i\hat{X}_{i+1} \ \hat{I}_{i+1}^A \ {}_{i+1}\hat{X}_i \ , \qquad \text{Eq. 57}$$

$$\hat{p}_i = \hat{p}_i^v - \hat{f}_i^{ext} + {}_i\hat{X}_{i+1} \ \hat{p}_{i+1} \qquad \text{Eq. 58}$$

$$+ {}_i\hat{X}_{i+1} (\ \hat{I}_{i+1}^A (\hat{v}_{i+1} \hat{\times} \hat{s}_{i+1} \ \dot{q}_{i+1} + \hat{s}_{i+1} \ \ddot{q}_{i+1})) \ {}_{i+1}\hat{X}_i \ .$$

Inverse dynamics can be calculated for the root (most proximal) body, as well, using Eq. 37, so that the overall 6 dimensional body motion is kinematically controlled. In this case, however, an external force, $\hat{f}_1$, not a joint force, must be applied to the root body in order to achieve the specified acceleration, $\hat{a}_1$. Unless there is a realistic source for their generation, arbitrary constraint forces should not be used. Using forward dynamics methods, it is a *control* problem to generate the desired body motion, using applied joint forces. In some cases, constraining the root motion is appropriate. For example, when the base is fixed, the root acceleration can trivially be set to zero.

The joint forces calculated by the inverse dynamics are very susceptible to the stability of the dynamic system being simulated. Some examples of sources of numerical instability are given below, in the discussion of contact forces. Because the joint forces computed by the inverse dynamics can vary widely, even over a short period of time, an average is made of the joint force. In *corpus* the numerical integrator, which is typically used to update velocity and position from acceleration, was augmented to compute an average joint force, over a range of time, with increased numerical stability, as in:

$$Q_{ave} = \frac{\int_{t_1}^{t_2} Q \, dt}{t_2 - t_1} \ . \qquad \text{Eq. 59}$$

5.6 First Order Dynamics

First order dynamics is an implementation of Aristotle's theory of dynamics, in which objects have a "preferred rest." In other words, unless a force is applied to an object, it will not move; the instantaneous velocity is proportional to the applied force. First order dynamics have been used in interactive computer graphics systems, in order to ease certain kinds of interactions. [Witkin 1990]

First order dynamics are very useful for quickly simulating the final rest state of a second order dynamic system. A first order simulation behaves somewhat like a damped, very low mass system, since the system has no momentum (using familiar terms from second-order dynamics). For example, a three-link pendulum, falling under the influence of gravity, is shown in Figure 24. When first order dynamics are used, the pendulum smoothly "falls" downward, to its fully vertical, extended position, without any overshoot or oscillations. But when second order dynamics are used, the pendulum overshoots its final equilibrium position, and oscillates back and forth. Depending on the damping factor, the pendulum may continue to move for a very long time. The final resting state of the pendulum is quite obvious. Other systems may be significantly more complex, however, with non-obvious final resting states. For example, a postural system may combine internal and external forces which must be balanced out.

The equations of motion for aristotelian dynamics are a straightforward extension to Featherstone's methods. The equations were derived by the author, and implemented in *corpus*.

The net applied force is proportional to body's velocity:

$$\hat{f}_i = \hat{I}_i^A v_i + \hat{p}_i \ . \qquad \text{Eq. 60}$$

Solving the forward dynamics problem, for the spatial velocity:

$$v_i = (\hat{I}_i^A)^{-1} (\hat{f}_i - \hat{p}_i) \ . \qquad \text{Eq. 61}$$

The articulated body inertia computation is the same as before:

$$\hat{I}_i^A = \hat{I}_i + \hat{I}_{i+1}^A - \frac{\hat{I}_{i+1}^A \hat{s}_{i+1} \hat{s}_{i+1}^S \hat{I}_{i+1}^A}{\hat{s}_{i+1}^S \hat{I}_{i+1}^A \hat{s}_{i+1}} \ , \qquad \text{Eq. 62}$$

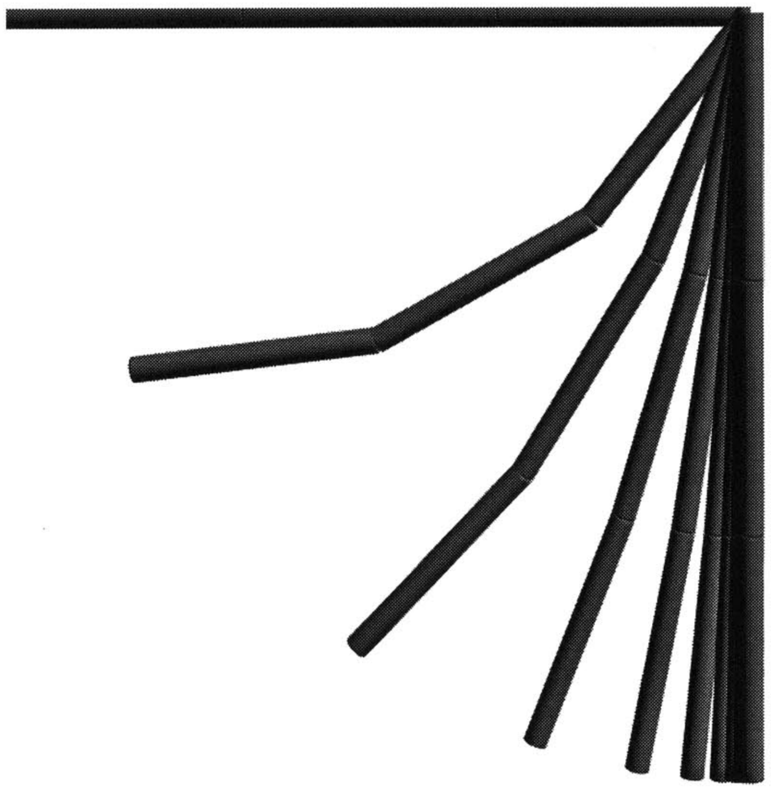
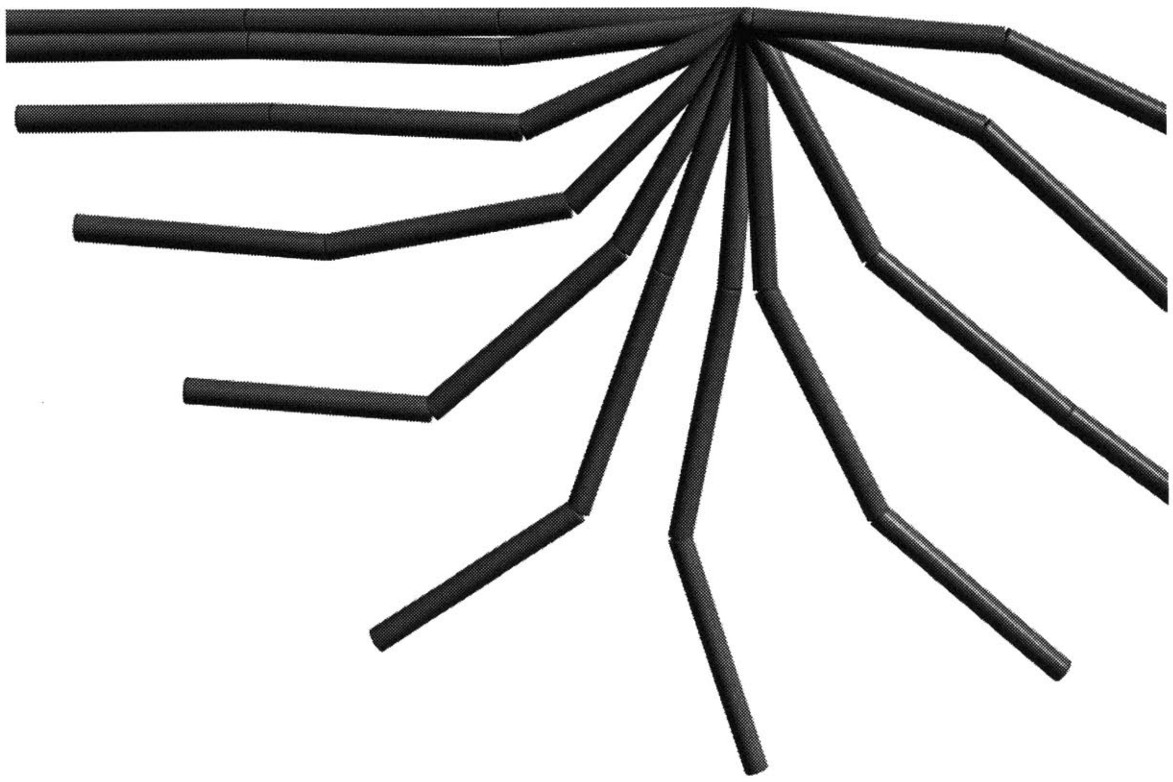


Figure 24: First order simulation of a 3 link pendulum (upper) compared to a second order simulation (below).

In both examples, the pendulum moves from left to right over time.

Using first order dynamics, the system has no momentum. The force of gravity applied to the links creates a 'downward' velocity at the joints. As soon as the pendulum is extended straight down, the force of gravity create no torques at the joints, and thus, there is no resultant velocity.



and

$$\hat{I}_n^A = \hat{I}_n \,. \qquad \text{Eq. 63}$$

The bias force now has no velocity-dependent terms:

$$\hat{p}_i = -\hat{f}_i + \hat{p}_{i+1} + \frac{Q_{i+1} - \hat{s}_{i+1}^S \hat{p}_{i+1}}{\hat{s}_{i+1}^S \hat{I}_i^A \hat{s}_{i+1}} \hat{I}_{i+1}^A \hat{s}_{i+1} \,, \qquad \text{Eq. 64}$$

and

$$\hat{p}_n = -\hat{f}_n \,. \qquad \text{Eq. 65}$$

The joint velocity is computed outwards, from the root body to the leaf bodies, using the following:

$$\dot{q}_i = \frac{Q_i - \hat{s}_i^S \hat{p}_i - \hat{s}_i^S \hat{I}_i^A \hat{v}_{i-1}}{\hat{s}_i^S \hat{I}_i^A \hat{s}_i} \,. \qquad \text{Eq. 66}$$

Using body local coordinate frames:

$$\hat{I}_i^A = \hat{I}_i + {}_i\hat{X}_{i+1} \hat{I}_{i+1}^A - \frac{\hat{I}_{i+1}^A \hat{s}_{i+1} \hat{s}_{i+1}^S \hat{I}_{i+1}^A}{\hat{s}_{i+1}^S \hat{I}_{i+1}^A \hat{s}_{i+1}} \; {}_{i+1}\hat{X}_i \,, \qquad \text{Eq. 67}$$

$$\hat{I}_n^A = \hat{I}_n \,, \qquad \text{Eq. 68}$$

$$\hat{p}_i = -\hat{f}_i + {}_i\hat{X}_{i+1} \left(\hat{p}_{i+1} + \frac{Q_{i+1} - \hat{s}_{i+1}^S \hat{p}_{i+1}}{\hat{s}_{i+1}^S \hat{I}_i^A \hat{s}_{i+1}} \hat{I}_{i+1}^A \hat{s}_{i+1} \right) , \qquad \text{Eq. 69}$$

$$\hat{p}_n = -\hat{f}_n \,, \qquad \text{Eq. 70}$$

and

$$\dot{q}_i = \frac{Q_i - \hat{s}_i^S \hat{p}_i - \hat{s}_i^S \hat{I}_i^A {}_i\hat{X}_{i-1} \hat{v}_{i-1}}{\hat{s}_i^S \hat{I}_i^A \hat{s}_i} \,. \qquad \text{Eq. 71}$$

As an implementation note, it is not necessary to include a separate set of computations, as above, to implement first order dynamics in a second order dynamics simulator. Rather, we can simply zero all of the velocities in the system as we enter the dynamics computations, and solve for the velocity, rather than acceleration. This eliminates all velocity-dependent terms automatically. However, this is not as efficient, since the zero accelerations are still integrated, and the velocity-dependent terms are still calculated (to be zero).

Velocity-dependent damping should not be used when simulating first-order dynamics, since the system would directly oppose its own motion. Velocity-dependent dampers are automatically eliminated from the system, if the velocities are set to zero before the simulation computations are initiated.

5.7 External Forces

External forces may be applied to any body in an articulated chain, in order to influence its motion, or simulate an environmental effect. These external forces are "environmental forces," because the forces are applied to the articulated figure from influences outside of itself, in the environment. These sources include: gravity, contact and collision forces, attachment forces, etc.

Gravity

Gravity is easily incorporated into the simulation system. The "downward" force due to gravity, $m\ g$, is applied to the center of mass (COM) of every rigid body in the simulation (including each body in an articulated figure). The gravitational constant, g, is a three dimensional vector, which specifies the strength and direction of gravity, and it is usually set to the normal value in the MKS system: [0, 0, -9.81] m/sec^2. The "Z" axis defines the vertical in the default *corpus* environment, with positive values representing the "up" direction. The g parameter can be set to any value (or direction) in the *corpus* parser. The gravitational force has been verified to induce the correct acceleration in several different simulation tests. (See **Appendix C**).

Ground Reaction Forces

The ground reaction forces, or, more generally, collision and contact forces, have two main components: normal and tangential forces. On "level" ground, these elements can be considered the vertical and horizontal force components. We will discuss the simple case of level ground forces. Contact (or support) and collision (or impact) between the articulated body and the ground are handled uniformly, by applying reaction forces at the point of contact, based on a spring and damper model.

The vertical reaction force is applied when the 3 dimensional geometric model of the body passes under the level of the ground. The ground model default in *corpus* is defined by the Z=0 plane. At all points (vertexes) of ground penetration a force is applied, dependent on the vertical penetration of the point, and the velocity of the point:

$$f_v = \begin{cases} \alpha_v (e^{-\beta_v x_v} - 1) - k_v x_v - b_v \dot{x}_v & \textbf{if } x_v < 0 \\ 0 & \textbf{if } x_v > 0 \end{cases}, \qquad \text{Eq. 72}$$

where f_v is the vertical reaction force, α_v and β_v are the linear and exponential spring constants for the exponential collision spring, k_v is the linear spring constant, and b_v is the collision damping constant. Damping is used in the above equation to dissipate energy in the collision; the percentage of kinetic energy lost depends on the spring constants as well as the damping constant. Alternately, the loss of energy can be directly parametrized by the *coefficient of restitution*, as in:

$$\dot{x}_{out} = e|\dot{x}_{in}| , \qquad \text{Eq. 73}$$

in which $\dot{x}_{in}$ is the velocity of the point of contact before the collision, $\dot{x}_{out}$ is the velocity after the collision, and e is the coefficient of restitution. A value of 1.0 for e would represent a perfectly elastic collision, with no loss of energy. The coefficient can be used to directly control the elasticity of a collision by modulating the vertical reaction force, as in:

$$f_v = \begin{cases} f_v & \textbf{if } \dot{x}_v < 0 \\ e f_v & \textbf{if } \dot{x}_v > 0 \end{cases}. \qquad \text{Eq. 74}$$

This technique was introduced to the computer graphics community by Moore and Wilhelms, [Moore] and was used by the author in previous work, with good results. The benefit of using the coefficient is that energy loss during collision can be directly specified, without matching damper constants to other simulation parameters (particularly, the ground spring stiffness constants). However, because the reaction force function is no longer continuous, a numerical instability is introduced, which creates a "microscopic" vertical jitter, especially during continuous contact (as opposed to collision). The magnitude of the jitter is dependent on the integrator time step size (and thus, on the adaptive integrator error tolerance). Numerical integrators which are dependent on the state history (such as predictorcorrectors) would have to initialized when such discontinuities occur (including the change between contact and non-contact) This greatly reduces the efficiency of such integrators.

The stability problems encountered using the coefficient of restitution (as in Eq. 73) interferes with the stable calculation of the joint force during a hybrid or inverse simulation. The instability also slows the simulation speed, because the integrator must take more substeps in order to calculate an accurate result. Because of this, the use of the coefficient of restitution was abandoned, in favor of the continuous vertical damper function, included in Eq. 72.

A similar vertical reaction force equation is used by Manko as a model of foot-soil interaction, between a robot and the ground. His equation is presented here, in a slightly different form:

$$f_v = \begin{cases} \alpha_v \left(e^{-(k_o - k_f) x_v} - k_f x_v - 1\right) & \textbf{if } x_v < 0 \\ 0 & \textbf{if } x_v > 0 \end{cases}, \qquad \text{Eq. 75}$$

where k_o and k_f are slope parameters. [Manko] Robot simulations using these functions have been experimentally confirmed using real robots.

Several different approaches have been used in *corpus* to model the tangential contact forces, which are due to friction. The friction models function primarily as velocity-dependent dampers to oppose sliding at the points of contact. Different models were added in order to reduce stability problems, similar to those encountered using the coefficient of restitution.

Early work by the author used a model of Coulombic friction in which a horizontal force is applied at the point of contact, in a direction to oppose the velocity, proportional to the vertical reaction force, as in:

$$f_h = -\gamma f_v \frac{\dot{x}_h}{|\dot{x}_h|}, \qquad \text{Eq. 76}$$

where f_h, is the horizontal friction force, γ is the coefficient of friction, and $\dot{x}_h$ is the horizontal velocity of the point of contact. This model was introduced to the graphics community by Wilhelms and Barsky. [Wilhelms 1985] The force response of this model is shown in

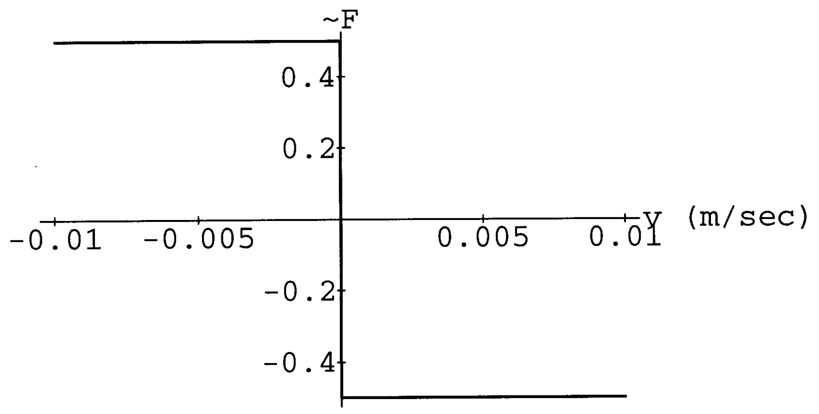
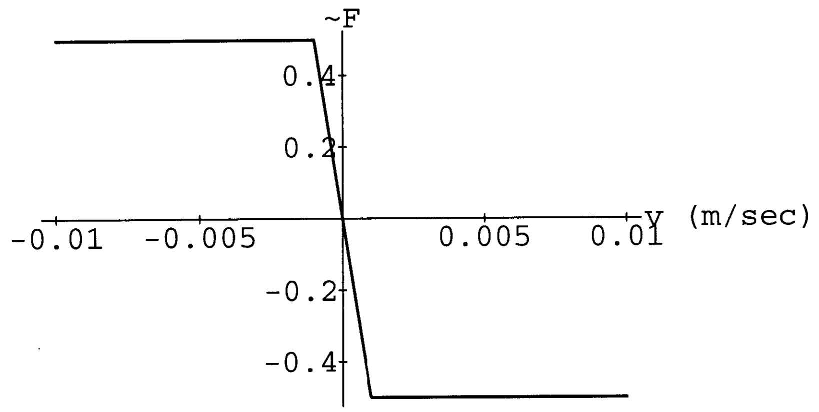


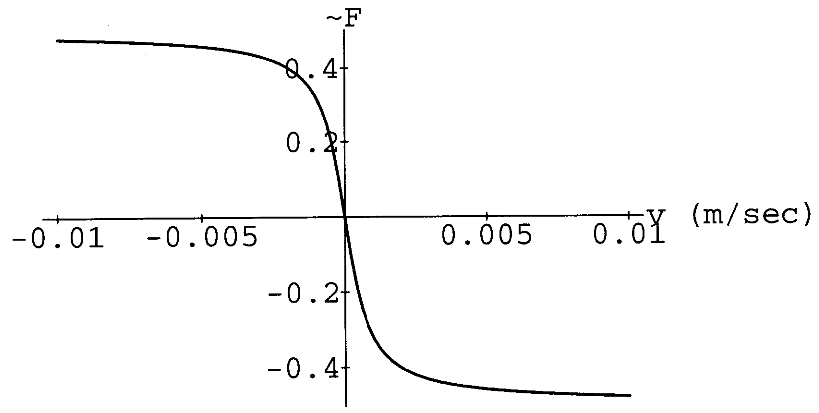
Figure 25: Force response of the friction functions.

top: Force is constantly applied, proportional to the normal force, as in Eq. 76.

The three plots are in terms of velocity (of the point of contact) and normalized force (~F = f_h/f_v), with a coefficient of friction, γ, of 0.5.



middle: The force response of the damper model of friction, with a clamped maximum/minimum, as in Eq. 77. The damping constant, b_h, is equal to 500 (N sec/m).



bottom: The force response of the arctangent friction function, as in Eq. 78. The damping constant, b_h, is equal to 500 (N sec/m).

Figure 25. This friction model has been used with success in previous research. However, this model creates instabilities due to the discontinuity in the friction force, when the velocity reverses direction. This disrupts the analysis of joint forces during inverse or hybrid simulation, and can slow simulation speed by creating a "stiffer" system.

A smoother friction function was implemented, using a clamped linear damping force:

$$f_h = \begin{cases} \gamma f_v & \textbf{if } b_h \dot{x}_h > \gamma f_v \\ -b_h \dot{x}_h & \textbf{if } -\gamma f_v \geq b_h \dot{x}_h > \gamma f_v \\ -\gamma f_v & \textbf{if } b_h \dot{x}_h < -\gamma f_v \end{cases} , \qquad \text{Eq. 77}$$

where b_h is the horizontal, "friction" damping constant. This increased the stability of the calculated joint forces, and decreased the number of simulation sub-steps required to integrate over a given time step, for a given error tolerance (decreased the stiffness of the system). The force response of this function is shown in the middle plot of Figure 25.

A similar, but smoother, more continuous friction function is used in simulations by Bogert, *et al* [Bogert]:

$$f_h = -\frac{2\gamma f_v}{\pi} \arctan\left(\frac{b_h \pi}{2\lambda f_v} \dot{x}_v\right) . \qquad \text{Eq. 78}$$

This is slightly more complex to compute, but it could lead to lower stiffness in the integrator, saving simulation steps, because the force response has a higher degree of continuity. This function has not, as yet, been implemented in *corpus*, however. The arctan friction function is shown in the lower plot of Figure 25.

Corpus also supports collision forces between articulated bodies and an uneven ground defined by a triangularized height array, similar to that used for uneven terrain in [Zeltzer 1982]. The same normal and tangential friction forces are used, but the direction of the normal force is no longer necessarily vertical, but is rather in the direction of the normal of the polygon which is penetrated by the body's geometry.

Collision forces between different bodies in *corpus* can also be simulated, using the same normal and tangential force models. Forces are applied equally and oppositely to the bodies involved. The normal of the penetrated polygon defines the normal and tangential directions.

For non-articulated, single bodies in *corpus*, a non-spring model can be used to simulate collisions. Using *collision analysis*, the momentum exchange of the colliding bodies is computed, such that their velocities are directly modified based on elastic impact dynamics in a single step. This is typically much more efficient than modifying the velocity of colliding bodies through the influence of acceleration, over a simulated interval of time. However, collision analysis responds poorly for situations involving continuous contact and support. This kind of collision model is discussed in [Moore] and [Hahn].

Other External Forces

Other environmental forces can be simulated in *corpus*. *Attachment forces* are linear and exponential springs, with dampers, which can be used to "attach" two bodies together. A point is specified on each body, and the distance between those two points is used as the feedback parameter for the spring. Equal and opposite linear forces are applied by the attachment spring to each body, at the attachment points. Muscle models which use linear attachment points can be simulated using these types of models. Attachment forces are considered external forces when the two bodies that they connect do not lie within the same articulated figure. If the attachment forces connect two bodies within the same articulated figure (as a muscle model would), the attachment forces are considered to be internal forces.

Terrestrial gravity, on a planetary surface, produces a constant acceleration. *Corpus* also supports *gravitational attraction* forces in which each body exerts a force on every other body, proportional to the two masses, as in:

$$F_g = G\frac{m\,M}{r^2}, \qquad \text{Eq. 79}$$

where m and M are the masses of the two bodies, G is the universal gravitation constant (6.673×10^{-11} N m^2/kg^2), and r is the distance between the two bodies' COMs. This can be used to simulate astronomical motions, such as orbital paths.

In addition, *corpus* commands and scripts can be used to set arbitrary applied forces to any body. In this manner, external programs can be used to provide force functions, and forces can be scripted.

5.8 Actuator Model and Joint Forces

Internal forces are applied within an articulated figure. These forces are related to the biomechanical model of the figure, since internal forces in real animals and people are generated by the mechanical properties of muscles and other tissues. Simulations in *corpus* typically use internal *joint* forces to control motion. Joint forces are forces which are applied directly at a joint location, in the direction of motion allowed by that joint. The force is applied to the two bodies that the joint connects, equally and oppositely. *Corpus* also allows for motion control using linear forces which are applied as internal forces to articulated bodies (via the attachment forces described above). The use of joint forces in the human figure model is discussed in the section **Biomechanical Model** (6).

There are several sources of joint forces available in *corpus*. Actuators, which deliver forces for active joint control, are based on spring models. Dampers create forces to oppose motion, and model the viscous properties of joints. Joint limit forces are created using springs and dampers which become active when the limb passes beyond a specified range. Joint limit forces model the ways in which the joints of real mechanisms, animals and people resist and stop movements beyond their ranges of motion.

For a body i, the total joint force, Q_i, is given as the sum of contributing joint forces:

$$Q_i = Q_S + Q_D + Q_L + \dots \, , \qquad \text{Eq. 80}$$

where Q_S is the exponential spring/actuator force, Q_D is the damping force, and Q_L is the (exponential) joint limit force. Other joint forces are available in *corpus*, including linear springs, linear spring joint limits, joint limit dampers, and bias forces (direct specification of a contributing joint force). Any number of springs, dampers, etc. can be set to operate at a joint.

Exponential springs are typically used at joints to control postures and motions of articulated figures in *corpus* simulations. These springs have an exponential relationship between the displacement of the joint position from the spring rest position and the resulting force. The force response of the springs is given as:

$$Q_S = \begin{cases} \alpha \left(e^{\beta (q_{target} - q)} - 1\right) & \textbf{if } q < q_{target} \\ -\alpha \left(e^{\beta (q - q_{target})} - 1\right) & \textbf{if } q > q_{target} \end{cases} \, , \qquad \text{Eq. 81}$$

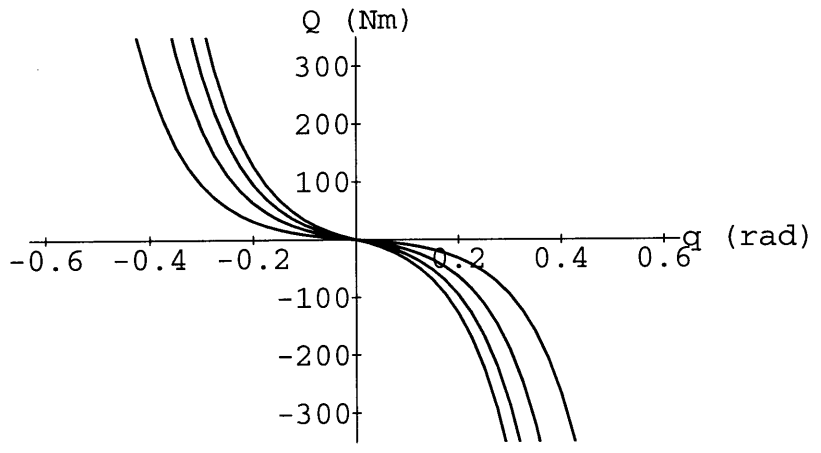
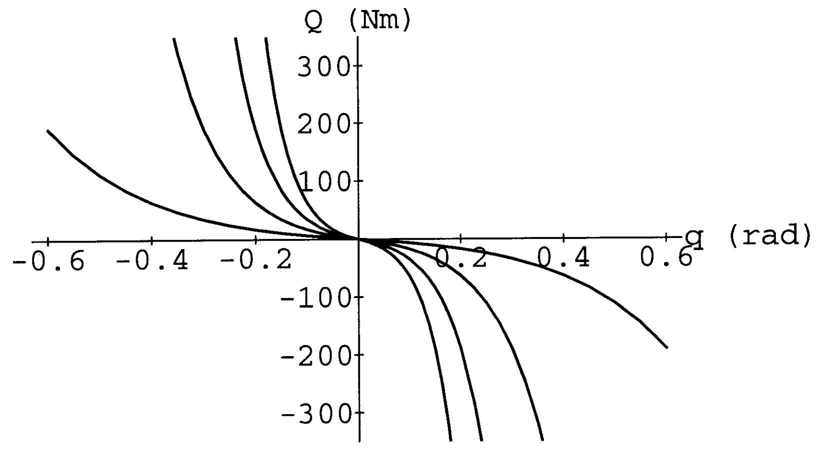


Figure 26: Force response of the exponential spring.

top: Varying the linear spring constant $\alpha = 5, 10, 15, 20$. $\beta=10$



lower: Varying the exponential spring constant: $\beta = 5, 10, 15, 20$. $\alpha = 10$. The exponential constant has a much more powerful influence on the force response, as compared to the linear parameter.

where α is the linear stiffness constant for the spring, β is the exponential stiffness constant, q is the joint position, and q_{target} is the spring rest position. This model was introduced to the graphics community by Armstrong, *et al.* [Armstrong 1987] The force response of the spring, for varying constants if shown in Figure 26.

An inverse model of the exponential springs is used to determine what spring rest angle corresponds to a specified joint force. When coupled with inverse or hybrid dynamics, this inverse actuator model provides not only the forces, but also the actuator control parameters that are need to execute the specified kinematics. Given the joint force and exponential spring stiffness parameters, the rest position is given as:

$$q_{target} = \frac{Q_s}{|Q_s|} \frac{\ln\left(\frac{|Q_s|}{\alpha} + 1\right)}{\beta} + q \ . \qquad \text{Eq. 82}$$

Linear springs can be used at joints to generate forces in *corpus* as well. The force of a linear spring, Q_{SL}, with a stiffness constant, k, is given as:

$$Q_{SL} = k\ (q_{target} - q) \ . \qquad \text{Eq. 83}$$

Dampers in *corpus* provide joint forces which oppose joint velocity. These forces dissipate kinetic energy, and model the viscous drag which arises from many sources in the real world. The force is linearly proportional to the velocity, and is applied in the opposite direction:

$$Q_D = -b\dot{q} , \qquad \text{Eq. 84}$$

where b is the joint damping coefficient.

Joint limits in *corpus* are springs which become active when the joint position passes beyond a specified range. These springs approximately model the resistive force in biological forms which are due to passive joint structures, including the elastic elements of the muscles, tendons, and other surrounding tissue. Exponential joint limits employ exponential springs in a similar form to those discussed above:

$$Q_L = \begin{cases} \alpha\,(e^{\beta(q_{limit1} - q)} - 1) & \textbf{if } q < q_{limit1} \\ -\alpha\,(e^{\beta(q - q_{limit2})} - 1) & \textbf{if } q > q_{limit2} \end{cases} \ . \qquad \text{Eq. 85}$$

Linear spring joint limit models can also be employed in *corpus*, and are given as:

$$Q_{L2} = \begin{cases} k\ (q_{limit1} - q) & \textbf{if } q < q_{limit1} \\ -k\ (q - q_{limit2}) & \textbf{if } q > q_{limit2} \end{cases} \ . \qquad \text{Eq. 86}$$

Joint limit dampers are also available in *corpus*, such that dissipating forces become active when the joint position passes into the joint limit region, as in:

$$Q_{Lb} = \begin{cases} -b\,\dot{q} \ \textbf{if } q < q_{limit1} \\ -b\,\dot{q} \ \textbf{if } q > q_{limit2} \end{cases} . \qquad \text{Eq. 87}$$

5.9 Motor Programs

Dynamic motor programs are used to modify the actuator parameters over time in order to generate actively-controlled movements. A motor program moves a spring's rest position from a start to an end position. Both exponential and linear springs can be used with motor programs in *corpus*. The spring rest angles are generally used as the means of motion control in *corpus*, therefore motor programs allow us to vary the control state from one setting to another. These control states could potentially have been "calibrated" using inverse/hybrid dynamics and the inverse actuator model.

Motor programs currently use a linear interpolation method, moving the spring rest position with a constant velocity until the goal position is reached. The spring rest position is set by the motor program as follows:

$$q_{target} = \frac{(q_{end} - q_{start})}{t_{dur}} (t - t_{start}) + q_{start} , \qquad \text{Eq. 88}$$

where q_{target} is the new spring rest position, q_{start} is the spring rest position at the beginning of the motor program q_{end} is the final spring rest position specified for the motor program, t_{dur} is the duration specified for the motor program, t is the current time, and t_{start} is the motor program starting time. Once the end goal is reached by the spring rest angle, the motor program terminates. The motor program function must be evaluated within the dynamics simulation step, not at a reduced sampling rate; because it is time-dependent, it must be sampled with the integrator's timestep, as opposed to the overall frame rate.

An alternate form of executing the motor program interpolation has also been explored in *corpus*, in which the numerical integrator is used to interpolate the spring rest position, as in:

$$q_{target}(t) = q_{start} + \int_{t_{start}}^{t_{end}} \dot{q}_{target} dt \,, \qquad \text{Eq. 89}$$

with a one-time calculation of:

$$\dot{q}_{target} = \frac{q_{end} - q_{start}}{t_{dur}} \,. \qquad \text{Eq. 90}$$

The numerical integrator then automatically updates the spring rest position over time, based on the spring rest position's velocity. Although it is an interesting approach to modifying the time-dependent rest position, use of the integrator does not offer any real advantage over Eq. 88, and likely runs at a slower execution speed.

New methods of interpolating the actuators via motor programs can be added to the system as necessary. Possibilities include smooth "in-out" functions, such as trigonometric functions or splines, and table-based interpolation, for matching recorded human movements or using pre-computed control trajectories.

Other types of motor programs can also be added to *corpus* in order to interpolate additional control values. For example, *corpus* supports motor programs to vary the linear stiffness constant (*ea*) of an exponential spring, from its starting value to a target value over time, providing a form of stiffness control.

6 Biomechanical Model

6.1 Introduction

This section describes the development of the human figure model, with all of its biomechanical parameters. These include both kinematic and dynamic elements. The kinematic structure describes how the figure is "put together," and in general it is defined by the anthropometrics of the human form. Kinematic parameters include the lengths of the limb segments (or "links"), the joint motions, and the three dimensional geometries that form the surfaces of the figure. In order to perform a kinematic simulation, these are the only parameters needed. To compute dynamic simulations, a number of dynamic parameters must be specified as well. These include the mass and inertia of the links (including the locations of their centers of mass) and the parameters for the force generators, such as the dampers, joint limits, and actuators.

The model developed here is fairly complex in nature, especially with regard to the foot. By including a higher level of kinematic complexity, the model is brought closer to the real structure of the human. This allows for the simulation and inspection of more nuances in motions such as walking, etc. Structure and function that are presently missing from animations and biomechanical analyses can be explored. Complexity can be a key ingredient in making animation seem interesting and realistic. In the real world, a physician obviously cannot ignore complexity and detail in the human body, because each part contributes to function.

A simplified version of the scripting commands used to "construct" or "build" and initialize the human figure model, with its biomechanical parameters, are given in **Appendix D Body Scripts**. Many of the kinematic and dynamic parameters for the model are listed in **Appendix E Body Tables**.

6.2 Kinematic Parameters

The kinematic model of the human was created in multiple steps. First, an intermediate kinematic model was generated, using a digitized skeleton [Stredney] as a three dimensional reference. The joints were parametrized from joint insertion to joint insertion, as matched to the geometric skeleton model, in its default, anatomical position. The kinematic model parametrized from the skeleton does not encompass the actual human bone shape, but simply the measurement from joint to joint, or the "link" length. The next step was to refine the model, using published anthropometric data, including link lengths and inertias. The joint axes, which specify the degrees of freedom in the figure, were based on the "major" degrees of freedom in the human body, and the "major" degrees of freedom in the foot. Finally, the model was completed using a set of scanned, 2D medical illustrations to "fine-tune" the model, filling in details that were difficult to extract from the literature.

Digitized Skeleton

A three dimensional geometric model of the human skeleton was manually digitized by Stredney, for the purposes of animation and medical education. [Stredney] This model was employed by Zeltzer, in his kinematic simulations of human walking. [Zeltzer 1984] The skeleton model, which we will refer to as the "Stredney" model henceforth, was used as a reference to assist in the initial design of the human figure model developed with this thesis (the "McKenna" model).

The Stredney model was loaded into the 3D *corpus* environment, and was used as a graphical "template" to construct the hierarchical, kinematic structure of the preliminary human figure model. Using *corpus* scripting commands, in an interactive fashion at the keyboard, the approximate locations of the joints of the skeleton were located. Simple representations of the body segments were placed along the length of the skeleton bone, from the proximal joint to the distal joint, adjusting the link length and direction appropriately. Thus the actual shape of the body is not encompassed in the kinematic description, only the locations of the joints, relative to one another. The actual bone shape is not important to the simulation, unless the internal stresses of the bones are of interest, or if detailed anatomical information is required for illustration or medical examination.

Higher level interactive techniques would have been of use during this process, to eliminate the time and tedium of using scripting commands. For example, graphical tools to automatically "surround" indicated bones would have helped to automatically locate their position in 3D. Also, interactive devices which allow for direct control over 3D cursors would have been of use to locate points of interest such as approximate joint centers of rotation. Nonetheless, scripting commands were sufficient to build the model, especially since only one model was being created. Because commands were used to build the model, this led to the direct building of the *corpus* scripts which construct and initialize the model. These scripts could then be edited to further adjust the figure, as was done in later stages of the modeling.

The human figure model was designed with three different "layers," such that each body segment has associated with it three different graphical objects. An internal structure is named the "skeletal" layer, which is the kinematic description derived from the Stredney model. In the dynamic simulation environment, these bodies form the articulated structure of the human figure model, defining the joints and the link lengths. These bodies are defined as being "non-colliding" such that no collision detection operations are performed between those bodies and the ground or any other body. Because they are internal and cannot collide with any objects, collision detection would be a waste of computation.

The skeletal layer is surrounded by two "skin" layers, which are essentially equivalent to each other. The skin layers approximate the human body's outer envelope, and serve three main purposes: they are used as the geometric surfaces for collision detection, the volume defined by their surfaces is used to compute the bodies' inertias, and they create a more realistic appearance for rendering (especially as compared to the skeletal layer). The "simple skin" layer is composed of objects that are very basic in nature, with few polygons. This layer is used to compute the bodies' inertias, for collision detection, and for display when rapid rendering is desired. The second skin layer is the "display skin" which is composed of graphical objects of the same size as the simple skin, but with rounded, beveled edges for more appealing visual rendered images. The intermediate model of the human figure, with its skeletal and skin layers, is shown with the Stredney skeleton in Figure 27.

The Stredney model was particularly useful in modeling the foot structure. Because the foot is a complex 3D form, difficult to describe and understand from text descriptions, a 3D model allowed for a rapid understanding of the relationships between bones. The Stredney model for the foot skeleton is shown together with the McKenna foot model in Figure 28.

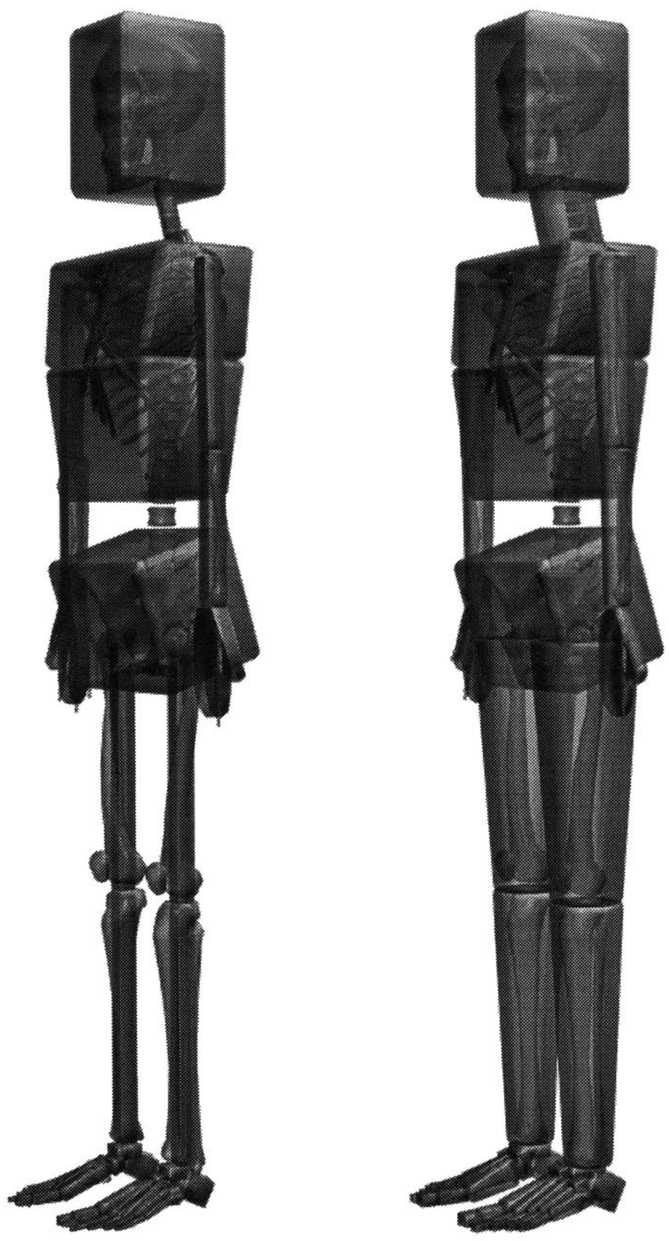


Figure 27: Stredney model of the digitized skeleton overlaid with parametrized model. Left: "skeletal," articulated layer. Right: the initial "skin" layer.

The rigid skin layer is used in the inertia computation. (The tapered cylinders in the legs and arm provide for the correct location of the limb segments' centers of mass.) The skin layer is also used for geometric collision detection and for a more realistic surface rendering.

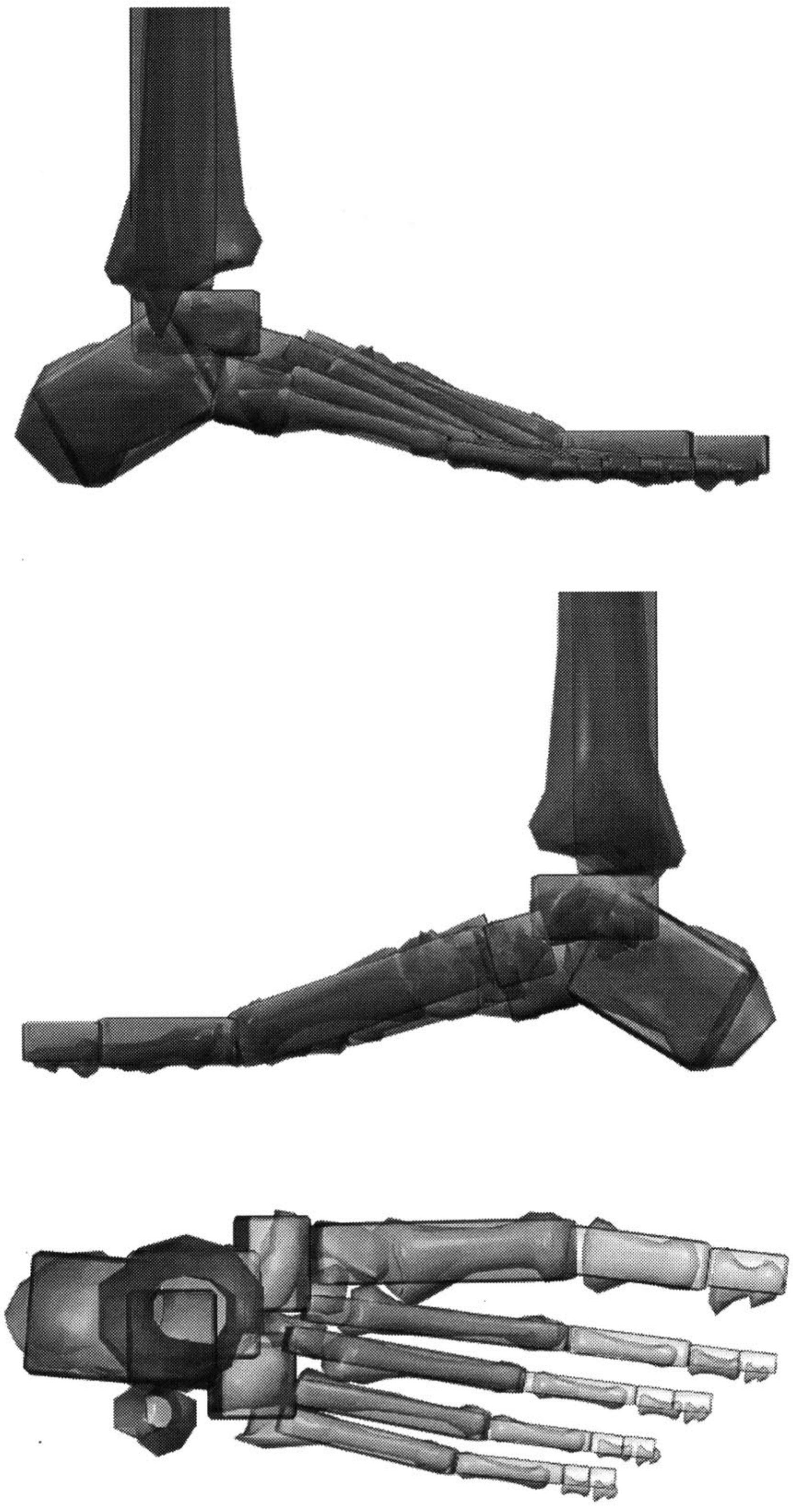


Figure 28: The right foot of the Stredney model overlaid with the parametrized *corpus* model.

Anthropometric Measures

The initial kinematic model was refined using anthropometric measures, published in the literature. The Stredney model was somewhat anomalous in the relative sizes of the different limb segments, in comparison to "average" values.

After considering a number of published studies of human anthropometrics, [Braune 1988; Dempster; Williams 1977] a model developed by Drillis and Contini was selected. [Drillis; Winter 1990] They present an averaged set of segment lengths, which are parametrized as a percentage of overall body height. A graphical representation of their model is shown in Figure 5, page 33. The body height was set to match an average male height. [Dempster] A table of the body segment lengths from different sources, including the final McKenna human figure model is given in Table 2.

Adjusting the McKenna model to the Drillis and Contini model was quite straightforward, since the kinematic structure had already been established. The *corpus* scripts which "build" the human figure model were easily modified simply by directly changing the normalized scaling factors for the limb segments.

Table 2: Segment (link) lengths, from joint to joint.

segment	*Stredney*	*Dempster*	*Drillis & Contini*	*McKenna*
total height	1.73	1.77	1.77	1.77
femur link	0.380	0.434	0.432	0.432
tibial link	0.380	0.409	0.434	0.434
foot length	0.274	0.267	0.268	0.28
mid-talus to floor	0.057	0.081	0.069	0.063
femur separation at pelvis	0.170		0.337	0.20
shoulder separation	0.350		0.457	0.36
head height	0.247		0.229	0.229
humerus link		0.302	0.328	0.328
radius link		0.272	0.258	0.258
hand length		0.191	0.191	0.191
femur to humerus height			0.508	0.498

Degrees of Freedom

The degrees of freedom in the figure were specified to capture the major ways in which the overall body moves, and also the major ways in which the foot moves. It is difficult to say what the "major" degrees of freedom are, but some choices were more clearly defined. For example, the knee allows primarily for flexion and extension, but small amounts of abduction/adduction and rotations are possible. In addition, the motion of the knee is not exactly defined by a hinge joint, but it actually has complex 6 dimensional motion. The knee is quite well approximated by a hinge joint, however, which is the model used in most animation and biomechanical models. Similarly, the middle and distal phalanges of the toes primarily flex and extend, but small amounts of abduction and adduction and rotation can occur there are well. These joints were also modeled as hinge joints. The other notable degrees of freedom that were not modeled are a complex spine and neck, and a jointed hand. A preliminary step was made in modeling the hand, described in the **Future Directions** subsection in the **Conclusions** section (8). Ultimately, all of the degrees of freedom in the human should be included in the human figure model, including very minor motions. Starting from the most sophisticated model, the complexity can be reduced to a level suitable for a given animation or simulation, and from that point, the complexity could be increased to observe the effect, if any, of additional DOFs.

A diagram of the degrees of freedom that were modeled in the body, except for the feet, is given in Figure 29. The DOFs modeled in the feet are shown in Figure 30.

The overall body has six degrees of freedom; it can translate in three directions, and rotate to any orientation in three dimensions. These six DOFs are associated with the abdomen (or torso) of the figure, which is the "root" object in the hierarchical kinematic structure. These degrees of freedom can be "free," when forward dynamics computes the acceleration of the figure, or they can be constrained to be motionless or to follow a specified acceleration.

Although a complex spine/neck has not been modeled, a number of degrees of freedom were included to approximate its motion. Three DOFs are included between the head and the neck, which allows the head to tilt or rotate in any direction with respect to the neck/abdomen. Similarly a three DOF "waist" is included, above the pelvis, about the location of the belly button. These DOFs are required to allow the pelvis and lower body to tilt and rotate without forcing the upper body to follow.

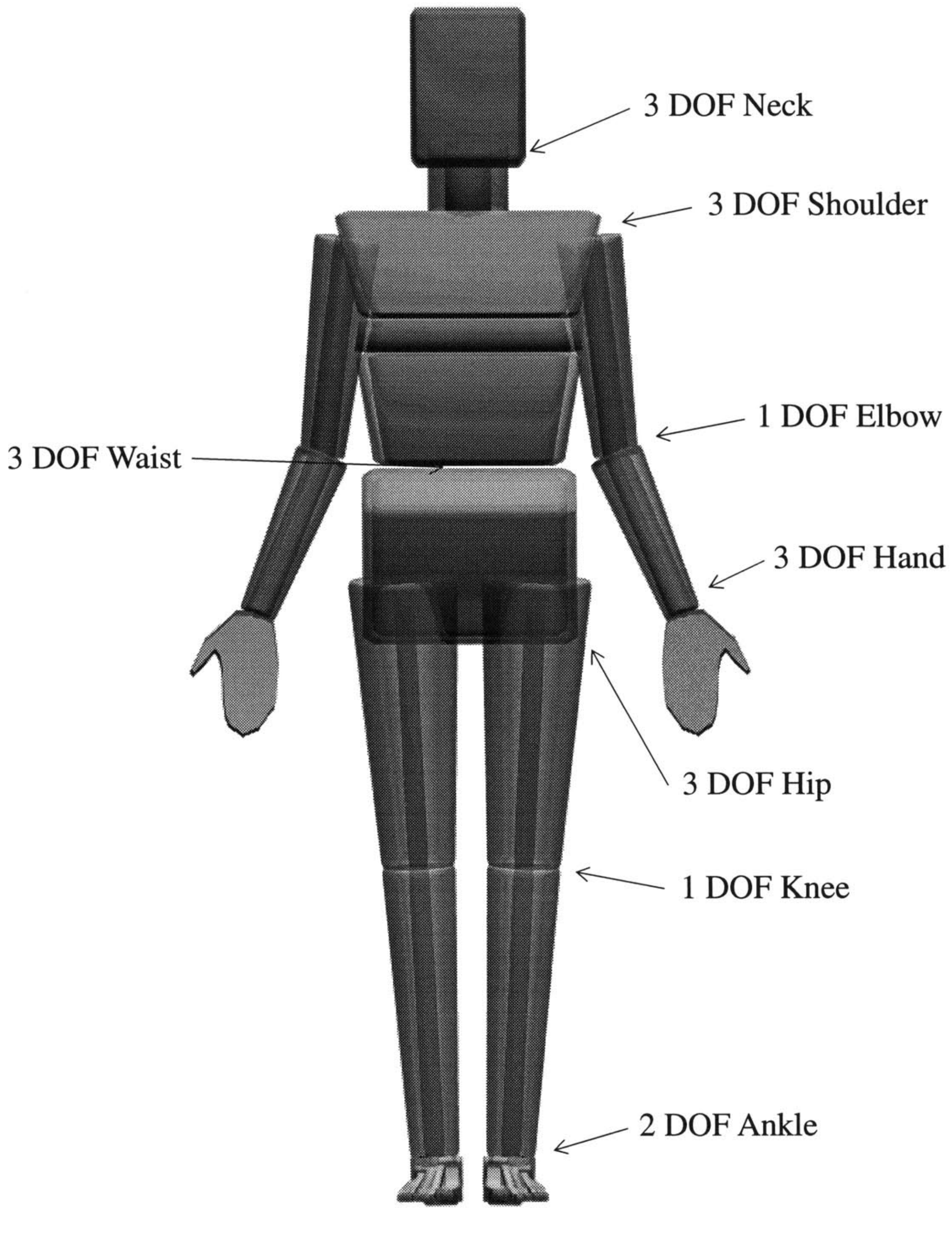


Figure 29: The degrees of freedom in the human figure model, above the foot.

The entire body also has 6 degrees of freedom, 3 translating and 3 rotating degrees of freedom, which allow the body to be positioned and oriented anywhere in space.

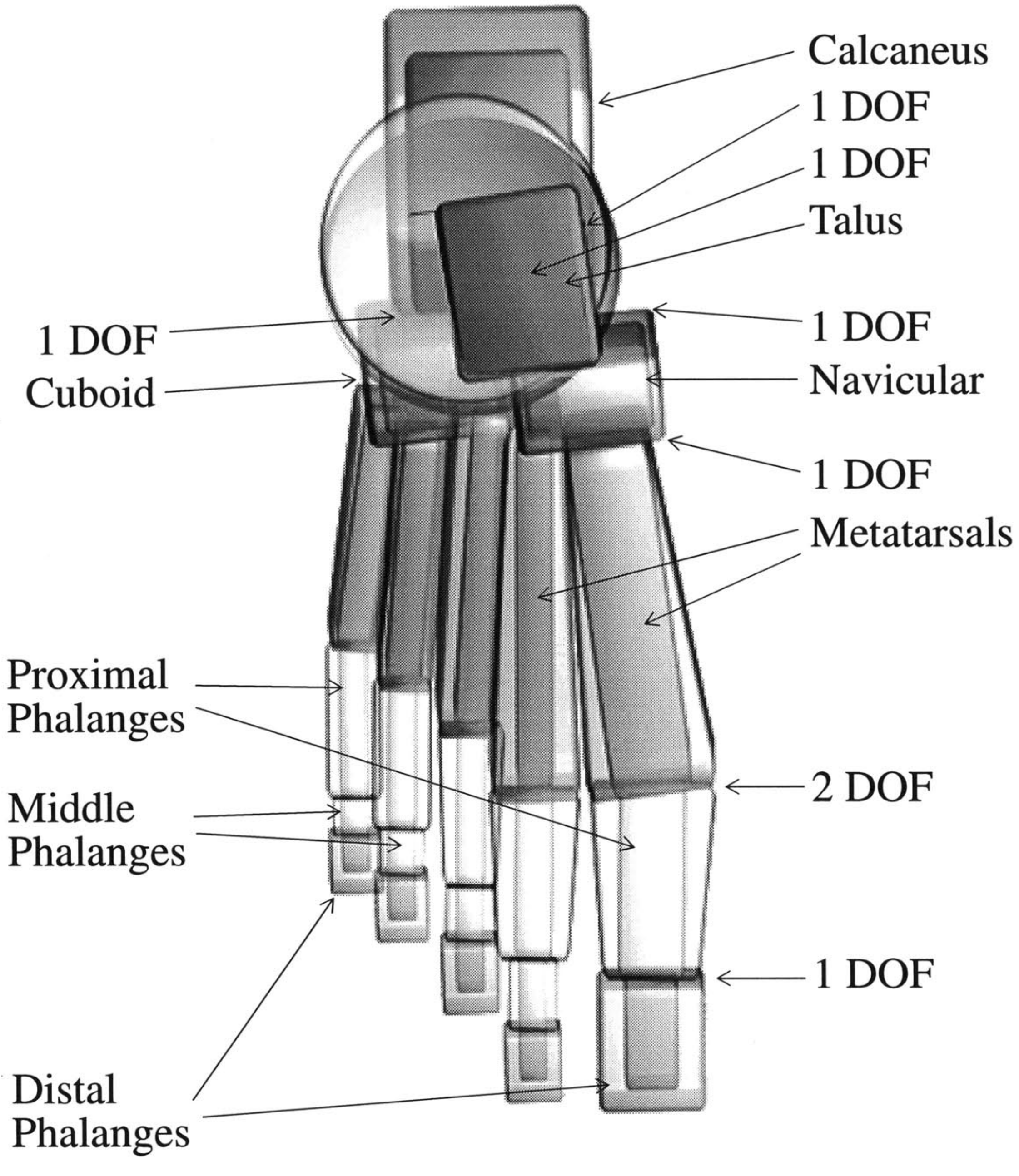


Figure 30: The degrees of freedom in the foot of the human figure model.

One of the 1 DOF ankle joints is above the talus, and one is below.

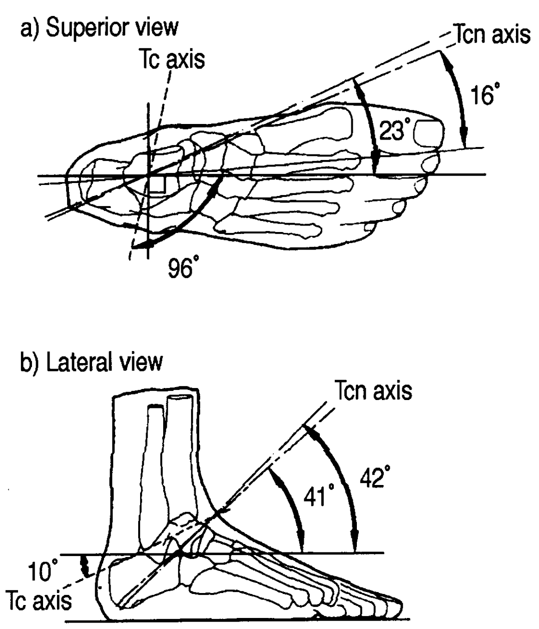


Figure 31: The Tc (talocrural or talar) and Tcn (talocalcaneonavicular or subtalar) joints in the ankle. [Procter]

The shoulders each have three DOFs, allowing the arm to rotate in any direction with respect to the abdomen. The elbows have one DOF each, allowing flexion and extension. The wrist, between the forearm and hand provides three DOFs. In the human, the rotation of the wrist (as opposed to flexion or abduction) occurs along the length of the forearm. In the human figure model, the wrist rotation is lumped together with the other two DOFs directly at the connection between the forearm and the hand. The hand itself (the palm, thumb, and fingers) is modeled as a single object, with no DOFs below the wrist. (However, a preliminary model of the hand was created, based on the foot's kinematic structure. See the **Future Directions** subsection in the **Conclusions** section (8)).

The hips, above the thighs are modeled with three DOFs each, like the shoulders. The knees have one DOF each, allowing flexion and extension. The ankles are modeled with two DOFs each, placed and aligned according to published biomechanical measurements. [Procter]. The talocrural (Tc) or upper ankle joint or talar joint allows for flexion and extension of the foot, and talocalcaneonavicular (Tcn) or lower ankle joint, or subtalar joint allows the foot to invert and evert. See Figure 31. The talar joint connects the distal end of the lower leg to the talus. The subtalar joint connects the "bottom" of the talus to the calcaneus, or hindfoot (the "heel bone").

The DOFs which articulated off of the hindfoot are based on Inman and Mann's analysis of foot movements, which is a simplified model compared to the complex sliding move-

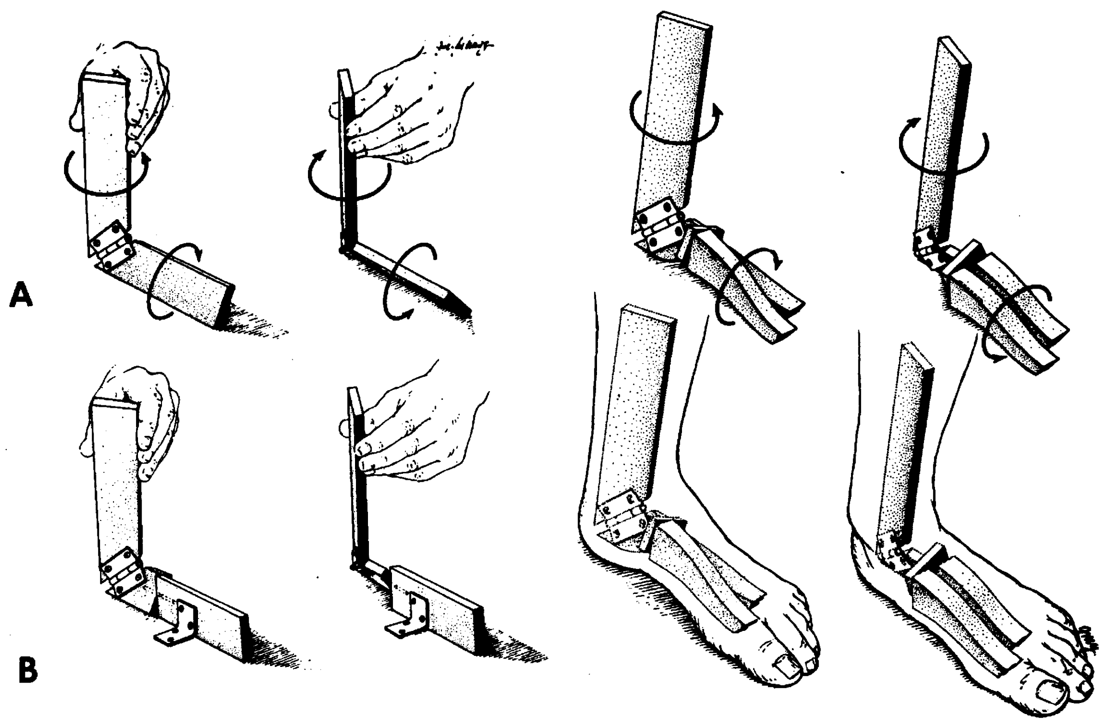


Figure 32: A depiction of the action of the subtalar joint and navicular and cuboid joints in the human foot.

First, consider the upper left quadrant, labeled "A." The hinge joint represents the subtalar joint. When the upper leg rotates, the subtalar joint everts and inverts the forefoot. In the lower left quadrant, labeled "B," a pin joint has been added which represents the combined action of the navicular and cuboid joints. Now, when the upper leg rotates, the hindfoot everts and inverts due to the subtalar joint, yet the forefoot remains flat and connected to the ground, due to the added pin joint. On the right side, the models include two pin joints between the hindfoot and forefoot. The medial joint represents the joint between the hindfoot and navicular. The ray emerging from the navicular represents the three medial rays in the foot (three metatarsals and their associated phalanges). The lateral pin joint represents the cuboid, and its single ray represents the two lateral rays which articulate off of the cuboid.

Image and model from [Inman].

ments which occur between the bones in the tarsal region of the foot (rear foot). The navicular and the cuboid bones articulate from the hindfoot, with one DOF each, using a rotary joint. The combination of their rotations and the rotations of the subtalar joint allow the leg to rotate while the forefoot maintains a non-sliding contact with respect to the ground. See Figure 32.

On each foot, the three medial metatarsals articulate off of the navicular, with one DOF each, allowing extension/flexion). The two lateral metatarsals articulate off the cuboid. The three cuneiforms, which lie between the medial metatarsals and the navicular are lumped together into the metatarsals. There is very little articulation provided by them, [Gray] although a more complete model should include the DOFs they provide. The metatarsals do not move very much in the foot, and stiff springs in the model restrict motion. The flexion and extension that they do provide allows the arches of the foot to change shape.

The proximal phalanges (singular "phalanx") articulate off of the metatarsals, with two DOFs each, allowing flexion/extension, and abduction/adduction. This is the start of toes, which would remain more flat on the ground, throughout the stance phase of walking. The middle and distal phalanges articulate with 1 DOF each, allowing flexion/extension. The big toe does not have a middle phalanx, but the other four toes do.

Anatomical Illustrations

A set of 2D anatomical illustrations were used in the final stages of modeling. Illustrations from [Gray] and other sources were used for general reference, but the primary source was [Goldfinger]. The illustrations were "scanned" into the computer, using a flatbed scanner, and were used as semi-transparent texture maps in the 3D *corpus* environment. The human figure model with one such illustration is shown from a perspective view in Figure 33. When an orthographic view is taken of the front or a side of the human figure model, and the illustration is appropriately scaled and positioned, the two graphical images overlap and the illustration can be used as a template for modeling or as a visual augmentation of the figure.

Illustrations of the skeleton were useful to adjust the kinematic structure of the human figure model. The illustrations indicated the locations of the attachment points at the head, shoulders, and hips. They also served as a validation of the established kinematic structure. Finally, they were used as a postural template, to position the various joint angles into a more "appealing" and more standardized anatomical position than had been constructed from the Stredney model. Front and side views of the skeleton structure are shown in Figure 34 and Figure 35.

Similarly, anatomical illustrations of the skin and musculature by Goldfinger were useful in refining the model of the skin layer of the human figure model. The final skin and skeletal layers are shown together in Figure 36. The Goldfinger musculature diagrams were used as a template for the specification of the location of the skin objects with respect to

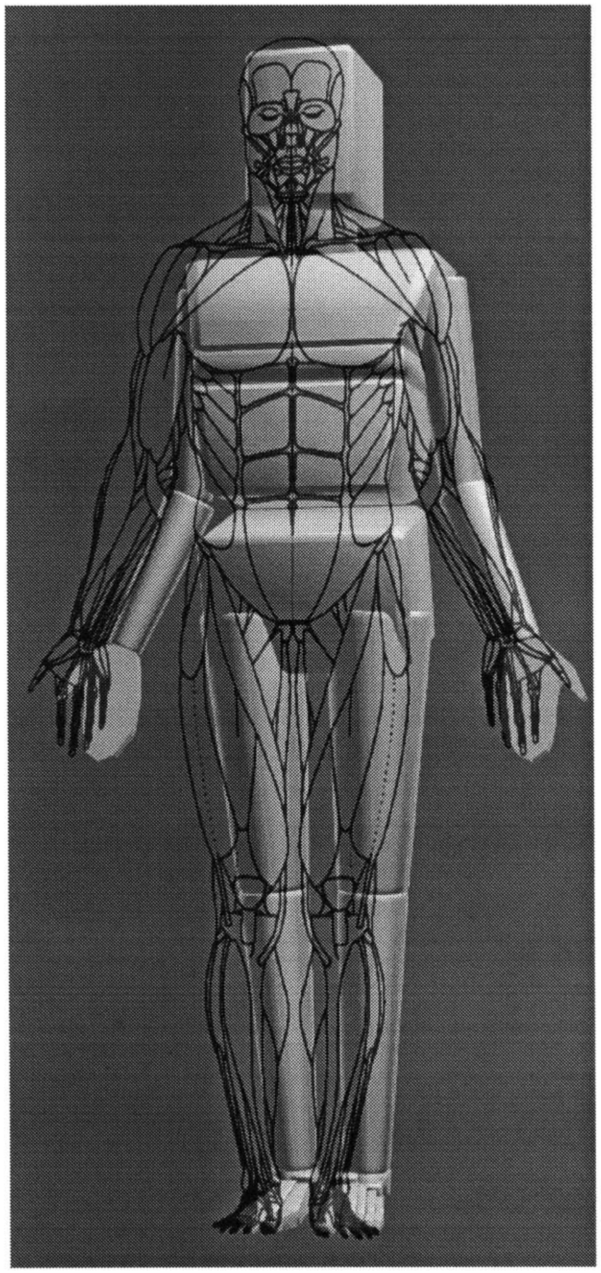


Figure 33: The final humanoid model, shown with a Goldfinger anatomical diagram, from a perspective view.

their internal skeletal objects. In some cases, especially with the torso, hands, neck and head, the diagrams were used as a visual template to help model the shapes and sizes of the skin objects. Front and side views of the skin layer, with the overlaid musculature diagrams are shown in Figure 37.

The "foot-box" is a set of four texture maps, scanned from [Goldfinger], which shows the skeleton of the foot from four different views. The maps are arranged in a box, which surrounds the model of the foot in the virtual environment. By viewing the model and the

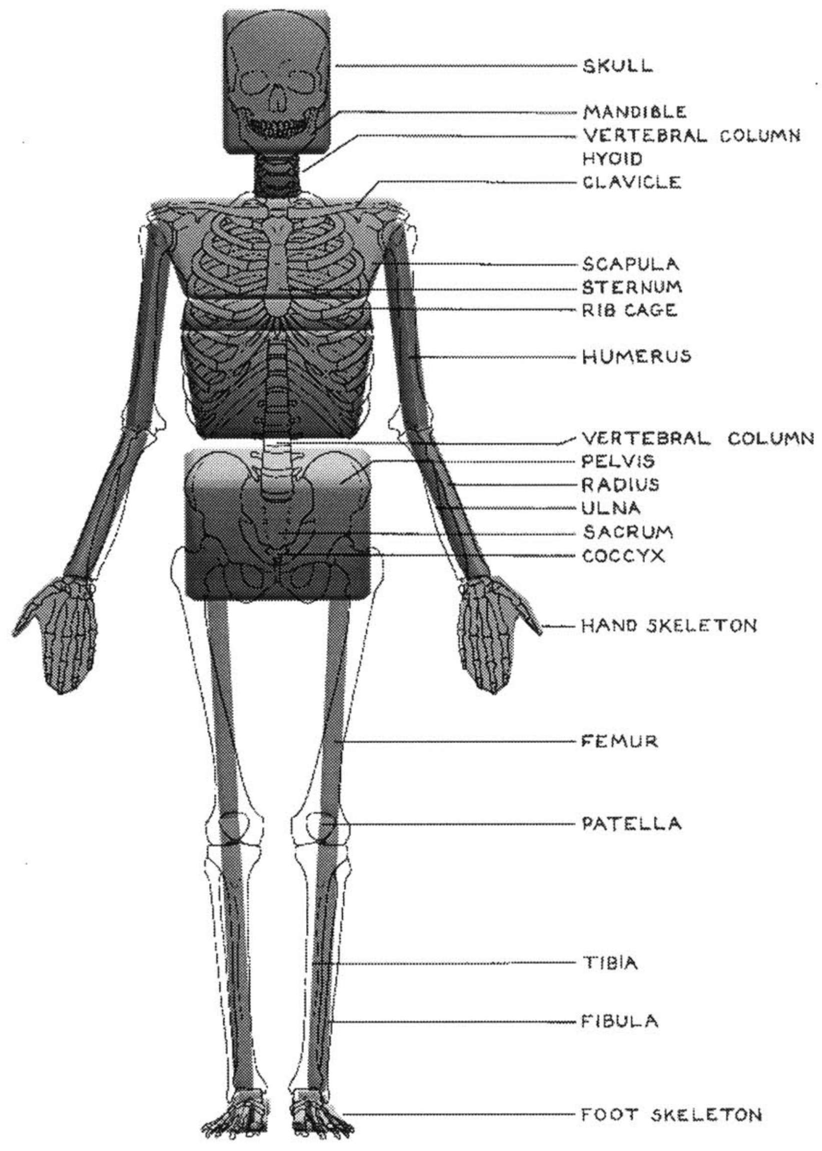
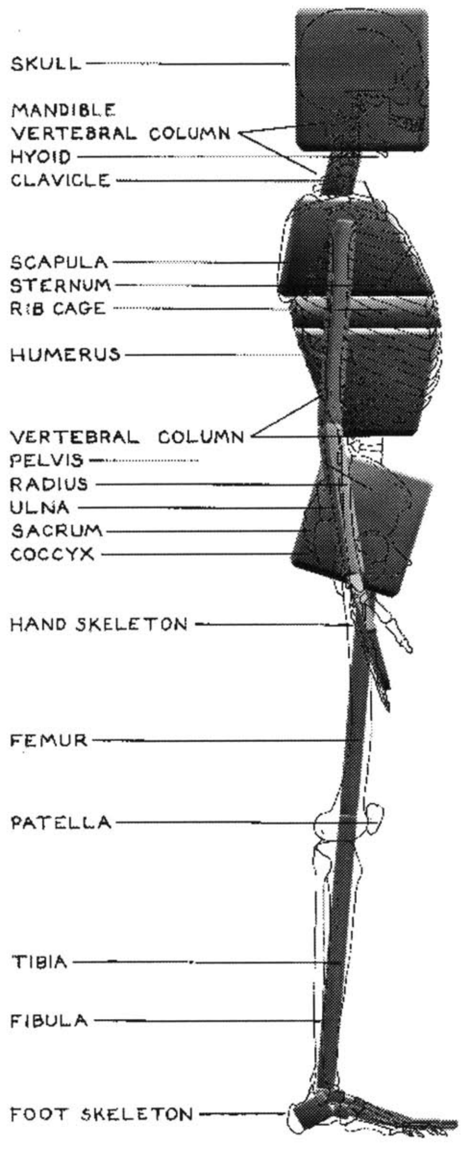


Figure 34: Human figure model, "skeleton" layer, overlaid with anatomical diagram, from the front.

Drawing from [Goldfinger].

semi-transparent foot-box from orthographic views, the two overlap, which allows for additional "fine-tuning" of the model, and also provides an augmented rendering of the model, with descriptive labels and detail from the illustrations. The foot-box and the skeletal layer of the foot model are shown in perspective views in Figure 38. An orthographic view, overlaying the illustration and model is shown in Figure 39.

After the Goldfinger diagrams had been scaled appropriately to match the height of the human figure, the link lengths were not adjusted, yet the diagrams matched very closely to



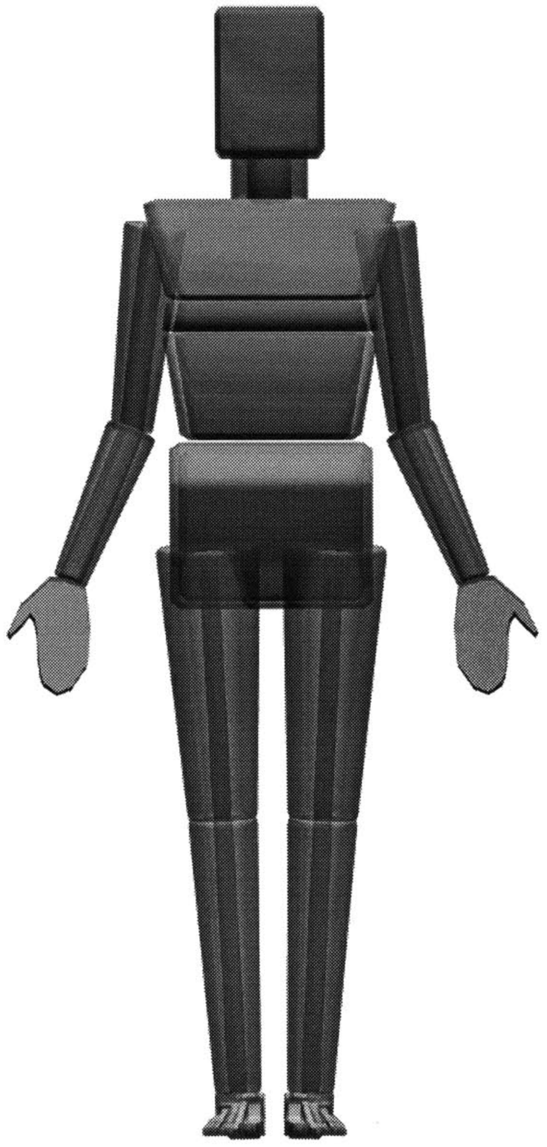


Figure 35: (Left) A side view of the human figure model "skeletal" layer, overlaid with an illustration of the human skeleton.

Drawings from [Goldfinger].

Figure 36: (Right) The human figure model with its "skeletal" and "skin" layers superimposed

The two layers are graphically "mixed" by making the two layers semi-transparent and rendering them both together.

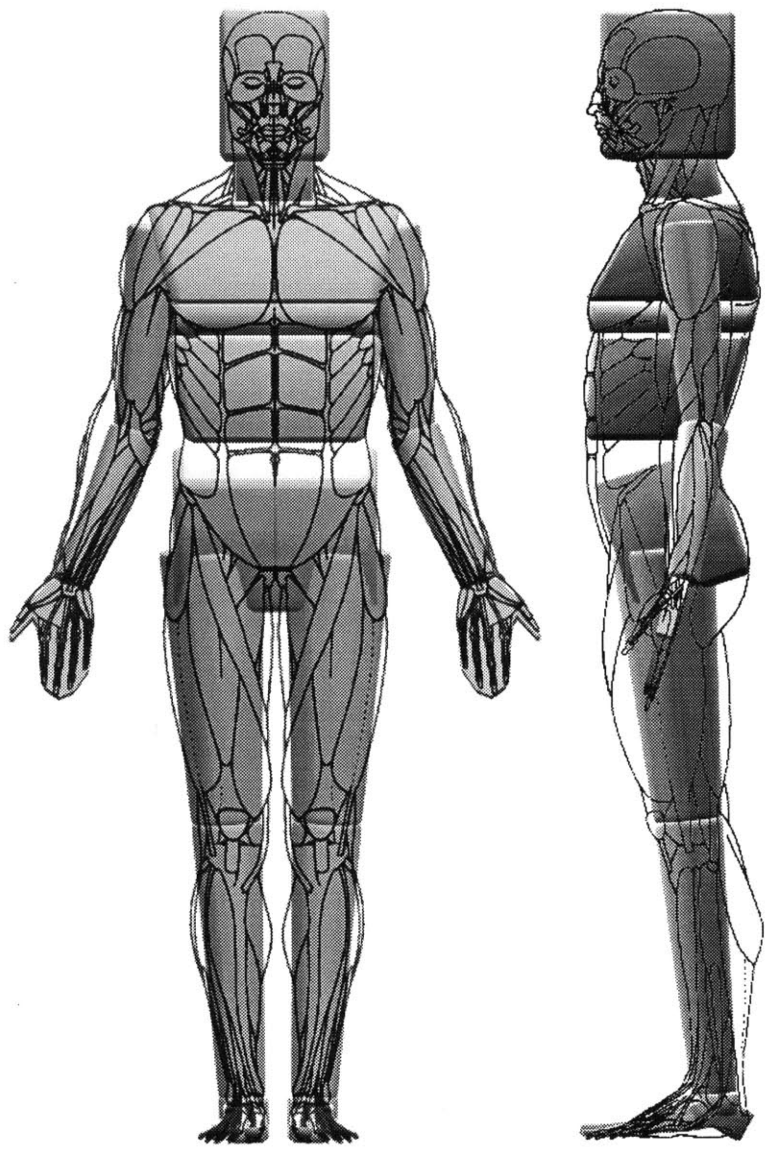
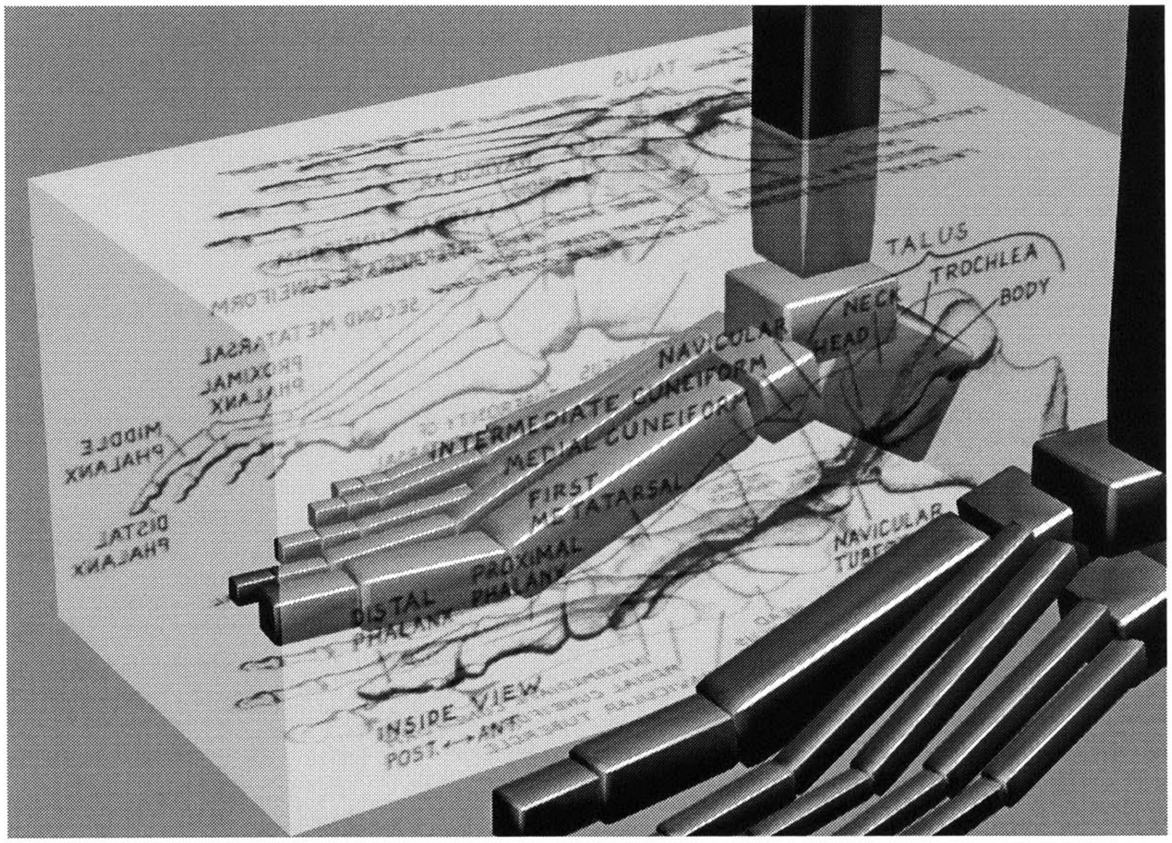


Figure 37: The human figure model "skin" layer with overlaid anatomical diagrams.

Drawings from [Goldfinger].

the model. This helps to show that both the illustrations and parametrized model are reasonably accurate.

As a final inspection of the kinematic structure of the human figure model, a comparison was made of the locations of the metatarsal heads relative to each other in the foot. Inman, *et al.*, measured the angle formed by the distal heads of the second and fifth metatarsal heads. [Inman] The third and fourth metatarsal heads also roughly lie on this line. This is the line along which the toes bend during walking. As shoes are "worn in," a crease will form along this line as well. The average angle measured by Inman was 62°, with a range



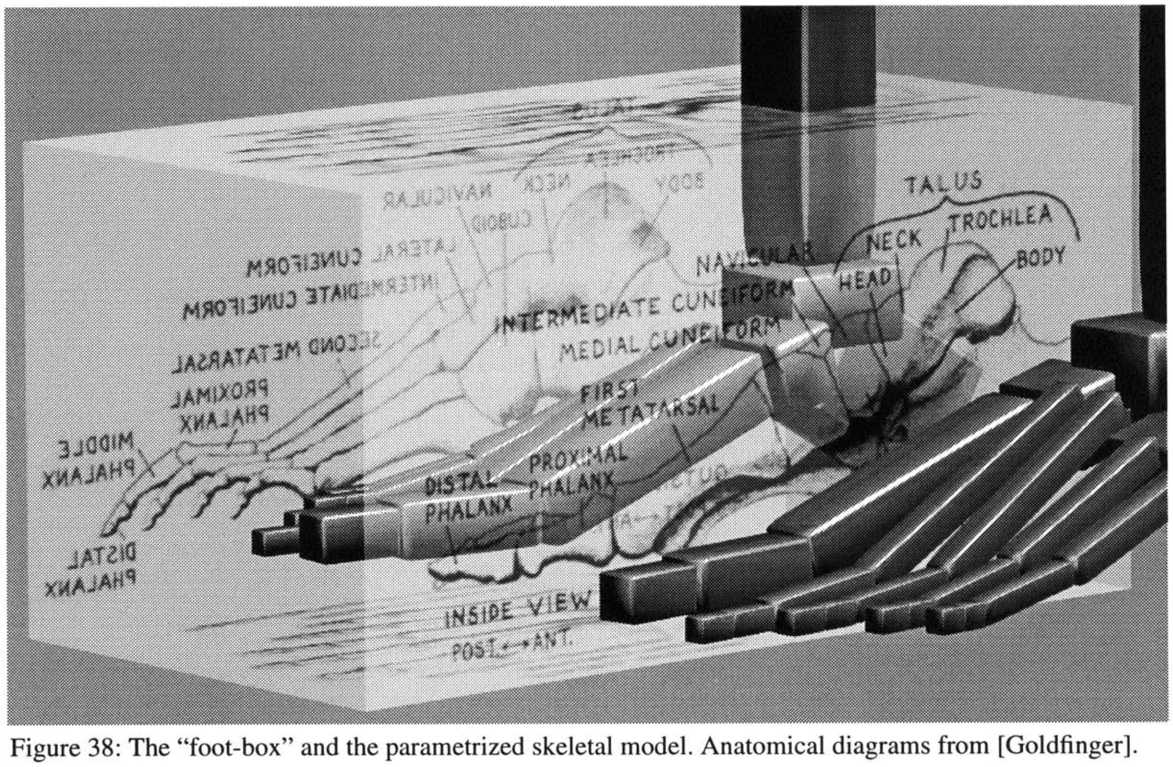


Figure 38: The “foot-box” and the parametrized skeletal model. Anatomical diagrams from [Goldfinger].

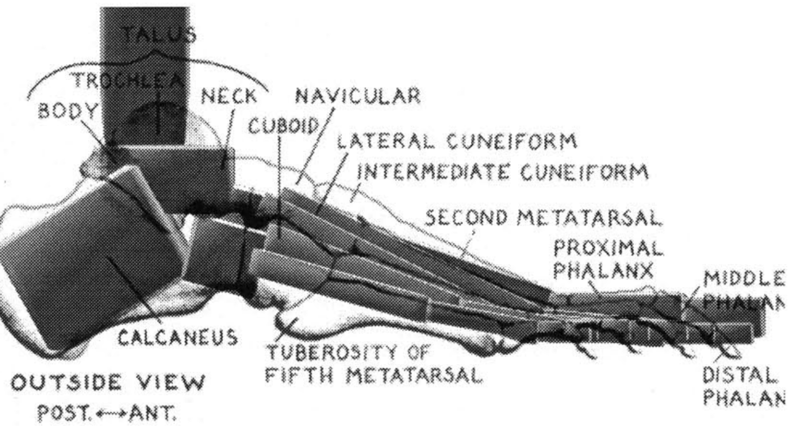


Figure 39: Skeleton of the right foot: McKenna model compared to Goldfinger illustration.

There are some mis-matches between the illustration and the model, as can be seen in the diagram. However, the differences may be within the range of variation seen between different people. A new posture (not shown here) was assigned to the foot, which gives the toes a more natural bend.

from 52°–74°. The angle formed by the human figure model is 54°, within the range measured, but near one of the extremes. See Figure 40.

The kinematic structure of the complete figure is provided in tabular form in **Appendix E Body Tables**.

6.3 Dynamic Parameters

In order to perform dynamic simulations, there are a number of parameters which must be specified beyond the kinematic structure. The mass and inertia tensors of the limbs are required for the equations of motion. The biomechanical models of the actuators, dampers, and joint limits have their associated parameters which are used in the force computations.

Inertia

The masses and centers of mass of the different body segments were set to match the values measured from humans. [Braune 1988; Dempster; Williams 1977; Winter 1990] The complexity of the mass distribution in the human figure model is tied to the level of detail present in its kinematic structure. Each link has its own mass, density and inertia. In *corpus*, the mass and inertia of the body segments is automatically defined from the specified density and the surface geometry of the "skin layer" graphical objects. A method is used to compute the mass, the rotational inertia tensors, and the center of mass, using a summation of the volume formed by each polygon and a reference point within the body.

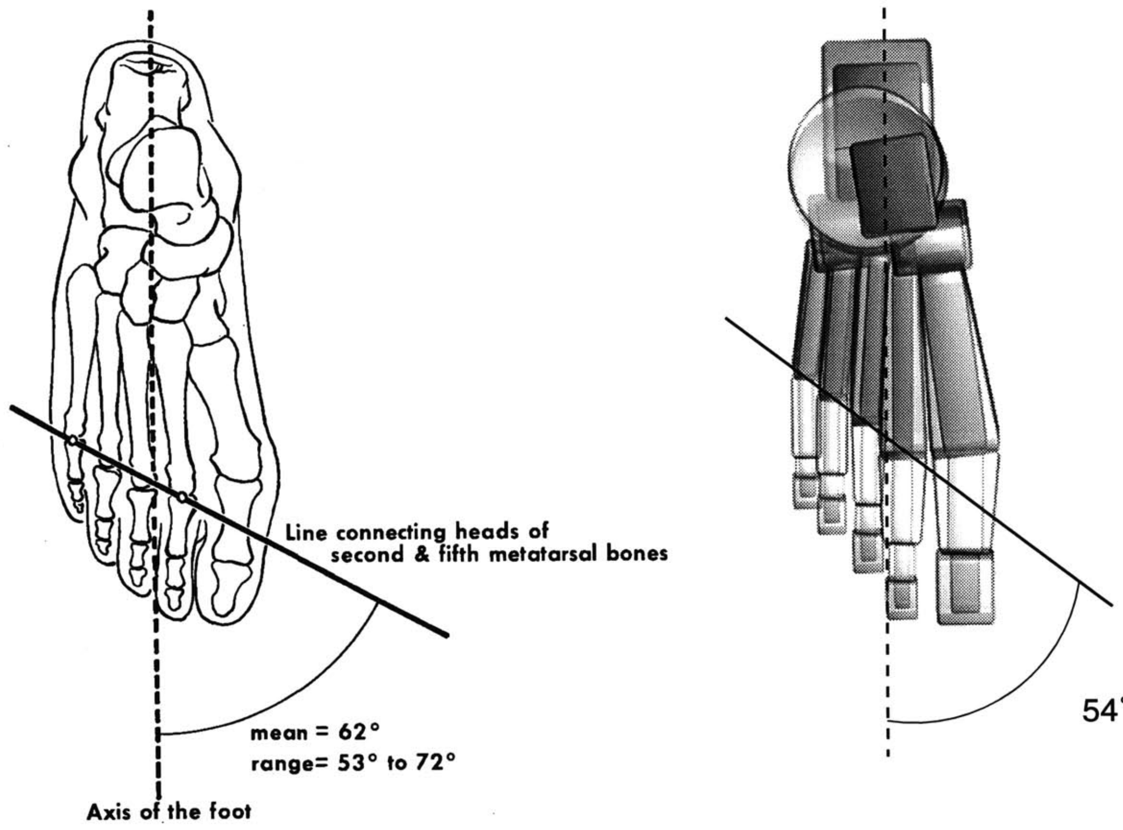


Figure 40: The line formed by the distal heads of the second and fifth metatarsal bones as measured from humans compared to the human figure model.

Left: The mean angle of a set of measurements from human subjects. From [Inman]

Right: The angle formed by the computer model. The angle is near the extreme of the range measured by Inman, *et al.* The third through fifth metatarsals seem to be somewhat short in the computer model, as parametrized from the Stredney model.

The densities of the body segments rises as more distal members are examined. [Winter 1990] This is because bone is more dense than other body tissues, and the distal members are composed of a greater proportion of bone.

For the arm and leg segments, the density of the objects were set to the values measured from humans. By then scaling the objects' diameters appropriately, their masses are computed to match the values measured from humans. Similarly, the cylindrical objects that make up the arms and legs were tapered, using geometric shearing operations, such that the segments' centers of mass shifted from the centers of the objects to the more proximal locations as measured from humans.

In contrast, the head, neck, abdomen, pelvis and foot objects were first scaled to a size to form the skin layer matching the Goldfinger illustrations. The density of the objects were then set so that the masses would result in the measured values.

A listing of the masses of the major body segments, as measured from the human body, and as modeled in the final human figure (McKenna) model is given in Table 3. The overall body weight of the model is 68 kg, or 150 lbs.

Table 3: Segment densities and masses

segment	*Dempster 1955 mass (kg)*	*Winter 1990- mass (kg)*	*Winter 1990 density (kg/m^3)*	*McKenna mass (kg)*	*McKenna density (kg/m^3)*
total	68.0	68.00	1078	68.01	
head	4.7	-	-	4.71	545
neck	0.7	-	-	0.70	565
head and neck	5.4	5.51	1110	5.41	
trunk	34.8	33.80	1030	33.93	780
thigh	6.6	6.80	1050	6.81	1050
shank	3.1	3.16	1090	3.15	1090
foot	1.0	0.99	1100	0.99	1413
upper arm	1.9	1.90	1070	1.90	1070
forearm	1.1	1.09	1130	1.08	1130
hand	0.4	0.41	1160	0.41	1160

The centers of mass for the major body segments, as measured from human studies and in the final human figure model are given in Table 4. A diagram by Dempster shows the COMs in a graphical form (see Figure 41).

Table 4: Centers of mass of different body segments.

The "COM/seg len" columns define the location of the center of mass, measured from the proximal joint, as a fraction of the segment length. The "dist from prox joint" columns give the distance of the COM from the proximal joint, in meters.

Clauser, *et al.*, data is from [Williams 1977].

segment	*Braune 1889 COM/ seg len*	*Clauser, 1969 COM/ seg len*	*Winter 1990 COM/ seg len*	*Dempster COM/ seg len*	*Dempster dist from prox joint (m)*	*McKenna COM/ seg len*	*McKenna dist from prox joint (m)*
head		0.466		0.433		0.500	0.124
thigh	0.440		0.433	0.433	0.187	0.433	0.187
shank	0.420	0.371	0.433	0.433	0.188	0.433	0.188
foot	0.444	0.449	0.500	0.429		0.2	0.053
upper arm	0.470	0.513	0.436	0.436	0.143	0.436	0.143
forearm	0.421	0.390	0.430	0.430	0.111	0.434	0.112
hand			0.506	0.506	0.097	0.424	0.081

Information regarding the masses of all of the body parts in the human figure model is given in **Appendix E Body Tables**.

Dynamic Joint Parameters

This research uses joint forces to control motion, in order to simplify the problem of modeling muscles. In humans, multiple muscles span each joint, and they attach at various locations along the limb (not simply at the joint). Although *corpus* has facilities for simulating multiple actuators, with attachment points onto limbs, the human figure biomechanical model uses actuators and other forces which work directly at the figure's joints. This choice was made in order to greatly simplify the modeling and control problem so that simulation work could begin in a timely manner with the kinematically complex model.

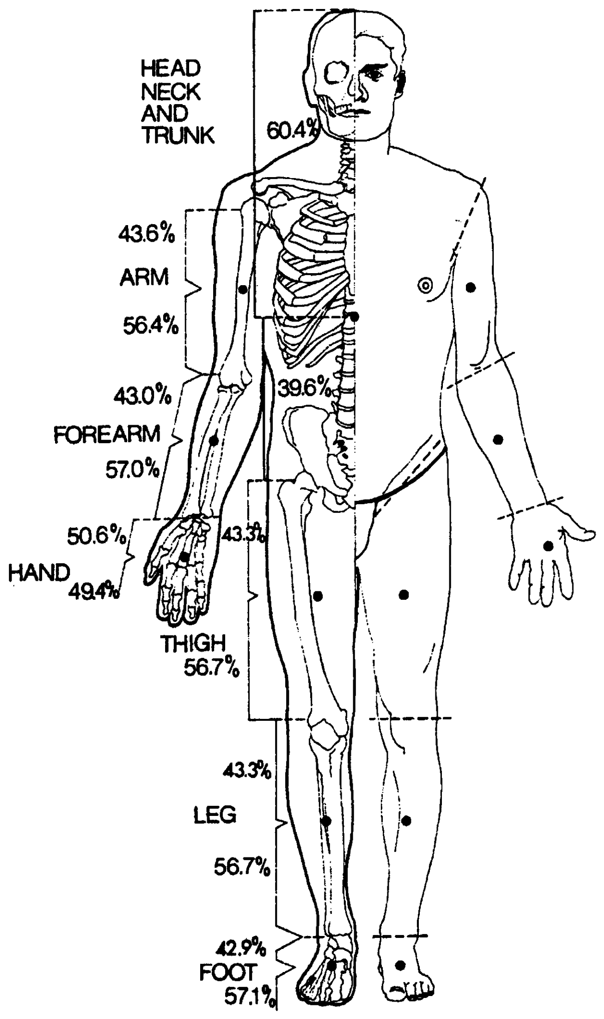


Figure 41: The centers of mass of the limbs.

Diagram by [Dempster], from Williams and Lissner. [Williams 1977]

The motivation for using dampers, spring actuators and exponential spring joint limits has been discussed previously in the **Background** (2) and **Approach** (3) sections. Dampers generate a force proportional to the joint velocity, in the direction to oppose the motion. The use of the dampers is twofold; they model the passive joint damping properties, and they model the damping effects generated by active muscle tissue. It might be more appropriate to include two dampers at each joint, one which is weaker to model the passive joint damping, and a stronger one which is activated along with the actuators. The joint limits are created by using exponential springs, which is a model that approximates the passive force response of the human limbs. The actuators are modeled as exponential springs as well, although human skeletal muscle is typically modeled as a linear spring. The author has had more success with motion and postural control for animation using exponential rather than linear springs. They have the attractive property that the limb will not stray grossly from the spring rest angle, because the force response rises so rapidly at larger displacements. In some simple ways, this approximates the higher level control in humans that allows us to begin to exert more when we encounter resistance, changing the stiffness and activation of the muscles to compensate. Also, informal observations have indicated that there are fewer numerical stiffness problems when exponential, rather than linear springs are used.

The force models for the dampers, exponential spring actuators, and exponential spring joint limits were described in the **Actuator Model and Joint Forces** subsection (5.8, page 99). The equations which describe the motor programs are discussed there as well.

The default values of the damping and spring constant for all of the joints in the human figure model are given in **Appendix E Body Tables**. Examples of using the biomechanical joint force models are given in the **Simulations** section (7).

The complete system for simulation is a synthesis of the dynamic simulator, the articulated figure, with its inertia and mass, the joint force model, and the environment. It is only after all of the elements have been combined that we can begin to perform meaningful simulations.

6.4 Visual Model

Although this work was driven primarily by academic and research goals, some aspects of the effort were of a visual nature as well. It was intended to design a visual form which was "appealing," with a level of detail that engages the eye, without detracting from the fundamental goals of articulated motion research. Fortunately, these two aspects were not at odds with one another, but rather, they were complimentary. Through the biomechanical modeling, form and complexity were "automatically" included in the structure.

Obviously, the modeling effort was directed towards generating an anthropomorphic form, but the human figure was intentionally designed without visual 'false detail' of structure that was not really present in the underlying biomechanical model. There were a few minor exceptions to this rule. The torso/abdomen link in the figure is modeled with three separate sheared cubes, and a cylinder for the neck, even though those separate objects cannot move with respect to one another, which may make the model appear to have more

DOFs than it truly has. The other exception was the inclusion of rounded, bevelled objects to create a more appealing form, with more shading effects.

The three dimensional graphical environment was designed with a number of supplemental texture maps, to create a stronger sense of space, as opposed to a featureless void, common in many virtual environments. These elements work well with video displays and continuous tone outputs, but not as well with dithered print media, such as this document, because they reduce the foreground-background contrast and can make the image seem cluttered. Therefore, they have been absent from most of the images in this document. However, Figure 1, page 17 and Figure 63, page 179 show some of these background elements. "Walls" were made to form a 3x3 meter room around the human figure, composed of texture maps which depict "grid-lines," spaced every 10 cm, with primary lines every meter. These elements were inspired by the backgrounds used by Muybridge in his photographic motion studies. These help give a better sense of scale and three dimensional location.

7 Simulations

A number of simulations were generated using the human figure model in the *corpus* dynamics simulation system. A set of basic simulations are presented first, including examples of the body passively falling to the ground, or drooping while the torso is rigidly held in place. Simple motor program examples are also demonstrated. This is followed by simulations of balanced standing posture, and its response to external perturbation forces. Next, an arm reaching task which is simulated in near real-time is described. A simulation of rising on the toes is then presented. This section closes with simulations of passive stepping, or "ballistic walking," which include passive dynamic motions of the complex foot.

7.1 General Simulations

These first simulations are fairly simple in design. They serve as a general introduction to the process of simulating the human figure model. Basic tests such as these were useful in the development of the model, including its biomechanical parameters. They also serve as a good introduction to creating simulations with the complex model.

Timings were made of many of the following example simulations. The computer platform used for these simulations is a Silicon Graphics *Onyx* workstation, using a 150 MHz R4400 processor which performs 128 million instructions per second (MIPS). The system also has real-time rendering hardware, the *Reality Engine*^{2} (RE^2). The manufacturer, Silicon Graphics, rates the performance of the RE^2 as being capable of rendering 900 thousand anti-aliased triangle mesh elements per second, and drawing 320 million texture-mapped pixels per second.

To begin a simulation, the program *corpus* is started, and the human articulated figure is created, using a *corpus* script which defines and initializes the model. Part of this initialization script is presented in **Appendix D Body Scripts**. This initialization takes approximately 6 seconds.

There are a number of global simulation parameters which can be set, including ones to control integration, and ground and gravity forces. With the human figure model, the adaptive step size integrator (runge-kutta "RK 4/5") should always be used. In all but the most trivial cases, the varying conditions of the simulation require the integrator to vary the number of sub-steps it must take in order to maintain a stable solution. Use of the adaptive integrator is specified with the *corpus* command: `integration rkf`.

The timestep size for the simulation is specified to 0.0333333, so that for every second of simulation time (time "experienced" by the human model), 30 frames of simulation results are generated. This matches the NTSC video frame rate (30 frames/sec), which allows simulation results to be animated directly to video. In some instances, the timestep is lowered, so that more samples are returned from the simulator, and the motions can be examined in more detail. In general, there is no extra expense, except for marginal data storage requirements, involved in using a smaller timestep, because the adaptive step size integrator takes multiple sub-steps per frame to compute the dynamics, in all but the most simple cases. The timestep is specified with the *corpus* command: `dt 0.0333333`.

The adaptive step size integrator has an error tolerance parameter which, in part, controls how many sub-steps the integrator uses to compute the dynamics. This parameter is typically set to 0.0001 in the human figure simulations. This specifies that the difference between the 4th and 5th order solutions for any term in the integrator computations cannot be greater than 0.0001. This is set in *corpus* using: `eps 0.0001`.

Ground collision and contact forces are delivered by the exponential spring and linear damping models. The ground force parameters which are typically used in the human figure simulations are set using the following *corpus* commands (refer to Eq. 72, page 94 and Eq. 77, page 97):

```
groundea 100
groundeB 100
groundb 100
groundfricb 100
# coefficient of friction: 1
groundfric 1
```

The gravitational acceleration constant is -9.81 m/sec^2 in the default *corpus* environment matching the real terrestrial value. *Corpus* uses the MKS (meters-kilograms-seconds) convention, so that length units are in meters, masses are in kilograms, etc. The gravitational constant can be set to any acceleration using the `setgrav` command, and the global application of gravitational forces is controlled with the `grav on` and `grav off` commands.

The simulations that we will examine first are very basic in nature, and are presented primarily to establish baseline computation rates. They should also help the reader become more familiar with simulation in the *corpus* environment.

Simulation 1: Basic rigid body

This is the simplest type of simulation, which uses a single rigid body for the human figure. To reduce the model to a single body, we first pose the figure into the desired configuration, and then "delete" all of the joints, which merges all of the bodies into a single rigid body. The posture is specified by loading a pre-saved configuration, or by setting it with joint commands. Interactive techniques could be added to simplify the problem of manually forming a posture, or inverse kinematics could be used to assist in the problem, as discussed previously. [Phillips] Programs which interface to knobs or other devices can be interfaced easily to *corpus*, via its parsing system, [McKenna 1992] or they could be added internal to the program itself. A command sequence which poses the figure to the anatomical position, is shown in Script 3. Once we have the desired posture, all of the joints are deleted, using the `deletejoint` command in *corpus*. The system will no longer perform any calculations regarding any of the joints, although the complex shape is still used for collision detection and display.

This rigid body is free to move under the influence of gravity, or any other external force. For the purposes of this simple simulation, no forces are applied, and thus, no motion is induced. Running this most basic simulation allows us to measure a baseline for the computation time involved in the dynamics equations. The script to setup this simulation is as follows:

```
# form anatomical posture, using pre-defined commandlist "anat"
anat
# delete all joints from the figure
commandtree deletejoint **
ground off
```

```
grav off
```

This simulation operates very quickly. In this and the following tests, the simulation was run for 1000 frames, yielding a little more than 33 seconds of simulation time. This simple simulation took 39 seconds of compute time, slightly slower than real-time. The simulation frame update rate was 25.6 frames/sec. Each timestep was calculated using a single step of the integrator; no subdivision occurred because of the trivial, unchanging integration. In fact, because this simulation is so simple, if we specify the timestep to cover the entire simulation time of 33 seconds (`dt 33`), the integrator returns after a single step, computing the solution in a fraction of a second. Nonetheless, this simulation provides us with our baseline simulation update rate for the complex model, of 25.6 frames/sec. Table 5 lists the timing results for this and the following simulations.

Note that, although we have reduced the complex model to a single dynamic object, there is still a significant overhead associated with the complexity of the numerous graphical objects that comprise that object. In a simulation test run with a single body, the simulation update rate was 143 frames/sec, over five times faster. The system could be optimized to remove much of the extra overhead associated with a deleted joint, but some flexibility in the system would be lost. In addition, the issue is not so important, since we are prima-

Script 3: A *corpus* script to pose the human figure to the anatomical position.

```
# define commandlist "anat"
cl anat
joint l_humerus1 q .05
joint r_humerus1 q .05

joint l_humerus2 q .1
joint r_humerus2 q -.1

joint l_humerus3 q 1
joint r_humerus3 q -1

joint l_forearm q -.35
joint r_forearm q -.35

joint l_hand2 q -.3
joint r_hand2 q .3

joint l_hand1 q .2
joint r_hand1 q .2
```

```
joint l_hand3 q -1
joint r_hand3 q 1

joint l_thigh2 q -.05
joint r_thigh2 q .05

joint l_thigh1 q .09
joint r_thigh1 q .09

joint l_talus q -.07
joint r_talus q -.07

joint l_hindfoot q -.04
joint r_hindfoot q -.04

# update the figure's posture
xform
.
```

Table 5: Computation times for basic simulations.

33.3 seconds of simulation time are generated, from 1000 frames of 0.0333 seconds each.

Simulation	*compute time (sec)*	*frames/ sec*	*sec/frame*	*integrator steps/ frame*	*integrator steps/sec*	*compute time / real-time*
1 - basic simulation	39	25.6	0.04	1	25.6	1.17
2 - gravity	39	25.6	0.04	1	25.6	1.17
3 - ground detection	55	18.2	0.06	1	18.2	1.65
4 - ground contact	850	1.2	0.85	11	13.2	25.5
5 - basic w/ subdivision	690	1.4	0.69	20	28.0	20.7
6 - fast rendering	28	35.7	0.03	-	-	0.84
7 - slow rendering	69	14.5	0.07	-	-	2.07
8 - basic sim w/ fast ren	60	16.7	0.06	1	-	2.0

rily interested in simulations which *do* use the joints, and, as we shall see, the computation time for the joint dynamics dominates when more than a few are included.

Simulation 2: Gravity

The second test is identical to the first, except that the force due to gravity is applied to the figure. The only change in the simulation script is that the command `grav on` was specified. There is no ground or other element to impede the acceleration of the figure. After 33 seconds of simulation time, the body had dropped over 5 kilometers, and was moving downward with a velocity over 300 meters/sec.

This simulation was also trivial for the integrator to compute, and it took a single step per frame, without subdividing. The simulation timings were the same as the first basic test of Simulation 1. The computations required to add in the force due to gravity is trivial in comparison to the main dynamics equations.

Simulation 3: Ground detection

In this simulation test, the ground detection and forces are activated, using the command `ground on`. Gravity is not applied, however, so that the body remains in place, "hovering" over the ground. This test was executed in order to measure the extra computational expense incurred by the ground collision detection algorithm. The timings are presented in Table 5.

The computation time slowed somewhat, yielding an update rate of 18.2 frames/sec, nearly 30% slower than the previous tests. The extra time is due to the ground collision detection, as each body is tested to check if it is penetrating the ground.

The collision detection algorithm uses a bounding box test, to quickly check if the graphical objects might be below the ground. If the values of the *Z* coordinates of an object's bounding box are greater than zero, it is not necessary to examine the actual surface geometry of the object. So, in this case, in which the body is completely above the ground, all of the bodies are trivially rejected. The system could incorporate additional methods to accelerate the computations of trivial rejections. A hierarchy of bounding boxes, encompassing a whole group such as the foot, or the whole body, could be used. In addition, the system could examine the velocities of the bodies to predict when collision detection will actually be necessary. [Dworkin] Because the body in this simulation was unmoving (as was the ground), the system could have bypassed all but the first collision detection pass.

Simulation 4: Ground collision

In this simulation, the ground is "activated" as well as gravity, and the body falls to the ground as a rigid body. This is the first test in which the integrator sub-divides, as the body impacts and is then supported by the ground, and the contact springs and dampers are activated. As a result, a simulation over the same interval takes much longer.

In the simulation, the body falls about 10 cm, starting from an upright, vertical position, with the anatomical posture "hardened" into one rigid object. This posture happens to be one which can stably support the body, i.e. when the feet rest flat in the ground, the COM of the entire body lies between the feet, and it does not topple and fall. After the body is pulled to the ground, it rocks slightly, because the initial orientation of the body was slightly tilted back with respect to the final, resting standing posture. After about 4 seconds of simulation time, the body has stopped rocking (due to damping in the ground). At this point the simulation timing for this test begins.

In the resting, standing state, the integrator took 11 sub-steps per frame, to maintain a stable, unchanging system, as the stiff ground springs pushed up on the figure's feet. For

1000 frames of simulation, corresponding to 33 seconds of simulation time, the computation time was 850 seconds, yielding a frame update rate of 1.2 frames/sec. However, because the integrator took 11 steps per frame, the internal update rate for the integrator was 13.2 steps/sec (1.2 * 11) — much more comparable to the previous test which executed 18.2 steps/sec. The extra computation time is required for the extra collision detection calculations that must be performed on the objects that are penetrating, or are very close to, the ground. The ground force computations also contribute to the slower rate.

Simulation 5: Basic simulation, with forced integrator subdivision

In order to verify that using additional integrator steps per frame does not create any additional overhead, this simple test was executed. The basic example from Simulation 1 was used, but the integrator was forced to subdivide, and use 20 steps/frame. The overall computation rate slowed, of course, by approximately 20 times. However, the integrator step frame rate was slightly faster, at 28 integrator steps/sec, compared to 25.6 steps/sec. There is a slight overhead involved with the internal programming interface to the integrator, which is reduced when the integrator subdivides internally.

Simulation 6: Rendering time, using simple objects

A simple test (not an actual simulation, however) was run to measure the time required to render the geometric human figure model, using the real time graphics hardware provided by the Silicon Graphics *Onyx* workstation with *Reality Engine*^{2} graphics. The scene was composed so that the entire body model was visible, filling the frame vertically. This first test timed the rendering speed when the basic objects in the simple “inertial skin” layer were displayed. There are only eight polygons for most of the objects, and the total polygon count is 1160. The rendering performance is fairly impressive, at 35 frames/sec, yielding 40 thousand anti-aliased polygons/sec.

Simulation 7: Rendering time, using smooth objects

This test is identical to the previous test, except that the smoother, more complex geometric objects were used for display of the human model (the “display skin”). This model incorporates a total of 10420 polygons. The update rate was reduced to 14.5 frames/sec, however the system drew 151 thousand anti-aliased polygons/sec, showing a greater overall efficiency, in terms of polygon rendering speed. This is presumably because the rendering hardware processing pipelines are used more effectively.

Simulation 8: Basic simulation, with rendering

In this test, each simulation step was accompanied by a rendering step, essentially combining Simulation 1 with Simulation 6. The simple "inertial skin" objects were used for rendering. The overall update rate was lowered because more work was done. However, the overall computation time for this simulation was less than the sum of the times for simulation and rendering performed individually, by approximately 11%. The final stages of rendering can be completed using the rendering hardware alone, allowing the simulation to begin in parallel. The system could be modified to render completely in parallel with the simulations, using another CPU in the system. In general, the simulations are too slow to make this of real benefit.

Simulation 9: Add joints

Simulation 9 actually represents several simulations, in which more and more joints are successively added to the articulated figure, and the computation time is measured. The joints are "added" by removing them from a list of joints which are deleted from the default, fully complex figure. In the final simulation, no joints are deleted. The joints are completely unpowered, and passive, and no external forces are active. In all simulations, the integrator took a single step per frame, with no subdivision.

Table 6: Results from Simulation 9. The computation time for increasing number of joints.

number of joints	*compute time (s)*	*frames/sec*	*sec/frame*	*compute time / real-time*
0	39	25.6	0.04	1.2
1	46	21.7	0.05	1.4
2	53	18.9	0.05	1.6
4	55	18.2	0.06	1.7
8	82	12.2	0.08	2.5
16	146	6.8	0.15	4.4
32	277	3.6	0.28	8.3
64	554	1.8	0.56	16.6
84	762	1.3	0.76	22.9

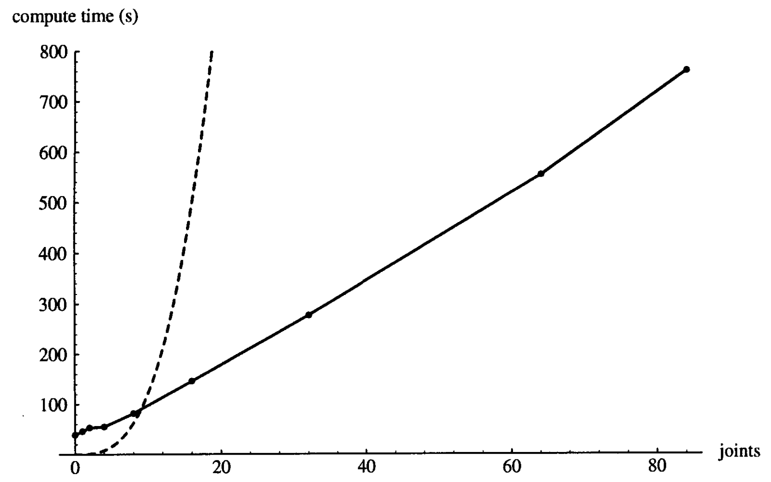


Figure 42: Computation time for simulation vs. the number of joints included in the articulated figure.

The plotted points, connected by the solid line, show the results of the simulation timings, from Table 6. The expense is linear with the number of joints, after the first few are included.

The simulations ran for 1000 frames, with a timestep of 0.03333. This yields 33.33 seconds of simulation time.

The dashed line shows a projection of the computation time for an order $O(n^3)$ system, such as the Walker-Orin method, which is efficient for figures with few joints. [Walker] The projection is based on Featherstone's calculation that the ABM becomes more efficient when more than 9 joints are included. With all 84 joints included, the Walker-Orin method would take nearly 100 times longer than the ABM. This emphasizes the importance of using a $O(n)$ simulation method with the complex model.

The timing results of the simulation are given in Table 6. When the first few joints are added, the computation time increases only marginally, since the equations for the root motion dominate. After approximately eight joints are added, the computation time of the joint accelerations dominates. From that point on, the computation time grows linearly with the number of joints, as is to be expected using Featherstone's recursive *Articulated Body Method.* After all joints are added, a single step of the integrator takes nearly one second (0.76 sec). The computation time vs. the number of joints is plotted in Figure 42.

Simulation 10: Inverse dynamics

In another simple simulation, inverse dynamics, instead of forward dynamics, was computed at all joints. The joints were kinematically controlled using the `jointmotion` command, and since there was no velocity or acceleration at the joints in the default initial state, a motionless state was maintained. The commands to generate this simulation are given as:

```
grav off
```

```
ground off
commandtree jointmotion ** kinematic
```

The compute time for 1000 steps was 714 seconds, slightly faster than the forward dynamics time of 762. The inverse dynamics equations are slightly less complex than the forward dynamics, yielding approximately a 6% increase in efficiency.

Simulation 11: Control equations

This simulation was used to time the additional overhead which arises from computing the typical joint forces used to control the figure, using forward dynamics. At all of the joints, a damper, an exponential spring, and a joint limit exponential spring/damper pair were activated. The control constants and the timestep size were set very low, to avoid any stiffness issues that would cause the integrator to sub-divide. The script for this simulation is:

```
grav off
ground off
# activate damper (1), exp spring (16) and exp joint limit (32) = 49
#   for all bodies in the articulated figure
commandtree joint ** Q_type 49
commandtree joint ** b .00001
commandtree joint ** ea .00001
commandtree joint ** eB .00001
commandtree joint ** e_q .001
commandtree joint ** jlim_ea1 .00001
commandtree joint ** jlim_ea2 .00001
commandtree joint ** jlim_eB1 .00001
commandtree joint ** jlim_eB2 .00001
commandtree joint ** jlim_b1 .00001
commandtree joint ** jlim_b1 .00001
commandtree joint ** jlim_q1 .001
commandtree joint ** jlim_q2 .002
dt 0.0000001
```

The addition of the three joint force functions at each joint slowed the simulation only slightly, by approximately 2%, retaining an update rate of 1.3 frames/second. The control is very fast, computationally; the force calculations are negligible compared to the dynamics equations. The control parameters may need to be calibrated using inverse dynamics and inverse control (discussed below), which is considerably slower than the forward control computations (not including the forward dynamics). However, the inverse computations are still quite fast in comparison to the forward simulation, because the equations are much less “stiff” and fewer integrator steps are required.

Simulation 12: Droop with joints

At this point we begin to examine more interesting simulations, which involve motion and control. First we will examine a number of passive simulations, which do not employ higher level motor control.

A set of simulations were run to examine the general behavior of the human model's biomechanical parameters, such as the dampers and joint limits. One class of these simulations are termed "drooping" tests, in which the abdomen (the central "root" body in the figure) is held rigidly in place, and the limbs of the body are allowed to fall passively, or "droop," under the influence of gravity. Dampers and joint limit exponential spring/damper pairs influence the resulting motion. This type of test can reveal much regarding the properties of the specified parameters. For example, an order of magnitude change in a parameter such as a joint damping constant almost always creates a gross change in the motion, which can easily be seen when animated. Direct comparisons to a real human body "drooping" was not performed. Nonetheless, whole ranges of values for the biomechanical parameters can be ruled out through simple simulations. For example, the damping constant for a limb can be varied, and through animation the changes can be examined, ruling out most values as far too weak or strong.

A command sequence to set up an example droop simulation, which uses all of the joints except for the ones in the foot below the ankle, is given as follows:

```
rotate abdomen y -45
# after transforming the graphical object, inform simulator
setrootpos
# lock abdomen in place
rootmotion fixed
grav on
ground off
foreach list-of-foot-bodies deletejoint **
```

By default, the biomechanical model is passive, using dampers and joint limits at the figure's joints, so this is not specified in the script. The resulting passive motion is shown in Figure 43. This simulation is quite rapid, using approximately 6 integrator steps per frame. With the 32 included joints, the simulation runs at about 0.6 frames/sec.

When the joints in the foot are included, the simulation slows considerably, not only because more joints need to be computed, but primarily because the stiffness of the system

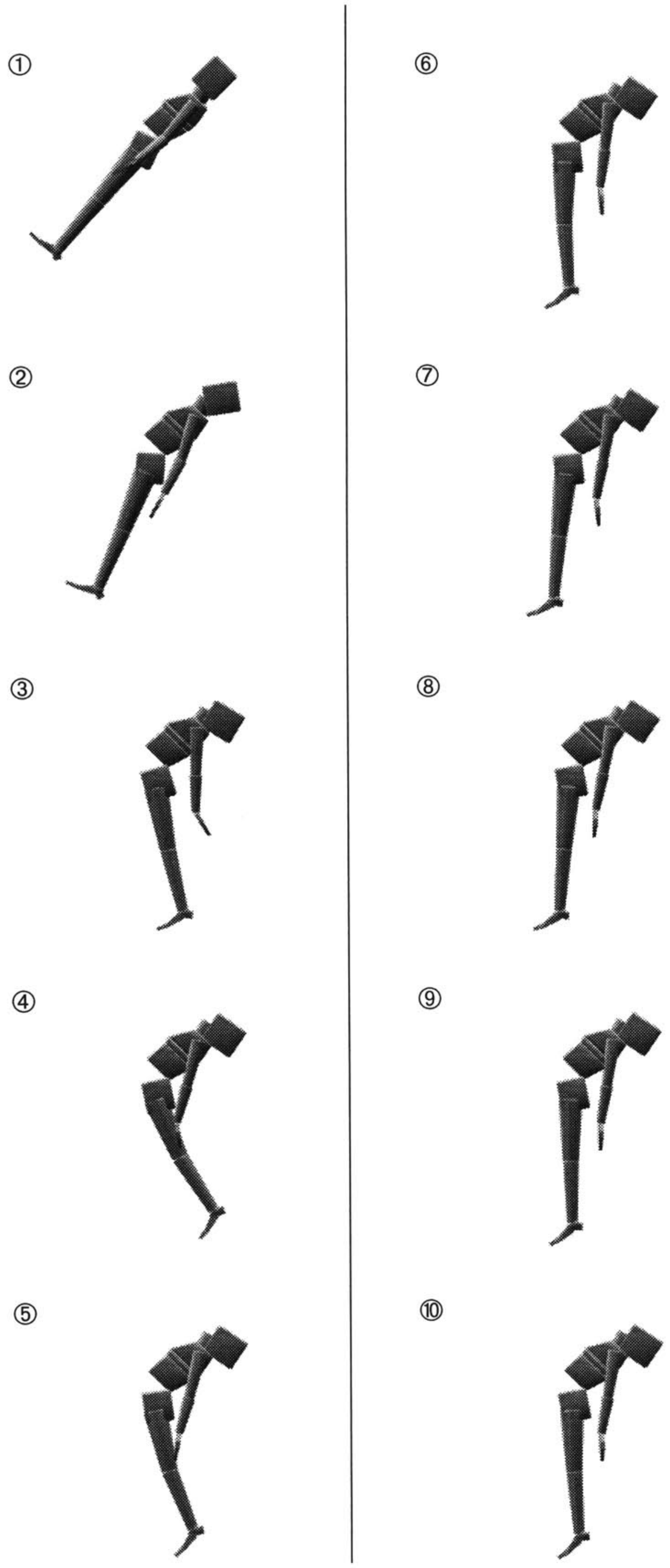


Figure 43: An animation sequence of the human figure “drooping” passively under the influence of gravity.

The torso is held in place, and the limbs are pulled down by gravity. The motion is resisted by the internal dampers and joint limits.

increases, and more sub-steps must be used in the integrator. Adding the five joints at the proximal end of the metatarsals, for a total of 42 joints in the figure, drives the simulator to take approximately 27 steps/frame, when their default dampers and springs are included. The corresponding simulation update rate falls to 0.1 frames/sec (10 sec/frame). Adding the flexion/extension joints at the proximal heads of the proximal phalanges (upper "toes"), increases the stiffness of the system again, so that the integrator takes approximately 56 integrator steps per frame, taking approximately 26 seconds per frame. Adding the abduction/adduction joints at the proximal heads of the proximal phalanges does not increase the integrator stiffness, but slows the computation linearly with the 10 added joints. Adding the medial and distal phalanges (the rest of the toes) slightly increases the stiffness, so that 61 integrator steps per frame are taken, slowing the computation to about 44 seconds/frame. The final joints are at the proximal ends of the navicular and cuboid in the foot. These joints increase the stiffness of the system somewhat, causing the integrator to use 79 steps/frame. The simulation timings are provided in Table 7.

Table 7: Results from Simulation 12. The computation time for increasing number of joints, while drooping passively.

number of joints	*integration steps/frame*	*frames/sec*	*sec/frame*	*integrator steps/sec*	*compute-time / real-time*
32- w/ major joints	6	0.60	1.7	3.6	100
42- w/ metatarsals	27	0.10	10	2.7	300
52- w/ proximal phalanges	56	0.039	26	2.2	770
62 - w/ proximal phalange abductors	56	0.034	29	1.9	880
80 - w/ medial and distal phalanges	61	0.023	44	1.4	1300
84 - w/ navicular and cuboid	79	0.018	55	1.4	1670

The small bodies in the foot have much lower mass than the larger bodies in the figure, and so they are much more sensitive to applied forces. Their associated spring and damper constants are much lower than those of the larger bodies. Nonetheless, because they have lower mass, they will oscillate under the influence of the spring, or gravity, with a higher frequency. Consider a short pendulum which swings rapidly vs. a long pendulum which swings slowly. The integrator must sample the motion more finely, because they move more rapidly, and with greater accelerations and changes in accelerations.

Simulation 13: Falling to the ground

Another very basic kind of simulation which can be executed is letting the figure fall passively to the ground. In a manner similar to the "drooping" experiments, these simple tests can be very informative regarding the damping, joint limit, actuator, and ground force parameters, as the model is developed.

In some tests, the figure falls passively, without the actuator springs included as force generators. In other simulations, the actuators are activated, but they are not calibrated to support the posture, and the figure topples. These latter experiments allow for a visual inspection of the effects of given actuator stiffnesses. An image sequence of the foot shape bending as the body falls to the ground and tips forward, with its actuators active but uncalibrated, is shown in Figure 44.

Simulation 14: Motor programs in zero gravity

In this example animation, we add active control of the figure's motion, using the exponential spring joint actuators. The figure hangs in zero gravity, and uses motor programs to move the spring rest positions to create motions in the figure's limbs. One motor program set is designed to draw the limbs inwards, and another extends the limbs outwards.

The joints in the foot, below the ankle are deleted, to simplify the simulation. The integrator took 6 steps per frame, with 26 joints included. The update rate was 0.77 frames per second.

A script which specifies the motor programs used to generate the limbs motions is provided in Script 4. An animation sequence of the body motions is shown in Figure 45.

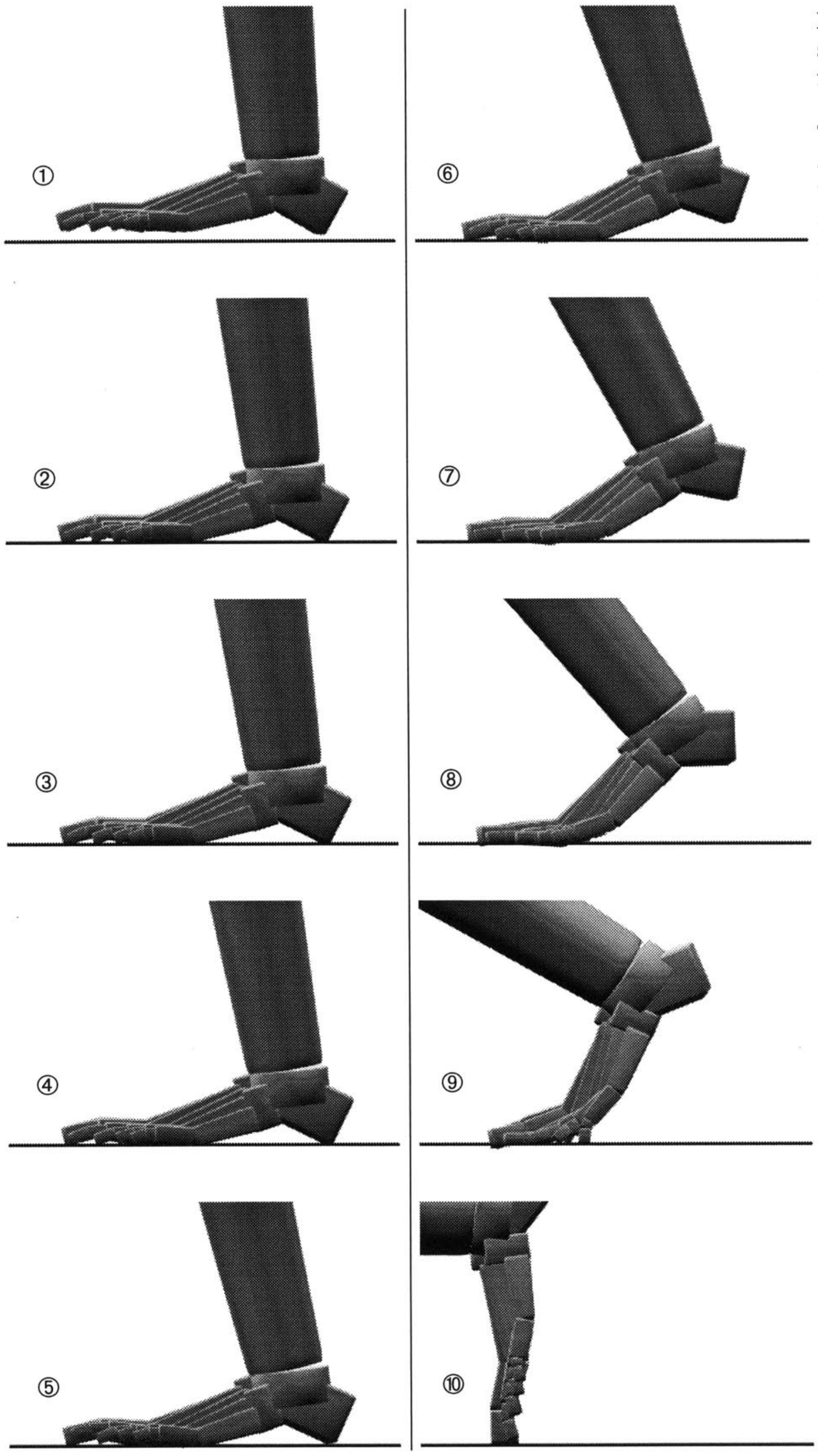


Figure 44: The foot shape as the body falls forward.

The joint actuators are activated, but their rest angles are simply set to match the current joint angles. Because the springs are not "preloaded" to support the body weight without first flexing somewhat, a balanced posture is not maintained, and the body pitches forward and falls.

Script 4: A script to generate motions of the limbs in zero gravity.

```
# add in ex spring (16) to 33
# (damper + exp joint limit)
commandtree jointmatchexp **

# first raise arms out
motor l_humerus1 etarget 0 1.0
motor l_humerus2 etarget 1.3 1.0
motor l_humerus3 etarget 0 1.0
motor l_forearm etarget 0 1.0

motor r_humerus1 etarget 0 1.0
motor r_humerus2 etarget -1.3 1.0
motor r_humerus3 etarget 0 1.0
motor r_forearm etarget 0 1.0

grav off
ground off

cl move-in
motor l_forearm etarget -2.5 1
motor r_forearm etarget -2.5 1
motor l_thigh1 etarget -2 1
motor r_thigh1 etarget -2 1

motor l_shank etarget 2.5 1
motor r_shank etarget 2.5 1

motor pelvis1 etarget -0.5 1
.

cl move-out
motor l_forearm etarget 0 1
motor r_forearm etarget 0 1

motor l_thigh1 etarget 0.09 1
motor r_thigh1 etarget 0.09 1

motor l_shank etarget 0 1
motor r_shank etarget 0 1

motor pelvis1 etarget 0 1
.
```

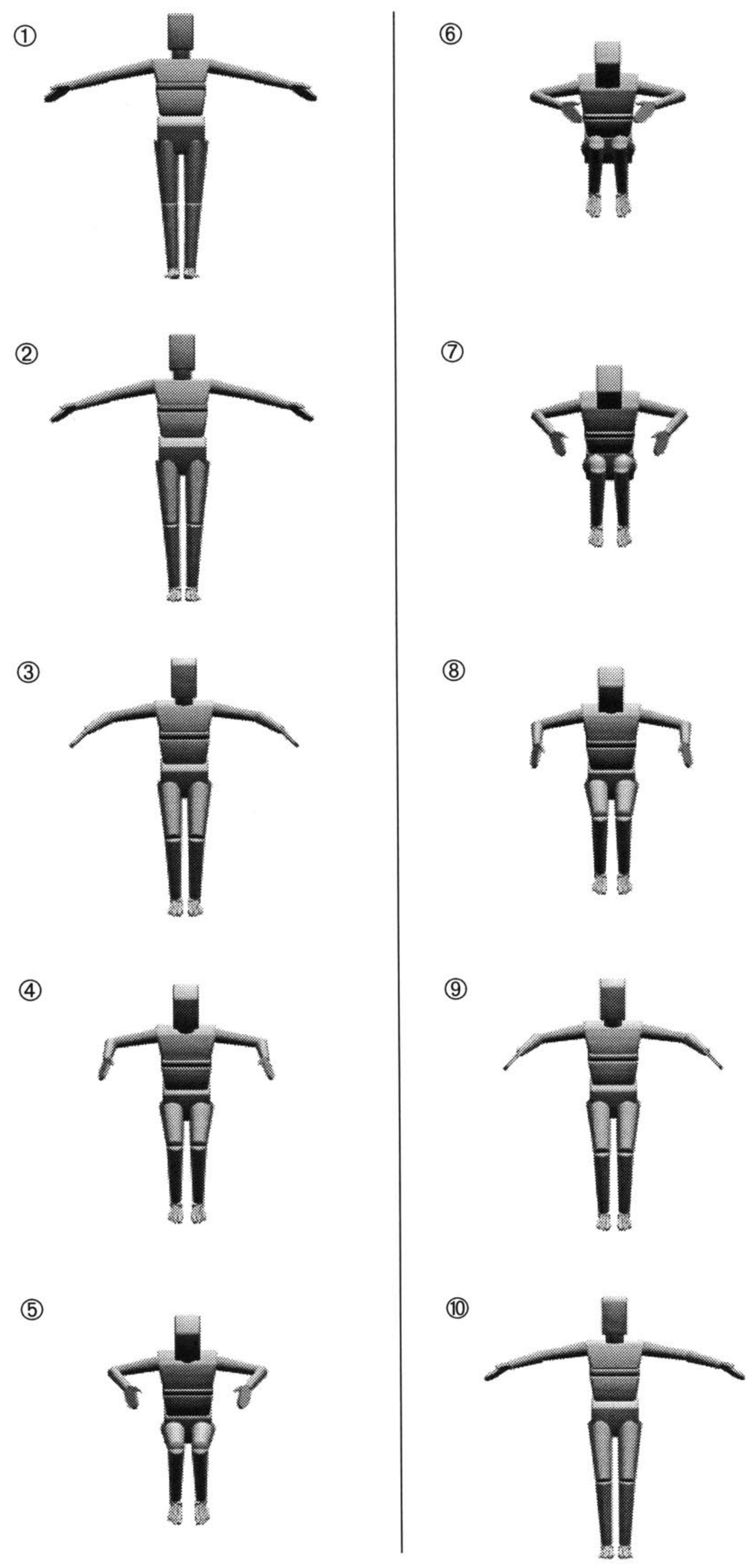


Figure 45: An "exercise" animation sequence using the human figure model.

The figure hangs in zero gravity, and uses motor programs to draw the limbs inwards then extend them outwards.

Script 5: A *corpus* script which defines motor programs to "wiggle" the toes.

```
cl extend
motor l_phal1.2 etarget -.9 .5
motor l_phal2.2 etarget -.6 .5
motor l_phal3.2 etarget -.5 .5
motor l_phal4.2 etarget -.45 .5
motor l_phal5.2 etarget -.41 .5

motor l_phal1.1 etarget -.05 .5
motor l_phal2.1 etarget .05 .5
motor l_phal3.1 etarget .1 .5
motor l_phal4.1 etarget .15 .5
motor l_phal5.1 etarget .15 .5

motor l_toe1 etarget .2 .5
motor l_toe2.1 etarget .2 .5
motor l_toe3.1 etarget .2 .5
motor l_toe4.1 etarget .2 .5
motor l_toe5.1 etarget .2 .5

motor l_toe2.2 etarget .2 .5
motor l_toe3.2 etarget .2 .5
motor l_toe4.2 etarget .2 .5
motor l_toe5.2 etarget .2 .5
.
```

```
cl flex
motor l_phal1.2 etarget .9 .5
motor l_phal2.2 etarget .9 .5
motor l_phal3.2 etarget .6 .5
motor l_phal4.2 etarget .4 .5
motor l_phal5.2 etarget .4 .5

motor l_phal1.1 etarget .05 .5
motor l_phal2.1 etarget -.1 .5
motor l_phal3.1 etarget .0 .5
motor l_phal4.1 etarget -.05 .5
motor l_phal5.1 etarget -.075 .5

motor l_toe1 etarget .9 .5
motor l_toe2.1 etarget .8 .5
motor l_toe3.1 etarget .8 .5
motor l_toe4.1 etarget .8 .5
motor l_toe5.1 etarget .8 .5

motor l_toe2.2 etarget .8 .5
motor l_toe3.2 etarget .8 .5
motor l_toe4.2 etarget .8 .5
motor l_toe5.2 etarget .8 .5
.
```

Simulation 15: Motor program to wiggle the toes

This simulation is similar to the previous. Motor programs are used to move the phalanges, flexing and extending the joints, with small amounts of abduction and adduction. The resulting motion is similar to "wiggling" the toes. The motor programs and initialization commands are given in Script 5.

The joints not in the left foot were deleted to simplify. Using the 26 joints in the left foot, the integrator took 54 steps per frame. The update rate was 0.083 frames per second, or 12 seconds per frame. The animated motions of the toes are shown in Figure 46.

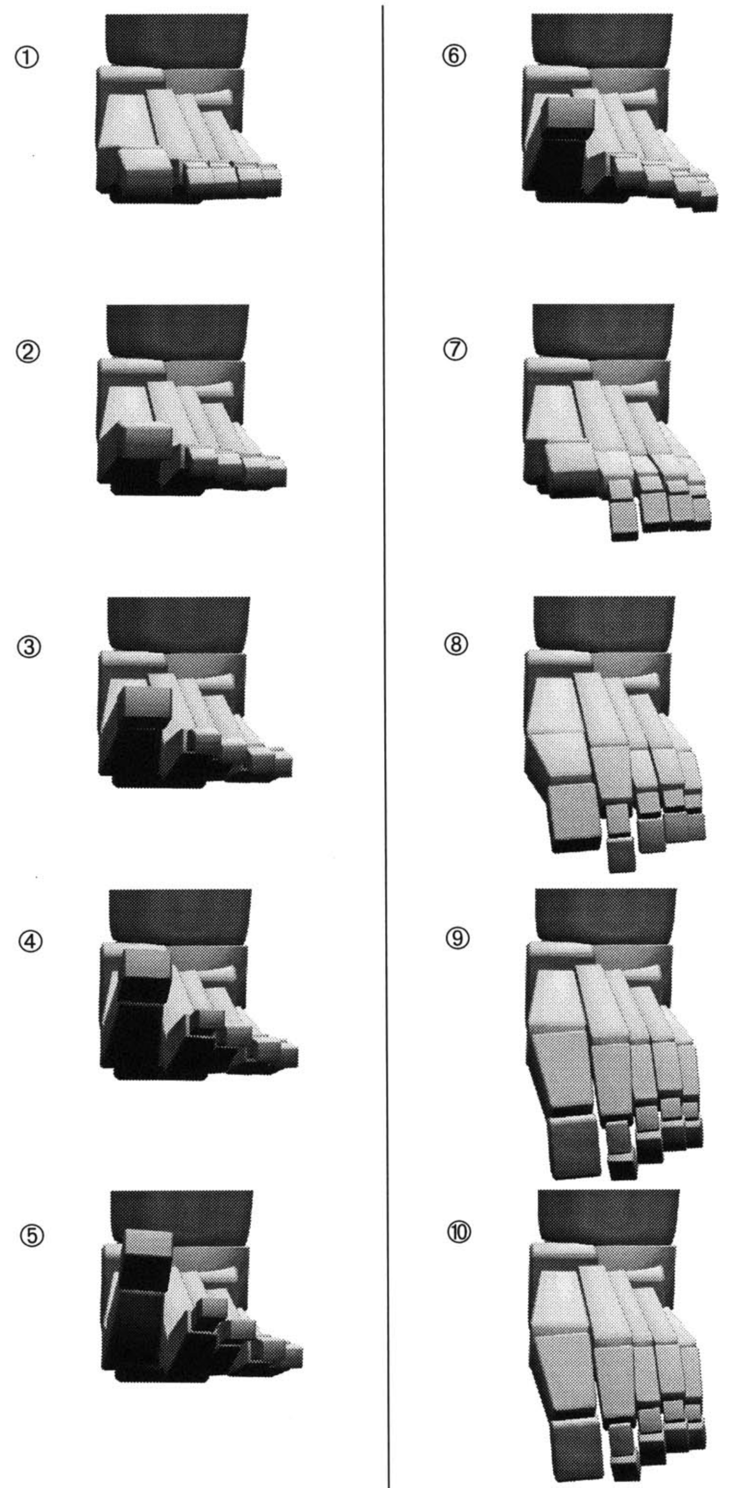


Figure 46: A motion sequence of "wiggling" the toes of the left foot, driven by motor programs.

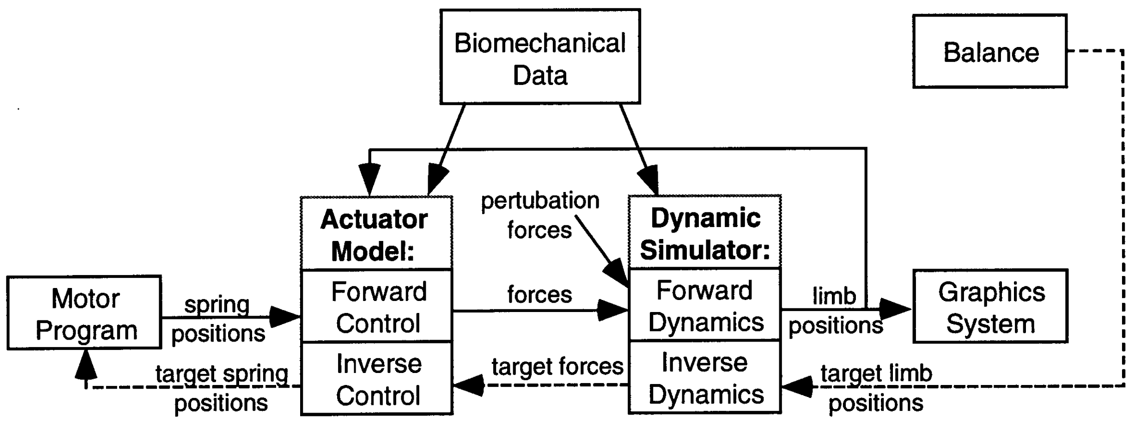


Figure 47: Block diagram of dynamic postural control.

The dashed lines represent control paths which are traversed less frequently than the solid, forward control paths. The 'Balance' block supplies a definition for a stable kinematic posture, and feeds the target posture into the dynamics simulator. Inverse (hybrid) dynamics is computed using the limb configuration and environmental forces to compute the joint forces required to achieve that posture. Inverse control then computes the actuator parameters which will deliver those forces. Motor programs mediate between the target actuator parameters and the current ones. The forward control loop takes the current actuator parameters, computes the joint forces, forward simulates the figure motion, and feeds the limb configuration back to the actuators to compute an updated force.

7.2 Standing Posture

The general approach to controlling a stable, balanced posture was covered in the **Approach** section (3). There are three main components of the static balance system. The first part defines a kinematic description for the figure, such that it has a stable posture. The second part uses hybrid dynamics and inverse control to calibrate the actuators to the specified posture. The third major component is the actual execution of balance through force application; the actuators control and maintain the posture by generating force, and the dynamics simulator generates the motion (or lack of motion). The actuators employ feedback which allows the system to adapt to and reject errors and perturbations to the system, to a limited degree. The balancing mechanism is depicted in block-diagram form in Figure 47.

The first major step in the calibration is to define the kinematic posture, in this case, a standing posture. This component could use kinematic analyses to determine the joint angles needed to maximize the stability of the figure, by centering the COM of the entire body within the support polygon formed by the points of contact with the ground, as the *Jack*™ system does. [Phillips] The main goal of this system is to maximize the *stability* *margin*, which is defined as the shortest distance from the projection of the COM onto the support surface to the boundary of the support polygon.

In this work, stable postures were interactively generated using the scripting language in *corpus* to create joint configurations which placed the COM of the figure within the region of support formed by the feet. To aid in this task, a COM marker was added to the system, as a visual guide to stability. The COM of the entire body is the average of the COMs of all the segments, weighted by their mass:

$$\mathbf{c} = \frac{\sum_{1}^{n} \mathbf{c}_i \, {}_i X_{\mathrm{w}} \, m_i}{n} , \qquad \text{Eq. 91}$$

where **c** is the three dimensional row vector defining the location of the body COM in world space, $\mathbf{c}_i$ is the (static) COM of body *i* in its local space, ${}_iX_w$ is the 3x3 matrix which transforms values from the space of body *i* to world space, and m_i is the scalar mass of body *i*.

Once a posture has been defined, the second step is to calibrate that posture for the control system. Hybrid dynamics uses the defined kinematic posture to compute the required joint forces, and inverse control then uses those forces to compute the required actuator parameters. The analysis phase employs the dynamics simulator to perform its calculations. However, this phase is "outside of" the normal, forward dynamics simulation which animates the figure; the simulation time spent in the analysis in not included in the overall simulation of the biped. The hybrid analysis could be considered a sort of detailed "thought experiment" performed by the control system.

The calibration process uses hybrid dynamics rather than inverse dynamics, because the overall body motion is unconstrained, and is forward simulated. The joints however, are kinematically constrained, and forces are computed using inverse dynamics. Some joints could be left to be forward simulated as well, with their posture being defined by the actuators at the joints and the other applied forces.

The hybrid analysis operates as follows: the joints are "locked" into the specified kinematic posture by setting the joint angles to the given target, and by setting the joint velocities and accelerations to zero. The figure then becomes essentially one rigid object. The motion of the entire body remains unconstrained, in 6D, and is simulated using forward dynamics. Gravity "pulls" the figure downwards, and support and friction forces are generated at the contact between the body and ground. This figure "settles" to the ground, coming to rest. Once the figure is unmoving, it is in a static case with respect to the external forces. There are small initial settling motions because it is difficult to place the figure exactly on the ground, in its final resting state, with the correct configuration of the feet touching the ground surface (and slightly penetrating by specific amounts at different points on the feet).

If the system is already in an operating state, the system does not have to "settle" before the hybrid analysis can begin. For example, if a standing posture, with its associated control state, is already defined, hybrid dynamics could be used to determine the control parameters to achieve an acceleration goal for a limb. The system does not need to settle because it is already in the proper starting configuration for the new goal.

During the hybrid simulation, joint forces are computed at the locked joints. These forces are integrated using the numerical integrator, in order to obtain a more stable "average" joint force, as described previously, in the sub-section **Hybrid Dynamics** (5.5) and Eq. 59, page 89. The forces are integrated over a short period of simulation time (typically one second).

Once a stable joint force has been computed, inverse control is used to compute the required "target" actuator parameters, as in Eq. 82, page 101. Motor programs mediate between the target actuator parameters and the current ones, so that the previous control state is interpolated to the newly determined state. The new actuator state can also be set instantaneously, which is the method used if no valid control state existed previously.

After the control system has been calibrated, forward dynamics can be used to simulate the figure. The actuators employ feedback which allows the system to adapt to and reject errors and perturbations to the system, to a limited degree. A certain range of error is introduced in the dynamic simulation due to numerical instabilities in the force functions, limited sampling of the dynamics function by the numerical integrator, and ultimately the numerical limitations of the machine. The actuators also allow for the adaptation to perturbation forces applied to the biped. When a perturbing force is applied, the figure will deviate from the specified posture. The magnitude of the deviation depends on the stiffness of the actuators, and the magnitude of the force. Forces above a certain level will cause the COM to move beyond the support polygon, and the biped will fall. When the force is removed, the actuators will return the figure to the original posture, assuming it has not fallen. The actuators are analogous to skeletal muscle with their proprioceptive feedback allowing for length and stiffness regulation.

In previous work by the author, simulating the locomotion of a hexapod, [McKenna 1990-B; McKenna 1990-C] a trial and error method was used to determine the actuator control parameters. The hybrid analysis system represents a significant advance over that method, because more control parameters are automatically determined, with much greater accuracy. An important issue from the previous work is that forward dynamics, rather than inverse dynamics, should be the means of motion production in the simulation. The concern is that motions and postures should not be overly constrained or prespecified, or much of the motivation for performing a detailed dynamic simulation will be lost. If motions are already kinematically specified, there is little point in dynamically simulating them for the purposes of animation. This philosophy remains important in this work. An emphasis is placed on producing motion using forward simulation. Inverse and hybrid analyses are used in lieu of trial and error methods. It should be noted that in some cases it is more beneficial to apply kinematic control, and in many case, the control is greatly simplified using kinematic constraints. Inverse dynamic control of some joints can also reduce the stiffness of the system, decreasing computation time. Because *corpus* allows for a mix of dynamic and kinematic control, the flexibility is provided to try different approaches for control.

The "anatomical position" posture for the human figure model that was set to match the Goldfinger illustrations turned out to be a stable, balanced posture which placed the COM of the entire body within the support region. This stable posture, and others, including a low squat were calibrated in the control system, and were maintained over time by the joint actuators when forward simulated. In one experiment, motor programs were used to vary the actuator control parameters from the state for a squatting posture to the state for the upright standing posture, successfully generating a "standing" motion. The motion sequence is shown in Figure 48. In general, a more sophisticated mechanism would be required to ensure that the COM remains within the support region during movements such as the one described. One solution would involve a subdivision of the movement tra-

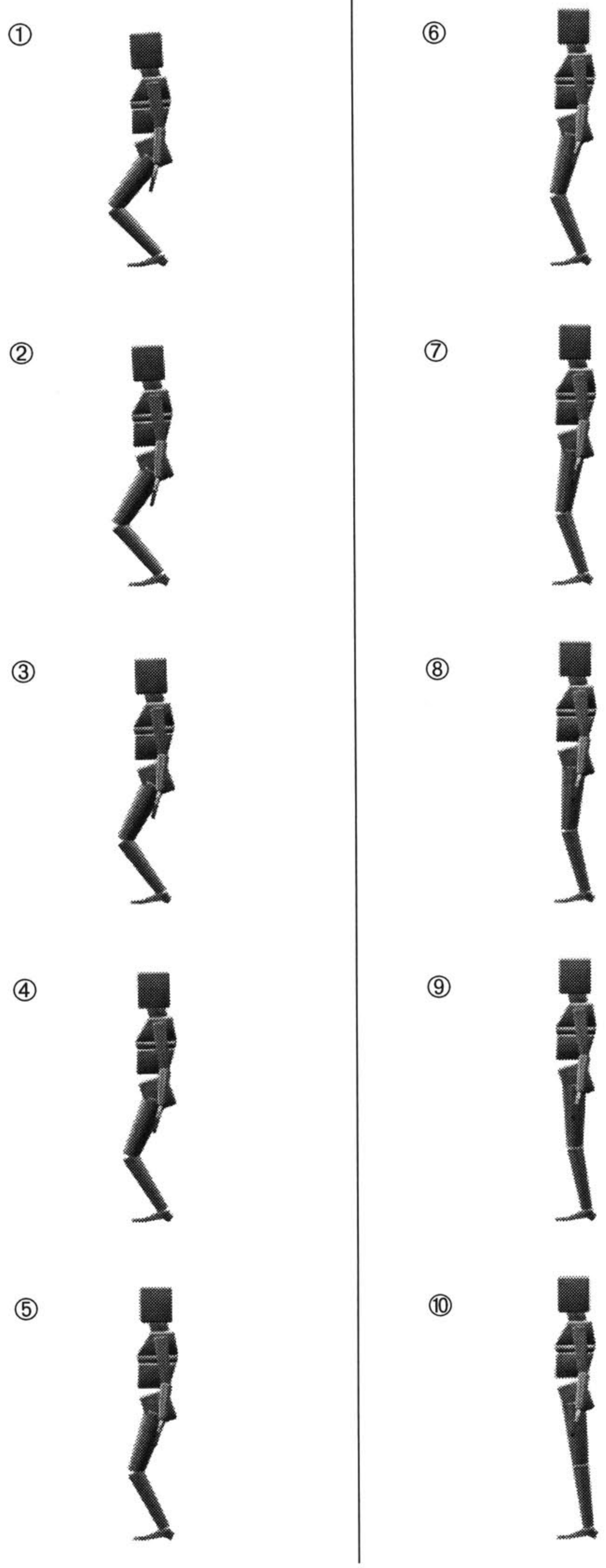


Figure 48: An animation sequence of the human figure rising to a standing posture from a knee bend.

jectory, with the kinematic posture formulation and hybrid analyses performed at several "points" along the way.

Other experiments dealt with the adaptive properties of the postural system to the application of an external force. There were two main types of experiments: ones in which the figure adapted passively using the actuators, and ones in which the control system actively adapted to the force. In all of the experiments, a linear external force was applied to the mid-region of the torso, with magnitudes in the range of 10 N (approximately equivalent in magnitude to the weight of 1 kg, on earth). The direction of the force was horizontal, parallel to the ground.

In the passively adaptive experiments, the body posture shifts, as the force is applied, in the direction of the force. The magnitude of the postural change depends on the stiffness of the actuators (the α and β parameters, Eq. 81, page 99). Beyond a certain level of applied force, the body COM shifts beyond the support region, and the figure falls.

When the complex model of the foot is included, the figure becomes much more sensitive to a perturbation force, and the body moves more. With the foot joints, there are many more places where the stress can be absorbed by spring deflection. In contrast, the rigid foot will supply any force needed to prevent bending. Although the complex foot makes the control task more difficult, the problem is more realistic.

In the actively adaptive experiments, the hybrid analysis takes into account the externally-applied force, so that, after the inverse control calculations, the actuators counteract the perturbing force. In these experiments, the body would move only very slightly in response to the force. It is likely that the small movements that occur are due to the compliant nature of the ground reaction forces. Even with the active control, forces can be applied which would push the figure over. At its best, the figure can respond only as a rigid object, and we all know that rigid objects can be toppled over. One potential drawback of the active adaptation is that it requires knowledge of the specifics of the applied force. In the real world, this information is not directly available to a robot or human, although the information is indirectly available though the various sense organs that are stimulated though the effects of the force (e.g. motion, pressure).

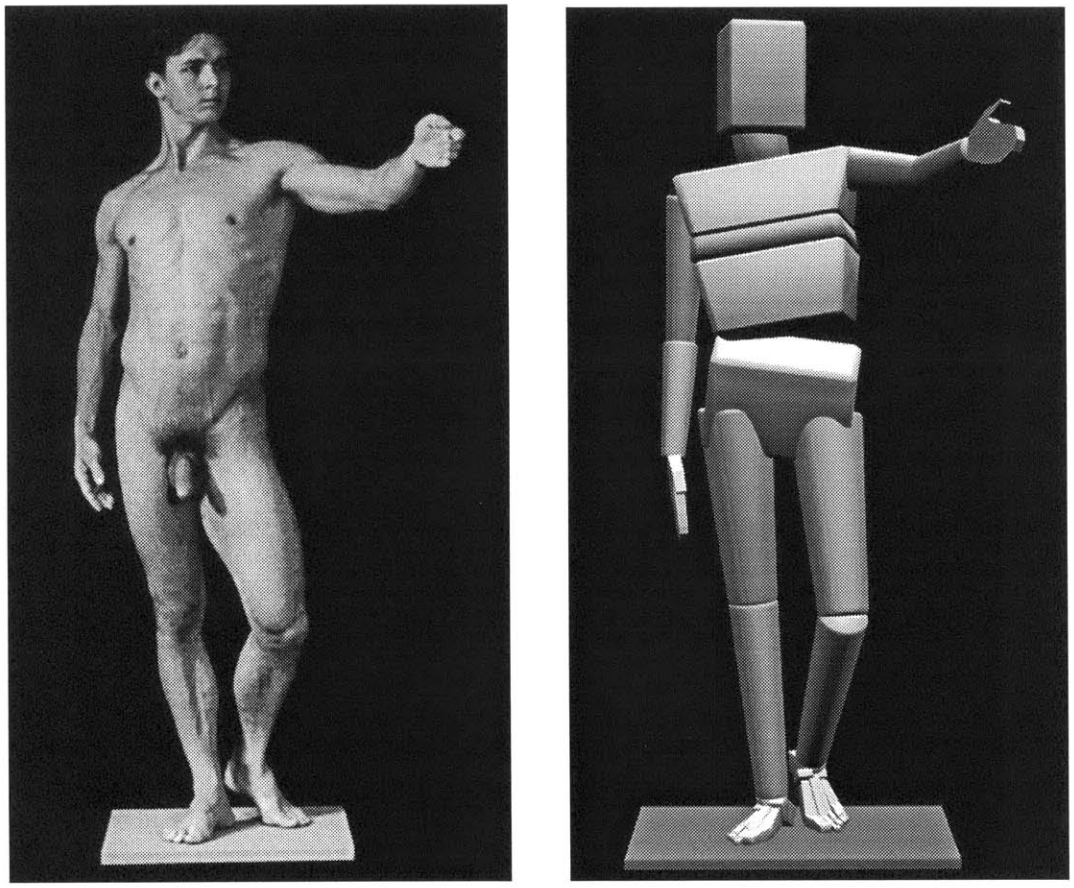


Figure 49: The human figure, in contraposition.

Photo on left from [Goldfinger].

To conclude the discussion on standing postures, the figure in contraposition is presented. “Contraposition” is a posture of the depicted human body, common in late Renaissance sculptures and paintings, in which twisting of the vertical axis of the body results in the head, shoulders, and hips being oriented in different directions. All of the major degrees of freedom are employed to form the pose, and the toes are bent on the figure’s left foot. See Figure 49.

7.3 Reaching Task

A simulation, which could be computed in near real-time, was generated of a reaching task with the arm. The task is to reach out with the left arm to touch, with the tip of the hand, one of five buttons arranged in a two dimensional array. The head also turns to “look” towards the indicated button. The approach for the control is similar to that used for the standing posture simulations. The actuators are first calibrated to a set of key postures, and during the execution of the task, motor programs are used to modify the actuator rest angles over time, from one posture setting to another, to generate the reaching motions.

The kinematic complexity of the model is greatly reduced, for efficiency, by deleting most of the joints. The overall body motion is fixed in place, and six joints are included: three DOFs at the shoulder, and one DOF at the elbow of the left arm. Two DOFs are included in the head/neck joint.

There are two main phases to the overall simulation: a "training" phase and a "performance" phase. The training process is used to form and calibrate key postures. Training occurs only once, before "performance" begins. During performance, forward dynamics is used to simulate the motion of the figure, as it reaches and looks towards the indicated targets.

The goal of the training phase is to generate one posture for each of the buttons, with the tip of the hand touching the given button, and the gaze of the head looking towards that button. Once the posture is specified, the training process calibrates the actuators for that posture, as they resist gravity and support the limb. The training calibration process could use an inverse kinematics technique to form the posture of the limb, as the hand touches a given button. Instead, a dynamic simulation process is used to form these key postures. This simulation process is part of the calibrating phase for the control system, and it is not intended to be a simulation of the figure's motion in the proper sense (for example, there is no gravity during most of the calibrating process).

The simulation and training start from an initial configuration, such that the figure looks straight ahead, with the arm held at its side. This initial posture is shown in Figure 50. A linear spring, in three dimensional space, is used to "pull" the tip of the hand to a given button. One end of the spring attaches to the button, and the other attaches to the tip of the hand. First order dynamics (see **First Order Dynamics**, page 90), rather than standard second order dynamics, is used to simulate the motion of the hand as it is drawn to the target button. First order dynamics allows a rapid descent to the solution, without overshooting the target.

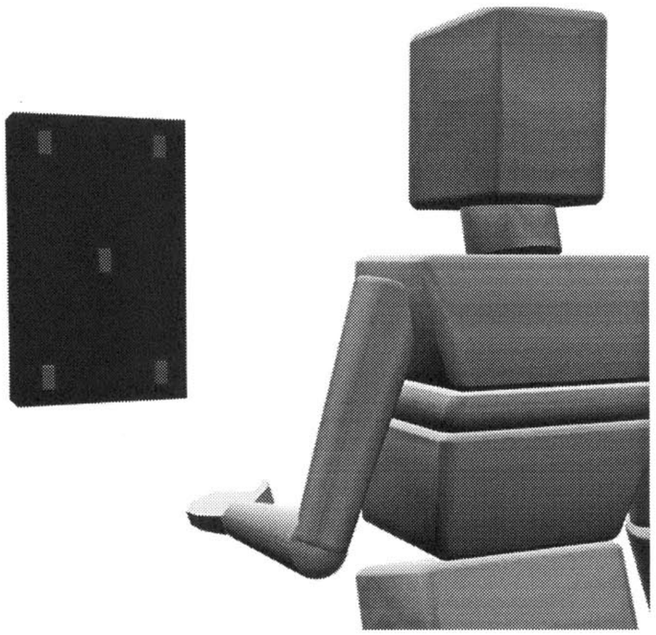


Figure 50: The setup for the reaching task.

This "over the shoulder" shot shows the initial, starting posture. The left arm is used to reach forward to press a specified button from the array of five grey boxes.

During this calibrating simulation, gravity and all internal forces in the arm are inactive. The limb takes a short path from its starting posture to the final posture, at which point the tip of the hand has been drawn almost completely to the target by the spring. The orientation of the head is similarly drawn towards the buttons. A linear spring is attached from the target button to a point in front of the head, similar to a "gaze" vector. Originally, the head motion seemed too exaggerated as it turned to orient the head directly at the target; presumably a person's head would turn towards the target, but their eyes would rotate further to center their gaze on the target. In order to modify the target head postures, springs were included in the head/neck joints, with the rest angles set to the initial, forward looking posture. The linear spring then pulls the head's gaze towards the target, but the neck springs resist, and the head moves until the forces are in equilibrium. The result is that the head turns towards the target button, but without directing the gaze completely at it. The neck springs provide a default posture, towards which the head will be biased. In general, such springs can be used to influence the type of posture that is formed. For example, joint limits can easily be included to keep movements within valid ranges, actuator springs can be used to bias towards a default posture, gravity can be included to influence the posture, and extra external springs (attractors or repulsors) can be used to shape the limb in a desired manner.

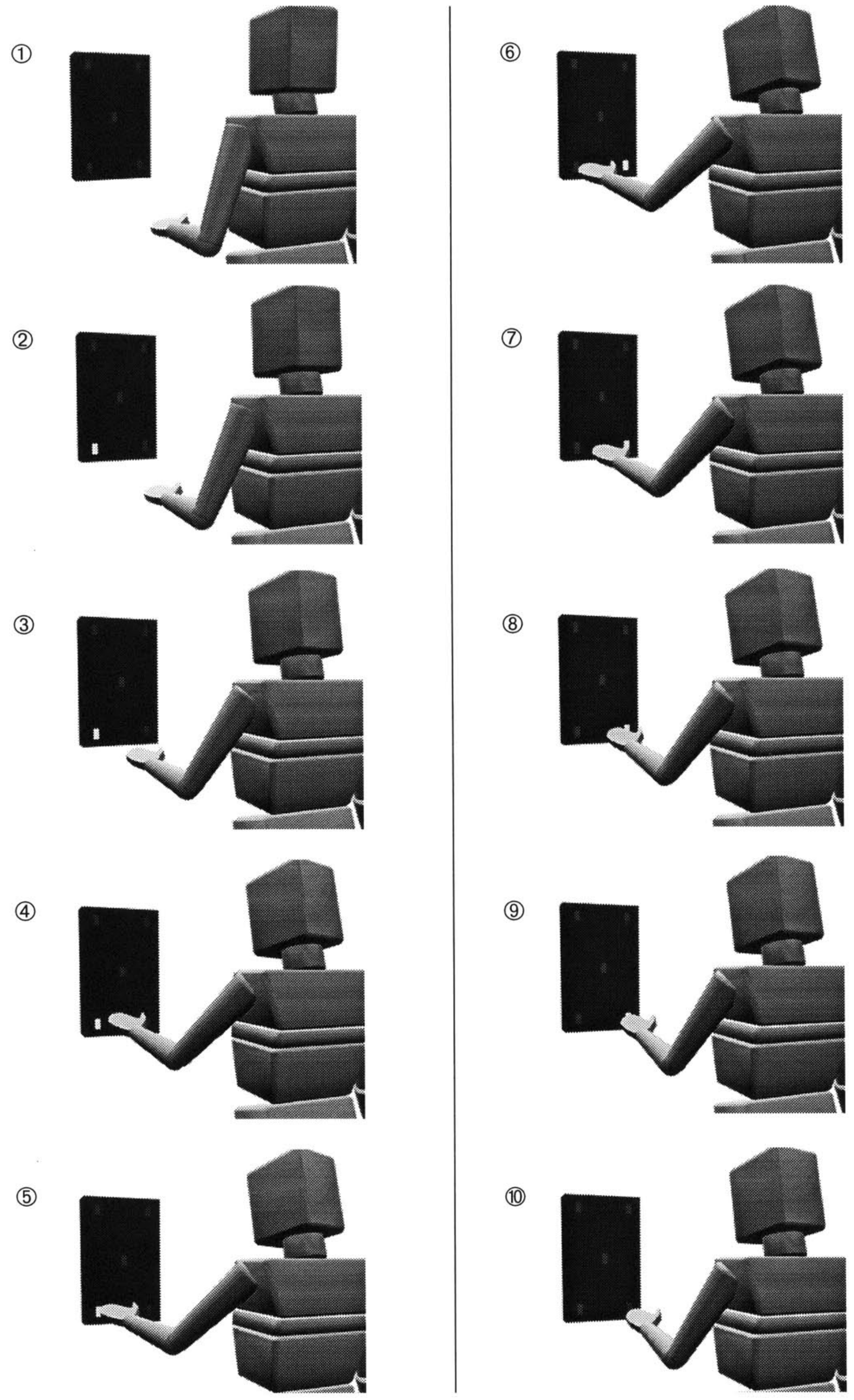


Figure 51: The reaching task simulation, in the performance phase.

As each target is indicated (by turning white), the figure turns the head towards it and reaches to touch it.

After reaching the target, the posture is calibrated for the spring controls, using inverse dynamics and inverse control. Gravity is activated, so that the springs will incorporate that force, and counteract it. The posture for touching each button is calibrated, and the initial posture with the arm at the figure's side is also calibrated.

In the performance phase, the motions of the figure are computed using forward dynamics, as different button targets are indicated, and the figure looks towards and reaches to touch the target. When a button is indicated, the control settings which correspond to the posture that touches that button are selected. If no target is indicated the control settings for the initial posture, with the arm at the side, are selected. Motor programs are then activated to move the spring controls to the selected, pre-calibrated posture. The spring angles move over time, pulling the limb to the target position, and the dampers dissipate the kinetic energy and smooth the motion. An animation sequence of the task is given in Figure 51.

The performance simulation runs at approximately 6.8 frames per second, about 1/4 real-time, including the time for both simulation and rendering. The training process is computed at a similar update rate. The entire training phase lasts approximately 40 seconds.

7.4 Toe Raise Simulation

In this simulation, the human figure is controlled to rise up on its toes, starting from a normal standing posture. Motor programs are used to vary the actuator parameters to generate motion. To simplify the balance problem, the figure is first controlled to raise its arms out in front of the torso, so that it leans against a "wall" in front of it. The proximal phalanges are then hyper-extended to push the foot and body upwards.

Starting from a calibrated standing posture, motor programs are used at the shoulders, elbows and wrists to raise the arms. As the arms are raised, the body's COM shifts forwards, the body tilts forward slightly. The hands touch a "wall" object in front of the figure, and collision and friction forces are exerted at the points of contact, preventing the figure from falling forward. Motor programs are then used at the proximal joints of the proximal phalanges, in both feet, to extend the phalanges, which has the effect of pushing up the hindfoot and the upper body, as the phalanges push down against the ground. There is a motor program which acts at the talar joint as well, to compensate for the change of orientation of the hindfoot as it rises, so that the upper body remains upright. The script for

Script 6: Script to control the human figure to rise on the toes.

```
addcorpus wall
get wall from ../data/unit_cube
facet wall

postmult
move wall .5 0 0
scale wall 100 100 100
addbody wall wall 0 0 1 rotary 1
setroot wall
corpusinit
move wall .73 0 0
setrootpos

rootmotion fixed
integrate wall off

setcorpus biped

collide l_hand3 wall
collide r_hand3 wall
collide l_forearm wall
collide r_forearm wall

collision on
collisionea 100
collisionb 100
collisionfric 1.0
collisionfricb 100
collisioneB 100

cl arms-out
motor l_humerus1 etarget -1 1.5
motor l_humerus2 etarget .4 1.5
motor l_humerus3 etarget -.5 1.5

motor r_humerus1 etarget -1 1.5
motor r_humerus2 etarget -.4 1.5
motor r_humerus3 etarget .5 1.5

motor l_forearm etarget -1 1.5
motor r_forearm etarget -1 1.5

motor l_hand1 etarget -1.5 1.5
motor l_hand2 etarget 0.8 1.5
motor l_hand3 etarget -3.2 1.5

motor r_hand1 etarget -1.5 1.5
motor r_hand2 etarget -0.8 1.5
motor r_hand3 etarget 3.2 1.5
.

cl flex
# move phals up
motor l_phal1.2 etarget -.8 1
motor l_phal2.2 etarget -.8 1
motor l_phal3.2 etarget -.8 1
motor l_phal4.2 etarget -.8 1
motor l_phal5.2 etarget -.8 1

motor r_phal1.2 etarget -.8 1
motor r_phal2.2 etarget -.8 1
motor r_phal3.2 etarget -.8 1
motor r_phal4.2 etarget -.8 1
motor r_phal5.2 etarget -.8 1

# compensate at ankle too
# toe's moved about .8, move the
e_q about .7 on ankle...
motor l_talus etarget .7 1

# and move hindfoot in a little
motor l_hindfoot etarget 0 1

motor r_talus etarget .7 1
motor r_hindfoot etarget 0 1
.
```

the toe raise simulation is given in Script 6. An animation sequence of the body rising is shown in Figure 52. A close-up of the side and front of the foot is shown in Figure 53.

The first simulation of the toe raise resulted in a final posture which employed mostly the big toes and the second toes, rather than rising evenly across all of the toes. Although it is certainly possible to perform a toe raise which places most of the stress on the big toes, it

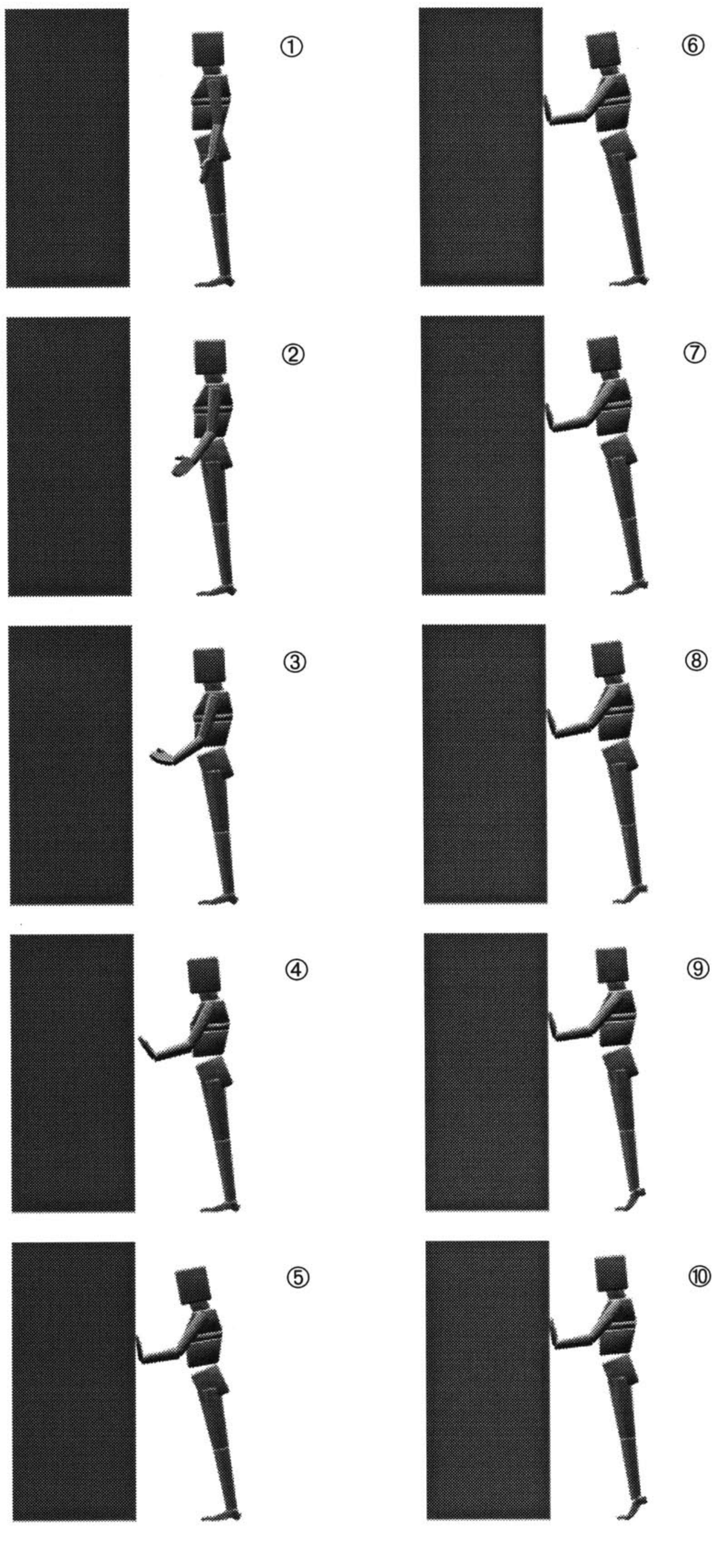


Figure 52: Simulation of rising on the toes.

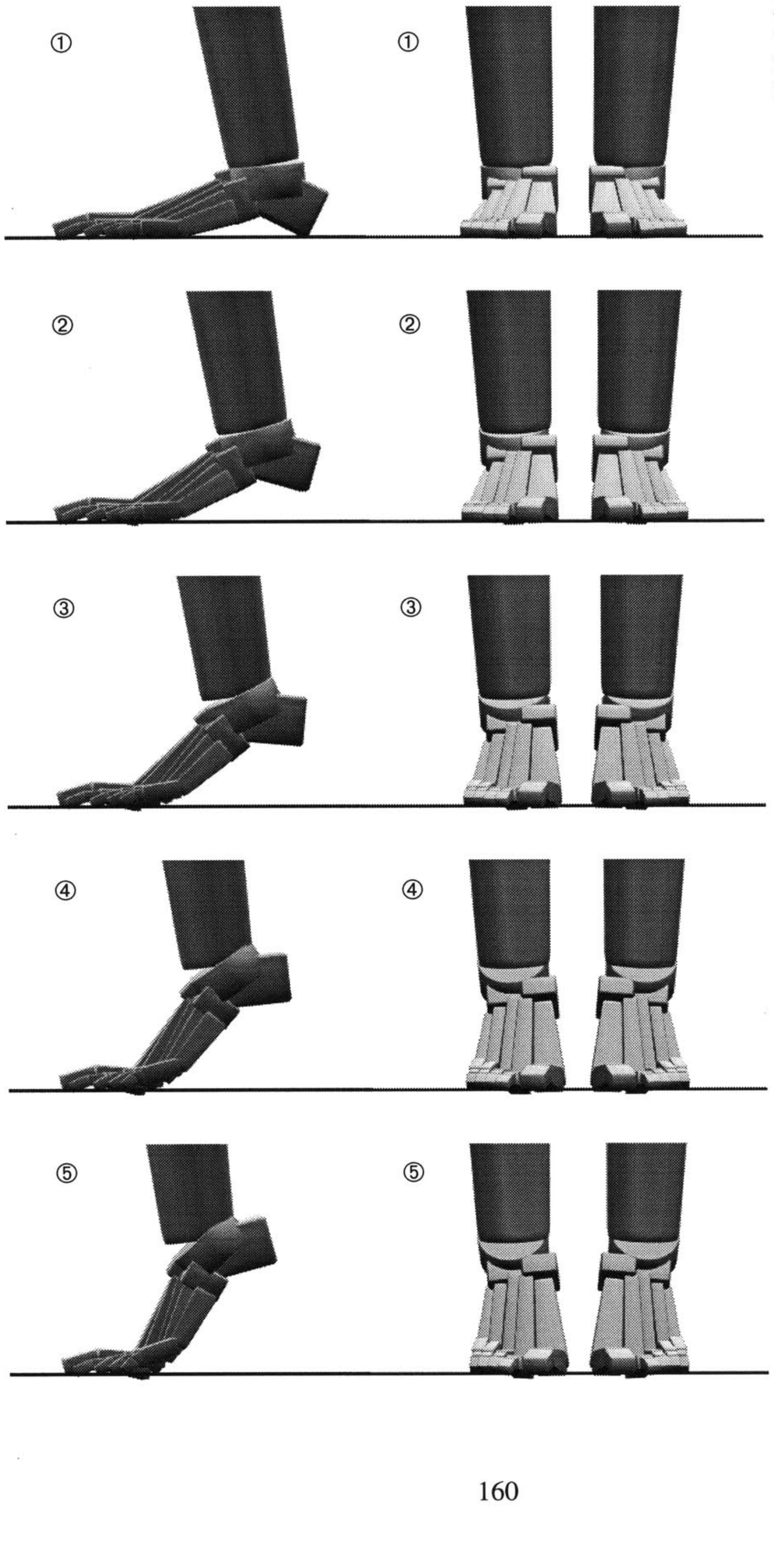


Figure 53: A close-up view of the foot during the toe raise simulation.

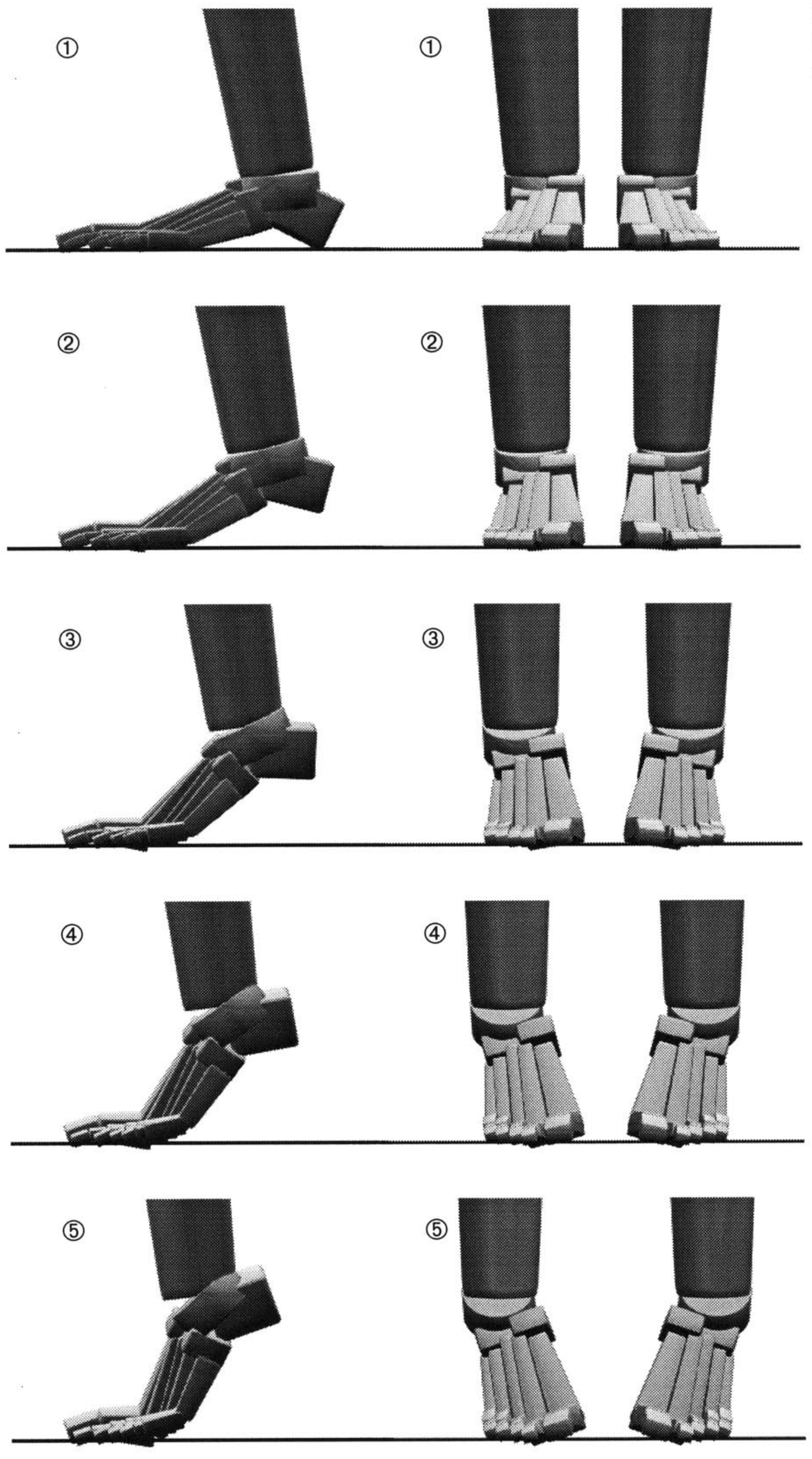


Figure 54: Rising on the toes, with an everted foot.

is not as comfortable as rising with the stress divided across the toes, bending evenly along the line of the distal heads of the second through fifth metatarsals (the "crease" of the toes).

A second simulation was run to generate this more natural posture. A motor program was added at the subtalar joint to evert the foot, which modified the posture to bend evenly at the toes, across the foot. An animation sequence of this second toe raise simulation is shown in Figure 54.

Informal comparisons were made between the animated simulations and video sequences that were shot of a human subject performing similar toe raises. The results were quite comparable, especially in contrast to any previous model of the foot used for animation. The most notable difference between the real and simulated were visible in the second toe raise simulation, with the everted foot. In the simulation, the toes rotated along their length axis, as the foot everted. In reality, this rotation is not seen, either because the toes joints (or some other parts of the foot) allow for rotational compensation, or because the surrounding tissues "mask" the rotations of the interior bones.

7.5 Passive Step

A set of "passive step" simulations were developed using the complex figure model, similar to Mochon and McMahon's passive "ballistic walking" experiments. [Mochon 1980-A; Mochon 1980-B] An emphasis was placed on using passive dynamic effects to generate the body motions, but a hybrid dynamics approach was used in order to simplify the problem. Some degrees of freedom were eliminated entirely, such as the waist and neck joints. The hips and shoulders were reduced to one DOF, allowing flexion and extension only. Eventually, it would be desirable to include all of the DOFs in the model, using either forward dynamics or kinematic constraints at the joints to make the motions more complete, with respect to real walking. The stance knee was kinematically locked, and the stance hip was kinematically controlled to undergo a constant angular velocity rotation, which is an approximation of the real motion in the human. The joints in the swing leg and the stance ankle were forward simulated, with no active joint forces, except for very weak dampers. Initial conditions were established for the joint positions and velocities of the step leg, the stance ankle, and for the initial overall body velocity. The system was then allowed to simulate forward, and a successful stepping motion was generated. The step leg acts as a pas-

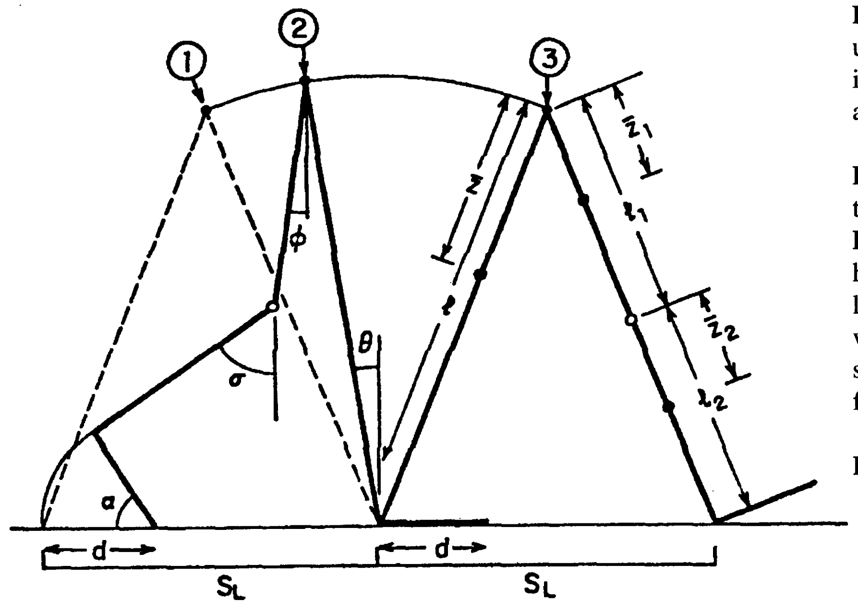


Figure 55: The model used in "ballistic walking" analyses by Mochon and McMahon.

Point 2 is the moment of toe-off for the swing leg. Point 3 indicates when heel strike of the swing leg occurs. The ballistic walking and passive step simulations generally run from point 2 to 3.

From [Mochon 1980-A].

sively-swinging double pendulum, and the stance leg acts as a passive inverted pendulum. The coupling of these two types of motions results in a natural appearing human step.

The model of the ballistic walking biped used by Mochon and McMahon is illustrated in Figure 55. If the joints in the arms and feet are also removed from the complex human figure, the result is a model that has a complexity similar to that used by Mochon and McMahon, except that their model was also restricted to the sagittal plane.

Because the passive step simulation was three dimensional, the human figure began to tilt laterally towards the swinging leg, as the center of mass was not above the region of the supporting foot. To address this, a simple ballistic motion was added to the simulation. As part of the initial conditions, a lateral velocity of the body was specified, so that the body moved towards the side of the supporting foot. About mid-stance, the body began to passively fall back towards the swinging leg. This motion takes advantage of the passive inverted pendulum formed by the stance leg, adding a new dimension to the one used previously.

Another problem arose because the simulation performed was not restricted to the sagittal plane. The momentum of the swing leg caused the body to rotate around its vertical axis, along with the swinging leg. Passive arm swinging was added to the simulation, which served to counteract the leg inertia and greatly reduce body rotation. The initial conditions for the arms to swing as passive double pendulums were established. The elbow of the left arm was kinematically constrained to maintain the same velocity, flexing the forearm, because passive motions were not sufficient to carry the arm up and forward.

Simulations of the passive step were executed both with and without the complex foot model. When the articulations were included in the foot, the foot shape was passively driven by the other body motions. When the stance leg became vertical, aligned straight upwards, the actuator spring in the talar (ankle) joint was activated, with its rest angle set to match the current joint angle. The motion of the ankle continued to bend somewhat, then stopped, as the spring deflected and took up a force load. As the body continued to move forward, it caused the angle between the leg and the ground to be reduced, which forced the foot to flex passively, since the ankle had stopped flexing. The foot flexion occurred at many DOFs, but primarily at the proximal joint of the proximal phalanges, bending the "toes."

An animation sequence of the passive step experiment, which shows the entire body, is given in Figure 56. A close-up image sequence of the side of the complex foot during the passive step is shown in Figure 57. A close-up of the front of the foot is shown in Figure 58. An informal comparison of the animated motions verses a video sequence of a human subject's stance foot revealed a reasonable correspondence between the two.

The simulation time varies during different parts of the simulation, depending primarily on how much stress the toes are placed under. The integrator took from 240 steps per frame to 2100 steps per frame (1/30 sec). The average time of computation was approximately ten minutes per frame, with 64 DOFs included in the model.

In addition to visual inspection of the animated results, we can also compare other aspects of the simulation to other simulations and to values measured from the real-world. A comparison between the motions of the joint angles in the legs from the ballistic walking experiments of Mochon and McMahon and from the passive step experiments with the complex model is shown in Figure 59. In this example, the ballistic walking simulation was designed such that the knee of the swing leg comes to full extension exactly at the time of heel strike. In the passive step simulation, the knee extends before heel strike. When the knee is nearly extended, its actuator is activated to maintain the knee position, with some flexion allowed from the spring. Mochon and McMahon also performed simu-

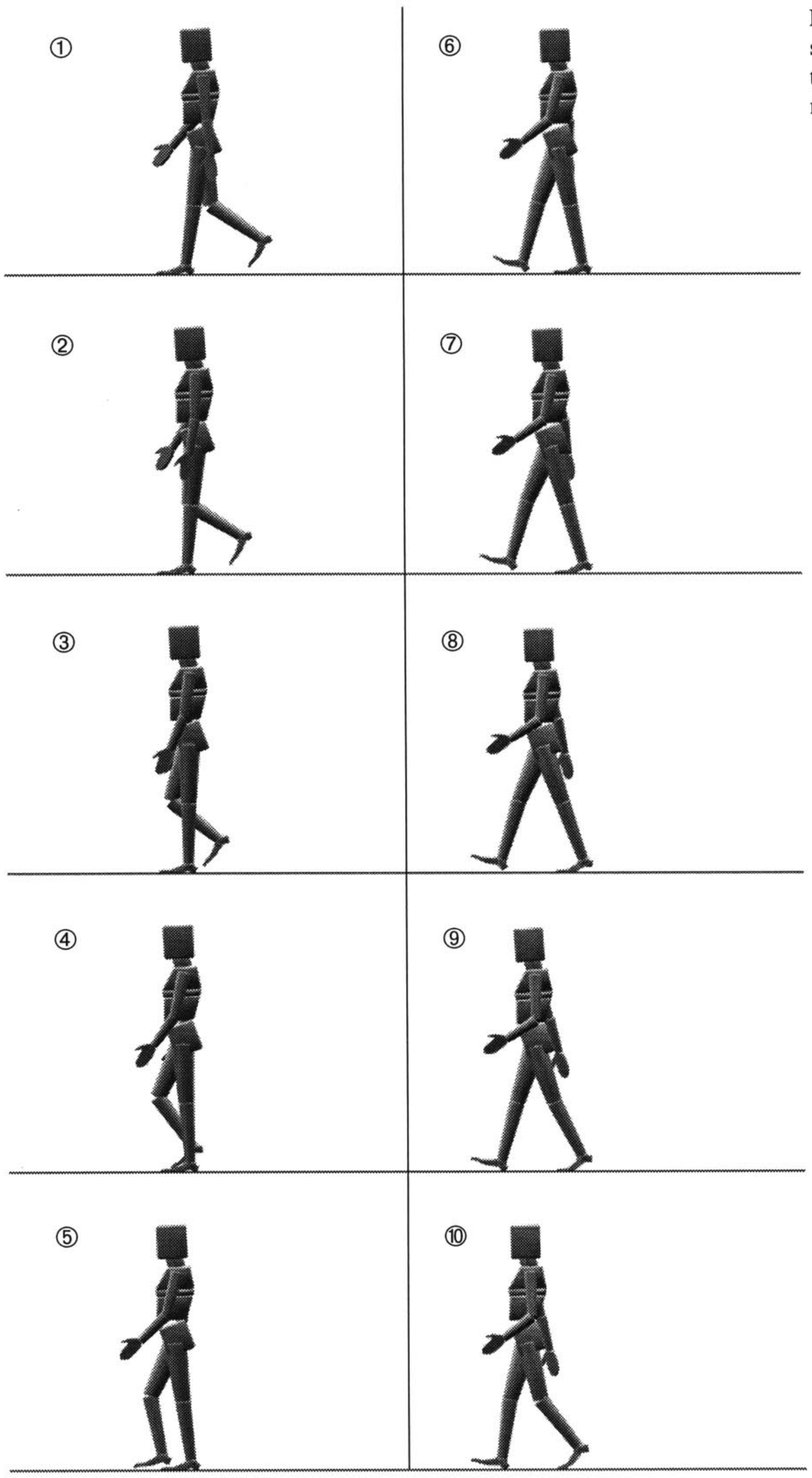


Figure 56: The passive step simulation, using the complex foot model.

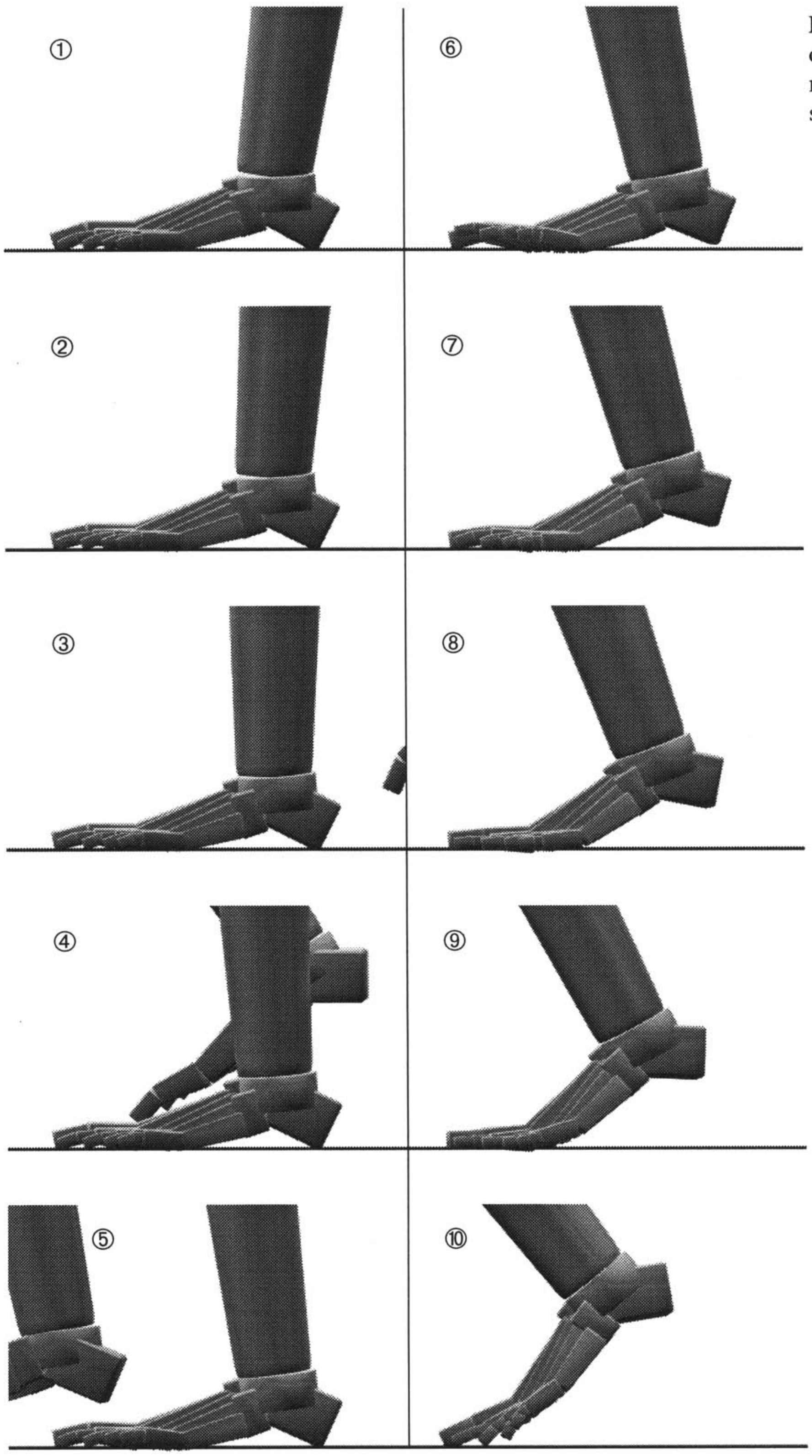


Figure 57: A close-up of the complex foot model during the passive step simulation.

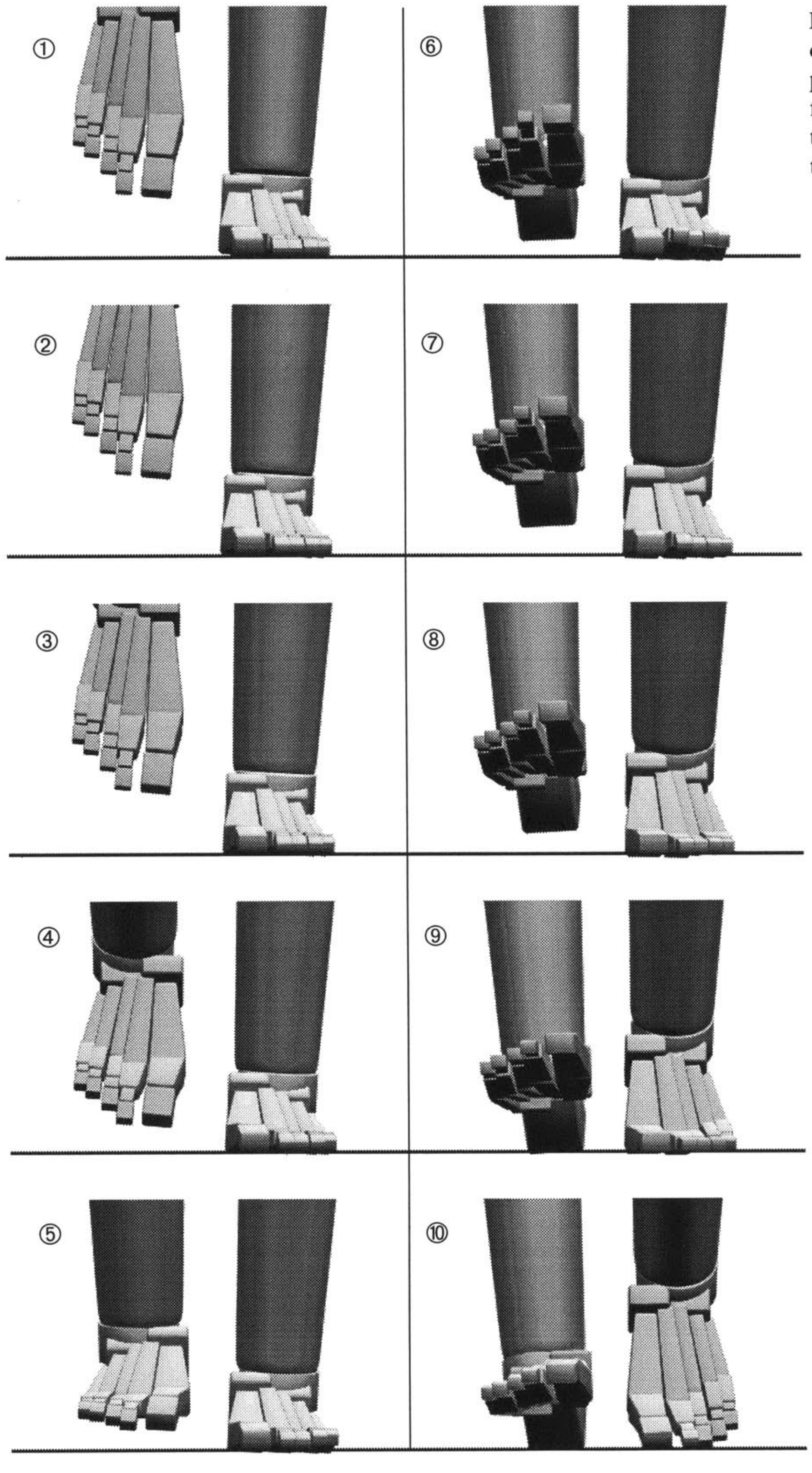
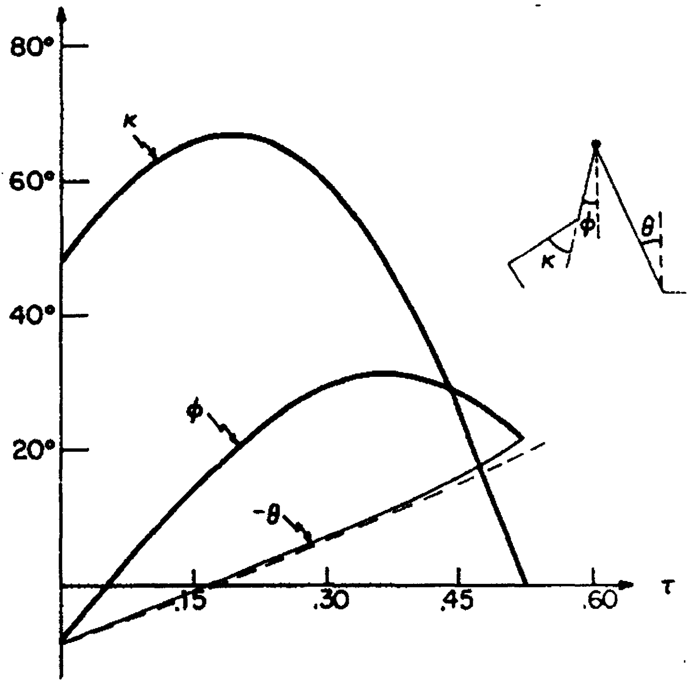
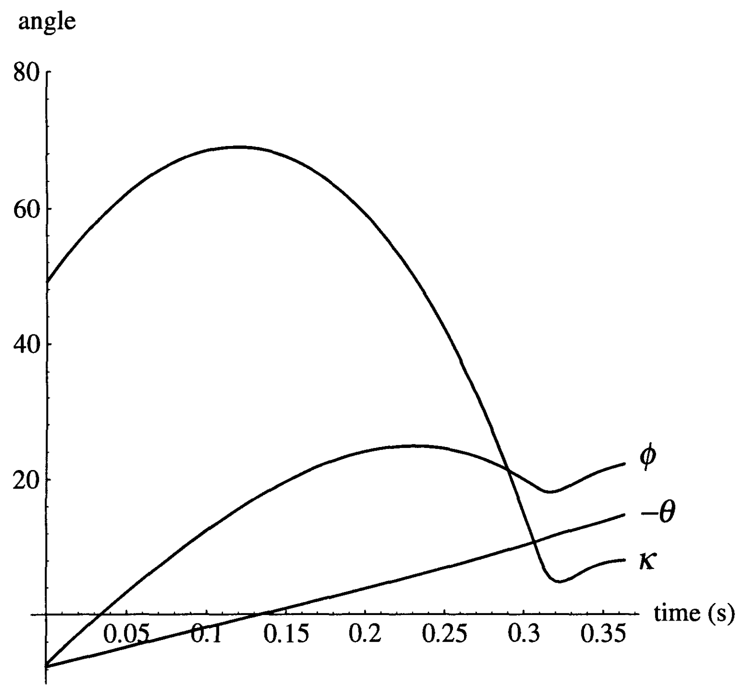


Figure 58: A close-up of the front of the complex feet of the human figure model during the passive step simulation.



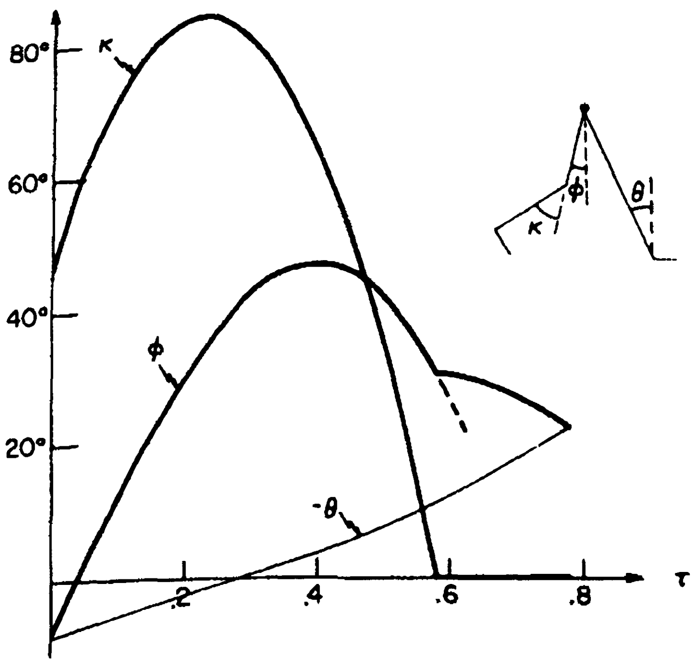
Mochon and McMahon



McKenna

Figure 59: Plot of the joint angles as a function of time calculated by Mochon and McMahon compared to the joint angles from the passive step simulation.

In this case, the ballistic walking simulation was designed such that the knee fully extends at the exact moment of heel strike in the swing leg. The foot in McKenna's passive step experiment was simplified to a single rigid body, in this example. Top image from [Mochon 1980-A].



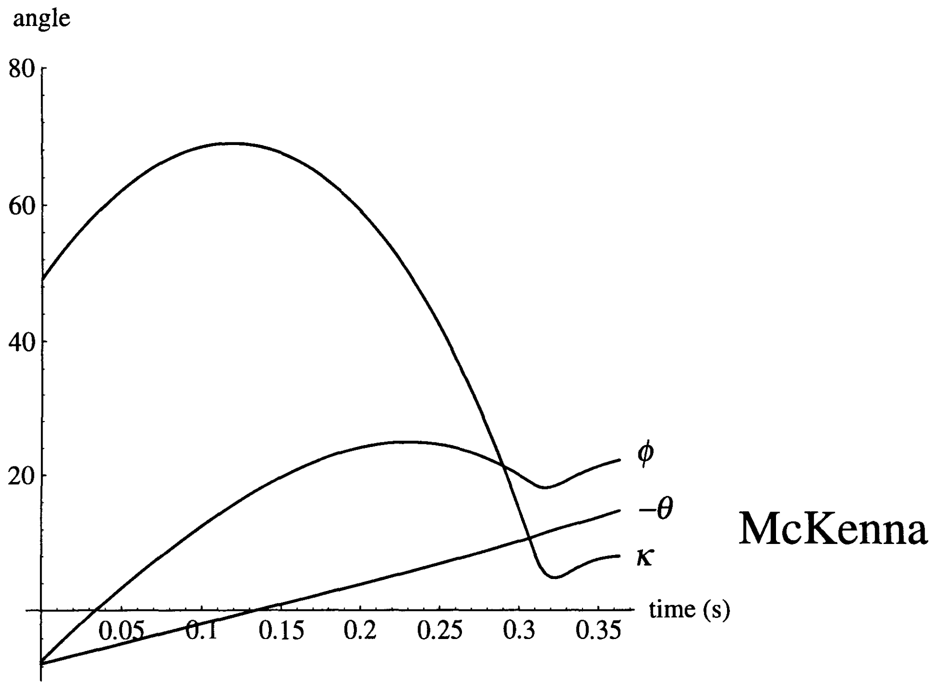
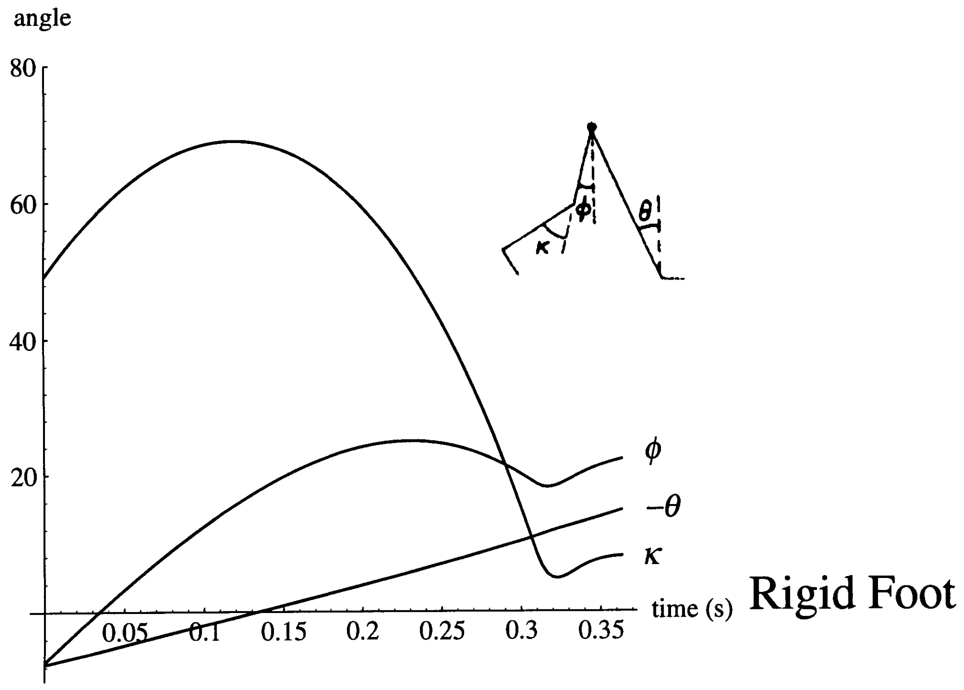


Figure 60: Joint angles from Mochon and McMahon's simulations with knee lock, compared to the passive step experiment.

The swing leg knee comes to full extension before heel strike, and is "locked." Again, the passive step simulation used the rigid foot. Top image from [Mochon 1980-A].



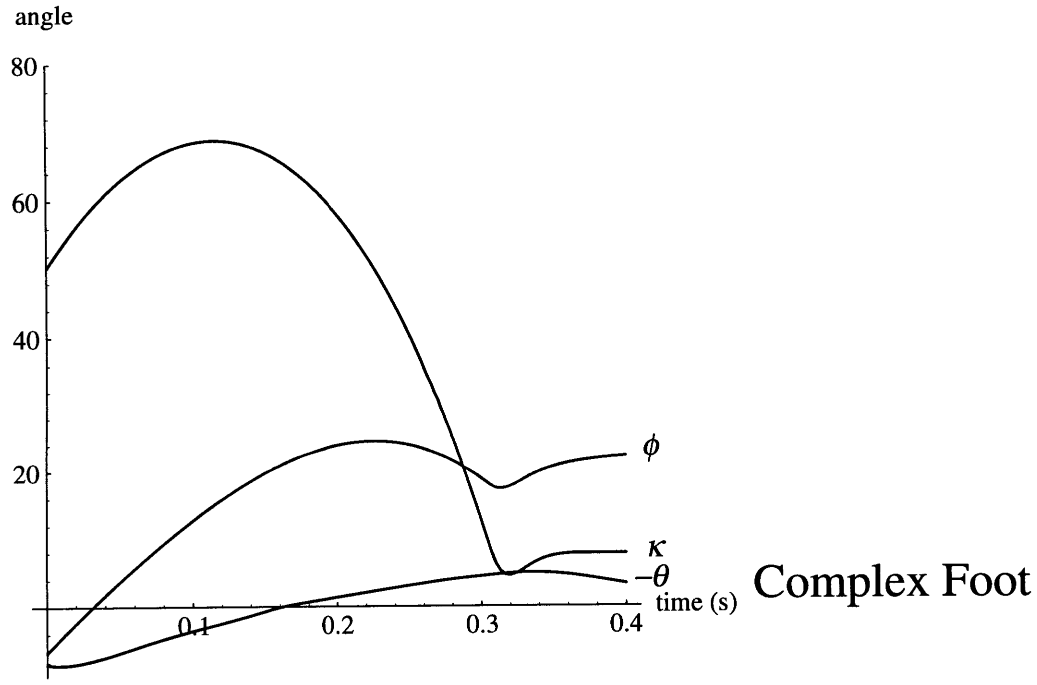


Figure 61: Joint angles from the passive step simulations, with the rigid foot (above) and the complex foot (below).

lations in which the swing knee fully extends, and then locks rigidly, before heel strike. A comparison of the passive step simulation with this type of ballistic walking is given in Figure 60. The comparison is quite favorable. The deflection of the knee when it "locks" in the passive step experiment is probably more realistic than the hard, rigid lock in the ballistic walking simulation. The major difference between the two is that the stance leg extended further before heel strike in the passive step simulations. This may be due to the manner in which Mochon and McMahon locked the swing leg. If its momentum was dissipated without being shifted to the thigh and body, the swing leg would extend less.

A comparison of the joint angles between passive step simulations that employ the rigid foot and the jointed foot is shown in Figure 61. The major difference is in the plot for the ankle joint. Because the ankle joint is locked with an actuator spring in the middle of the support phase, when the complex foot is modeled, its angle is not comparable unless the total deflection of the foot joints are also included.

We can also examine the ground reaction forces. The ground reaction forces (GRFs) as measured from the human are compared to the GRFs computed during the passive step experiments, both with and without the complex foot. See Figure 62. The GRFs are quite similar, and certainly have the same overall characteristics. When the complex foot is included in the model, more variation is seen in the force plots, in some ways bringing them closer to the measured GRFs, in other ways making them less similar.

The simulator, with the biomechanical model, can be used to generate "what-if" types of experiments. As a simple and preliminary example of the kind of clinical application that could be developed from this work, simulations of the passive step experiment were developed, in which parts of the big toe of the stance leg were removed, to observe the effect on the developing motion. It is a fairly common reconstructive surgery practice to transfer some part of the toes in order to create a replacement for an amputated thumb. There are several different options regarding how much of a toe to remove, and whether to use the big toe or the second toe. [Wei] A simulator system, with a biomechanical model tailored to a given patient could be used to help examine some of the trade-offs between the different options.

In the first simulation, the distal phalanx of the big toe of the left foot was removed (the stance leg in the simulation). The same passive step simulation was executed, and the

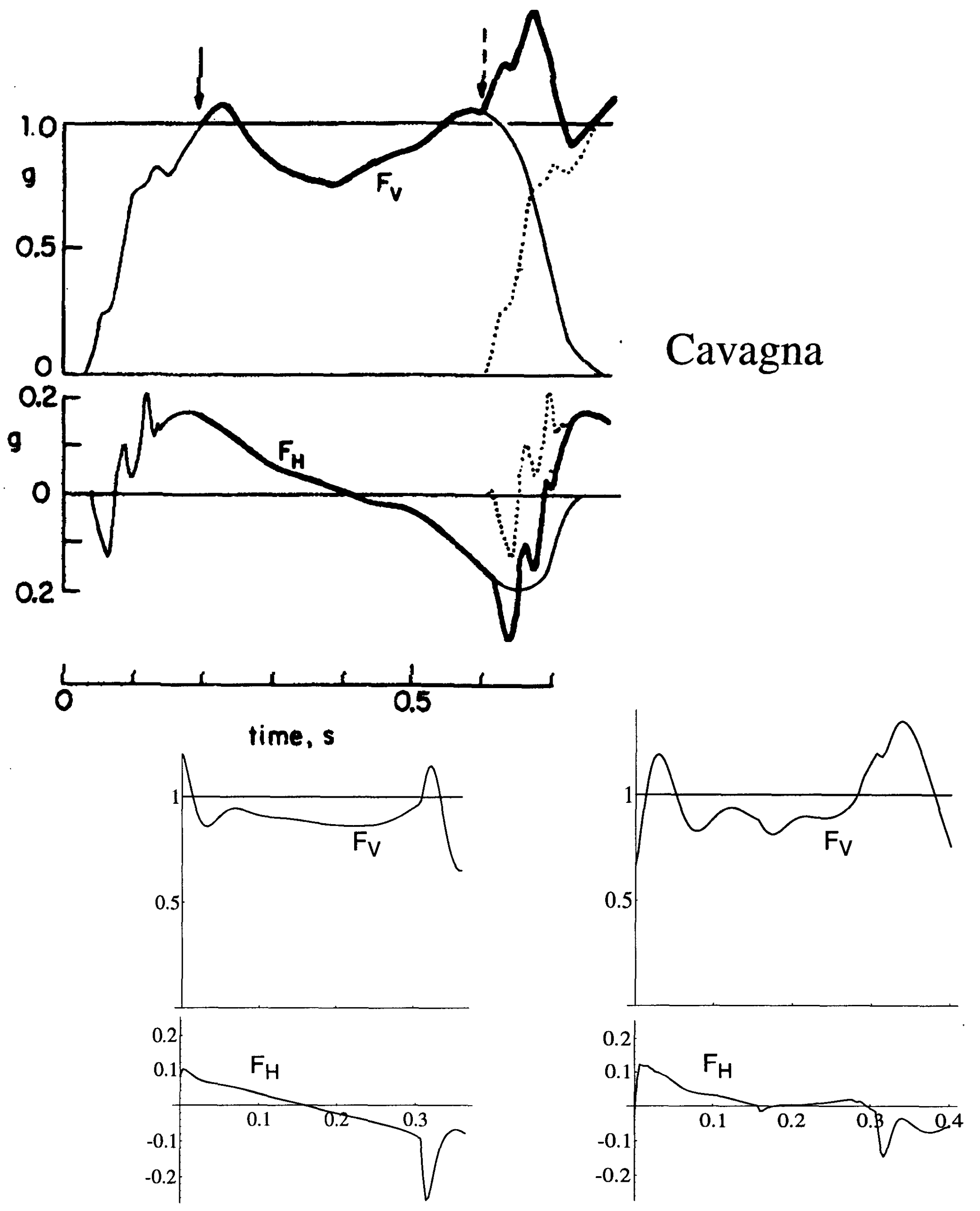


Figure 62: Plots of the ground reaction forces measured from humans compared to those computed in the passive step simulation.

The ground reaction forces during normal human walking, as measured by Cavagna and Margaria are shown in the upper plots. The solid arrow to the left indicates the time of toe-off for the swing leg. The dashed arrow to the right indicates when heel strike of the swing leg occurs. Top plot from [Cavagna 1966]

The ground reaction force computed during the passive step experiment are shown in the bottom plots. The left plots show the forces from the rigid foot, the right ones show the forces from the articulated foot. The left plots are approximately aligned to the corresponding time below the Cavagna plots. The upper plot shows the vertical reaction force (F_V) on the supporting foot. The lower plot shows the forward, horizontal reaction force (F_H).

change in the motion could be observed. In fact, there was not much change in the motion, except that the body tilted further laterally, towards the swing leg, near the end of the support phase, because there was less of a support on the medial side of the stance foot. A second simulation was run, in which the proximal phalanx of the big toe was also removed. In this case, the lateral body tilt was greater, but overall, the change in motion was not very large. With both phalanges missing from the big toe, the relative lateral distance between the body's center of mass and the point of heel strike was reduced by 1 cm, in comparison to the simulation that had an intact foot. In reality, a person would certainly attempt to compensate for the missing toe, but this type of simulation at least provides a sort of baseline, to examine the effects of a structural change without any compensation.

Future Work: Extensions and Issues Concerning Walking Simulation

Human walking is certainly not a completely passive activity. The muscles are not inactive during locomotion — muscles are active to support the stance leg, muscles are used to accelerate and decelerate the swing leg, [Yoon] and many other muscles are used to support the upper body, etc. These factors are "designed-around" in McGeer's purely passive walking mechanisms, and they are ignored in McMahon and Mochon's Ballistic walking work.

Active control can be provided through motor programs, by modifying actuator control parameters over time. Active control can be used, in part, to "guide" the passive control, establishing initial conditions which allow passive motions to be successfully executed. Kinematic constraints can also be used to setup initial conditions by accelerating the limbs to match target velocities. These constrains could also be formulated as a type of motor program. How can the motor programs for walking be established? By careful examination of the requirements; trial and error; and hybrid analyses. Hybrid dynamics and inverse control can be used to calibrate the actuators to target motions, either on the boundary conditions of the motions, or throughout entire motions, using a more continuous method to calibrate the motor program parameters over a time interval — forming table-based motor programs. The target motions could be derived from clinical gait studies. However, this method is less desirable, because the motions become more prescribed. However, it is not assured that table-based motor programs would completely dictate the motion, because of the forward nature of the motion simulation, especially in the presence of errors or unexpected force or terrain perturbations.

The trade-offs of kinematic versus dynamic control, and the mix of the two, is an interesting issue. Hybrid dynamics can be used as a means of determining control parameters for forward simulation, and it can also be used as a direct part of the motion simulation. Thus, certain DOFs can be kinematically controlled at certain times, while others are dynamically simulated. This can greatly increase the computational efficiency of the simulation, since the inverse dynamics calculations are somewhat more efficient than the forward dynamics computations, and problems of numerical stiffness are greatly reduced at a kinematically-controlled joint. In addition, mixing kinematics and dynamics can provide more flexibility of control— an important consideration for animation. An example of hybrid control for walking can be given: before toe-off, the step leg is kinematically accelerated to the correct initial conditions required for a dynamic, passive step. Dynamics can be employed where they are most critical to the motions— passive motions being a case in point.

In previous work by the author, a gait controller, based on coupled oscillators, was used to coordinate the stepping actions of a simulated hexapod, i.e. designating when each leg should step and when it should stand. [McKenna 1990-B; McKenna 1990-C] The same mechanisms can be used to control figures with any number of opposing legs, including bipeds. However, the complexity of this mechanism is not especially required for biped stepping coordination. In fact, a central pattern generator may not be appropriate for this system. The biped stepping pattern may result more from the system dynamics than from central planning, especially when the passive effects dominate. Also, free gaits (non-periodic gaits, such as those used to walk across stepping stones) would rely far less on a central stepping pattern. Under highly controlled conditions, such as walking on level ground without disturbances, a gait controller would be an appropriate means to sequence motor programs. Otherwise, motor programs could be sequenced by higher order planning controllers, and/or 'reflexively' triggered, based on events that occur during the walking cycle (such as heel strike, etc.).

Another topic of interest is the use of passive "return springs" during walking. These springs deform during part of the locomotion cycle, and return their energy in another part of the cycle. Alexander describes the use of return springs in locomoting animals. [Alexander 1985; Alexander 1990] The use of passive joint limit forces, as well as the actuator spring forces, can be used as return springs. The toes bend significantly during the passive step simulation, near the end of the stance phase. The springs become compressed and store energy, which could be used, if properly “channeled,” to launch the leg and push the body forward, restoring lost energy.

8 Conclusions

8.1 Human Figure Model

This work has demonstrated that complex human kinematics can be dynamically simulated, including small bones placed under high stress forces. Simulation times can range from real-time for simplified models, to approximately half an hour per frame (1/30 second of simulated time) for complex models involving standing postures, using high-end computer workstations.

Corpus, a computer program for the simulation and animation of articulated figures has been developed and implemented by the author. Physically based motions are generated in *corpus* by the efficient dynamic simulator sub-system, based on the *Articulated Body Method* algorithm by Featherstone. [Featherstone 1987] The system is flexible and general purpose, allowing for the simulation of any branching articulated figure using forward, inverse, hybrid, and first order dynamics. Gravity, collision, contact, friction, damping, joint limit, actuator, and other force models are available in the system. The simulator is integrated with a graphics sub-system that provides a three dimensional environment for real-time computer graphics and animation.

A new, complex human biomechanical model has been developed and described. The figure has 90 degrees of freedom overall, with a foot model that incorporates 28 DOFs each. The kinematic and dynamic structure of the model has been designed based on anthropometrics and other measures from humans. This model is useful as a structural basis for physically based computer animation and biomechanical research, and ultimately, with additional modeling and verification, the system should prove useful as a tool for clinical analyses.

A low level control system for movement and posture forms a foundation for a variety of dynamic tasks. The use of spring actuators and dampers at the joints provides a stable feedback system capable of generating postural support and motion control. The control state for the actuators can be calibrated using inverse dynamics and inverse control to automatically determine the parameters required for a specified postural goal.

The dynamic simulation system and the biomechanical model, with its low level behavior control for motion have been used to generate a number of animated simulations, demonstrating their utility. From simple simulations of passively falling under gravity, to complex simulations of rising on the toes and walking, the human figure model has proven to be a sophisticated tool for the creation of realistic animation.

8.2 Future Directions

The *corpus* system can be used as a platform for performing simulations for biomechanical studies. For example, recently a dynamic simulation of a frog leg, with a hip, 'knee' and 'ankle' has been developed to verify a model developed by experimental biologists. After activating the joint spring actuators in the simulated frog leg, we measure the force-field that is generated at the end of the foot or at the ankle, as the foot is pulled through a range of kinematic locations. The results of the simulations are compared to the measured force fields from *in vivo* experiments. The simulation model can be used to test and validate the biologists' hypotheses. The results from the frog simulation also shed some light on more general motion control issues, which could lead to control techniques for dynamically simulated animation. Using the human figure model, biomechanical analyses of human motions can also be investigated.

The human figure model developed in this work represents a single human form. In order to adjust the model to a given person, accurate information regarding their body structure is required. The technology currently exists to extract bone and tissue geometry from MRI or CAT scans, and to tailor the kinematic structure and biomechanical function to that person-specific, or "patient-specific," geometry. The technology also exists to manipulate that data and biomechanical model in three dimensions, in real-time. [Delp; Pieper 1994] The use of patient-specific data will be a crucial element in the surgical planning systems of the near future.

As our models increase in kinematic and biomechanical complexity, the computational and organization complexity also grows steeply. However, the number of real problems and questions that can be addressed also grows with the details from which the models are formed. In a well designed system, all functionality and complexity need not be employed simultaneously, in order to simplify computation and control. Localized areas and functions can be simulated, and "multi-resolution" simulations can be run, at different levels of complexity, to see if and how changes in one system affect another.

With the addition of other kinds of models to the human figure model, we can investigate new, multi-faceted problems. Three dimensional, biomechanical models of human skeletal muscle [Chen] and human skin tissue [Pieper 1992] have been demonstrated. There is no intrinsic barrier to the unification of these finite-element models with the rigid, articulated body model. With the inclusion of other models, such as organ function, a highly sophisticated human figure model can be developed, creating a computationally based, artificial physiological person.

It is also desirable to enhance the current biomechanical model of the articulated figure. The current kinematics could be refined and verified. Additional degrees of freedom, to make the model more complete, should also be added. In particular, a complex spine, neck, and hand would be welcome. In addition, more complete biomechanical models of the many muscles, ligaments, and passive forces should be included, so that their functions can be analyzed as well.

Because the structure of the hand is similar to the structure of the foot, a preliminary model of the hand was generated, using the structural definition of the foot as a starting point. The resulting model is shown in Figure 63. The entire human figure, with the newly modeled hands, includes 136 joints, for a 142 DOF model. Although the kinematic structure was defined, other biomechanical parameters have not yet been modeled, so simulations have not yet been performed using the hand.

Finally, high level behavior control is required in order to select from and control different behaviors, based on the stimulus the model receives and its internal goals.

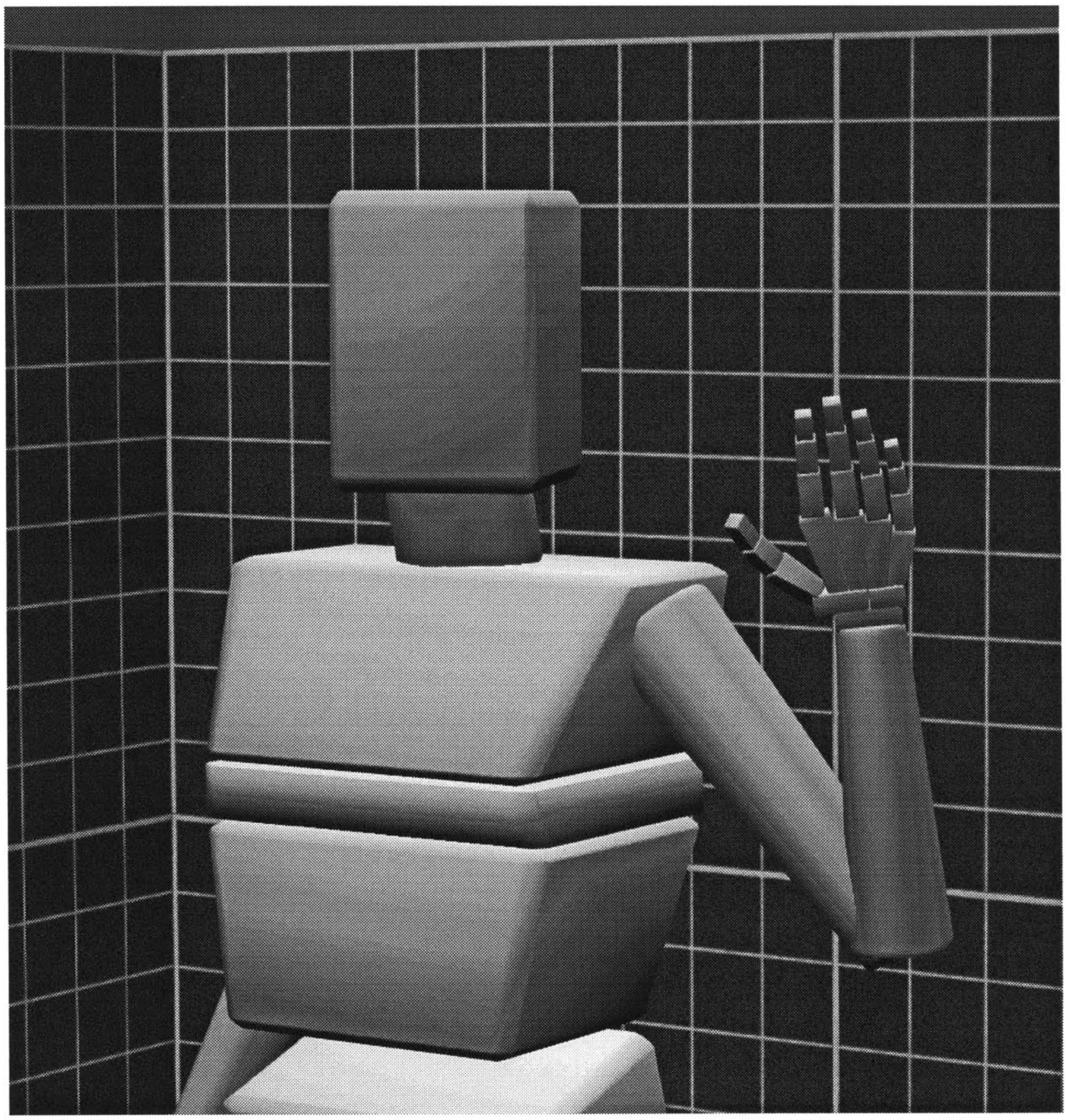


Figure 63: A preliminary hand model is added to the complex human figure model.

9 Acknowledgments

I would like to thank a number of people who were influential in seeing this project through.

To my sets of parents: Lynn and Don, Paul and Debbie, and Howard and Phyllis, thanks for the many years of love and support.

More than thanks go to my academic advisor of seven-plus years, David Zeltzer — this work would never have happened without his guidance and encouragement. Others on my thesis committee were Professors Marc Raibert and Joe Rosen. I'd like to thank Marc for his support and interest in my work — his respect has always meant a lot to me. And to Joe, many thanks for the boundless enthusiasm and generous friendship.

Thanks to Nicholas Negroponte for creating a remarkable environment for technical and creative work with media technology. And thanks as well to Steve Benton for advice and guidance over the years.

Dave Small gets more than my thanks for his many years of standing by me.

A resounding "HOOT" to Stevie Pieper, Dave Chen, Peter Schröder, Steve Drucker, and Tinsley Galyean — otherwise known as the playpen guys — for all the required diversions and inspirations, long discussions on dynamics, animation, simulation — the works! And, of course a "HOOT" to the rest of the Snakepit crew over the years: Michael Johnson, Paul Dworkin, Karl Sims, Steve Strassmann, Clea Waite, Jim Puccio, Fran Taylor, and Margaret Minsky. Lots of other Media Lab students, staff and faculty deserve my thanks and regards as well, so I'd like to offer my thanks to the following friends, in no particular order: Henry Holtzman, Bob Sabiston, John Underkoffler, Janet Noss, Greg Tucker, Steve Librande, Chris Schmandt, Michael Halle, Anne Russell, Peg Schafer, Alan Lasky, Walter

Bender, John Wadlington, Toru Nagamachi, Alice Markunas, Patti Maes, Ron MacNeil, Betsy Connors, Muriel Cooper, Tod Machover, Gayle Sherman, David Berger, Michael Klug, Anh V. Ho, Lena Davis, Gloriana Davenport, David Blank-Edelman, Laura Robin, Dave Young, Irfan Essa, Janet Cahn, Barry Arons, Judith Donath, Gilberte Houbart, Robin Kullberg, Suguru Ishizaki, Michael Travers, Alice Lei, Grace Colby, Mark Lucente, Wendy Plesniak, Stan Sclaroff, Radhika Nagpal, Jory Bell, Wayne Ross, and Pierre St. Hilaire. I know I'm bound to leave out people I want to thank, so please forgive me!

Of course, I extend many thanks to the course secretaries, Linda Peterson and Santina Tonelli. Thanks for the help and friendship.

I would like to thank and acknowledge Simon Giszter, for the collaborative frog simulation work.

A number of individuals contributed their computer software for use in this research, and so I would like to thank: David Chen, for *rendermatic* and *librobot*; Peter Schröder for *retepmatic* (enhanced anti-aliasing scan converter) and the inertia computation function; John Underkoffler for *awf* (anti-aliased wire-frame rendering) , and David Small and others from the Visible Language Workshop, for anti-aliased text and line functions.

I would like to acknowledge Donald Stredney for creating the skelton model "George," used as an initial template for the human figure model. He created the model under a grant from the National Science Foundatation, Project NO. MCS-7923670, titled *Complex Animation and Complex Object Synthesis*, at the Advanced Computing Center for the Arts and Design, The Ohio State University.

I would like to gratefully thank the following organizations for their support of my work at MIT, through research and equipment grants: NHK (Japan Broadcasting Corp.), the National Science Foundation (NSF), MIT UROP Program, Hewlett Packard Co., Apple Computer, Inc., Silicon Graphics, Inc., Bitstream, Inc., Stardent Computer Corp., Symbolics, Inc., Thinking Machine Corp., and Wacom, Inc.

I would also like to thank Medical Media Systems (MMS) for the use of equipment which greatly facilitated the timely completion of this thesis. I'd also like to thank all of the staff of MMS for their generous personal support.

10 Biographical Note

Michael Allen McKenna was born and raised in southern Florida. He had an early interest in science and mathematics, and spent a summer with the Florida Foundation for Future Scientists at the University of Florida, in Gainesville while in high school. He graduated valedictorian from a small private school.

Mike is a member of the Massachusetts Institute of Technology undergraduate class of 1987. He studied both Computer Science and Visual Arts, and had an undergraduate research position at the Architecture Machine Group, and then its successor, the Media Laboratory. Mike received a Bachelor of Science from the Department of Architecture in 1987.

Continuing his education at MIT, Mike attended graduate school at the Computer Graphics and Animation Group in the Media Laboratory. While conducting his research, Mike created the award winning computer animations *Cootie Gets Scared* and *Grinning Evil Death* (the latter with Bob Sabiston). Mike received a Master of Science in 1990.

Mike has authored and co-authored over a dozen articles in conference proceedings, journals, and books. He has given numerous presentations on his research, around the world. Mike's interests include Lego bricks, visual arts, science fiction, and home repair. Mike currently lives in Cambridge, Massachusetts with his partner, David Small.

11 Bibliography

1. Alexander, R.M. (1976). Mechanics of Bipedal Locomotion. *Perspectives in Experimental Biology.* Edited by P.S. Davies. Pergamon Press, Oxford.

2. Alexander, R.M., N.J. Dimery and R.F. Ker. (1985). Elastic structures in the back and their role in galloping in some mammals. *J. Zool., Lond.* 207:467–482.

3. Alexander, R.M. (1990). Three Uses for Springs in Legged Locomotion. *International J. Robotics Research.* 9(2):53–61.

4. Amirouche, F.M.L., S.K. Ider and J. Trimble. (1990). Analytical Method for the Analysis and Simulation of Human Locomotion. *Journal of Biomedical Engineering.* 112 (November):379–386.

5. Amkraut, S. and M. Girard. (1989). *Eurhythmy.* Computer animation. ACM SIGGRAPH, Film Video Show 1989.

6. An, C.H., C.G. Atkeson and J.M. Hollerbach. (1988). *Model-Based Control of a Robot Manipulator.* MIT Press, Cambridge, MA.

7. Armstrong, W.W. (1979). Recursive solution to the equations of motion of an n-link manipulator. *Proc. of 5th World Congress Theory Mach. Mechanisms* (Montreal). 2:1343–1346.

8. Armstrong, W.W. and M. Green. (May 1985). The Dynamics of Articulated Rigid Bodies for Purposes of Animation. *Proc. Graphics Interface 85* (Montreal, Canada). pp.407–415.

9. Armstrong, W.W., M. Green and R. Lake. (June 1987). Near-Real-Time Control of Human Figure Models. *IEEE Computer Graphics and Applications.* 7(6):52–61.

10. Audu, M.L. and D.T. Davy. (1985). The Influence of Muscle Model Complexity in Musculoskeletal Motion Modeling. *Journal of Biomechanical Engineering.* 107:147–157.

11. Badler, N.I., J.K. Korein, J.U. Korein, G.M. Radack and L.S. Brotman. (1985). Positioning and Animating Human Figures in a Task-Oriented Environment. *Visual Computer.* 1(4):212–220.

12. Badler, N.I. (June 1987). Articulated Figure Positioning by Multiple constraints. *IEEE Computer Graphics and Applications.* 7(6):28–38.

13. Barzel, R. and A.H. Barr. (August 1988). A Modeling System Based on Dynamic Constraints. *Proc. SIGGRAPH '88* (Atlanta, Georgia), in *Computer Graphics* 22(4):179–188.

14. Bizzi, E., W. Chapple and N. Hogan. (1982). Mechanical Properties of Muscle: Implications for Motor Control. *Trends in Neuroscience.* 5(11):395–398.

15. Bizzi, E., N. Accornero, W. Chapple and N. Hogan. (1984). Posture Control and Trajectory Formation During Arm Movement. *Journal of Neuroscience.* 4(11):2738–2744.

16. Bogert, A.J. von den., H.C. Schamhardt and A. Crowe. (1989). Simulation of Quadrupedal Locomotion Using a Rigid Body Model. *Journal of Biomechanics.* 22(1):33–41.

17. Brady, M., J.M. Hollerbach, T.L. Johnson, T. Lozano-Pérez and M.T. Mason. (1982). *Robot Motion: Planning and Control.* MIT Press, Cambridge, MA.

18. Braune, W. and O. Fischer. (1987). *The Human Gait.* Springer-Verlag, Berlin. (Originally published in German, between 1895 and 1904).

19. Braune, W. and O. Fischer. (1988). *Determination of the Moments of Inertia of the Human Body and Its Limbs.* Springer-Verlag, Berlin. Translators: P Maquet, R. Furlong (originally published in 1892).

20. Brotman, L.S. and A. Netravali. (August 1988). Motion Interpolation by Optimal Control. *Computer Graphics.* 22(4):309–315.

21. Bruderlin, A. (1988). *Goal-Directed, Dynamic Animation of Bipedal Locomotion.* Master's Thesis, School of Computing Science, Simon Fraser University. CMPT TR 88-10.

22. Bruderlin, A. and T.W. Calvert. (July 1989). Goal-Directed, Dynamic Animation of Human Walking. *Proc. of SIGGRAPH '89* (Boston, MA), in *Computer Graphics* 23:233–242.

23. Cavagna, G.A. and R. Margaria. (1966). Mechanics of walking. *Journal of Applied Physiology.* 21:271–278.

24. Cavagna, G.A., N.C. Heglund and C.R. Taylor. (1977). Mechanical Work In Terrestrial Locomotion: Two Basic Mechanisms For Minimizing Energy Expenditure. *Am. J. Physiology.* 233(5):R243–R261.

25. Cavagna, G.A. (1985). Force Platforms as Ergometers. *J. Applied Physiology.* 39(1):174–179.

26. Chen, D.T. (1991). *Pump It Up: Computer Animation of a Biomechanically Based Model of Muscle using the Finite Element Method.* Ph.D. Thesis, Massachusetts Institute of Technology.

27. Darrell, T., P. Maes, B. Blumberg and A.P. Pentland. (1994). *Situated Vision and Behavior for Interactive Environments.* M.I.T. Media Laboratory Perceptual Computing Technical Note No. 261. Jan. 1994.

28. Delp, S., P. Loan, M. Hoy, F. Zajac, S. Fisher and J. Rosen. (1990). An Interactive Graphics-Based Model of the Lower Extremity to Study Orthopaedic Surgical Procedures. *IEEE Transactions on Biomedical Engineering.* 37. Special issue on interaction with and visualization of biomedical data.

29. Dempster, W.T. (1955). *Space Requirements of the Seated Operator: Geometric, Kinematic, and Mechanical Aspects of the Body With Special Reference to the Limbs.* Wright-Patterson Air Force Base, Ohio. WADC-TR-55-159.

30. Drillis, R. and R. Contini. (1966). *Body Segment Parameters,* Rep. 1163-03. Office of Vocational Rehabilitation, Department of Heath, Education, and Welfare, New York.

31. Dworkin, P. and D. Zeltzer. (1993). A New Model for Efficient Dynamic Simulation. *Proc. Fourth Eurographics Workshop on Animation and Simulation.* pp.35–147.

32. Essa, I.A. and A. Pentland. (1994). A Vision System for Observing and Extracting Facial Action Parameters. *Proc. 1994 Computer Vision and Pattern Recognition Conference.* IEEE Computer Society. (to appear).

33. Featherstone, R. (1983). The Calculation of Robot Dynamics Using Articulated-Body Inertias. *Robotics Research.* 2(1):13–29.

34. Featherstone, R. (1987). *Robot Dynamics Algorithms.* Kluwer Academic Publishers.

35. Foley, J.D., A. van Dam, S.K. Feiner and J.F. Hughes. (1990). *Computer Graphics: Principles and Practice.* 2nd edition. Addison-Wesley, Reading, MA.

36. Forsythe, G.E., M.A. Malcolm and C.B. Moler. (1977). *Computer Methods for Mathematical Computations.* Prentice-Hall, Inc., New Jersey.

37. Frank, A.A. (1971). On the Stability of an Algorithmic Biped Locomotion Machine. *Journal of Terramechanics*. 8(1):41–50.

38. Freeman, P.S. and D.E. Orin. (1991). Efficient Dynamic Simulation of a Quadruped Using a Decoupled Tree-Structured Approach. *International J. Robotics Research*. 10(6):619–627.

39. Furusho, J. and M. Masubuchi. (1986). Control of a Dynamical Biped Locomotion System for Steady Walking. *ASME J. Dyn. Sys. Meas. Control*. 108:111–118.

40. Ginsberg, C. and D. Maxwell. (April 1983). Graphical Marionette. *Proc. ACM SIGGRAPH/SIGART Workshop on Motion* (Toronto, Canada). pp.172–179.

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

42. Girard, M. (June 1987). Interactive Design of 3D Computer-Animated Legged Animal Motion. *IEEE Computer Graphics and Applications*. 7(6):39–51.

43. Goldfinger, E. (1991). *Human Anatomy for Artists: The Elements of Form*. Oxford University Press, New York.

44. Gray, H. (1977). *Gray's Anatomy*. Gramercy Books, New York. (American edition originally published in 1901).

45. Gubina, F., H. Hemami and R.B. McGhee. (1974). On the Dynamic Stability of Biped Locomotion. *IEEE Transactions on Biomedical Engineering*. BME-21(2):102–108.

46. Hahn, J.K. (August 1988). Realistic Animation of Rigid Bodies. *Computer Graphics*. 22(4):299–308.

47. Hatze, H. (1976). The Complete Optimization of a Human Motion. *Mathematical Biosciences*. 28:99–135.

48. Heppenheimer, T.A. (1985). Man Makes Man, in *Robotics*. Edited by M. Minsky. Omni Press (Anchor Press/Doubleday), Garden City, New York. pp.28–69.

49. Herman, R., T. Cook, B. Cozzens and W. Freeman. (1973). Control of Postural Reactions in Man: The Initiation of Gait. *Control of Posture and Locomotion*. Edited by R.B. Stein, K.G. Pearson, R.S. Smith and J.B. Redford. Plenum Press, New York.

50. Hof, A.L. and Jw. Van den Berg. (1981). EMG to Force Processing II: Estimation of Parameters of the Hill Muscle Model for the Human Triceps Surae by Means of a Calfergometer. *Journal of Biomechanics*. 14(11):759–770.

51. Hollerbach, J.M. (1980). A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity. *IEEE Transactions on Systems, Man, and Cybernetics*. SMC-10(11):730–736.

52. Huelke, D.F. (1986). Anatomy of the Lower Extremity – An Overview. *Biomechanics and Medical Aspects of Lower Limb Injuries* (P-186). Society of Automotive Engineers, Warrendale, PA.

53. Hughes, G.M. and P.J. Mill. (1974). Locomotion: Terrestrial. *The Physiology of Insecta*. Edited by M. Rockstein. Academic Press, New York and London. pp.335–379.

54. Inman, V.T., H.J. Ralston and F. Todd. (1981). *Human Walking*. Williams & Wilkins, Baltimore.

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

56. Isaacs, P.M. and M.F. Cohen. (1988). Mixed Methods for Complex Kinematic Constraints in Dynamic Figure Animation. *Visual Computer* 4(6):296–305.

57. Kajita, S., K. Tani and A. Kobayashi. (1990). Dynamic Walk Control of a Biped Robot along the Potential Energy Conserving Orbit. *Proc. of IEEE International Workshop on Intelligent Robots and Systems* (Tsuchiura, Ibaraki, Japan). 2:789–794.

58. Kato, T., A. Takanishi, H. Jishikawa and I. Kato. (1983). The Realization of the Quasi-Dynamic Walking by the Biped Walking Machine. *Fourth Symposium on Theory and Practice of Robots and Manipulators*. Edited by A. Morecki, G. Bianchi and K. Kedzior. Polish Scientific Publishers, Warsaw. pp.341–351.

59. Kochanek, D.H.U. and R.H. Bartels. (July 1984). Interpolating Splines with Local Tension, Continuity, and Bias Control. *Computer Graphics*. 18(3):33–41.

60. Lasseter, J. (1987). Principles of Traditional Animation Applied to 3D Computer Animation. *Computer Graphics*. 21(4):35–44.

61. Lathrop, R.H. (1986). Constrained (Closed-Loop) Robot Simulation By Local Constraint Propagation. *Proc. 1986 IEEE Int. Conf. on Robotics and Automation* (San Francisco). 2:689–694.

62. Lee, P., S. Wei, J. Zhao and N.I. Badler. (1990). Strength Guided Motion. *Computer Graphics*. 24(4):253–262.

63. Liston, R.A. and R.S. Moser. (1968). A Versatile Walking Truck. *Proc. Transportation Engineering Conf.* (Institution of Civil Engineers, London).

64. Maes, P. (1990). Situated Agents Can Have Goals. *Journal of Robotics and Autonomous Systems.* 6 (1&2).

65. Manko, D.J. (1992). *A General Model of Legged Locomotion on Natural Terrain.* Kluwer Academic Publishers, Boston.

66. Mann, R.W. and E.K. Antonsson. (1983). Gait Analysis— Precise, Rapid, Automatic, 3-D Position and Orientation Kinematics and Dynamics. *Bulletin of the Hospital for Joint Diseases Orthopaedic Institute.* Vol. XLIII(2):137–146.

67. Marion, J.B. and W.F. Hornyak. (1982). *Physics for Science and Engineering, Part 1.* Saunders College Publishing, Philadelphia.

68. Marsolais, E.B. and R. Kobetic. (1987). Functional Electrical Stimulation for Walking in Paraplegia. *Journal of Bone and Joint Surgery.* 69-A(5):728–733.

69. McGeer, T. (1990-A). Passive Dynamic Walking. *The International Journal of Robotics Research.* 9(2):62–82.

70. McGeer, T. (1990-B). Passive Walking with Knees. *Proc. of the 1990 IEEE Robotics and Automation Conference.*

71. McGeer, T. (1990-C). Passive Bipedal Running. *Proc. of the Royal Society of London.* B 240:107–134.

72. McGhee, R.B. and G.I. Iswahdhi. (April 1979). Adaptive Locomotion of a Multilegged Robot over Rough Terrain. *IEEE Trans. on Systems, Man, and Cybernetics.* SMC-9(4):176–182.

73. McKenna, M. (1988). *Cootie Gets Scared.* Computer animation. Produced at the Computer Graphics and Animation Group, Media Laboratory, Massachusetts Institute of Technology.

74. McKenna, M.A., S. Pieper and D. Zeltzer. (1990-A). Control of Virtual Actor: The Roach. *Proc. of the 1990 Symposium on Interactive 3D Graphics* (Snowbird, Utah). In *Computer Graphics*, 24(2):165–174.

75. McKenna, M.A. (1990-B). *A Dynamic Model of Locomotion for Computer Animation.* Master's Thesis, Massachusetts Institute of Technology.

76. McKenna, M. and D. Zeltzer. (1990-C). Dynamic Simulation of Autonomous Legged Locomotion. *Proc. of SIGGRAPH '90* (Dallas, TX). In *Computer Graphics* 24(4):29–38.

77. McKenna, M. and B. Sabiston. (1990-D). *Grinning Evil Death.* Computer animation, produced at the Massachusetts Institute of Technology's Media Laboratory.

78. McKenna, M. (1992). Interactive Viewpoint Control and Three Dimensional Operations. *Proceeding of 1992 Symposium on Interactive 3D Graphics* (Cambridge, MA). Association for Computing Machinery, New York, NY. pp.53–56.

79. McMahon, T.A. (1984). *Muscles, Reflexes, and Locomotion.* Princeton University Press.

80. Meglan, D.A. (1991). *Enhanced Analysis of Human Locomotion.* Ph.D. Thesis, The Ohio State University.

81. Messuri, D.A. and C.A. Klein. (1985). Automatic Body Regulation for Maintaining Stability of a Legged Vehicle During Rough-Terrain Locomotion. *IEEE Journal of Robotics and Automation.* RA-1(3):132–141.

82. Minsky, M. (1987). *The Society of Mind.* Simon and Schuster, New York.

83. Mochon, S. and T.A. McMahon. (1980-A). Ballistic Walking. *Journal of Biomechanics.* 13:49–57.

84. Mochon, S. and T.A. McMahon. (1980-B). Ballistic Walking: An Improved Model. *Mathematical Biosciences.* 52:241–260.

85. Moore, M. and J. Wilhelms. (August 1988). Collision Detection and Response for Computer Animation. *Computer Graphics.* 22(4):289–288.

86. Morlock, M. (1989). *A Generalized Three-Dimensional Six-Segment Model of the Ankle and Foot.* Ph.D. Thesis, The University of Calgary.

87. Murphy, M.C. and R.W. Mann. (1988). A Method for Estimating The Total Freedom of the Knee. *Modeling and Control Issues in Biomedical Systems.* DSC-Vol 12, BED-Vol. 11:55–65.

88. Muybridge, E. (1955). *The Human Figure in Motion.* Dover, New York.

89. Nashner, L.M. (1980). Balance Adjustments of Humans Perturbed While Walking. *J. Neurophysiology.* 44(4):650–664.

90. Onyshko, S. and D.A. Winter. (1980). A Mathematical Model for the Dynamics of Human Locomotion. *Journal of Biomechanics.* 13:361–368.

91. Ousterhout, J.K. (1993). *Tcl and the Tk Toolkit.* Addison-Wesley Publishing Co., Inc., (in press).

92. Pai, D.K. (1991). Least Constraint: A Framework for the Control of Complex Mechanical Systems. *Proc. of 1991 American Control Conference* (Boston, MA).

93. Pandy, M.G. and N. Berme. (1989). Quantitative Assessment of Gait Determinants During Single Stance Via a Three-dimensional Model— Part 1. Normal Gait. *J. Biomechanics.* 22(6/7):717–724.

94. Patriarco, A.G., R.W. Mann, S.R. Simon and J.M. Mansour. (1981). An evaluation of the approaches of optimization models in the prediction of muscle forces during human gait. *Journal of Biomechanics.* 14(8):513–525.

95. Pearson, K. (December 1976). The Control of Walking. *Scientific American.* 235(6):72–86.

96. Pfeiffer, F. and B. Gebler. (1988). A Multistage-Approach to the Dynamics and Control of Elastic Robots. *IEEE International Conf. on Robotics and Automation* (Philadelphia). 1:2–8.

97. Phillips, C.B. and N.I. Badler. (1991). Interactive Behaviors for Bipedal Articulated Figures. *Proc. of SIGGRAPH '91* (Las Vegas, Nevada). In *Computer Graphics* 25(4):359–362.

98. Pieper, S., J. Rosen and D. Zeltzer. (1992). Interactive Graphics for Plastic Surgery: A Task-Level Analysis and Implementation. *Proc. of 1992 Symposium on Interactive 3D Graphics* (Cambridge, MA).

99. Pieper, S., M. McKenna, D. Chen and I. McDowall. (1994). Computer Animation for Minimally Invasive Surgery: Computer System Requirements and Preferred Implementations. *SPIE '94: The Engineering Reality of Virtual Reality* Edited by S. Fisher and M. Bolas.

100. Press, W.H., B.P. Flannery, S.A. Teukolsky and W.T. Vetterling. (1988). *Numerical Recipes in C.* Cambridge University Press, Cambridge.

101. Procter, P. and J.P. Paul. (1982). Ankle Joint Biomechanics. *J. Biomechanics.* 15(9):627–634.

102. Pugh, D.R., E.A. Ribble, V.J. Vohnout, T.E. Bihari, T.M. Walliser, M.R. Patterson and K.J. Waldron. (1990). Technical Description of the Adaptive Suspension Vehicle. *The International Journal of Robotics Research.* 9(2):24–42.

103. Raibert, M.H. (1986). *Legged Robots That Balance.* MIT Press, Cambridge, MA.

104. Raibert, M.H. and J.K. Hodgins. (1991). Animation of Dynamic Legged Locomotion. *Proc. of SIGGRAPH '91* (Las Vegas, Nevada). In *Computer Graphics* 25(4):349–358.

105. Reynolds, C.W. (1982). Computer Animation with Scripts and Actors. *Computer Graphics.* 16(3):289–296.

106. Reynolds, C.W. (July 1987). Flocks, Herds and Schools: A Distributed Behavioral Model. *Computer Graphics.* 21(4):25–34.

107. Russell, M. (1983). Odex 1: the first functionoid. *Robot. Age.* 5(5):12–18.

108. Saunders, J.B., V.T. Inman and H.D. Eberhart. (1953). The major determinants in normal and pathological gait. *Journal of Bone and Joint Surgery.* 35A:543–558.

109. Schröder, P. and D. Zeltzer. (1990). The Virtual Erector Set: Dynamic Simulation with Linear Recursive Constraint Propagation. *Proc. of the 1990 Symposium on Interactive 3D Graphics* (Snowbird, Utah). Association for Computing Machinery, New York, NY. pp.23–31.

110. Siegler, S., R. Seliktar and W. Hyman. (1982). Simulation of Human Gait with the Aid of a Simple Mechanical Model. *Journal of Biomechanics.* 15(6):415–425.

111. Simkin, A. (1982). *Structural Analysis of the Human Foot in the Standing Posture.* Ph.D. Thesis, Tel-Aviv University.

112. Sims, K. (June 1987). *Locomotion of Jointed Figures over Complex Terrain*, M.S.V.S Thesis. Massachusetts Institute of Technology.

113. Singh, B., J.C. Beatty, K.S. Booth and R. Ryman. (1983). A Graphics Editor for Benesh Movement Notation. *Computer Graphics.* 17(3):51–62.

114. Steketee, S.N. and N.I. Badler. (1985). Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control. *Computer Graphics.* 19(3):255–262.

115. Stokes, V.P. and C. Anderson. (1989). Rotational and Translational Movement Features of the Pelvis and Thorax During Adult Human Locomotion. *J. Biomechanics.* 22(1):43–50.

116. Stredney, D. (March 1982). *The Representation of Anatomical Structures Through Computer Animation for Scientific, Educational and Artistic Applications.* M.A. Thesis. The Ohio State University.

117. Sturman, D. (1986). Interactive Keyframe Animation of 3-D Articulated Figures. *Graphics Interface '86, Tutorial on Computer Animation.*

118. Sutherland, I.E. (1983). *A Walking Robot.* Marcian Chronicles, Inc., Pittsburgh, PA.

119. Takanishi, A., H. Lim, M. Tsuda and I. Kato. (July 1990). Realization of Dynamic Biped Walking Stabilized by Trunk Motion on a Sagittally Uneven Surface. *Proc. of IEEE International Workshop on Intelligent Robots and Systems* (Tsuchiura, Ibaraki, Japan). Vol 1:323–330.

120. Terzopoulos, D. and K. Fleischer. (1988). Deformable Models. *Visual Computer.* 4(6):306–331.

121. Townsend, M.A. (1981). Dynamics and Coordination of Torso Motions in Human Locomotion. *J. Biomechanics.* 14(11):727–738.

122. Walker, M.W. and D.E. Orin. (1981). Efficient dynamic computer simulation of robotic mechanisms. *Proc. Joint Automatic Contr. Conf.* (Charlottesville, VA).

123. Walters, G. (August 1989). The Story of Waldo C. Graphic. ACM SIGGRAPH '89 Course Notes, *3D Character Animation by Computer.*

124. *Webster's Third New International Dictionary.* (1986). Merriam-Webster, Inc., Springfield, MA.

125. Wei, F.-C., H.-C. Chen, C.-C. Chuang and S.H. T. Chen. (1994). Microsurgical Thumb Reconstruction with Teo Transfer: Selection of Various Techniques. *Plastic and Reconstructive Surgery.* February 1994:345–357.

126. Wilhelms, J. and B. Barsky. (May 1985). Using Dynamic Analysis to Animate Articulated Bodies Such As Humans and Robots. *Proc. of Graphics Interface 85* (Montreal, Canada). pp.97–115.

127. Wilhelms, J. (June 1987). Using Dynamic Analysis for Realistic Animation of Articulated Bodies. *IEEE Computer Graphics and Applications.* 7(6):12–27.

128. Williams, M. and H.R. Lissner. (1977). *Biomechanics of Human Motion.* B. Le Veau, Editor. W. B. Saunders Company, Philadelphia.

129. Williams, L. (1990). Performance-Driven Facial Animation. *Computer Graphics.* 24(4):235–242.

130. Wilson, D.M. (1966). Insect Walking. *Annual Review of Entomology.* 11:162–169.

131. Winter, D.A. and D.G.E. Robertson. (1978). Joint Torque and Energy Patterns in Normal Gait. *Biological Cybernetics.* 29(3):137–142.

132. Winter, D.A. (1990). *Biomechanics and Motor Control of Human Movement.* John Wiley & Sons, Inc., New York.

133. Witkin, A. and M. Kass. (August 1988). Spacetime Constraints. *Proc. of SIGGRAPH '88* (Atlanta, Georgia). In *Computer Graphics* 22(4):159–168.

134. Witkin, A., M. Gleicher and W. Welch. (1990). Interactive Dynamics. *Proc. of the 1990 Symposium on Interactive 3D Graphics* (Snowbird, Utah). In *Computer Graphics* 24(2):11–21.

135. Wolfram, S. (1988). *Mathematica™, A System for Doing Mathematics by Computer.* Addison-Wesley Publishing Co, Inc., Redwood City, CA.

136. Yaeger, L. (1994). Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context. *Artificial Life III.* Edited by C.G. Langton. Addison-Wesley, Reading, MA. pp.263–298.

137. Yamaguchi, G.T. and F.E. Zajac. (1989). Restoring Natural Gait to Paraplegics Through Functional Neuromuscular Stimulation: A Feasibility Study. *Issues in the Modeling and Control of Biomechanical Systems.* DSC-Vol. 17:49–57.

138. Yang, G.-B. and M. Donath. (1988). Dynamic Model of a One-Link Robot Manipulator with Both Structural and Joint Flexibility. *IEEE Int. Conf. on Robotics and Automation* (Philadelphia). 1:476–481.

139. Yoon, Y.S. and J.M. Mansour. (1982). The Passive Elastic Moment at the Hip. *Journal of Biomechanics.* 15(12):905–910.

140. Zajac, F.E., E.L. Topp and P.J. Stevenson. (1986). A Dimensionless Musculotendon Model. *Proc. 8th Annual Conf. of the IEEE Eng. in Med. and Biol.*

141. Zajac, F.E. (1989). Muscle and Tendon: Properties, Models, Scaling, and Applications to Biomechanics and Motor Control. *Critical Reviews in Biomedical Engineering.* 17(4):359–411.

142. Zeltzer, D. (November 1982). Motor Control Techniques for Figure Animation. *IEEE Computer Graphics and Applications.* 2(9):53–59.

143. Zeltzer, D. (August 1984). *Representation and Control of Three Dimensional Computer Animated Figures.* Ph.D. Thesis., Dept. of Computer and Information Science, Ohio State University.

144. Zeltzer, D., S. Pieper and D. Sturman. (1989). An Integrated Graphical Simulation Platform. *Proc. of Graphics Interface 89* (London, Ontario). pp.266–274.

145. Zeltzer, D. (1990). Task Level Graphical Simulation: Abstraction, Representation and Control. *Making Them Move: Mechanics, Control and Animation of Articulated Figures.* Edited by N. Badler, B. Barsky and D. Zeltzer. Morgan Kaufmann Publishers, San Mateo, CA. pp.3–33.

146. Zeltzer, D. and M.B. Johnson. (1991). Motor Planning: An Architecture for Specifying and Controlling the Behavior of Virtual Actors. *Journal of Visualization and Computer Animation.* 2:74–80.

Appendix A *Corpus* Help

The following is the help file available in corpus, by typing “help.”

```
Parser/Control commands:
  quit
  version (print executable name)
  searchpath path_list
  do file_name <off> <#_of_times> (read in and execute a file script)
  dostop (in a script (do) file - terminates execution of script)
  toplevel (return to interactive input, from script)
  if val1[ `=' | `<' | `>' |`#' ]val2 command_string(# remainder-see blocksize)
  blocksize (denominator for `#' remainder function, see `if' above )

  (Integer Variable commands)
  set var value (eg "set F 10")
  %V=value (set integer variable. eg "%F=10")
  %V+ (increment variable. eg "%F+")
  %V+value (increment variable by value. eg "%F+10")
  %V- (decrement)
  %V-value (decrement)
  %V*value (multiple variable by value. eg "%F*10"
  %V/value (divide)
  %V#value (set variable to variable modulo value. eg "%F#10")

  # COMMENT (eg "# This is a comment line")
  ! Printed COMMENT
  @ Printed comment w/o '@' - ALWAYS printed

  verbose [silent, basic, extra, super or detailed]
       (set level of printed information for dynamics statistics)
  help (how else did you get here?)
  basicin [on|off] (set basic non-line-editiing input)
  silent (stop all optional output printing (such as the prompt))
  unsilent (restore normal output)

  "command" > "filename"  ( write 'printed' result of "command" into file )
  "command" >> "filename"  ( append 'printed' result of "command" into file )

  write file_name {multple lines until `newline-.'}  ( write to a file, like
       UNIX 'cat'. e.g. :
              write /tmp/ba
              this is the first line written...
```

```
          this is the second line
          .
     )

  commandlist | cl command-list-name {multiple lines till newline-.}
     The list can then be called using its command-list-name.
     Valid calls for a commandlist named "func" :
     func         (execute each command in the list "func")
     func 10      (execute "func" 10 times)
     func param(execute "func", and replace each "***" with "param")
     func param 10(execute "func" 10 times, substituting "***")
     (Warning: avoid mixing "***" substitutions with "%" variables)
  commandloop | cloop [#_of_times] {multiple lines till newline-.}

  savelist | sl save_list_name {multiple lines till newline-.}
  playlist | pl save_list_name [command preamble]
     (execute next command in save-list with command preamble)
  playlistn | pln save_list_name list_index_# [command preamble]
     (execute nth command in save-list with command preamble)
  playlistfull save_list_name [command preamble]
     (execute every command in save-list with command preamble)
  foreach save_list_name command-string (command-string is executed, once
     for each element in the save_list, replacing "***" with the element)
     (Warning: avoid mixing "***" substitutions with "%" variables)


  frame [frame#] (set internal frame #)
  trackflush (remove all tracks)
  track start_frame end_frame command-string (Sets %A to rel frame #- called
     only in the start-end interval)
  trackf start_frame end_frame command-string (appends rel frame # to command
     end- called for all frames, less than start passes 0, greater than
     end passes (end-start))
  tracka start_frame end_frame command-string (Sets %A to frame number- called
     for all frames, less than start %A=0, greater than end %A=(end-start))
  trackc start_frame end_frame command-string (Sets %A to frame number-
     cycles, called all frames)
  frametrack | ft - execute track commands for current frame

  system shell-command-string
  wait (performs wait() to wait for a vforked process to finish)
  vfork shell-command-string (five arg's max, fork off shell command)

  ppopen name command-string
  ppread name
  ppwrite name command-string
  ppdo name

Animation control commands:
  render <{on|off}>
  renderprint {on|off} (toggle whether or not "rendering...done" is printed)
  wireframe | wf
  retep (render using alternate scan converter)
  filtertype val (val: box=0, pyramid=1, bartlet=2, stochastic=3) RETEP ONLY
  dither float_val
  renderwindowsize x y [not yet implemented in retep]
```

```
renderwindow {on|off}
renderpasses #_of_strips
renderdisplay (enable rendering to display)

render2file filename alpha-filename z-filename
autofilename filename (for auto file2render)
autofilenumber #_to_tag_onto_autofilename
incrfilenumber
autorender2file

antialias | aa {on|off}
mssize samples z-buffer-bits stencil-buffer-bits(0) (sgi antialiasing params)
hardware | hw {on|off}
clear {on|off}  (clear between renders)
doublebuffer | db {on|off}
doublebufferautoswitch | dbas {on | off} (switch double buffer on render)
doublebufferswitch | dbs (switch double buffer)

fb_open (open the framebuffer)
fb_close (close the framebuffer)

backgroundcolor r g b
setpenwidth #_of_pixels (for wireframe)
setpencolor r g b

(Hardware Drawing Modes)
edgeson r g b
edgesoff
hatch
hollow
outline
point
solid

(Camera commands)
lookat <obj> <x y z>
eye <obj> <x y z>
forward distance (move forward along view normal)
back distance (move back along view normal)
roll angle
up x y z
uprel [obj] x y z (up direction relative to eye)
perspective field_of_view_degrees aspect_ratio near far
apectratio | ar [screen aspect ratio]
fov [field_of_view_value]
nearfar | nf [near-dist far-dist]
windowsize center-x center-y half-size-x y
windowcenter center-x center-y
windowhalfsize half-size-x y
viewdistance | vd {val}

screenmix x y z
screenmax x y z
screensize x y

(Special camera commands)
```

```
holoshear frame_# (0-99, special purpose shearing for stereogram)
dumpcam {file-name} (dump (some) camera information)
dumpcambob {file-name}
     (dump camera information in a different aspect ratio format)
vdcam <obj> <x y z> (special view-dependent camera)

(Transformation modes, for "move", "rotate", "scale")
premult
postmult
localxforms
xformcenter x y z

get obj_name [from] instance_file_name
closeobj obj_name
whereis obj_name (dump x y z of centroid of object in world space)
setmatrix obj [next four lines are matrix
dumpmatrix obj <filename>
dumpmatrixa obj <filename>
dumpmatrixwsp obj <filename> (dump world space matrix)
mcopy obj-src obj-dest
mcopywsp obj-src obj-dest
boliodump (dump ascii matrixes of all objects)
countpolys
countpostedpolys
filtermat obj-name filter-obj-name predition_scale(try 1)
     (filter an objects motions, based on current and last matrix)

sharememory on|off (duplicate instance point data with share memory off)

(Hierarchy commands)
pushobj obj_name (specify a parent object-
     until "popobj" any object which you "get" will be a child)
popobj

(Graphical Object Transformation commands)
init obj
harden obj (harden object points based on xform matrix, then init matrix)
scale obj x y z
transscale obj x y z ( scale an objects translationsl values only)
center obj (puts centroid at origin)
placex obj (puts objects centroid at a specific x loc)
placez obj (puts objects centroid at a specific z height)
move obj <obj> <x y z>
movely obj1 obj2 x y z ( move obj1 by the amount of (x y z)
     transformed from the coordinate frame of obj2 )
rotate obj {x|y|z} angle
rotateaxis obj x y z angle
align object object_axis_x y z align_axis_x y z
interpolateobj obj1 obj2 t(0-1) obj_dest (point interpolation)
shear obj axis-x y z scale-initial scale-slope
     (shear an object- proportional scale along an axis -  hardens object)
shear1d obj axis-x y z up-x y z scale-initial scale-slope
     (shears along one direction only - 'up' vector is unaffected)

(Graphical Object Shading commands)
color obj r g b (0-1)
icolor obj r g b (0-255)
pcl obj pcl-filename (reassign poly colors with pcl file)
joinall .asc-filename (merges all objects into one, in wsp)
defaultcolor obj r g b (0-1, set color for all new objects)
defaulticolor obj r g b (0-255)
defaultdefaultcolor (restores default (1, 1, 1))
specularcolor obj r g b (0-1)
ispecularcolor obj r g b (0-255)
defaultspecularcolor r g b (0-1, set specular color for all new objects)
defaultispecularcolor r g b (0-255, set specular color for all new objects)
defaultdefaultspecularcolor (restores default (1 1 1))
shadeparam obj diffuse% specular% specular-exponent ambient%
defaultshadeparam diffuse% specular% specular-exponent ambient%
defaultdefaultshadeparam (restores to .6 .4 30. .025)
trans obj val (0-1, 0=opaque)
pointtrans obj index val (0-1, 0=opaque, set transparancy at point)
polycolor obj poly# [r g b]
polyicolor obj poly# [r g b]
shademodel obj {p|g|h}  (phone or gouraud or hand-grenade)
defaultshademodel {p|g|h}  (phone or gouraud or hand-grenade)
cull obj {on|off} (perform back-face cull)
concave obj {on|off} (set concave polygons for an object, so that it renders
     correctly (currently needed on SGI hardware) )
facet obj
defaultfacet
smooth obj
defaultsmooth
groupsave filename    (??)

post obj (show object)
unpost obj (hide object)

(Special Object Commands)
objbound objname x1 y1 x2 y2 filename.asc
     (write out .asc file, clips object to x-y bounds, obj local space)
objibound objname x1 y1 x2 y2 filename.asc
     (write out .asc file, clips object outside of x-y bounds, local space)
objzlimit objname z-min z-max filename.asc
     (writes out object setting all points to fit into z-min, max in wsp)
objxyuv objname [filename.uv]
     (writes out a uv file using the xy bounding box to map flat,
      AND sets the object uv)
renderuv objname filename
     (writes out uv file using display space as texture coords-
      like a slide projector over rendered image, AND sets obj uv)
objuv inst_name file-name(.uv file) (loads uv file for an object)

(Texture and Transparency Map commands)
getmap <mapname> from <fname> x-size y-size x-repeat y-repeat\n");
getmapa <mapname> from <fname> x-size y-size x-repeat y-repeat\n");
     (get rgba texture map)
getmapbw <mapname> from <fname> x-size y-size x-repeat y-repeat\n");
     (get black & white map (one byte))
getmapbwa <mapname> from <fname> x-size y-size x-repeat y-repeat\n");
     (get black & white map with an alpha channel)
objmap objname textmap reflmap obj% text% refl%
```

```
transmap objname map-name (assign transparency map)
pointuv inst_name vertex_ind# texture_u texture_v
mergergba rgb-file alpha-file width height rgba-file-dest
     (make rgba file from rgb and alpha file)
makemip rgb-file width height mip-file-dest mip-dimension
makemipa rgba-file width height mip-file-dest mip-dimension
makemipbw a-file width height mip-file-dest mip-dimension

(Light commands)
(default light is light.1)
lightmake light_name (make light in global, default list)
lightmakeone light_name (make light not in default list)
lightclose light_name (kill light)
lighton light_name
lightoff light_name
pointlight light_name (make light point source)

lightobj light_name obj_name (light object with light- private list)
defaultlightobj obj_name (lightobject with default lights- private list)
defaultlightallobj (light al objects with default lights- private list)
unlightobj light_name obj_name (remove light from object- private list)

lightcolor light_name [r g b] (0-1)
ilightcolor light_name [r g b] (0-255)
lightwhite light_name b (0-1)
lightdimmer light_name val (0-1, set brightness (not color) of light)
lightlinear light_name (linear falloff)
lightconstant light_name
lightexp light_name
lightexponent light_name exponent (exponent for exponential falloff)
lightpos light_name <obj> <x y z>
lightpoint light_name <obj> <x y z>
lightangle light_name theta
lightdeltaangle light_name theta (falloff angle for cone light)
lightshadow light_name [ file-name| NULL ]

(Shadow commands)
shadow obj
shadows {on|off}
unshadow obj
shadowbias bias1 bias2
shadowres resfactor
shadowminsize val
shadowmapsize width height
shadowlightdensity lightname density(0-1, 1=fully dark)

hazelevel val
hazediatance val

(Video Deck Control commands)
animate {on|off}
     (when "animate off", "vpranimate" and "bvw-animate" will do nothing)
vpranimate #_of_frames (ampex or beta)
bvw-animate #_of_frames (beta)
animateinit (set new animation starting timecode to current value)
bvw-search hh:mm:ss:fr
```

```
bvw-forward hh:mm:ss:fr
bvw-readtc
eject


(General Bitmap Manipulation commands)
blitread {filename|`screen'} blitname w h [x-start y-start] [alphafile]
           (save bitmap on screen for specified size in memory blit)
blitreadblur {filename|`screen'} blitname width height
     (same as blitsave, but with a `blur' style file- argb banks)
blitreadbyte filename blitname x-width y-height
blitwrite filename blitname
blitwritebyte filename blitname - save only red
blitmake new_blitname width height red green blue (ints)
blit blitname x y (put up memore blit starting at x y)
blitlayer blit-background blit-foreground blit-dest
blitlayerb blit-background blit-foreground blit-dest (alpha reduction not
           built into blit-foreground-rgb)
blitcopy blit_src blit_dest start_x y width height
blitinsert blit_src blit_dest dest_x y
blitlayertrans blit1 blit2 blitdest trans-factor
blitlayerred blit-background blit-foreground blit-dest (uses red as alpha)
blitramp blit blit-dest ramp (brightness scale)
blitscale blit blit-dest int-scale (size scale)
alphablit rgb_file alpha_file (mat in file over saved memorey blit)
glopen width height - special window open for using blits w/ gl on SGI

(Screen Bitmap Manipulation commands)
bitsave x y width height file_name
bitload x y width height file_name
bankload x y width height file_name
bitmove x_src y_src x_len y_len x_dest y_dest
alphaload bit_file alpha_file (mat in 640x512 file over upper left 640x512)
turnover x y width height (mirror bitmap 180 degrees)
turnaround x y width height (mirror bitmap 180 degrees)
bitgamma x y width height gamma (perform hard gamma correct on bitmap-
           permanently changes bitmap)
bitramp x y width height val (perform hard brightness scale on bitmap)
bitrotate x y w h

(Save potion of screen, in the old "bob" file format)
bobdump x_src y_src x_len y_len file_name (old bob-format bitmap save)
bobload x y filename
bobloadb x y file r g b (0-255, load with transparant background color)
bobloadt x y t file_name (load with transparancy=(255 0 0))

enlarge (pixel duplicate for 4x enlargement)

label (draw fonts to screen. see /cga/bin/label for options)

(Antialiased Line Drawing commands)
lineinit
linecolor r g b (0-255)
linetrans val (0-255)
lineclip x-min y-min x-max y-max
lineclipoff
```

```
  linemove x y thick
  linedraw x y thick
  linedone

  fill_color r g b (0-1  doesn't do much)
  rectangle x1 y1 x2 y2 (doesn't do much)


Dynamics:
  addworld world_name (create a dynamic environment for bodies)
  setworld world_name

  addcorpus corpus_name
       (create an articulated body collection, in the current world)
  setcorpus corpus_name
  corpusinit
  addbody body_name instance_name joint_axis_x y z { sliding | rotary }
    density [1=inertial  1=colliding 1=convex 1=colliding_vel_normal_test]
       (create a rigid body, in the current corpus)
  addpart part_name inst_name body_name density
    [1=inertial  1=colliding 1=convex 1=colliding_vel_normal_test]
  setroot body_name
  setjoint body_name { joint_axis_x y z joint-type }
  setjointp b_name joint_axis_x y z joint-type
    (set joint axis direction in parent's frame)
  jointalign body object object_axis_x y z
  deletejoint body-name
       (removes body as one with a joint- making it a rigid part of parent)
  killjoint body-name
       (entirely removes body- no more mass or collision detection)
  bodygroundstick b_name on|off
  bodygroundtest b_name on|off (set individual body ground testing on or off)
  maxv body_name max_linear max_angular
  addv body_name linear_x y z angular_x y z (specified in WORLD space)
  setv body_name [ linear_x y z angular_x y z ] (specified in LOCAL space)
  rmsv (returns the RMS of all of the joint velocities, for the current corpus)
  ifrmsv tolerance command (if the RMS velocity of the joints is below the
       given tolerance, then do the command-string)
  setrootmatrix [4 lines of 4 floats, the transform matrix for the local space]
  setrootpos (harden current position of the corpus)
  xform (update the xforms for current corpus)
  updatev (update all spatial velocities in the current corpus,
    useful after reading in a state file)
  scalelengthv scale-val - scale all bodies linear velocities
  scaletimev scale-val - scale all bodies velocities by a time factor
  linkbodies parent child
  integrate corpus_name [on|off]
  go { # of frames }
  motor body { etarget | ... } value1 value2
       motor etarget target-joint-pos motor-time (exp spring)
       motor eatarget target-exp-spring-ea motor-time (exp spring ea param)
       motor ktarget target-joint-pos motor-time (linear spring)
  motorfile motor-state-file etarget duration
  jointmatchexp body_name (set exp spring to the current joint angle)
  jointmatchqtoexp body_name (set joint angle to the current exp spring)
  joint body { q | dq | ... } value (set a joint parameter)
```

```
    q, dq, ddq, Q, b, k, k_q, ea, eB, e_q, e_maxq,
    jlim_q1, jlim_q2, jlim_k1, jlim_k2, jlim_b1, jlim_b2,
    jlim_ea1, jlim_e2, jlim_eB1, jlim_eB2,
    springtype: constant, mass, distalmass, totalmass
    Q_type: QOPEN (0), QDAMP (1), QSPRING (2), QDAMPSPRING (3), QBIAS (4),
            QJLIM (8), QEXPSPRING (16), QJLIM_EXP (32) - "or" these together
  jointstatus body_name
  jointQsum body [0 | off]
  jointQabssum body [0 | off]
  jointQsumtoQ body
  QtoQbias - copy each body's Q force to the Q bias term
  QbiastoQ
  QtoE_q - do inv control from Q force to exp spring rest angles

  addmuscle muscle-name body-name
  muscle muscle-name { k | b | ... } [value]
    (set or query a "muscle" force parameter)
    Q (bias force),
    b (linear damping),
    k, k_q (linear spring stiffness and rest angle),
    ea, eB, e_q, e_maxq (exponential spring stiffness parans, rest angle,
        maximum clamping force),
    jlim_q1, jlim_q2, jlim_k1, jlim_k2, jlim_b1, jlim_b2,
    jlim_ea1, jlim_e2, jlim_eB1, jlim_eB2 (joint limit angle, stiffnesses,
        linear dampers, and exponential springs),
    springtype: constant, mass, distalmass, totalmass (scaling factor for
        forces),
    body2 (name of second body for 2 joint springs, attach forces, etc.),
    k2a, k2b, k2a_q, k2b_q, (2 joint spring stiffness, rest angles)
    attach_k, attach_ea, attach_eB, atatch_e, attach_break,
        attach_insert1 (x y z), attach_insert2 (x y z) (spatially linear
        forces- linear and exponential stiffnesses, w/ an
        optional break length, and 2 insertion coordinates local
        to the 2 bodies)
    pulleypos [x y z], pulleyaxis [x y z], pulleybody [body-name],
       pulleyr [value]
    Q_type: QOPEN (0), QDAMP (1), QSPRING (2), QDAMPSPRING (3), QBIAS (4),
        QJLIM (8), QEXPSPRING (16), QJLIM_EXP (32), Q2LINEAR (64),
        QATTACH (128), QPULLEY (256)
        (reference # for the "type" of muscle; "or" these together)

  extbias body_name wsp_linear_force_x y z local_body_point__x y z
    (constant external linear bias force applied to body-wsp force,local point)
  rootmotion {fixed|free|kinematic|constrained}
  jointmotion {free|kinematic}
  treepoints body-name local_x y z [file_name]
  totaljoints (print total # of DOF's (degree of freedom) of current corpus)
  totalmass (print total mass of corpus)
  mass bodyname (mass of body, distal mass, total-fraction...)
  massonly bodyname (only the mass of the one obj- ok before corpusinit)
  partmass partname (mass of a 'part' in a body)
  density bodyname (returns d of first part only)
  corpuscog marker_obj [NOTE: sets localxforms]

  (State Saving and Loading Commands)
  dumps file_name (dump main state of corpus)
```

```
loads file_name
dumpQs file_name (dump joint force values)
dumpas file_name (dump acceleration values)
loadas file_name
dumpmotors file_name (dump motor progams state)
loadmotors file_name
dumpmatcorpus file_name (dump script to position all objects for rendering)
dumpIs (show Inerta tensor)
dumppv (show bias force)
dumpcog body_name (show center of gravity for a body)
listbodies {file-name}
dumpbodyq body-name [file-name]
dumpbodyqa body-name file-name ( append to file )
dumpbodyX body-name [file-name]   (dump body 6x6 Xforms)
dumpcontacts file_name (dump special contact data)
loadcontacts file_name
dumpcontactforces [file_name]
dumpcontactforce body-name [file_name]

contactfsum body-name [0 | off]

commandtree command-string-for-each-body(**=body-name)
commandtreeparts command-string-for-each-part(**=part-name)
commandtreein command-string-for-each-body(**=body-name) (leaves-inward)
commandtreeout command-string-for-each-body(**=body-name) (leaves-outward)

addmatlist
initmatlist
setmatlist
loadmatlist

(Integrator Commands)
iter int (loop each frame int times)
dt [time] (set time step for frame)
h [time] (set time step for next integration)
eps [val] (set integrator error tolerance)
time [val] (set current time)
inctime (increment time by dt)
pushtime
poptime  (one level of push only)

integration [rkfixed | rkvar | rkf | euler]
order newt|arist (set 2nd order (Newtonian) or 1st order (Aristotelian)
                  dynamics solution)

(Constant Acceleration Gravity)
setgrav val (set gravitational acceleration value)
setgvect x y z xr yr zr (set gravitational acceleration value)
grav [on|off]

("Equal-Attraction" Gravity Commands)
SetG val (set Newtonian gravitational constant)
Gravity [on|off]
Gattract body1 body2 (put two bodies on gravity attaction list)

ground [on|off]
```

```
  groundtype [flat | trigrid]
  addtrigrid grid_name instance_name
  groundk [val] (set ground linear spring const)
  #  grounde [val] (set ground coefficient of elasticity) NOW INACTIVE
  groundfric [val] (set coefficient of sliding friction)
  groundfricb [val] (set damping for pseduo-static friction)
  groundb [val] (set ground damping const)
  groundea [val] (set ground exponential spring linear strength)
  groundeB [val] (ground exp rise val)
  groundstickea [val] (sticky ground exp spring linear strength)
  groundstickeB [val] (sticky ground exp spring rise strength)
  groundstickemaxz [val] (maximum force penetration val)
  groundsticke [val] (ground stick force coefficient of restitution)
  groundz [val] (set world space value for z-ground plane)
  groundmassscale [on|off]
       (scale all vertical reaction forces by the corpus mass)
  collide body1 body2 (set collision detection between b1 and b2)
  collision [on|off]
  collisionanalysis [on|off]
  collisione [val] (set collision restitution for collision analysis)
  collisionfric [val] (sliding friction value for collisions)
  collisionfricb [val] (sliding friction damping value for collisions)
  collisionb [val] (set collision damping)
  collisionea [val] (exp spring linear strength)
  collisioneB [val] (exp spring rise)
  collisionemaxz [val] (maximum force depth value)
  collisionlog file_name

  attach body1 body2 body1_x y z body2_x y z k ea eB e [break_length]
  attachb body damping_value
  attachf body (prints last attach force vector)
  attachfsum body-name [0 | off]

  alarm time command_string
  timer delta_time command_string
  timerflush (activate all timers which are triggered by current time)


Hexapod Gait control commands:
  addroach roach_name
  setroach roach_name (set current roach for other commands below)
  metabolism time (set's protraction, dleg, retraction, cycle time)
  speed val  (1.0=top speed, 2.0=1/2 speed of 1.0)
  deltaspeed val
  incrspeed
  topspeed val
  bottomspeed val
  protime time
  rettime time
  cycletime time
  dlegtime time
  gaitgo
  gaitinit
  procom leg_number [list of commands for protraction]
  retcom leg_number [list of commands for retraction]
  stepreflex {on|off}
```

```
legstepreflex leg_number body_name trigger_angle trigger_dir(`+' or `-')
loadreflex {on|off}
legloadreflex leg_number body_name joint_force
legstatus leg_number
```

Appendix B *Corpus* Tutorial

B.1 Starting with *Corpus*

This appendix is intended to help familiarize the reader with the control of *corpus*. This section is not intended to be a complete manual for *corpus*, but rather, it should serve as a guide to the basic operations and concepts in *corpus*. Users of *corpus* need to be familiar and comfortable with standard computer graphics concepts such as polygonal graphical objects, viewing parameters, object transformation, rendering concepts, etc. To fully utilize *corpus*, users should also be adept with command line interfaces and programming concepts.

ASCII commands are used to control *corpus*, either through keyboard input, or through file scripts. "Keyboard input" actually refers to UNIX `stdin`, therefore input to *corpus* can be redirected from any text source, including files, and the output of other programs.

Corpus is invoked at the command line by typing:

```
corpus
```

Corpus can be invoked in a "silent" mode in which it prints out information only when specifically instructed to, using a command. This silent mode can be useful when the input and output of *corpus* is linked to another controlling program. To run *corpus* in the "silent" mode, type:

```
corpus -s
```

The "silent" mode can also be toggled with the *corpus* command `silent`, as in:

```
silent on
silent off
```

Be sure to refer to **Appendix A Corpus Help** where the full *corpus* command set is listed. Not all *corpus* commands will be discussed here, but **Corpus Help** should be a useful guide, once the basic concepts are learned from this Appendix. In addition, **Appendix C Dynamics Verification** presents a set of dynamic simulation scripts for *corpus*, which can provide additional tutorial material.

B.2 Graphical Operations

This first *corpus* script performs some basics: loading an object from a file and rendering a view of it.

Script 7: A *corpus* script to render an image of a graphical object

```
# any line which begins with a "#" symbol is a comment, in corpus

# set a dark blue background color for rendering
backgroundcolor 0 0 .2

# load a graphical object in the "osu" format
# the object will be named b, the file name is cube.asc
get b from cube.asc

# set some object shading parameters: color, etc.
color b .5 0 0
#   shadeparam obj diffuse% specular% specular-exponent ambient%
# we want an object which is mostly diffuse, with a little ambient color
shadeparam b .8 0 0 .2

# By default objects are smooth shaded, but the cube should appear
#     faceted
facet b

# set some viewing parameters
# set the camera or "eyepoint" location in space
eye 5 5 1
lookat 0 0 0
# alternately, we could look at the object's centroid, using:
#lookat b

# render the scene.
render

# by default, rendering will be accomplished using high speed hardware
# to turn off hardware rendering, to use software rendering:
hw off
```

```
# set the file to render into
render2file /tmp/frame

# set the size of the rending output
screensize 320 256

# we will have 'square' pixels- set the aspect ratio (320/256=1.25)
ar 1.25

# turn on anti-aliasing
aa on

# render the scene, using software scan conversion
render

# we can execute a shell command or program at any time.
# call compress function to reduce the size of the file
# which was just rendered
system compress /tmp/frame
```

Graphical objects can be transformed in a number of ways in *corpus*. Transformations effect an object's transformation matrix, used for display and geometric computations. The following is a partial list of commands that transform graphical objects:

```
init obj
scale obj x y z
move obj <obj> <x y z>
rotate obj {x|y|z} angle
rotateaxis obj x y z angle
```

These commands create transformation matrixes using the standard computer graphics forms. [Foley]

The way in which each of the transformation commands affects the object's matrix depends on the current *transformation mode*. The `premult` command sets the transformation mode to "pre-multiply," such that additional transformations to a graphical object will be pre-multiplied into the objects transformation matrix. The effect is as if the transformation becomes the "first" one performed on the object. In contrast, the `postmult` command specifies the "post-multiply" transformation mode, such that an additional transformation command is the "last" one performed. For example, consider the sequence:

```
move obj 1 0 0
scale obj 2 2 2
```

In post-multiply mode, the object would first move by 1 in *X*, and then, when it is scaled, it will not only double in its linear dimension, but also move an additional 1 in *X*, because the previous transformation is taken into consideration and the original offset of 1 is scaled by 2 as well. In pre-multiple mode, the object moves, and then doubles in size, without moving again, because the scale is performed "first," before the move. When multiple transformations are made on an object, the pre-multiply mode can become difficult to conceptualize; in post-multiply mode, all new transforms are taken with respect to the previous ones.

The `localxforms` command is a post-multiply mode which allows scale and rotate transforms to occurs with respect to an object's geometric centroid, so the object scales or rotates about that point, without regard to the object's location in worldspace. Similarly, the `xformcenter` command allows the user to set the point in worldspace about which rotations and scales will occur.

B.3 Dynamics in *Corpus*

The first step to use dynamics in *corpus* is to create a "world," or a dynamic environment. The *corpus* command `addworld` is used for this purpose. The first `addworld` command prepares *corpus* to begin handling dynamic objects. A "world" hold parameters, such as the gravitational force vector and ground contact parameters, which are shared among multiple dynamic objects. Multiple "worlds" can be created in *corpus*, representing different environmental conditions.

The next step to using dynamics in *corpus* is to create a new (potentially articulated) body, known as a "corpus." The `addcorpus` command is used for this purpose. "Bodies" and "Parts" (discussed below) are then added to the new "corpus." When the articulated body (or single rigid body) has been constructed, the new "corpus" is initialized, with the `corpusinit` command. A new "corpus" could now be defined, or we could begin simulation.

As an example, let us now look at a simple script to construct a single "corpus," which consists of a single cube.

Script 8: A *corpus* script to create a dynamic object

```
# create a default dynamic environment
```

```
addworld w

# create a new dynamic object (a corpus)
addcorpus cube-corpus

# load a graphical object
get cube-obj from .cube.asc

# addbody body_name inst_name joint_axis_x y z {sliding|rotary} density
# Add a dynamic body (body_name), defined from a graphical object
#          (inst_name) to the current corpus
# Bodies in general have a joint axis, for a single body the joint
#     params do not make a difference
# the density is in kg/m^3 . 1000 = water
addbody cube-body cube-obj 0 0 1 rotary 1000

# set that body to be the "root" object in the corpus (even though there
#     is only one body), and initialize
setroot cube-body
corpusinit
```

We could now simulate the motion of this body. For example, we could simple activate the gravitational force with the *corpus* command `grav on`, and the simulator would generate a downward acceleration in the body (See **Appendix C Dynamics Verification**). To compute the simulation, the *corpus* command `go` is used, which instructs the system to simulate forward in time, for the duration of the current timestep, which is specified by the `dt` command. For example:

```
# set the timestep to 1/30 of a second (like video)
dt 0.033333
# simulate 1/30 sec ahead
go
# look at the results
render
# simulate 30 timesteps- 1 second
go 30
```

There are a number of parameters in *corpus* used to control the numerical integrator. The `integration` command selects the type of integration: `euler`, `rkfixed` (the 4th order fixed step Runge-Kutta), and `rkf` (the 4-5 order adaptive step size Runge Kutta). The error tolerance of the `rkf` method is set with the `eps` command. For example:

```
integration rkf
# if the diff between the 4th and 5th order solution is > 0.0001, subdivide
eps .0001
```

To construct an articulated figure, multiple links or "bodies" are added to the corpus. First, their local coordinate frames and joint axes are defined. Then, the parent and child connections are established. To conclude the initialization process, the `rootbody` is specified, and the `corpusinit` command is entered.

To define the local coordinate frames for the bodies, the first step is to load the graphical objects which correspond to them. Using the appropriate transformations, the graphical objects are placed into their local coordinate frame (using the standard *corpus* transformation commands, such as `scale`, `move`, etc.). During this process the graphical object must be scaled to the intended size of the dynamic object (*corpus* uses the MKS system (Meter-Kilogram-Second), so a distance of 1 in the graphical environment correspond to 1 meter). The root body can be placed anywhere within its own local frame, but every other body must be positioned (at this stage of the process) such that its joint axis has its origin at the world-space origin. For example, let's examine the process of creating a 2-link pendulum with the following script.

Script 9: A script to create a dynamic articulated figure in *corpus*.

```
addworld w
addcorpus c

get base from ../data/unit_cube
# the base will be fixed in place. It serves as the immobile connection
#          for the top of the pendulum. (it is the root object)
# make the base object 1 cm per side
scale base .01 .01 .01
unpost base

get b1 from ../data/unit_cube.asc
get b2 from ../data/unit_cube.asc

# scale it to be 1 long in Y, w/ a cross-section of .05 m
postmult
scale b1 .05 1 .05
scale b2 .05 1 .05

# move the joint to the origin
move b1 0 .5 0
move b2 0 .5 0
```

Both of the pendulum body-link segments, b1 and b2, have been scaled so that they are 1 meter long in *y*, and 5 cm wide in *x* and *z*. This is their "physical" size, which they will maintain from this point on. They have both been positioned, currently overlapping in space, such that one of their long ends lies at the worldspace origin (the point {0, 0, 0} in the graphical environment). By positioning each body in this manner, we have defined the location of the body's joint (at {0, 0, 0}) with respect to its graphical object (at one of its ends). Now that the local coordinate frames are ready, they are set using the addbody command.

```
# 'base' is the rootbody,without a real joint
addbody base base 0 0 1 rotary 1000

# addbody body_name inst_name joint_axis_x y z {sliding|rotary} density
# Add a dynamic body (body_name), defined from a graphical object
#          (inst_name) to the current corpus
addbody b1 b1 1 0 0 rotary 1000
addbody b2 b2 1 0 0 rotary 1000
```

We have specified rotary joints, along the *X* axis, and a density of 1000 kg/m^3 (like water).

The next step is to define the transformations which are used to place each body into the coordinate frame of its parent body. In our example, the body "b1" does not need any additional transformation to connect to its parent, the "base." This body can now be connected to its parent. We define the parent-child relationship with the linkbodies command:

```
# define transform between parent and child, and define link
# linkbodies parent-body-name child-body-name
linkbodies base b1
```

The joint of body "b2" is located at the distal end of body "b1." The location of the distal end of body "b2," in its initial configuration is {0, 1, 0}. Recall that "b2" is 1 meter long. We place "b2" in the frame of "b1" by moving it as follows:

```
move b2 0 1 0
linkbodies b1 b2
```

We can now initialize the articulated corpus:

```
setroot base
corpusinit

# we do not want the base to move.
rootmotion fixed
```

```
grav on
ground off

integration rkf
dt .03333333
eps .0000001
```

Additional graphical objects can be added to a link during the corpus building process. Additional objects are called "parts." "Bodies" are the main objects associated with a link and its joint. The joint and link properties are accessed through the "body's" name. When a "part" is added to a body, it is attached to it rigidly, as if it were a part of the link. The part (optionally) contributes mass to the link, and (optionally) its geometric surface is used for collision detection. The addpart command adds additional objects to a link (body):

```
addpart part_name inst_name body_name density
```

Other options exist for the `addbody` and `addpart` commands. Four binary flag terms may be added to the end of either command:

```
addpart part_name inst_name body_name density
    [1=inertial 1=colliding 1=convex 1=colliding_vel_normal_test]
```

The first flag (a "1" or "0") is the "inertial" flag, which sets whether the body or part should contribute any mass to the articulated figure. Each link should always have some mass associated with it, whether from its primary body or an added part (or any combination). The second flag, the "colliding" flag indicates whether the body or part should be used for collision detection. The final two flags are less important, and are used to assist the collision analysis (impulse mechanics, for non-articulated bodies only). The third flag is used to indicate if the part of body is geometrically convex. The fourth flag indicates whether or not to use a particular test in collision analysis (the "colliding-velocity-normal-test," which will cancel an impulse response of the "colliding" bodies if they are moving apart from each other). Without specifying these flags they are assumed to all be "1."

Use of these flag allows for special structure to be built into an articulated figure. For example, parts could be added to a link which have a great amount of geometric detail, but which are not considered during collision detection for efficiency. Or, geometrically simplified objects can be added as parts which contribute no mass, but are used for collision detection to reduce computation time. Different densities can be used with different parts to encode special geometric mass distribution of a link.

Joint parameters are accessed through the `joint` command. Values such as the joints position (q), velocity (dq), and acceleration (ddq) can be set or queried, as follows:

```
joint body-name q 1
joint body-name dq 10.2
joint body-name dq
          10.2
```

Parameters pertaining to joint spring, dampers, etc. are also set using `joint`:

```
# set joint damper constant to 10
joint body-name b 10
# set joint linear spring constant to 1.2
joint body-name k 1.2
# set joint linear spring rest angle to 3.1 (radians)
joint body-name k_q 3.1
```

The `jointstatus` command displays the values of the major joint parameters. The joint parameter, "Q_type," sets the kind of joint forces which are active at the joint. Each type of joint force generator (damper, spring, etc.) has a unique value (see **Appendix A Corpus Help**), and to combine types, their values are added (or logically ORed) to form the Q_type. For example:

```
# set active joint forces to damper (1) and linear spring (2).(1 + 2 = 3)
joint body-name Q_type 3
```

Additional force generators can be added to a body, using the `addmuscle` command. A "muscle" in *corpus* contains the same type of force generators that are available via the `joint` command (as well as a few additional types). The muscle parameters are accessed via the `muscle` command, in the same manner as the `joint` command.

The root body of a given corpus can be free to move about in space, under the influence of the applied forces, or it can be constrained to be immobile, with the remainder of the articulated body behaving appropriately. To allow free motion of the root body of the current corpus, use:

```
rootmotion free
```

To constrain the base to be immobile, use:

```
rootmotion fixed
```

Similarly, the motion of any joint can be left free, to respond to the forces (forward dynamics), or constrained to follow a specified joint velocity and acceleration (inverse dynamics). Free motion is specified with:

```
jointmotion body-name free
```

Constrained joint motion is specified with:

```
jointmotion body-name kinematic
```

In such a situation, the joint moves according to its kinematic joint parameters, so to generate a joint acceleration of 1.0 radians/sec^2, starting from rest use the following commands:

```
joint body-name dq 0
joint body-name ddq 1.0
```

When a joint's motion is kinematically controlled, the joint force required to maintain the motion constraint is computed during the dynamics computations. The computed joint force (Q) can be accessed with the `joint` or `jointstatus` commands.

B.4 Language Features

Corpus has rudimentary commands to control execution flow, create loops, etc. *Corpus* can be successfully used without knowledge of these language features, however complex environments are more easily managed when using them. The scripts used to generate the humanoid figure studied in this thesis, as presented in **Appendix D Body Scripts**, make use of these language commands, as do some of the simulations presented in **Appendix C Dynamics Verification**.

The text output response from any *corpus* command can be redirected from the screen output to a file, using the special characters ">" and ">>". For example, the command: `eye > /tmp/eyepoint` will write the output "`eye 0 0 0`" in the file: `/tmp/eyepoint`. The characters ">>" will append the output to the specified file, rather than overwrite.

Integer variables may be used with *corpus* commands. This simple mechanism is typically used to store and access quantities such as the current "frame number" or status flags. Integer variables are accessed using the special character '%' followed by the variable name, which is a single ASC-II character. If a line begins with '%', a numerical operation or assignment is performed on the variable. At any other location on a command line, the '%' character is replaced with the variable's value, and the command-line is executed. For example:

```
%A=10
%F=%A
%F+1
! lines which begin with the '!' character are printed comments.
            ! lines which begin with the '!' character are printed
comments.
! The current value of 'F' is %F
            ! The current value of 'F' is 11
# load a dynamic state, for the current Frame
loads StateData/StateFile%F
```

Appendix A Corpus Help lists the operations available for the integers.

The integers can be used to test basic conditionals using the `if` command. The tail end of the `if` command line is executed if the testing condition is true. For example:

```
if %F=10 !this comment will print if F = 10

# should we simulate?
if %s=1 go

# should we render?
if %r=1 render

if %a<%g %a=%g
```

Using a command-list, described below, a sequence of commands can be triggered to execute after the test.

A "command list" is a sequence of *corpus* commands which are executed when the command list's name is entered (like a simple macro). A command list is defined using the `commandlist` command or with the abbreviation `cl`. For example, the following script creates a command list named "sim" which would simplify generating a simulated animation:

```
# a command list is defined using "cl cl-list-name", followed by a list
#    of command lines. The list is terminated with a line starting with "."
cl sim
```

```
go
render
.
```

We could then simulate a step and render the results by entering `sim`. Command lists can be instructed to run multiple times. We could watch 30 sequential simulation steps by entering `sim 30`. The following is a more sophisticated simulation command list:

```
cl sim
go
! done simulating frame %F
render
# save the current dynamic state
dumps StateData/StateFile%F
%F+
.
```

Note that the integer variables can be used within the command lists. Command lists function as loops when run multiple times, as described above. (Loops can also be made using the `commandloop` or `cloop` commands, without naming and saving the commands as a command list.)

Command lists can also accept an input string variable. Each occurrence of the character sequence '**' is replaced with the input string, which is specified when invoking the command list. For example:

```
cl shade-em
facet **
shadeparam ** .8 .1 30 0
color ** .1 .2 .8
.

shade-em object1
shade-em object2
```

A list of arbitrary text lines can be stored using the `savelist` command. A savelist is named and entered in the same manner as a command list. For example:

```
savelist body-list
head
torso
left-arm
right-arm
.
```

Savelists can be used to execute *corpus* command, based on their contents. The `foreach` command loops over a savelist, and substitutes each element of the savelist into a command string, which is executed. For example:

```
# color all body parts red
foreach body-list color ** 1 0 0
```

The `playlistn` command uses a specific element from a savelist to build a corpus command. For example:

```
# unpost the 2rd element (left-arm) of body-list (starting from 0).
playlistn body-list 2 unpost
```

A command can be executed for each body in an articulated figure using the `commandtree` command. For example:

```
# set the joint velocity of every body in the current corpus to 0
commandtree joint ** dq 0
```

Similarly, a command can be executed for every “part” in an articulated figure (which includes every “body”) using the `commandtreeparts` command, as in:

```
# color each part of the current corpus green. (requires the graphical
#    object and part to share the same name)
commandtreeparts color ** 0 1 0
```

Rather than improve the computer language facilities built into *corpus*, effort would be better directed into replacing the command line parsing interface sub-system with a well-developed system such as the *tcl* library. [Ousterhout] Such a task is considerable undertaking, however, due to the large number of *corpus* parsing commands which would have to modified to be compatible with a new system.

Appendix C Dynamics Verification

A suite of test simulations were developed to verify that the dynamics computations performed by *corpus* were properly implemented, and result in numerically "accurate" results. Because the computations are numerical, the results will usually not be exact. We can define an "accurate" result as one in which the errors are insignificant when compared to the results, and one in which the error does not lead to gross changes in the dynamics of the system. With each test case, we will discover how accurate the dynamics computations are with respect to the given conditions.

To solve the forward dynamics problem, there are several key factors which must be correctly computed: spatial and joint accelerations are solved based on the applied forces, the accelerations are integrated to velocities and positions, and the forces functions are computed based on their underlying model.

For the inverse dynamics problem, based on specified velocities and accelerations, the forces are computed, accelerations are integrated to velocities and positions, and acceleration functions are computed.

The most difficult equations to evaluate are the force functions (for forward dynamics) and acceleration functions (for inverse dynamics). We can verify that the functions create the appropriate accelerations or forces. However, it is more difficult to evaluate their correspondence to real world phenomena.

C.1 Constant Linear Force: Gravitational Free Fall Test

The most basic test verifies that bodies accelerate correctly under the influence of a linear force. To test this, simulated gravitational forces are applied to a single dynamic body (non-articulated). The gravitational force applies a constant linear force of -9.81, in the z direction, to the body's center of mass. After one simulated second of free fall, the body's position and velocity are queried, and the results are compared to the analytic result. For one second of free fall, an analytic solution gives us:

$$v = \int_0^1 -9.81 \text{ m/sec}^2 dt = -9.81\ t = -9.81 \text{ m/sec} \qquad \text{Eq. 92}$$

$$p = \iint_0^1 -9.81 \text{ m/sec}^2 dt = \frac{1}{2}(-9.81)\, t^2 = -4.905 \text{ m} \qquad \text{Eq. 93}$$

A *corpus* script to simulate a falling body is shown in Script 10. Because the acceleration is unchanging, the simulation is trivial to solve accurately. In fact, euler integration is implemented using equations similar to those given in equations Eq. 92 and Eq. 93, and the exact solution can be computed using a single euler step. The more advanced integration methods — the fixed-step, fourth-order runge-kutta (RK4), and variable step-size runge-kutta (RK4/5) — also give exact solutions in one step.

Script 10: A *corpus* script to simulate a body in free fall.

```
# create a default dynamic
# environment
addworld w

# create a new dynamic object
# (a corpus)
addcorpus c

# load a graphical object
get b from ../data/unit_cubeb.asc

# add the graphical object (a body)
# to the current corpus
addbody b b 0 0 1 rotary 1000

# set that body to be the root
# object in the corpus, and
# initialize
setroot b
corpusinit

# set some world properties
grav on
ground off
```

```
# pick integration type
#integration euler
#integration rkfixed
integration rkf

# set simulation timestep
dt .0333333333333333333333

# set the rkf error tolerance
eps .0000001

# print more dynamics status
# during simulation
verbose extra

# simulate 30 steps (1 sec)
go 30

# print out information on the
# body. It should be at 0 0 -4.905,
# v: 0 0 -9.81
whereis b
setv b
```

In a typical simulation, using a smaller simulation step-size will generally result in a more accurate result, because the integrator samples the changing state variables more frequently. In this simple test case, however, the result becomes slightly less accurate with a smaller time step, presumably because the numerical limits of the computer allow more errors to accumulate when more mathematical operations are executed (See Table 8).

Table 8: Free fall test results.

method	*simulation steps/ sec*	*position (m)*	*velocity (m/sec)*
Analytic Solution	1	-4.905	-9.81
Euler	1	-4.905	-9.81
Euler	30	-4.905	-9.81
Euler	300	-4.90498	-9.80993
Euler	3000	-4.90496	-9.80989
RK4	1	-4.905	-9.81
RK4	300	-4.90495	-9.80987
RK4	3000	-4.90496	-9.80989
RK4/5	1	-4.905	-9.81
RK4/5	300	-4.90499	-9.80998
RK4/5	3000	-4.90496	-9.80988

No rotational accelerations were inadvertently introduced when simulating a gravitational acceleration. In addition, no rotations, or joint accelerations, were introduced when simulating a falling articulated body. Bodies which have their center of mass offset from the coordinate frame origin also simulate correctly, even though technically the gravitational force creates a torque at the coordinate frame origin, where the computations occur. No rotational acceleration is created because the spatial inertia tensor also encodes the offset of the COM.

C.2 Conservation of Momentum: Constant Velocity Tests

Another set of basic tests validates that, in the absence of applied forces, a body moves with a constant velocity. The first test simulates a body moving with a constant linear velocity. The second tests a body with a constant angular velocity. The third tests the

Script 11: A *corpus* script to simulate a body moving with a constant velocity.

```
addworld w
addcorpus c
get b from ../data/unit_cubeb.asc
addbody b b 0 0 1 rotary 1000
setroot b
corpusinit
grav off
ground off

integration rkf
dt .033333333333333333333
eps .0000001
```

```
# set the linear velocity of the
# body. Set angular v to 0
setv b 1.1 2.2 4.432 0 0 0

# set angular velocity:
#setv b 0 0 0 1.1 2.2 4.432

# set both linear and angular v:
#setv b 1.1 2.2 4.432 1.1 2.2 4.432

# simulate 30 steps (1 second)
go 30
whereis b
setv b
```

motion of a body with a constant linear and angular velocity. The computations involved in the third test are more complex than they might seem upon first examination. Because local coordinate frames are used in *corpus*, a motion involving rotations and translations requires that the body move in a locally rotating space. Although the motion is straight-line in world space, an examination of the linear velocities in local space reveals a curving path. This curve must be accurately integrated, or errors in the motion will result.

The *corpus* script to simulate a body moving with a constant linear velocity, in the absence of external forces, is given in Script 11. This test is even more basic than the gravitational test above, and it is not surprising that exact results are computed by the three different integrators, at a variety of simulation step sizes. Similarly, for the second test, which examines constant angular velocities, exact results can be obtained from any of the integrators.

The third test verifies that bodies with both a constant linear and angular velocity continue to move with that same velocity along a straight line. This test reveals some differences in the accuracy of the integrators; the euler integrator introduced errors which built up over time, while the other integrators created an accurate, stable result. See Figure 64.

C.3 Oscillatory Motion: Linear Spring Test

The following experiment/test simulates a simple translating joint with a linear spring. An oscillatory movement is created when the joint position is offset from the spring rest

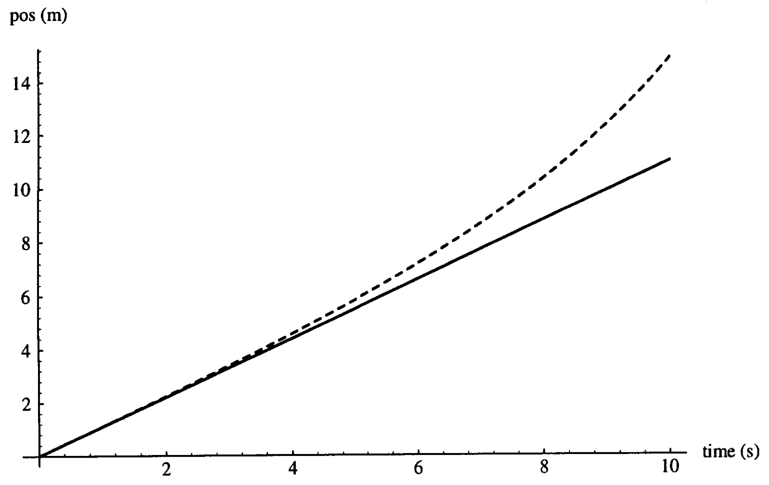


Figure 64: Results from the constant linear and angular velocity test.

The plots to the left show the x position for the dynamic body over time, using the script given in Script 10. After 10 seconds of simulation time, the Euler plot (dashed) begins to deviate from a straight line, gaining energy. The lower, solid-line plots shows the RK4, RK4/5, and analytic solutions (which co-exists on the same straight line).

angle. The simulation progresses forward in time, simulating accelerations from the applied spring forces. A feedback system is created, as the acceleration integrates to velocity and position, and the spring changes its force, based on the changing joint position. Without any damping in the system, the oscillation continues indefinitely.

The analytic solution to this spring/body system is:

$$F_k = -k\,\Delta x\ , \qquad \text{Eq. 94}$$

where k is the sprint constant, F_k is the spring force, and x is the joint position. The natural frequency, ω_0, is given as:

$$\omega_0 = \sqrt{\frac{k}{m}}\ , \qquad \text{Eq. 95}$$

where m is the body mass. The amplitude of the oscillation is given as:

$$A = \sqrt{{x_0}^2 + {v_0}^2/{\omega_0}^2}\ , \qquad \text{Eq. 96}$$

where v_0 is the starting joint velocity, and x_0 is the starting joint position. The phase of the oscillation is defined using:

$$\tan\phi_0 = \frac{\omega_0\,x_0}{v_0}\ . \qquad \text{Eq. 97}$$

The value of the joint position at a time, t, is then:

$$x(t) = A\,\sin(\omega_0 t + \phi_0)\ . \qquad \text{Eq. 98}$$

Script 12: A *corpus* script to simulate a linear spring system.

```
addworld w
addcorpus c
get b from ../data/unit_cubeb.asc
get b2 from ../data/unit_cubeb.asc
addbody b b 0 0 1 rotary 1000
addbody b2 b2 1 0 0 sliding 1000
linkbodies b b2
setroot b
corpusinit
grav off
ground off
integration rkf
dt .03333333
```

```
verbose extra
joint b2 q 4
joint b2 Q_type 2
# equal to mass of b2
joint b2 k 984.857
joint b2 k_q 3
rootmotion fixed

# for 3 integrator steps/frame
eps .000000000001
# for 1 step/frame
#eps .1
```

The *corpus* script to simulate such a system is given in Script 12. The RK4/5 integrator provided excellent results, matching the analytic solution with great accuracy. Even after 100 seconds of simulation time, the simulated solution remained in step with the analytic, and there was no net energy gain or dissipation. A plot of the simulation results, vs. the analytic solution is shown in Figure 65.

C.4 Damped Oscillation: Linear Spring and Damper Test

The following test compares the analytic solution to the simulated *corpus* solution for a body moving under the influence of a spring and damper combination. The resulting motion, when the body is released from a position offset from the spring rest position is an oscillatory movement, which reduces its amplitude over time.

The differential equation describing the body's motion is given as:

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = 0 \ , \qquad \text{Eq. 99}$$

where b is the damping constant (other terms carry over from previous examples).

The position of the body, as a function of time then is given as:

$$x(t) = A\ e^{-\beta t}\sin(\omega_0 t + \phi_0) \qquad \text{Eq. 100}$$

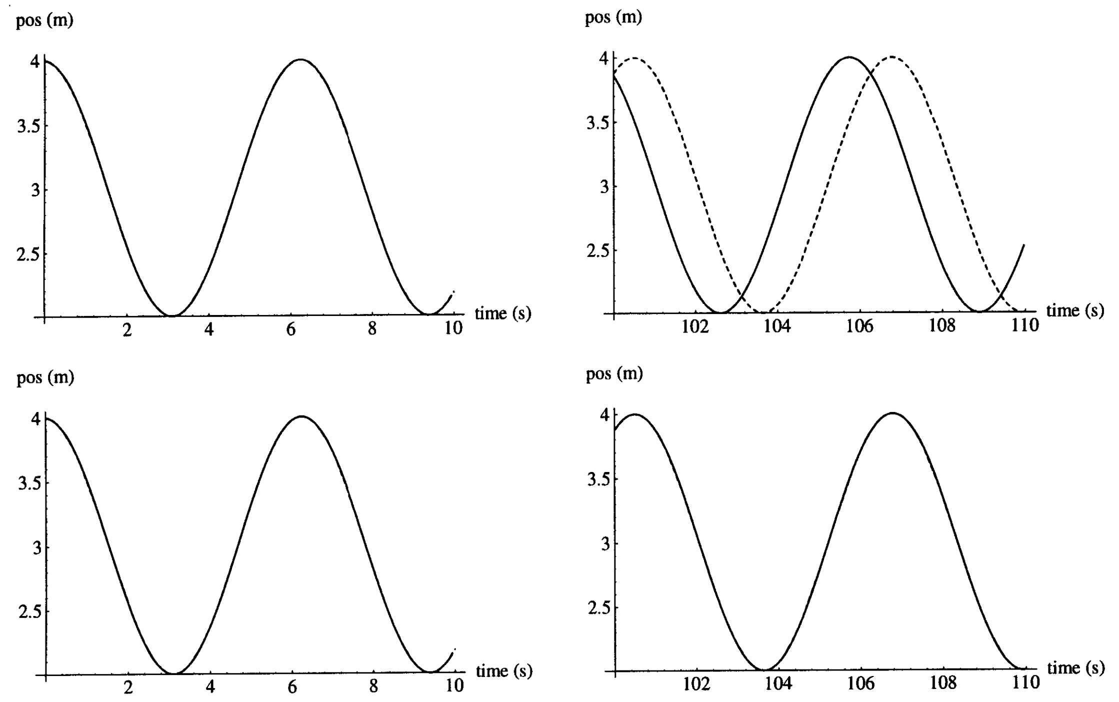


Figure 65: Linear spring oscillations.

The plots show the motion of the body over time, as it is driven by the spring force.

Two different simulations are shown, at two different times ranges. The upper row shows a simulation which took one integrator step per frame (using the RK 4/5 integrator). The bottom row shows a simulation which took three integrator steps per frame for increased accuracy. On the left, the plots show the results from time 0–10 seconds. The simulation results match the analytic very closely; on the plots, the two curves overlap and cannot be distinguished. The right plots show the results for simulation time 100-110 seconds. Here, the top plot shows that the simulation has lost approximately 30° of phase with the analytic solution, which is shown as the dashed line. Note, however, that the simulation has remained stable, with a constant net amplitude. The lower plot shows that the simulation which took three times more integrator time steps remains exactly in sync with the analytic solution.

Script 13: A *corpus* script to simulate a linear spring and damper system.

```
addworld w
addcorpus c
get b from ../data/unit_cubeb.asc
get b2 from ../data/unit_cubeb.asc
addbody b b 0 0 1 rotary 1000
addbody b2 b2 1 0 0 sliding 1000
linkbodies b b2
setroot b
corpusinit
grav off
ground off
```

```
integration rkf
dt .03333333
eps .1
joint b2 q 4
joint b2 Q_type 3
# equal to mass of b2
joint b2 k 984.857
joint b2 k_q 3
joint b2 b 100
rootmotion fixed
```

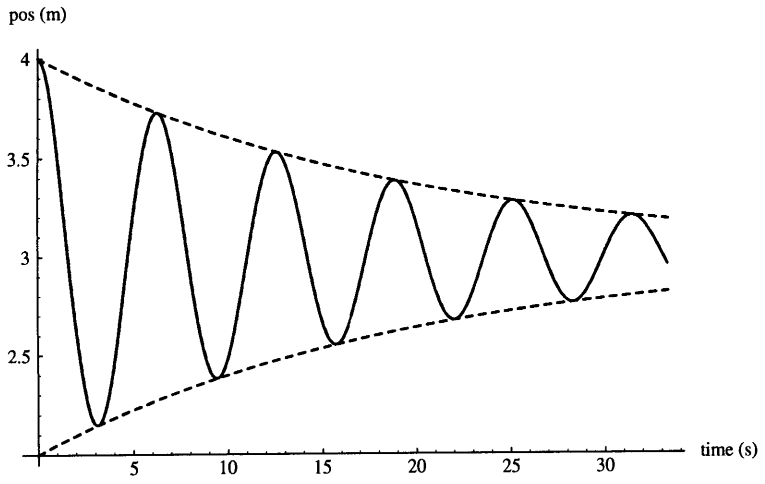


Figure 66: Damped oscillation simulation results.

The solid line on the plot shows the motion of the body over time, in response to the applied spring and damper forces. The outer dashed lines show the analytic solution for the envelope of the damped oscillation. The analytic solution and simulation results for the body motion are extremely similar; another dashed line is plotted for the analytic solution, but it is very difficult to discern from the simulation plot (solid line).

The *corpus* script to simulate this spring and damper system is given in Script 13. A plot of the analytic and simulated solutions is given in Figure 66. The results agree with great accuracy.

C.5 Exponential Spring Test

The next test examines the motion of a body which is influenced by an exponential spring. The equation of motion for the body, derived from the exponential spring equations described in **Dynamic Simulator** (5), is given as:

$$m\frac{d^2x}{dt^2} + \mathrm{sign}(x-x_s)\ \alpha\, e^{\beta\,|x-x_s|} = 0 \qquad \text{Eq. 101}$$

Script 14: A *corpus* script to simulate an exponential spring system.

```
addworld w
addcorpus c
get b from ../data/unit_cubeb.asc
get b2 from ../data/unit_cubeb.asc

addbody b b 0 0 1 rotary 1000
addbody b2 b2 1 0 0 sliding 1000
linkbodies b b2
setroot b
corpusinit
grav off
ground off

integration rkf
dt .03333333
```

```
eps .000000000001
verbose extra

joint b2 q 4
joint b2 Q_type 16

# b2 mass: 984.857

joint b2 ea 1
joint b2 eB 10
joint b2 e_q 3

rootmotion fixed
```

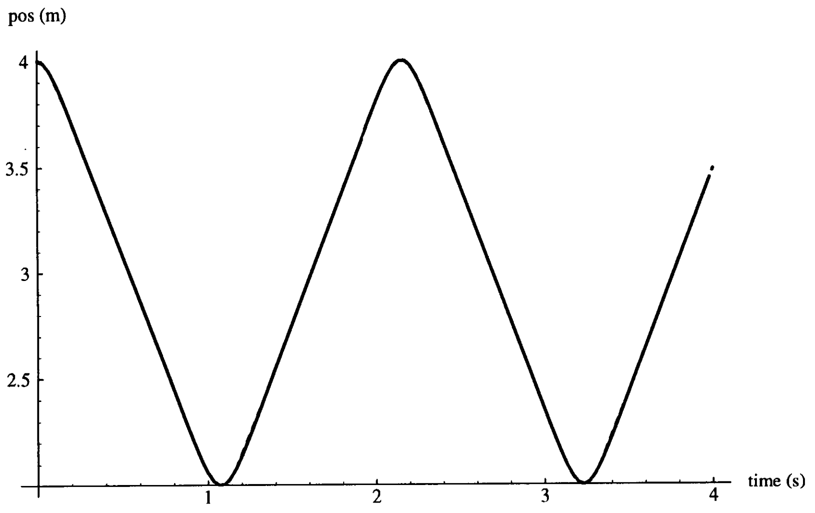


Figure 67: Exponential spring oscillation.

The plot shows the position of the body over time, in response to the applied forces from an exponential spring.

Again, the plots of the simulation results and analytic solution overlap.

The script to simulate a body moving under the influence of an exponential spring in *corpus* is given in Script 14. The motion of the body was first simulated in *corpus*, and then computed in the math analysis program Mathematica™, [Wolfram] which was used to numerically integrate Eq. 101. The results, once again, were in very close agreement. A plot of the simulation results is shown in Figure 67.

C.6 Double Pendulum: Two Link, Two Joint Arm

The following test creates a simulation of a two link arm. The equations of motion for such a system are developed in Brady, *et al.* [Brady] Part of that work is reviewed here. The introduction of joints greatly complicates the equations of motion, even in such a simple

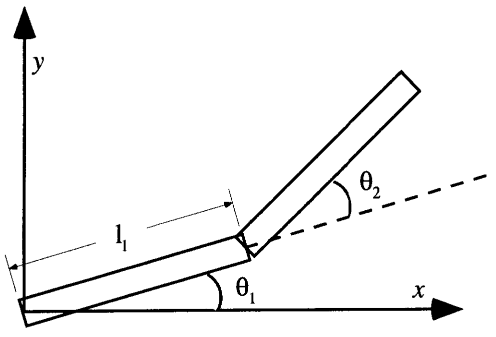


Figure 68: A two-link, two-DOF arm.

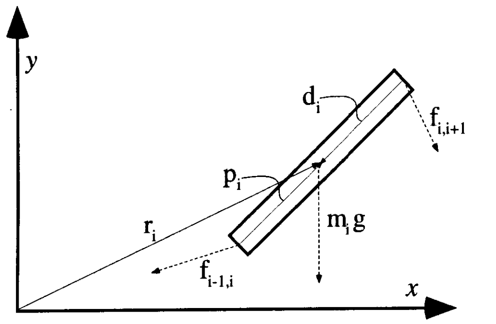


Figure 69: Free body diagram for link *i*.

system. Construction of the simulation in *corpus*, however, remains a simple matter. The equations of motion for the two link arm were entered into Mathematica™, to allow comparison to the results of the *corpus* computations.

For reference, a diagram of the two link arm system is shown in Figure 68. A free body diagram for one of the links in the arm is shown in Figure 69. A *corpus* script used to construct such a system for simulation is given in Script 15.

The arm kinematics are defined in terms of the link lengths and the joint angles. The centers of mass of the two links are given as:

$$r_1 = \frac{l_1}{2}\begin{bmatrix}\cos\theta_1 \\ \sin\theta_1\end{bmatrix}, \qquad \text{Eq. 102}$$

$$r_2 = \begin{bmatrix} l_1 \cos\theta_1 + 0.5\ l_2\cos(\theta_1+\theta_2) \\ l_1 \sin\theta_1 + 0.5\ l_2\sin(\theta_1+\theta_2)\end{bmatrix}. \qquad \text{Eq. 103}$$

The velocity of the body i's center of mass, $\dot{r}_i$, and acceleration, $\ddot{r}_i$, are found by differentiating the position vector, r_i, yielding fairly complex terms (not given here).

Script 15: A *corpus* script to simulate a double pendulum.

```
postmult

addworld w
addcorpus c

get base from ../data/unit_cubeb
unpost base

# cyl: aligned w/ Y axis-
# {-10 10, -5 5, -10 10}
get b1 from ../data/cylinder.asc
get b2 from ../data/cylinder.asc

# scale it to be 1 long in Y, w/
# a radius of .05 m
scale b1 .005 .1 .005
scale b2 .005 .1 .005

# move the joint to the origin
move b1 0 .5 0
move b2 0 .5 0

addbody base base 0 0 1 rotary 1000
addbody b1 b1 1 0 0 rotary 1000
addbody b2 b2 1 0 0 rotary 1000

# define transform between parent
# and child, and define link
linkbodies base b1

move b2 0 1 0
linkbodies b1 b2

setroot base
corpusinit

rootmotion fixed

grav on
ground off

integration rkf
dt .03333333
eps .0000001
verbose extra
```

The vector p_i defines the location of the center of mass of body i, taken from the body's proximal joint, and d_i defines the center of mass taken from the body's distal joint:

$$p_1 = \frac{l_1}{2}\begin{bmatrix}\cos\theta_1 \\ \sin\theta_1\end{bmatrix}, \qquad \text{Eq. 104}$$

$$p_2 = \frac{l_2}{2}\begin{bmatrix}\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2)\end{bmatrix}, \qquad \text{Eq. 105}$$

$$d_i = -p_i\,. \qquad \text{Eq. 106}$$

The *dynamics* equations relate the applied force (f_i) and torque (n_i) to the linear and rotational accelerations:

$$f_i = m_i \ddot{r}_i \ , \qquad \text{Eq. 107}$$

$$n_i = I_i \dot{\omega}_i + \omega_i \times I_i \omega_i \ . \qquad \text{Eq. 108}$$

In our two dimensional case, we can simplify:

$$\omega_i \times I_i \omega_i = 0 \ , \qquad \text{Eq. 109}$$

and, in terms of the joint angles in our 2-link arm case:

$$n_1 = I_1 \ddot{\theta}_1 \ , \qquad \text{Eq. 110}$$

$$n_2 = I_2 (\ddot{\theta}_1 + \ddot{\theta}_2) \ . \qquad \text{Eq. 111}$$

The net forces and torques are computed by the *statics* equations, given as:

$$f_i = f_{i-1,i} - f_{i,i+1} + m_i g \ , \qquad \text{Eq. 112}$$

$$n_i = n_{i-1,i} - n_{i,i+1} - p_i \times f_{i-1,i} + d_i \times f_{i,i+1} \ . \qquad \text{Eq. 113}$$

In our example, there are no external forces applied except for gravity, thus no forces are applied at the arm's tip, so that $f_{2,3} = 0$, and $n_{2,3} = 0$. The gravitational vector is given as:

$$g = \begin{bmatrix} 0 \\ -9.81 \end{bmatrix} \ . \qquad \text{Eq. 114}$$

We can now solve the inverse dynamics equations, by substituting the dynamics equations (Eq. 107 – Eq. 111) into the statics equations (Eq. 112 – Eq. 114), and extracting the applied joint torques, given joint positions, velocities, and accelerations:

$$n_{0,1} = I_1 \ddot{\theta}_1 + n_{1,2} + p_1 \times (m_1 \ddot{r}_1 + m_2 \ddot{r}_2 - m_1 g - m_2 g) \qquad \text{Eq. 115}$$

$$-d_i \times (m_2 \ddot{r}_2 - m_2 g) \ ,$$

$$n_{1,2} = I_2 (\ddot{\theta}_1 + \ddot{\theta}_2) + p_2 \times (m_2 \ddot{r}_2 - m_2 g) \ . \qquad \text{Eq. 116}$$

We next insert our kinematics equations (Eq. 102–Eq. 106), and simplify and collect terms. The inverse dynamics solution for the joint torques can then be given as:

$$n_{1,2} = \ddot{\theta}_1\left(I_2 + \frac{m_2 l_1 l_2}{2}\cos\theta_2 + \frac{m_2 l_2^2}{4}\right) + \ddot{\theta}_2\left(I_2 + \frac{m_2 l_2^2}{4}\right) \qquad \text{Eq. 117}$$

$$+ \frac{m_2 l_1 l_2}{2}\dot{\theta}_1^2 \sin\theta_2 + 4.905\ m_2 l_2 \cos\left(\theta_1 + \theta_2\right)\ ,$$

$$n_{0,1} = \ddot{\theta}_1\left(I_1 + I_2 + m_2 l_1 l_2 \cos\theta_2 + \frac{m_1 l_1^2 + m_2 l_2^2}{4} + m_2 l_1^2\right) \qquad \text{Eq. 118}$$

$$+ \ddot{\theta}_2\left(I_2 + \frac{m_2 l_2^2}{4} + \frac{m_2 l_1 l_2}{2}\cos\theta_2\right) - \frac{m_2 l_1 l_2}{2}\dot{\theta}_2^2 \sin\theta_2$$

$$-m_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \sin\theta_2 + 4.905 m_2 l_2 \cos\left(\theta_1 + \theta_2\right) + 9.81\ l_1\left(\frac{m_1}{2} + m_2\right)\cos\theta_1\ .$$

To solve for the forward dynamics, we apply a torque function, and solve for the joint accelerations. For our test, we allow the arm to passively fall, as a double pendulum, under the force of gravity, so that:

$$n_{0,1} = n_{1,2} = 0 \qquad \text{Eq. 119}$$

The rather lengthy solution for the forward dynamics to compute joint accelerations is given below in the Mathematica™ script, Script 16. These equations are too complex for Mathematica to integrate, and since we have already verified the accuracy of the integrator in *corpus*, we compare instantaneous solutions for the joint accelerations. Examining these instantaneous solutions (given joint angles and velocities) shows that the Mathematica™ and *corpus* equations of motion yield exactly the same results.

Script 16: A Mathematica™ script to define the equations of motion for a two link pendulum.

```
(* 2- link arm- statics (net force) *)

f01[t_]:=f1[t] + f12[t] - m1 g
f12[t_]:=f2[t] - m2 g
```

```
Cross[a_, b_]:= a[[1]] b[[2]] - a[[2]] b[[1]]

n01[t_]:= n1[t] + n12[t] + Cross[(pl1[t] + rl1[t]), f01[t] ] - \
            Cross[rl1[t], f12[t]]
n12[t_]:= n2[t] + Cross[(pl2[t] + rl2[t]), f12[t]]

(* 2 link dynamics *)

f1[t_]:= m1 r1''[t]
f2[t_]:= m2 r2''[t]

g:= {0, -9.81}

n1[t_]:= i1 th1''[t]
n2[t_]:= i2 (th1''[t] + th2''[t])

(* Kinematics *)
(* pl is the 'local' vector- from proximal to distal *)

pl1[t_]:= { l1 Cos[th1[t]], l1 Sin[th1[t]] }
pl2[t_]:= { l2 Cos[th1[t] + th2[t]], l2 Sin[th1[t] + th2[t]]}

(* rl is 'local' vector - from distal to COM *)

rl1[t_]:= -0.5 pl1[t]
rl2[t_]:= -0.5 pl2[t]

(* p - vector from base to distal end *)

p1[t_]:= pl1[t]
p2[t_]:= pl1[t] + pl2[t]

(* r- vector from base to COM - r1 also equals p1 + rl *)

r1[t_]:= 0.5 p1[t]
r2[t_]:= p2[t] + rl2[t]

(* Define derivatives *)

r1'[t_]:= D[r1[t], t]
r1''[t_]:= D[r1'[t], t]
r2'[t_]:= D[r2[t], t]
r2''[t_]:= D[r2'[t], t]

(* Simplify and Collect terms for the joint torques: n01, n12 *)

nn01[t_]:= Simplify[n01[t]]
nn12[t_]:= Simplify[n12[t]]

nnN12[t_]:=Collect[nn12[t], th1''[t]]
```

```
nnn12[t_]:=Collect[nnN12[t], th2''[t]]

nnN01[t_]:=Collect[nn01[t], th1''[t]]
nnn01[t_]:=Collect[nnN01[t], th2''[t]]

(* Make a 2D array,solving for joint accelerations when torques are zero *)

th'':= Solve[{nnn01[t]==0, nnn12[t]==0},
            {th1''[t], th2''[t]}]

(* aa[t] is the joint acceleration of the 1st joint *)
aa[t_]:= th1''[t] /. th''[[1,1]]
(* bb[t] is the joint acceleration of the 2nd joint *)
bb[t_]:= th2''[t] /. th''[[1,2]]

nnn01[t]

4.905 l1 m1 Cos[th1[t]] + 9.81 l1 m2 Cos[th1[t]] +

  4.905 l2 m2 Cos[th1[t] + th2[t]] -

  1. l1 l2 m2 Sin[th2[t]] th1'[t] th2'[t] -

                                            2
  0.5 l1 l2 m2 Sin[th2[t]] th2'[t]  +

                      2           2              2
  (i1 + i2 + 0.25 l1  m1 + 1. l1  m2 + 0.25 l2  m2 +

     1. l1 l2 m2 Cos[th2[t]]) th1''[t] +

                 2
  (i2 + 0.25 l2  m2 + 0.5 l1 l2 m2 Cos[th2[t]]) th2''[t]


nnn12[t]

4.905 l2 m2 Cos[th1[t] + th2[t]] +

                                            2
  0.5 l1 l2 m2 Sin[th2[t]] th1'[t]  +

                 2
  (i2 + 0.25 l2  m2 + 0.5 l1 l2 m2 Cos[th2[t]]) th1''[t] +

                 2
  (i2 + 0.25 l2  m2) th2''[t]


Simplify[aa[t]]
```

```
                       2
-(((1. i2 + 0.25 l2  m2 + 0.5 l1 l2 m2 Cos[th2[t]])

       (4.905 l2 m2 Cos[th1[t] + th2[t]] +

                                                 2
         0.5 l1 l2 m2 Sin[th2[t]] th1'[t] )) /

                                   2              2
     (-1. i1 i2 - 0.25 i2 l1  m1 - 1. i2 l1  m2 -

                  2                 2   2
       0.25 i1 l2  m2 - 0.0625 l1  l2  m1 m2 -

               2   2   2           2   2   2            2
       0.25 l1  l2  m2  + 0.25 l1  l2  m2  Cos[th2[t]] )) -

                        2
   ((-1. i2 - 0.25 l2  m2)

     (4.905 l1 m1 Cos[th1[t]] + 9.81 l1 m2 Cos[th1[t]] +

       4.905 l2 m2 Cos[th1[t] + th2[t]] -

       1. l1 l2 m2 Sin[th2[t]] th1'[t] th2'[t] -

                                               2
       0.5 l1 l2 m2 Sin[th2[t]] th2'[t] )) /

                                   2              2
    (-1. i1 i2 - 0.25 i2 l1  m1 - 1. i2 l1  m2 -

                  2                 2   2
      0.25 i1 l2  m2 - 0.0625 l1  l2  m1 m2 -

               2   2   2           2   2   2            2
      0.25 l1  l2  m2  + 0.25 l1  l2  m2  Cos[th2[t]] )


Simplify[bb[t]]

                                    2           2             2
-(((-1. i1 - 1. i2 - 0.25 l1  m1 - 1. l1  m2 - 0.25 l2  m2 -

         1. l1 l2 m2 Cos[th2[t]])

       (4.905 l2 m2 Cos[th1[t] + th2[t]] +

                                                 2
         0.5 l1 l2 m2 Sin[th2[t]] th1'[t] )) /
```

```
                          2              2
   (-1. i1 i2 - 0.25 i2 l1  m1 - 1. i2 l1  m2 - 
 
               2               2   2
     0.25 i1 l2  m2 - 0.0625 l1  l2  m1 m2 - 
 
            2   2   2          2   2   2            2
     0.25 l1  l2  m2  + 0.25 l1  l2  m2  Cos[th2[t]]  )) - 
 
                 2
((1. i2 + 0.25 l2  m2 + 0.5 l1 l2 m2 Cos[th2[t]]) 
 
   (4.905 l1 m1 Cos[th1[t]] + 9.81 l1 m2 Cos[th1[t]] + 
 
     4.905 l2 m2 Cos[th1[t] + th2[t]] - 
 
     1. l1 l2 m2 Sin[th2[t]] th1'[t] th2'[t] - 
 
                                     2
     0.5 l1 l2 m2 Sin[th2[t]] th2'[t]  )) / 
 
                        2              2
 (-1. i1 i2 - 0.25 i2 l1  m1 - 1. i2 l1  m2 - 
 
             2               2   2
   0.25 i1 l2  m2 - 0.0625 l1  l2  m1 m2 - 
 
          2   2   2          2   2   2            2
   0.25 l1  l2  m2  + 0.25 l1  l2  m2  Cos[th2[t]]  )
```

C.7 Self Consistency: Multiple Geometric Structures

Four different articulated bodies were created in a *corpus* simulation. The four structures were identical in their overall geometric structures, however, each articulated figure was specifically created in a different manner. Each 4-link articulated body had a different link specified as its root object. The root object is the only object whose spatial acceleration is directly calculated; otherwise joint accelerations are computed. Therefore, very different root motions must be calculated for the different structures.

The structures are given energy by the constant application of a torque at a single joint in each structure. The different structures undergo very similar simulated motion. They do begin to drift apart slightly in their solutions after simulating for some time. After approximately 5 seconds of simulation time, a very large amount of energy had been entered into the systems, through the constant application of the joint torques. Each powered joint had been accelerated to a velocity of over 18 radians/sec, within a structure 4 meters in length. The joint parameters between different structures were still within 99% agreement.

The following, Script 17, creates the multiple articulated body test.

Script 17: A *corpus* script which creates 4 different articulated bodies which have identical structures, but different links specified as the root link.

```
postmult
addworld w

# define the first of 4 corpora ("corpus"es)
addcorpus c1

# Define a "commandlist" (cl), like a simple procedure, to load and
# initialize bodies.
# The "%" symbols indicate an integer variable.
# This list is names 'load1'.
# Lists are terminated by "." as the first character of a line.
cl load1
get b.%c.%b from ../data unit_cubeb.asc
scale b.%c.%b 1 .1 .1
move b.%c.%b .5 0 0
addbody b.%c.%b b.%c.%b 0 0 1 rotary 1000
```

```
%b+
.

# set the corpus number to 1
%c=1
# set the body number to 1
%b=1
# call the load command list, 4 times, to load 4 bodies into the corpus, c1
load1 4

# define the transformations between parent and child links, and specify
# the hierarchy
move b.1.2 1 0 0
linkbodies b.1.1 b.1.2
move b.1.3 1 0 0
linkbodies b.1.2 b.1.3
move b.1.4 1 0 0
linkbodies b.1.3 b.1.4

setroot b.1.1
corpusinit

# define the second corpus, with a different root body, and different
# local connections, but the same overall geometry
# Subsequent operations are performed on the new, current corpus
# (see "setcorpus")
addcorpus c2
%c=2
%b=1
load1 4

move b.2.2 1 0 0
linkbodies b.2.1 b.2.2
move b.2.3 1 0 0
linkbodies b.2.2 b.2.3
rotate b.2.4 z 180
linkbodies b.2.1 b.2.4

setroot b.2.1
corpusinit

# move the entire articulated corpus by moving the root body and
# dynamically setting the new position with "setrootpos"
setrootpos

addcorpus c3
%c=3
%b=1
load1 4

move b.3.2 1 0 0
```

```
linkbodies b.3.1 b.3.2
rotate b.3.3 z 180
linkbodies b.3.1 b.3.3
#rotate b.3.4 z 180
move b.3.4 1 0 0
linkbodies b.3.3 b.3.4

setroot b.3.1
corpusinit

move b.3.1 2 0 -.6
setrootpos

addcorpus c4
%c=4
%b=1
load1 4

rotate b.4.2 z 180
linkbodies b.4.1 b.4.2
move b.4.3 1 0 0
linkbodies b.4.2 b.4.3
move b.4.4 1 0 0
linkbodies b.4.3 b.4.4

setroot b.4.1
corpusinit

move b.4.1 3 0 -.9
setrootpos


grav off
ground off

integration rkf
dt .03333333
eps .000000001
verbose extra

# set the joint force type to be a "bias force" (type 4)
# the bias force value is set by specifying the "Q".
# In this manner, a constant torque of 10 is set for the joints below.
# The remaining joints are unpowered, with zero applied force.
joint b.1.2 Q_type 4
joint b.1.2 Q 10

joint b.2.4 Q_type 4
joint b.2.4 Q -10

joint b.3.4 Q_type 4
joint b.3.4 Q -10

joint b.4.4 Q_type 4
joint b.4.4 Q -10

# color the root objects
color b.1.1 1 0 0
color b.2.1 1 0 0
color b.3.1 1 0 0
color b.4.1 1 0 0

# for convenience, define a command list to simulate one step, then render
cl s
go
render
.

# set some viewing parameters
backgroundcolor 0 0 .3
eye 2 -10 3
lookat 2 0 0
fov 30

# run the simulation, and display the animated results,
# for 5 simulation second
s 150
```

Appendix D Body Scripts

The following is a simplified *corpus* script, used to assemble the articulated biped used in this research, and to initialize its dynamic parameters. In the interest of saving space, only the "skeletal" kinematic-definition layer of the humanoid model is described here. The "skin" layer definitions are not included.

Script 18: *Corpus* script to build the "skeleton" layer of the human figure model, with the biomechanical joint parameters.

```
# DEFINE WORLD, AND SET GLOBAL PARAMETERS

addworld w

ground on
groundtype flat
grounde 0
groundk 0
groundb 100
groundfric 1.0
groundfricb 100
groundea 100
groundeB 100

eps .0001

grav on


# BODY DEFINITION
addcorpus biped

# SKEL OBJECTS

defaultshadeparam ** .8 0 0 0
defaultshademodel p
defaulticolor ** 162 162 162
sharememory off

get head1 from ../data/unit_cubeb
get head2 from ../data/unit_cubeb
unpost head1
unpost head2
get head3 from ../data/unit_cubeb
pushobj head3

# abdomen the root object- main dyn body
    is a small, internal body
#   the abd skin will have the tapered
    body, w/ inertia
get abdomen from ../data/unit_cubeb

# parts for abdomen skel
get abdomen1 from ../data/unit_cubeb
get abdomen2 from ../data/unit_cubeb
get abdomen3 from ../data/unit_cubeb

get neck from ../data/unit_cyl.asc

get pelvis1 from ../data/unit_cubeb
get pelvis2 from ../data/unit_cubeb
get pelvis3 from ../data/unit_cubeb

get l_humerus1 from ../data/unit_cubeb
get l_humerus2 from ../data/unit_cubeb
get l_humerus3 from ../data/unit_cubeb
get r_humerus1 from ../data/unit_cubeb
get r_humerus2 from ../data/unit_cubeb
get r_humerus3 from ../data/unit_cubeb

get l_forearm from ../data/unit_cubeb
get r_forearm from ../data/unit_cubeb

get l_hand1 from ../data/unit_cubeb
get l_hand2 from ../data/unit_cubeb
get r_hand1 from ../data/unit_cubeb
get r_hand2 from ../data/unit_cubeb

get l_hand3 from /u/mikey/ribconv/hand2/
    Hand.asc
get r_hand3 from /u/mikey/ribconv/hand2/
    Hand.asc
facet l_hand3
facet r_hand3
concave l_hand3 on
concave r_hand3 on

get l_thigh1 from ../data/unit_cubeb
get l_thigh2 from ../data/unit_cubeb
get l_thigh3 from ../data/unit_cubeb
get r_thigh1 from ../data/unit_cubeb
get r_thigh2 from ../data/unit_cubeb
get r_thigh3 from ../data/unit_cubeb
get l_shank from ../data/unit_cubeb
get r_shank from ../data/unit_cubeb

# talus = upper ankle,  hindfoot- lower
    ankle
get l_talus from ../data/unit_cubeb
get l_hindfoot from ../data/unit_cubeb
get r_talus from ../data/unit_cubeb
# hindfoot- like "os calcis"
get r_hindfoot from ../data/unit_cubeb

get l_nav from ../data/unit_cubeb
get l_cuboid from ../data/unit_cubeb
get r_nav from ../data/unit_cubeb
get r_cuboid from ../data/unit_cubeb

get l_metat1 from ../data/unit_cubeb
get l_metat2 from ../data/unit_cubeb
get l_metat3 from ../data/unit_cubeb
get l_metat4 from ../data/unit_cubeb
get l_metat5 from ../data/unit_cubeb
get r_metat1 from ../data/unit_cubeb
get r_metat2 from ../data/unit_cubeb
get r_metat3 from ../data/unit_cubeb
get r_metat4 from ../data/unit_cubeb
get r_metat5 from ../data/unit_cubeb

# phal#.1 = abd/add joint
# phal#.2 = flex/ext joint
get l_phal1.1 from ../data/unit_cubeb
get l_phal1.2 from ../data/unit_cubeb
get l_phal2.1 from ../data/unit_cubeb
get l_phal2.2 from ../data/unit_cubeb
get l_phal3.1 from ../data/unit_cubeb
get l_phal3.2 from ../data/unit_cubeb
get l_phal4.1 from ../data/unit_cubeb
get l_phal4.2 from ../data/unit_cubeb
get l_phal5.1 from ../data/unit_cubeb
get l_phal5.2 from ../data/unit_cubeb

get r_phal1.1 from ../data/unit_cubeb
get r_phal1.2 from ../data/unit_cubeb
get r_phal2.1 from ../data/unit_cubeb
get r_phal2.2 from ../data/unit_cubeb
get r_phal3.1 from ../data/unit_cubeb
get r_phal3.2 from ../data/unit_cubeb
get r_phal4.1 from ../data/unit_cubeb
get r_phal4.2 from ../data/unit_cubeb
get r_phal5.1 from ../data/unit_cubeb
get r_phal5.2 from ../data/unit_cubeb


# toe#.1 = mid phalange (big toe (#1)
    doesn't have the third phalange)
# toe#.2 = distal phalange
get l_toe1 from ../data/unit_cubeb
get l_toe2.1 from ../data/unit_cubeb
get l_toe3.1 from ../data/unit_cubeb
get l_toe4.1 from ../data/unit_cubeb
get l_toe5.1 from ../data/unit_cubeb
get r_toe1 from ../data/unit_cubeb
get r_toe2.1 from ../data/unit_cubeb
get r_toe3.1 from ../data/unit_cubeb
get r_toe4.1 from ../data/unit_cubeb
get r_toe5.1 from ../data/unit_cubeb
get l_toe2.2 from ../data/unit_cubeb
get l_toe3.2 from ../data/unit_cubeb
get l_toe4.2 from ../data/unit_cubeb
get l_toe5.2 from ../data/unit_cubeb
get r_toe2.2 from ../data/unit_cubeb
get r_toe3.2 from ../data/unit_cubeb
get r_toe4.2 from ../data/unit_cubeb
get r_toe5.2 from ../data/unit_cubeb

# abductor mini-links- do not display
unpost l_phal1.1
unpost r_phal1.1
unpost l_phal2.1
unpost r_phal2.1
unpost l_phal3.1
unpost r_phal3.1
unpost l_phal4.1
unpost r_phal4.1
unpost l_phal5.1
unpost r_phal5.1

defaultdefaultcolor
defaultdefaultshadeparam


# DEFINE OBJECTS IN THEIR LOCAL FRAMES

postmult

# main root object abdomen is a small,
    dummy body
#   its has 3 torso parts
scale abdomen .01 .01 .01

postmult
scale head1 .01 .01 .01
scale head2 .01 .01 .01

move head3 0 0 .5
# match Drillis and Contini for height:
    0.229
scale head3 0.21 0.167 0.229
# move to match anat illustation
move head3 .02 0 0

move neck 0 0 .5
scale neck 0.05 0.07 0.11
rotate neck y 15
# move up above of abdomen
move neck 0 0 .22
# matched to skel illustation
move neck -.025 0 0

scale pelvis1 .01 .01 .01
scale pelvis2 .01 .01 .01

cl pelvis3-make
```

```
move ** 0 0 -.5
scale ** .18 .29 .2
rotate ** y 18
move ** 0.026 0 -.1
move ** 0 0 0.016
move ** 0 0 0.06
move ** 0.02 0 -.022
move ** .02 0 0
.
pelvis3-make pelvis3

move l_humerus3 0 0 -.5
scale l_humerus1 .05 .05 .05
scale l_humerus2 .05 .05 .05
# match Drillis & Contini: 0.328
scale l_humerus3 .03 .03 .328
rotate l_humerus3 x 2

unpost l_humerus1
unpost l_humerus2

move r_humerus3 0 0 -.5
scale r_humerus1 .05 .05 .05
scale r_humerus2 .05 .05 .05
scale r_humerus3 .03 .03 .328
rotate r_humerus3 x -2
unpost r_humerus1
unpost r_humerus2

move l_forearm 0 0 -.5
scale l_forearm .03 .03 .258
move r_forearm 0 0 -.5
scale r_forearm .03 .03 .258

scale l_hand1 .01 .01 .01
scale l_hand2 .01 .01 .01
scale r_hand1 .01 .01 .01
scale r_hand2 .01 .01 .01
scale l_hand3 1 1 2
scale r_hand3 1 1 2
rotate l_hand3 x 90
rotate l_hand3 z 90
# taper hand distally, but not in y
    (width) direction
shear1d l_hand3 0 0 -1 0 1 0 1 -2
move l_hand3 0 0.01 0
rotate r_hand3 x 90
rotate r_hand3 z -90
shear1d r_hand3 0 0 -1 0 1 0 1 -2
move r_hand3 0 -0.01 0

scale l_thigh1 .1 .1 .1
scale l_thigh2 .1 .1 .1
move l_thigh3 0 0 -.5
scale l_thigh3 .03 .03 .432
unpost l_thigh1
unpost l_thigh2
move r_thigh3 0 0 -.5
scale r_thigh1 .1 .1 .1
scale r_thigh2 .1 .1 .1
scale r_thigh3 .03 .03 .432
unpost r_thigh1
unpost r_thigh2

move l_shank 0 0 -.5
# match Drillis & Contini : 0.434
scale l_shank .03 .03 .434
move r_shank 0 0 -.5
scale r_shank .03 .03 .434

# upper ankle - move down so axis is on
    top of talus
move l_talus 0 0 -.5
scale l_talus .045 .035 .022
move r_talus 0 0 -.5
scale r_talus .045 .035 .022

# offset hindfoot, to simplify scales &
    moves
move l_hindfoot 0 0 -.5
scale l_hindfoot .04 .04 .055
move r_hindfoot 0 0 -.5
scale r_hindfoot .04 .04 .055
rotate l_hindfoot y 60
rotate l_hindfoot z 5
move l_hindfoot 0 .007 0
rotate r_hindfoot y 60
rotate r_hindfoot z -5
move r_hindfoot 0 -.007 0

# move points to "low" part of hindfoot-
    the part that hits the ground
# points run along x axis- rotate to y
rotate l_hindfoot.points z 90
move l_hindfoot.points .5 0 -.5
move l_hindfoot.points 0 0 -.5
scale l_hindfoot.points .05 .05 .065
rotate l_hindfoot.points y 60
rotate l_hindfoot.points z 5
move l_hindfoot.points 0 .007 0
rotate r_hindfoot.points z 90
move r_hindfoot.points .5 0 -.5
move r_hindfoot.points 0 0 -.5
scale r_hindfoot.points .05 .05 .065
rotate r_hindfoot.points y 60
rotate r_hindfoot.points z -5
move r_hindfoot.points 0 -.007 0

# navicular & cuboid 'bones'
move l_nav .5 0 0
move r_nav .5 0 0
move l_cuboid .5 0 0
move r_cuboid .5 0 0
# these scales do not include the
    cuniforms !
scale l_nav 0.022 0.031 0.022
scale r_nav 0.022 0.031 0.022
scale l_cuboid 0.026 0.028 0.022
scale r_cuboid 0.026 0.028 0.022
rotate l_nav y 15
rotate r_nav y 15
rotate l_cuboid y 15
rotate r_cuboid y 15

move l_metat1 .5 0 0
move l_metat2 .5 0 0
move l_metat3 .5 0 0
move l_metat4 .5 0 0
move l_metat5 .5 0 0
move r_metat1 .5 0 0
move r_metat2 .5 0 0
move r_metat3 .5 0 0
move r_metat4 .5 0 0
move r_metat5 .5 0 0

scale l_metat1 .091 .02 .02
scale r_metat1 .091 .02 .02
scale l_metat2 .098 .0089 .0089
scale r_metat2 .098 .0089 .0089
scale l_metat3 .088 .0089 .0089
scale r_metat3 .088 .0089 .0089
scale l_metat4 .070 .0089 .0089
scale r_metat4 .070 .0089 .0089
scale l_metat5 .063 .0089 .0089
scale r_metat5 .063 .0089 .0089
rotate l_metat1 z -1
rotate r_metat1 z 1
rotate l_metat2 z 6
rotate r_metat2 z -6
rotate l_metat3 z 9
rotate r_metat3 z -9
rotate l_metat4 z 10
rotate r_metat4 z -10
rotate l_metat5 z 13
rotate r_metat5 z -13

rotate l_metat1 y 18
rotate r_metat1 y 18
rotate l_metat2 y 24
rotate r_metat2 y 24
rotate l_metat3 y 25
rotate r_metat3 y 25
rotate l_metat4 y 20
rotate r_metat4 y 20
rotate l_metat5 y 15
rotate r_metat5 y 15

move l_phal1.1 .5 0 0
move l_phal2.1 .5 0 0
move l_phal3.1 .5 0 0
move l_phal4.1 .5 0 0
move l_phal5.1 .5 0 0
move l_phal1.2 .5 0 0
move l_phal2.2 .5 0 0
move l_phal3.2 .5 0 0
move l_phal4.2 .5 0 0
move l_phal5.2 .5 0 0
move r_phal1.1 .5 0 0
move r_phal2.1 .5 0 0
move r_phal3.1 .5 0 0
move r_phal4.1 .5 0 0
move r_phal5.1 .5 0 0
move r_phal1.2 .5 0 0
move r_phal2.2 .5 0 0
move r_phal3.2 .5 0 0
move r_phal4.2 .5 0 0
move r_phal5.2 .5 0 0

scale l_phal1.1 0.016 0.016 0.016
scale r_phal1.1 0.016 0.016 0.016
rotate l_phal1.1 z 7
rotate r_phal1.1 z -7
scale l_phal1.2 0.048 0.016 0.016
scale r_phal1.2 0.048 0.016 0.016
rotate l_phal1.2 z 7
rotate r_phal1.2 z -7
scale l_phal2.1 0.0089 0.0089 0.0089
scale r_phal2.1 0.0089 0.0089 0.0089
rotate l_phal2.1 z 7
rotate r_phal2.1 z -7
scale l_phal2.2 0.042 0.0089 0.0089
scale r_phal2.2 0.042 0.0089 0.0089
rotate l_phal2.2 y 5
rotate r_phal2.2 y 5
rotate l_phal2.2 z 7
rotate r_phal2.2 z -7
scale l_phal3.1 0.0067 0.0067 0.0067
scale r_phal3.1 0.0067 0.0067 0.0067
rotate l_phal3.1 z 7
rotate r_phal3.1 z -7
scale l_phal3.2 0.04 0.0067 0.0067
scale r_phal3.2 0.04 0.0067 0.0067
rotate l_phal3.2 y 5
rotate r_phal3.2 y 5
rotate l_phal3.2 z 7
rotate r_phal3.2 z -7
scale l_phal4.1 0.0067 0.0067 0.0067
scale r_phal4.1 0.0067 0.0067 0.0067
rotate l_phal4.1 z 7
rotate r_phal4.1 z -7
scale l_phal4.2 0.037 0.0067 0.0067
scale r_phal4.2 0.037 0.0067 0.0067
rotate l_phal4.2 y 10
rotate r_phal4.2 y 10
rotate l_phal4.2 z 7
rotate r_phal4.2 z -7
scale l_phal5.1 0.0067 0.0067 0.0067
scale r_phal5.1 0.0067 0.0067 0.0067
rotate l_phal5.1 z 6
rotate r_phal5.1 z -6
scale l_phal5.2 0.038 0.0067 0.0067
scale r_phal5.2 0.038 0.0067 0.0067
rotate l_phal5.2 y 10
rotate r_phal5.2 y 10
rotate l_phal5.2 z 6
rotate r_phal5.2 z -6

move l_toe1 .5 0 0
move l_toe2.1 .5 0 0
move l_toe3.1 .5 0 0
move l_toe4.1 .5 0 0
move l_toe5.1 .5 0 0
move l_toe2.2 .5 0 0
move l_toe3.2 .5 0 0
move l_toe4.2 .5 0 0
move l_toe5.2 .5 0 0
move r_toe1 .5 0 0
move r_toe2.1 .5 0 0
move r_toe3.1 .5 0 0
move r_toe4.1 .5 0 0
move r_toe5.1 .5 0 0
move r_toe2.2 .5 0 0
move r_toe3.2 .5 0 0
move r_toe4.2 .5 0 0
move r_toe5.2 .5 0 0

scale l_toe1 0.027 .013 .013
scale r_toe1 0.027 .013 .013
rotate l_toe1 z 6
rotate r_toe1 z -6
scale l_toe2.1 .017 .0067 .0067
scale r_toe2.1 .017 .0067 .0067
rotate l_toe2.1 z 7
rotate r_toe2.1 z -7
scale l_toe3.1 .013 .0067 .0067
scale r_toe3.1 .013 .0067 .0067
rotate l_toe3.1 z 7
rotate r_toe3.1 z -7
scale l_toe4.1 .011 .0067 .0067
scale r_toe4.1 .011 .0067 .0067
rotate l_toe4.1 z 7
rotate r_toe4.1 z -7
scale l_toe5.1 .0089 .0067 .0067
scale r_toe5.1 .0089 .0067 .0067
rotate l_toe5.1 z 5
rotate r_toe5.1 z -5
scale l_toe2.2 .013 .0067 .0067
scale r_toe2.2 .013 .0067 .0067
rotate l_toe2.2 z 5
rotate r_toe2.2 z -5
scale l_toe3.2 .013 .0067 .0067
scale r_toe3.2 .013 .0067 .0067
rotate l_toe3.2 z 5
rotate r_toe3.2 z -5
scale l_toe4.2 .011 .0067 .0067
scale r_toe4.2 .011 .0067 .0067
rotate l_toe4.2 z 5
rotate r_toe4.2 z -5
scale l_toe5.2 .0089 .0067 .0067
scale r_toe5.2 .0089 .0067 .0067
rotate l_toe5.2 z 5
rotate r_toe5.2 z -5

# abdomen1 is the upper, double sheared
    object
move abdomen1 0 0 -.5
scale abdomen1 0.255 0.40 0.16
shear1d abdomen1 0 0 -1 1 0 0 1 -1.8

# add 2nd shear, in 1 direction only
# move up (by its height), so bottom stays
    the same
move abdomen1 0 0 .16
# move, so one side is more affected than
    the other
move abdomen1 .06 0 0
# Taper the thickness down, upwards
    direction
shear1d abdomen1 0 0 1 0 1 0 1.0 -3
# move back to center
move abdomen1 -.06 0 0
#move back down
move abdomen1 0 0 -.16
# move to place in abdomen frame (about 1/
    2 of the total abdomen height)
move abdomen1 0.01 0 .225

# abdomen2 is the middle part of the chest
move abdomen2 0 0 -.5
scale abdomen2 0.255 0.40 0.05
move abdomen2 0 0 -.16
shear1d abdomen2 0 0 -1 1 0 0 1 -1.8
scale abdomen2 .9 1 1
move abdomen2 0 0 .16
shear1d abdomen2 0 0 -1 1 0 0 1 4.5
move abdomen2 .3 0 0
shear1d abdomen2 0 0 -1 0 1 0 1 .5
move abdomen2 -.3 0 0
move abdomen2 .008 0 0
move abdomen2 0 0 -.16
move abdomen2 0.01 0 .225

move abdomen3 0 0 -.5
scale abdomen3 0.2352378 0.30478 0.17
move abdomen3 0.0255 0 0
shear1d abdomen3 0 0 -1 1 0 0 1 -1.1
move abdomen3 -.05 0 0
shear1d abdomen3 0 0 -1 0 1 0 1 -2.6
move abdomen3 .05 0 0
move abdomen3 0 0 .015


# make into bodies

# density: 0.8 water
addbody abdomen abdomen 0 0 1 rotary 10 1
    0 1 1
addpart abdomen1 abdomen1 abdomen 0.0 0 0
    1 1
addpart abdomen2 abdomen2 abdomen 0.0 0 0
    1 1
addpart abdomen3 abdomen3 abdomen 0.0 0 0
    1 1
addbody pelvis1 pelvis1 0 1 0 rotary 10 1
    0 1 1
addbody pelvis2 pelvis2 1 0 0 rotary 10 1
    0 1 1
addbody pelvis3 pelvis3 0 0 1 rotary 0.0 0
    0 1 1

# head joints: y, x, then z: nodding,
    tilting (abd, add), rotation
addbody head1 head1  0 1 0 rotary 10 1 0 1
    1
addbody head2 head2  1 0 0 rotary 10 1 0 1
    1
```

```
addbody head3 head3  0 0 1 rotary 0.0 0 0
    1 1
addpart neck neck abdomen 0.0 0 0 1 1

addbody l_humerus1 l_humerus1 0 1 0 rotary
    10 1 0 1 1
addbody l_humerus2 l_humerus2 1 0 0 rotary
    10 1 0 1 1
addbody l_humerus3 l_humerus3 0 0 1 rotary
    10 1 0 1 1
addbody r_humerus1 r_humerus1 0 1 0 rotary
    10 1 0 1 1
addbody r_humerus2 r_humerus2 1 0 0 rotary
    10 1 0 1 1
addbody r_humerus3 r_humerus3 0 0 1 rotary
    10 1 0 1 1

addbody l_forearm l_forearm 0 1 0 rotary
    10 1 0 1 1
addbody r_forearm r_forearm 0 1 0 rotary
    10 1 0 1 1

addbody l_hand1 l_hand1 0 1 0 rotary 10 1
    0 1 1
addbody l_hand2 l_hand2 1 0 0 rotary 10 1
    0 1 1
addbody l_hand3 l_hand3 0 0 1 rotary 0.0 0
    0 1 1
addbody r_hand1 r_hand1 0 1 0 rotary 10 1
    0 1 1
addbody r_hand2 r_hand2 1 0 0 rotary 10 1
    0 1 1
addbody r_hand3 r_hand3 0 0 1 rotary 0.0 0
    0 1 1

addbody l_thigh1 l_thigh1 0 1 0 rotary 10
    1 0 1 1
addbody l_thigh2 l_thigh2 1 0 0 rotary 10
    1 0 1 1
addbody l_thigh3 l_thigh3 0 0 1 rotary 10
    1 0 1 1
addbody r_thigh1 r_thigh1 0 1 0 rotary 10
    1 0 1 1
addbody r_thigh2 r_thigh2 1 0 0 rotary 10
    1 0 1 1
addbody r_thigh3 r_thigh3 0 0 1 rotary 10
    1 0 1 1
addbody l_shank l_shank 0 1 0 rotary 10 1
    0 1 1
addbody r_shank r_shank 0 1 0 rotary 10 1
    0 1 1

# see Procter and Paul- Ankle Joint
    Biomech paper for joint angles
addbody l_talus l_talus -0.104528 0.994522
    -0.018431106 rotary 10 1 0 1 1
addbody r_talus r_talus 0.104528 0.994522
    0.018431106 rotary 10 1 0 1 1
addbody l_hindfoot l_hindfoot -0.920505
    0.390731 -0.828826 rotary 10 1 0 1 1
addbody r_hindfoot r_hindfoot 0.920505
    0.390731 0.828826 rotary 10 1 0 1 1

addbody l_nav l_nav 1 0 0 rotary 10 1 0 1 1
addbody l_cuboid l_cuboid 1 0 0 rotary 10
    1 0 1 1
addbody r_nav r_nav 1 0 0 rotary 10 1 0 1 1
addbody r_cuboid r_cuboid 1 0 0 rotary 10
    1 0 1 1

addbody l_metat1 l_metat1 0 1 0 rotary 10
    1 0 1 1
addbody l_metat2 l_metat2 0 1 0 rotary 10
    1 0 1 1
addbody l_metat3 l_metat3 0 1 0 rotary 10
    1 0 1 1
addbody l_metat4 l_metat4 0 1 0 rotary 10
    1 0 1 1
addbody l_metat5 l_metat5 0 1 0 rotary 10
    1 0 1 1
addbody r_metat1 r_metat1 0 1 0 rotary 10
    1 0 1 1
addbody r_metat2 r_metat2 0 1 0 rotary 10
    1 0 1 1
addbody r_metat3 r_metat3 0 1 0 rotary 10
    1 0 1 1
addbody r_metat4 r_metat4 0 1 0 rotary 10
    1 0 1 1
addbody r_metat5 r_metat5 0 1 0 rotary 10
    1 0 1 1

addbody l_phal1.1 l_phal1.1 0 0 1 rotary
    10 1 0 1 1
addbody l_phal2.1 l_phal2.1 0 0 1 rotary
    10 1 0 1 1
addbody l_phal3.1 l_phal3.1 0 0 1 rotary
    10 1 0 1 1
addbody l_phal4.1 l_phal4.1 0 0 1 rotary
    10 1 0 1 1
addbody l_phal5.1 l_phal5.1 0 0 1 rotary
    10 1 0 1 1
addbody l_phal1.2 l_phal1.2 0 1 0 rotary
    10 1 0 1 1
addbody l_phal2.2 l_phal2.2 0 1 0 rotary
    10 1 0 1 1
addbody l_phal3.2 l_phal3.2 0 1 0 rotary
    10 1 0 1 1
addbody l_phal4.2 l_phal4.2 0 1 0 rotary
    10 1 0 1 1
addbody l_phal5.2 l_phal5.2 0 1 0 rotary
    10 1 0 1 1
addbody r_phal1.1 r_phal1.1 0 0 1 rotary
    10 1 0 1 1
addbody r_phal2.1 r_phal2.1 0 0 1 rotary
    10 1 0 1 1
addbody r_phal3.1 r_phal3.1 0 0 1 rotary
    10 1 0 1 1
addbody r_phal4.1 r_phal4.1 0 0 1 rotary
    10 1 0 1 1
addbody r_phal5.1 r_phal5.1 0 0 1 rotary
    10 1 0 1 1
addbody r_phal1.2 r_phal1.2 0 1 0 rotary
```

```
   10 1 0 1 1
addbody r_phal2.2 r_phal2.2 0 1 0 rotary
   10 1 0 1 1
addbody r_phal3.2 r_phal3.2 0 1 0 rotary
   10 1 0 1 1
addbody r_phal4.2 r_phal4.2 0 1 0 rotary
   10 1 0 1 1
addbody r_phal5.2 r_phal5.2 0 1 0 rotary
   10 1 0 1 1

addbody l_toe1 l_toe1 0 1 0 rotary 10 1 0
   1 1
addbody l_toe2.1 l_toe2.1 0 1 0 rotary 10
   1 0 1 1
addbody l_toe3.1 l_toe3.1 0 1 0 rotary 10
   1 0 1 1
addbody l_toe4.1 l_toe4.1 0 1 0 rotary 10
   1 0 1 1
addbody l_toe5.1 l_toe5.1 0 1 0 rotary 10
   1 0 1 1
addbody l_toe2.2 l_toe2.2 0 1 0 rotary 10
   1 0 1 1
addbody l_toe3.2 l_toe3.2 0 1 0 rotary 10
   1 0 1 1
addbody l_toe4.2 l_toe4.2 0 1 0 rotary 10
   1 0 1 1
addbody l_toe5.2 l_toe5.2 0 1 0 rotary 10
   1 0 1 1
addbody r_toe1 r_toe1 0 1 0 rotary 10 1 0
   1 1
addbody r_toe2.1 r_toe2.1 0 1 0 rotary 10
   1 0 1 1
addbody r_toe3.1 r_toe3.1 0 1 0 rotary 10
   1 0 1 1
addbody r_toe4.1 r_toe4.1 0 1 0 rotary 10
   1 0 1 1
addbody r_toe5.1 r_toe5.1 0 1 0 rotary 10
   1 0 1 1
addbody r_toe2.2 r_toe2.2 0 1 0 rotary 10
   1 0 1 1
addbody r_toe3.2 r_toe3.2 0 1 0 rotary 10
   1 0 1 1
addbody r_toe4.2 r_toe4.2 0 1 0 rotary 10
   1 0 1 1
addbody r_toe5.2 r_toe5.2 0 1 0 rotary 10
   1 0 1 1


# move to parent positions

movely l_toe2.2 l_toe2.1 .5 0 0
movely l_toe3.2 l_toe3.1 .5 0 0
movely l_toe4.2 l_toe4.1 .5 0 0
movely l_toe5.2 l_toe5.1 .5 0 0
movely r_toe2.2 r_toe2.1 .5 0 0
movely r_toe3.2 r_toe3.1 .5 0 0
movely r_toe4.2 r_toe4.1 .5 0 0
movely r_toe5.2 r_toe5.1 .5 0 0

# move toes (also phalanges, actually)
   down the full proximal phalange lengths
movely l_toe1 l_phal1.2 .5 0 0
movely r_toe1 r_phal1.2 .5 0 0
movely l_toe2.1 l_phal2.2 .5 0 0
movely r_toe2.1 r_phal2.2 .5 0 0
movely l_toe3.1 l_phal3.2 .5 0 0
movely r_toe3.1 r_phal3.2 .5 0 0
movely l_toe4.1 l_phal4.2 .5 0 0
movely r_toe4.1 r_phal4.2 .5 0 0
movely l_toe5.1 l_phal5.2 .5 0 0
movely r_toe5.1 r_phal5.2 .5 0 0

# move phalanges down the metatarsal
   lengths
movely l_phal1.1 l_metat1 .5 0 0
movely r_phal1.1 r_metat1 .5 0 0
movely l_phal2.1 l_metat2 .5 0 0
movely r_phal2.1 r_metat2 .5 0 0
movely l_phal3.1 l_metat3 .5 0 0
movely r_phal3.1 r_metat3 .5 0 0
movely l_phal4.1 l_metat4 .5 0 0
movely r_phal4.1 r_metat4 .5 0 0
movely l_phal5.1 l_metat5 .5 0 0
movely r_phal5.1 r_metat5 .5 0 0

# move metatarsals down the nav/cuboid
   lengths + the cuniform size
#  measurements from Gray
move l_metat1 0.025 0 0
move r_metat1 0.025 0 0
move l_metat2 0.022 0 0
move r_metat2 0.022 0 0
move l_metat3 0.016 0 0
move r_metat3 0.016 0 0
move l_metat4 0.027 0 0
move r_metat4 0.027 0 0
move l_metat5 0.022 0 0
move r_metat5 0.022 0 0
# move metat's off to side, on their nav/
   cuboid
move l_metat1 0 -0.003 0
move r_metat1 0 0.003 0
move l_metat1 0 0 -.005
move r_metat1 0 0 -.005
move l_metat2 0 0.014 0
move r_metat2 0 -0.014 0
move l_metat2 0 0 .005
move r_metat2 0 0 .005
move l_metat3 0 0.024 0
move r_metat3 0 -0.024 0
move l_metat3 0 0 .003
move r_metat3 0 0 .003
move l_metat4 0 .005 0
move r_metat4 0 -.005 0
move l_metat4 0 0 .003
move r_metat4 0 0 .003
move l_metat5 0 .014 0
move r_metat5 0 -.014 0
move l_metat5 0 0 -.005
move r_metat5 0 0 -.005

movely l_nav l_hindfoot 0 0 .5
movely r_nav r_hindfoot 0 0 .5
move l_nav .016 0 0
move r_nav .016 0 0
movely l_cuboid l_hindfoot 0 0 .5
movely r_cuboid r_hindfoot 0 0 .5
move l_cuboid .007 0 0
move r_cuboid .007 0 0
move l_nav 0 -.020 0
move r_nav 0 .020 0
move l_cuboid 0 .016 0
move r_cuboid 0 -.016 0
move l_cuboid 0 0 -.01
move r_cuboid 0 0 -.01

# down full talus height
movely l_hindfoot l_talus 0 0 -.5
movely r_hindfoot r_talus 0 0 -.5
movely l_talus l_shank 0 0 -.5
movely r_talus r_shank 0 0 -.5
move l_talus 0 -.01 0
move r_talus 0 .01 0
move l_talus .0075 0 0
move r_talus .0075 0 0

move l_thigh1 0 0 -.24
move l_thigh1 0 0 0.016
move l_thigh1 0 0.1 0
move l_thigh1 .03 0 0
move l_thigh1 0 0 -.01
move l_thigh1 0 0 .06
move l_thigh1 .02 0 0
move r_thigh1 0 0 -.24
move r_thigh1 0 0 0.016
move r_thigh1 0 -0.1 0
move r_thigh1 .03 0 0
move r_thigh1 0 0 -.01
move r_thigh1 0 0 .06
move r_thigh1 .02 0 0
movely l_shank l_thigh3 0 0 -.5
movely r_shank r_thigh3 0 0 -.5

movely l_hand1 l_forearm 0 0 -.5
movely r_hand1 r_forearm 0 0 -.5
# move down humerus length
movely l_forearm l_humerus3 0 0 -.5
movely r_forearm r_humerus3 0 0 -.5

# half abdomen = .225
move l_humerus1 0 0 .195
move l_humerus1 0 0 -.01
move l_humerus1 0 0.18 0
# matched to skel illustration
move l_humerus1 -0.01 0 0
move r_humerus1 0 0 .195
move r_humerus1 0 0 -.01
move r_humerus1 0 -0.18 0
# matched to skel illustration
move r_humerus1 -0.01 0 0

move pelvis1 0 0 -.105
move pelvis1 0 0 0.016
move pelvis1 -0.02 0 -.06

move head1 0 0 .295


# create tree structure

linkbodies abdomen head1
linkbodies head1 head2
linkbodies head2 head3
linkbodies abdomen pelvis1
linkbodies pelvis1 pelvis2
linkbodies pelvis2 pelvis3
linkbodies abdomen l_humerus1
linkbodies abdomen r_humerus1
linkbodies l_humerus1 l_humerus2
linkbodies l_humerus2 l_humerus3
linkbodies r_humerus1 r_humerus2
linkbodies r_humerus2 r_humerus3
linkbodies l_humerus3 l_forearm
linkbodies r_humerus3 r_forearm
linkbodies l_forearm l_hand1
linkbodies r_forearm r_hand1
linkbodies l_hand1 l_hand2
linkbodies r_hand1 r_hand2
linkbodies l_hand2 l_hand3
linkbodies r_hand2 r_hand3
linkbodies pelvis3 l_thigh1
linkbodies pelvis3 r_thigh1
linkbodies l_thigh1 l_thigh2
linkbodies l_thigh2 l_thigh3
linkbodies r_thigh1 r_thigh2
linkbodies r_thigh2 r_thigh3
linkbodies l_thigh3 l_shank
linkbodies l_shank l_talus
linkbodies r_thigh3 r_shank
linkbodies r_shank r_talus
linkbodies l_talus l_hindfoot
linkbodies r_talus r_hindfoot
linkbodies l_hindfoot l_nav
linkbodies r_hindfoot r_nav
linkbodies l_hindfoot l_cuboid
linkbodies r_hindfoot r_cuboid
linkbodies l_nav l_metat1
linkbodies l_nav l_metat2
linkbodies l_nav l_metat3
linkbodies r_nav r_metat1
linkbodies r_nav r_metat2
linkbodies r_nav r_metat3
linkbodies l_cuboid l_metat4
linkbodies l_cuboid l_metat5
linkbodies r_cuboid r_metat4
linkbodies r_cuboid r_metat5
linkbodies l_metat1 l_phal1.1
linkbodies l_metat2 l_phal2.1
linkbodies l_metat3 l_phal3.1
linkbodies l_metat4 l_phal4.1
linkbodies l_metat5 l_phal5.1
linkbodies r_metat1 r_phal1.1
linkbodies r_metat2 r_phal2.1
linkbodies r_metat3 r_phal3.1
linkbodies r_metat4 r_phal4.1
linkbodies r_metat5 r_phal5.1
linkbodies l_phal1.1 l_phal1.2
linkbodies l_phal2.1 l_phal2.2
linkbodies l_phal3.1 l_phal3.2
linkbodies l_phal4.1 l_phal4.2
linkbodies l_phal5.1 l_phal5.2
linkbodies r_phal1.1 r_phal1.2
linkbodies r_phal2.1 r_phal2.2
linkbodies r_phal3.1 r_phal3.2
linkbodies r_phal4.1 r_phal4.2
linkbodies r_phal5.1 r_phal5.2
linkbodies l_phal1.2 l_toe1
linkbodies r_phal1.2 r_toe1
linkbodies l_phal2.2 l_toe2.1
linkbodies l_phal3.2 l_toe3.1
linkbodies l_phal4.2 l_toe4.1
linkbodies l_phal5.2 l_toe5.1
linkbodies r_phal2.2 r_toe2.1
linkbodies r_phal3.2 r_toe3.1
linkbodies r_phal4.2 r_toe4.1
linkbodies r_phal5.2 r_toe5.1
linkbodies l_toe2.1 l_toe2.2
linkbodies l_toe3.1 l_toe3.2
linkbodies l_toe4.1 l_toe4.2
linkbodies l_toe5.1 l_toe5.2
linkbodies r_toe2.1 r_toe2.2
linkbodies r_toe3.1 r_toe3.2
linkbodies r_toe4.1 r_toe4.2
linkbodies r_toe5.1 r_toe5.2


# initialize the articulated figure

setroot abdomen

corpusinit
rootmotion free

move abdomen 0 0 1.4
setrootpos


# JOINT BIOMECHANICAL PARAMETERS

# initial setting for all
commandtree joint ** springtype constant
commandtree joint ** dq 0
commandtree joint ** ddq 0
commandtree joint ** Q_type 33

# setting for abdomen dummy 'joint' to
   avoid error during inverse control
joint abdomen ea 1

cl head-params
joint ** ea 62
joint ** eB 1
joint ** e_q 0
joint ** b 1
joint ** jlim_ea1 6.2
joint ** jlim_ea2 6.2
joint ** jlim_eB1 1
joint ** jlim_eB2 1
joint ** jlim_b1 .62
joint ** jlim_b2 .62
.
head-params head1
head-params head2
head-params head3

# joint limits:
# tilt back a fair amount
joint head1 jlim_q1 -0.8
# bend forward- chin to chest (penetrates
   a little)
joint head1 jlim_q2 0.9
# tilt to the side some - symmetrical
joint head2 jlim_q1 -0.8
joint head2 jlim_q2 0.8
# rotate to the side somewhat less than 90
   degrees: 1.571 rad
joint head3 jlim_q1 -1.3
joint head3 jlim_q2 1.3

cl leg-params
joint ** ea 62
joint ** eB 10
joint ** e_q 0
joint ** b 10
joint ** jlim_ea1 6.2
joint ** jlim_ea2 6.2
joint ** jlim_eB1 10
joint ** jlim_eB2 10
joint ** jlim_b1 .62
joint ** jlim_b2 .62
.
leg-params l_thigh1
leg-params r_thigh1
leg-params l_thigh2
leg-params r_thigh2
leg-params l_thigh3
leg-params r_thigh3
leg-params l_shank
leg-params r_shank
leg-params l_talus
leg-params r_talus
leg-params l_hindfoot
leg-params r_hindfoot
# use same values for pelvis
leg-params pelvis1
leg-params pelvis2
leg-params pelvis3

# joint limits
# bend forward quite a bit
joint l_thigh1 jlim_q1 -2.2
# bend back somewhat
joint l_thigh1 jlim_q2 1
joint r_thigh1 jlim_q1 -2.2
joint r_thigh1 jlim_q2 1
# don't bend in too much
```

```
joint l_thigh2 jlim_q1 -.3
# bend out somewhat
joint l_thigh2 jlim_q2 0.7
joint r_thigh2 jlim_q1 -.7
joint r_thigh2 jlim_q2 .3
# foot rotates in some
joint l_thigh3 jlim_q1 -0.7
# foot rotates out a fair amount- about 90
   degrees
joint l_thigh3 jlim_q2 1.4
joint r_thigh3 jlim_q1 -1.4
joint r_thigh3 jlim_q2 0.7

# knee bends forward just a little
joint l_shank jlim_q1 0
# knee bends back quite a bit -
joint l_shank jlim_q2 2.5
joint r_shank jlim_q1 0
joint r_shank jlim_q2 2.5

# foot bends forward some
joint l_talus jlim_q1 -0.8
joint l_talus jlim_q2 0.8
joint r_talus jlim_q1 -0.8
joint r_talus jlim_q2 0.8
# allow only a little motion
joint l_hindfoot jlim_q1 -0.5
joint l_hindfoot jlim_q2 0.5
joint r_hindfoot jlim_q1 -0.5
joint r_hindfoot jlim_q2 0.5

# bend forward a fair amount
joint pelvis1 jlim_q1 -1.5
# bend back a little
joint pelvis1 jlim_q2 0.5
# bend from side to side somewhat
joint pelvis2 jlim_q1 -1
joint pelvis2 jlim_q2 1.0
# twist (rotate) about a fair amount
joint pelvis3 jlim_q1 -1.2
joint pelvis3 jlim_q2 1.2

cl arm-params
joint ** ea 1.0
joint ** eB 10
joint ** e_q 0
joint ** b 5
joint ** jlim_ea1 1.0
joint ** jlim_ea2 1.0
joint ** jlim_eB1 10
joint ** jlim_eB2 10
joint ** jlim_b1 .1
joint ** jlim_b2 .1
.
arm-params l_humerus1
arm-params l_humerus2
arm-params l_humerus3
arm-params r_humerus1
arm-params r_humerus2
arm-params r_humerus3
arm-params l_forearm
arm-params r_forearm

# joint limits
# bend forward and up quite a bit
joint l_humerus1 jlim_q1 -3.0
# bend back a little
joint l_humerus1 jlim_q2 0.5
joint r_humerus1 jlim_q1 -3.0
joint r_humerus1 jlim_q2 0.5
# although the arm can bend in, I will
   restict it away from the body
joint l_humerus2 jlim_q1 0.1
# bend out quite a bit
joint l_humerus2 jlim_q2 2.5
joint r_humerus2 jlim_q1 -2.5
joint r_humerus2 jlim_q2 -0.1
# staring at default, w/ thumbs pointed
   outwards
# rotate inwards a fair amount
joint l_humerus3 jlim_q1 -1.7
# bend backwards just a little more
joint l_humerus3 jlim_q2 1.2
joint r_humerus3 jlim_q1 -1.2
joint r_humerus3 jlim_q2 1.7
# bend up a fair amount
joint l_forearm jlim_q1 -2.5
# bend back none
joint l_forearm jlim_q2 0
joint r_forearm jlim_q1 -2.5
joint r_forearm jlim_q2 0

cl hand-params
joint ** Q_type 17
joint ** b 1
joint ** ea 1
joint ** eB 10
.
hand-param l_hand1
hand-param l_hand2
hand-param l_hand3
hand-param r_hand1
hand-param r_hand2
hand-param r_hand3

cl nav-params
joint ** Q_type 17
joint ** b 0.5
joint ** ea 5
joint ** eB 10
.
nav-params l_nav
nav-params l_cuboid
nav-params r_nav
nav-params r_cuboid

cl metat-params
joint ** Q_type 17
joint ** b 0.2
joint ** ea 5
joint ** eB 10
.
```

```
metat-params l_metat1
metat-params l_metat2
metat-params l_metat3
metat-params l_metat4
metat-params l_metat5
metat-params r_metat1
metat-params r_metat2
metat-params r_metat3
metat-params r_metat4
metat-params r_metat5
# bigger toe gets more
joint l_metat1 b 0.4
joint r_metat1 b 0.4

cl phal-param
joint ** Q_type 17
joint ** b .003
joint ** ea .25
joint ** eB 10
.
cl phal-param.1
joint ** Q_type 17
joint ** b .003
joint ** ea .05
joint ** eB 10
.
phal-param.1 l_phal1.1
phal-param.1 l_phal2.1
phal-param.1 l_phal3.1
phal-param.1 l_phal4.1
phal-param.1 l_phal5.1
phal-param l_phal1.2
phal-param l_phal2.2
phal-param l_phal3.2
phal-param l_phal4.2
phal-param l_phal5.2
# bigger toe gets more
joint l_phal1.1 b .01
joint l_phal1.2 b .01
# big phals stiffer
joint l_phal1.2 ea .75
joint l_phal1.2 eB 10

phal-param.1 r_phal2.1
phal-param.1 r_phal3.1
phal-param.1 r_phal4.1
phal-param.1 r_phal5.1
phal-param r_phal2.2
phal-param r_phal3.2
phal-param r_phal4.2
phal-param r_phal5.2
# bigger toe gets more
joint r_phal1.1 b .01
joint r_phal1.2 b .01
# big phals stiffer
joint r_phal1.2 ea .75
joint r_phal1.2 eB 10

cl toe-param
joint ** Q_type 17
joint ** b .0001
joint ** ea 0.005
joint ** eB 10
.
toe-param l_toe1
toe-param l_toe2.1
toe-param l_toe2.2
toe-param l_toe3.1
toe-param l_toe3.2
toe-param l_toe4.1
toe-param l_toe4.2
toe-param l_toe5.1
toe-param l_toe5.2
# big toe more
joint l_toe1 b .000333
joint l_toe1 ea 0.015

toe-param r_toe1
toe-param r_toe2.1
toe-param r_toe2.2
toe-param r_toe3.1
toe-param r_toe3.2
toe-param r_toe4.1
toe-param r_toe4.2
toe-param r_toe5.1
toe-param r_toe5.2
# big toe more
joint r_toe1 b .000333
joint r_toe1 ea 0.15
```

Appendix E Body Tables

This appendix provides extra details on the structure of the humanoid model.

Table 9 provides a list of the kinematic parameters (excluding the geometric surfaces) for the complex human figure model. The "Parent" body name is the name of the body segment to which the body connects with its proximal joint. The three "Joint Axis" columns give the 3D orientation of the rotary joint axis of the body's proximal joint, in world-space. The three "Joint Offset" columns give the translational offset of the body's proximal joint, from the proximal joint of its parent.

Table 10 lists some of the mass parameters of the different body segments in the human figure model. The mass of each body segment is listed, as well as the fractional weight of each body part with respect to the whole body weight (68 kg). The weight of all of the body segments distal to a given body, including its own weight is also listed. Finally, the translational distance of a body's center of mass, from its proximal joint is provided.

Table 11 gives the joint parameters for the dampers and exponential springs, for all of the joints. The joint position (angle) and spring rest position for each joint is listed, as calibrated to a standing posture, in the anatomical position. The b, α, β, q, and q_{target} parameters are those used in Eq. 81, page 99 and Eq. 84, page 101.

Table 9: Kinematic link and joint parameters of the human figure model.

Body Name	*Parent*	*Joint Axis X*	*Joint Axis Y*	*Joint Axis Z*	*Joint Offset X*	*Joint Offset Y*	*Joint Offset Z*
r_hand3	r_hand2	0	0	1	0	0	0
r_hand2	r_hand1	1	0	0	0	0	0
r_hand1	r_forearm	0	1	0	0	0	-0.258
r_forearm	r_humerus3	0	1	0	0	-0.0114	-0.328
r_humerus3	r_humerus2	0	0	1	0	0	0
r_humerus2	r_humerus1	1	0	0	0	0	0
r_humerus1	abdomen	0	1	0	-0.01	-0.18	0.185
l_hand3	l_hand2	0	0	1	0	0	0
l_hand2	l_hand1	1	0	0	0	0	0
l_hand1	l_forearm	0	1	0	0	0	-0.258
l_forearm	l_humerus3	0	1	0	0	0.0114	-0.328
l_humerus3	l_humerus2	0	0	1	0	0	0
l_humerus2	l_humerus1	1	0	0	0	0	0
l_humerus1	abdomen	0	1	0	-0.01	0.18	0.185
r_toe5.2	r_toe5.1	0	1	0	0.00887	-0.00078	0
r_toe5.1	r_phal5.2	0	1	0	0.0372	-0.00391	-0.0066
r_phal5.2	r_phal5.1	0	1	0	0	0	0
r_phal5.1	r_metat5	0	0	1	0.0593	-0.0142	-0.0159
r_metat5	r_cuboid	0	1	0	0.022	-0.014	-0.005
r_toe4.2	r_toe4.1	0	1	0	0.0109	-0.00134	0
r_toe4.1	r_phal4.2	0	1	0	0.0362	-0.00444	-0.00642
r_phal4.2	r_phal4.1	0	1	0	0	0	0
r_phal4.1	r_metat4	0	0	1	0.0648	-0.0122	-0.0236
r_metat4	r_cuboid	0	1	0	0.027	-0.005	0.003
r_cuboid	r_hindfoot	1	0	0	0.007	-0.023	-0.01
r_toe3.2	r_toe3.1	0	1	0	0.0129	-0.00158	0
r_toe3.1	r_phal3.2	0	1	0	0.0396	-0.00486	-0.00349
r_phal3.2	r_phal3.1	0	1	0	0	0	0

Table 9: Kinematic link and joint parameters of the human figure model.

Body Name	*Parent*	*Joint Axis X*	*Joint Axis Y*	*Joint Axis Z*	*Joint Offset X*	*Joint Offset Y*	*Joint Offset Z*
r_phal3.1	r_metat3	0	0	1	0.0788	-0.0138	-0.0367
r_metat3	r_nav	0	1	0	0.016	-0.024	0.003
r_toe2.2	r_toe2.1	0	1	0	0.0169	-0.00207	0
r_toe2.1	r_phal2.2	0	1	0	0.0415	-0.0051	-0.00366
r_phal2.2	r_phal2.1	0	1	0	0	0	0
r_phal2.1	r_metat2	0	0	1	0.089	-0.0102	-0.0396
r_metat2	r_nav	0	1	0	0.022	-0.014	0.005
r_toe1	r_phal1.2	0	1	0	0.0476	-0.00585	0
r_phal1.2	r_phal1.1	0	1	0	0	0	0
r_phal1.1	r_metat1	0	0	1	0.0865	0.00159	-0.0281
r_metat1	r_nav	0	1	0	0.025	0.003	-0.005
r_nav	r_hindfoot	1	0	0	0.016	0.013	0
r_hindfoot	r_talus	0.70872	0.300834	0.638134	0	0	-0.022
r_talus	r_shank	0.10451	0.994353	0.018428	0.0075	0.01	-0.434
r_shank	r_thigh3	0	1	0	0	0	-0.432
r_thigh3	r_thigh2	0	0	1	0	0	0
r_thigh2	r_thigh1	1	0	0	0	0	0
r_thigh1	pelvis3	0	1	0	0.05	-0.1	-0.174
l_toe5.2	l_toe5.1	0	1	0	0.00887	0.000776	0
l_toe5.1	l_phal5.2	0	1	0	0.0372	0.00391	-0.0066
l_phal5.2	l_phal5.1	0	1	0	0	0	0
l_phal5.1	l_metat5	0	0	1	0.0593	0.0142	-0.0159
l_metat5	l_cuboid	0	1	0	0.022	0.014	-0.005
l_toe4.2	l_toe4.1	0	1	0	0.0109	0.00134	0
l_toe4.1	l_phal4.2	0	1	0	0.0362	0.00444	-0.00642
l_phal4.2	l_phal4.1	0	1	0	0	0	0
l_phal4.1	l_metat4	0	0	1	0.0648	0.0122	-0.0236
l_metat4	l_cuboid	0	1	0	0.027	0.005	0.003

Table 9: Kinematic link and joint parameters of the human figure model.

Body Name	*Parent*	*Joint Axis X*	*Joint Axis Y*	*Joint Axis Z*	*Joint Offset X*	*Joint Offset Y*	*Joint Offset Z*
l_cuboid	l_hindfoot	1	0	0	0.007	0.023	-0.01
l_toe3.2	l_toe3.1	0	1	0	0.0129	0.00158	0
l_toe3.1	l_phal3.2	0	1	0	0.0396	0.00486	-0.00349
l_phal3.2	l_phal3.1	0	1	0	0	0	0
l_phal3.1	l_metat3	0	0	1	0.0788	0.0138	-0.0367
l_metat3	l_nav	0	1	0	0.016	0.024	0.003
l_toe2.2	l_toe2.1	0	1	0	0.0169	0.00207	0
l_toe2.1	l_phal2.2	0	1	0	0.0415	0.0051	-0.00366
l_phal2.2	l_phal2.1	0	1	0	0	0	0
l_phal2.1	l_metat2	0	0	1	0.089	0.0102	-0.0396
l_metat2	l_nav	0	1	0	0.022	0.014	0.005
l_toe1	l_phal1.2	0	1	0	0.0476	0.00585	0
l_phal1.2	l_phal1.1	0	1	0	0	0	0
l_phal1.1	l_metat1	0	0	1	0.0865	-0.00159	-0.0281
l_metat1	l_nav	0	1	0	0.025	-0.003	-0.005
l_nav	l_hindfoot	1	0	0	0.016	-0.013	0
l_hindfoot	l_talus	-0.70872	0.300834	-0.63813	0	0	-0.022
l_talus	l_shank	-0.10451	0.994353	-0.01843	0.0075	-0.01	-0.434
l_shank	l_thigh3	0	1	0	0	0	-0.432
l_thigh3	l_thigh2	0	0	1	0	0	0
l_thigh2	l_thigh1	1	0	0	0	0	0
l_thigh1	pelvis3	0	1	0	0.05	0.1	-0.174
pelvis3	pelvis2	0	0	1	0	0	0
pelvis2	pelvis1	1	0	0	0	0	0
pelvis1	abdomen	0	1	0	-0.02	0	-0.149
head3	head2	0	0	1	0	0	0
head2	head1	1	0	0	0	0	0
head1	abdomen	0	1	0	0	0	0.295

Table 10: Mass parameters of the body parts in the human figure model.

Body Name	*Mass (kg)*	*Mass/ BodyMass*	*Distal Mass (including self) (kg)*	*COM dist. from proximal (m)*
r_hand3	0.408	0.006	0.408	0.0824
r_hand2	9.85e-06	1.45e-07	0.408	0
r_hand1	9.85e-06	1.45e-07	0.408	0
r_forearm	1.08	0.0159	1.49	0.112
r_humerus3	1.9	0.0279	3.39	0.143
r_humerus2	0.00123	1.81e-05	3.39	0
r_humerus1	0.00123	1.81e-05	3.39	0
l_hand3	0.408	0.006	0.408	0.0824
l_hand2	9.85e-06	1.45e-07	0.408	0
l_hand1	9.85e-06	1.45e-07	0.408	0
l_forearm	1.08	0.0159	1.49	0.112
l_humerus3	1.9	0.0279	3.39	0.143
l_humerus2	0.00123	1.81e-05	3.39	0
l_humerus1	0.00123	1.81e-05	3.39	0
r_toe5.2	0.00283	4.17e-05	0.00283	0.007
r_toe5.1	0.00153	2.24e-05	0.00436	0.00445
r_phal5.2	0.0116	0.000171	0.016	0.0185
r_phal5.1	2.96e-06	4.36e-08	0.016	0.00335
r_metat5	0.0239	0.000352	0.0399	0.0304
r_toe4.2	0.00324	4.76e-05	0.00324	0.008
r_toe4.1	0.00189	2.77e-05	0.00512	0.0055
r_phal4.2	0.0133	0.000196	0.0184	0.0179
r_phal4.1	2.96e-06	4.36e-08	0.0184	0.00335
r_metat4	0.0338	0.000497	0.0522	0.0334
r_cuboid	0.0564	0.000829	0.149	0.013
r_toe3.2	0.00392	5.77e-05	0.00392	0.009
r_toe3.1	0.00243	3.57e-05	0.00635	0.0065
r_phal3.2	0.0133	0.000195	0.0196	0.0185

Table 10: Mass parameters of the body parts in the human figure model.

Body Name	*Mass (kg)*	*Mass/ BodyMass*	*Distal Mass (including self) (kg)*	*COM dist. from proximal (m)*
r_phal3.1	2.96e-06	4.36e-08	0.0196	0.00335
r_metat3	0.0472	0.000694	0.0668	0.0414
r_toe2.2	0.00392	5.77e-05	0.00392	0.009
r_toe2.1	0.00318	4.67e-05	0.0071	0.0085
r_phal2.2	0.0216	0.000317	0.0287	0.0188
r_phal2.1	6.94e-06	1.02e-07	0.0287	0.00445
r_metat2	0.0794	0.00117	0.108	0.0485
r_toe1	0.0212	0.000312	0.0212	0.016
r_phal1.2	0.0427	0.000628	0.0639	0.0218
r_phal1.1	4.03e-05	5.93e-07	0.064	0.008
r_metat1	0.113	0.00166	0.177	0.0456
r_nav	0.0505	0.000743	0.402	0.011
r_hindfoot	0.23	0.00339	0.781	0.0327
r_talus	0.204	0.003	0.985	0.0234
r_shank	3.15	0.0463	4.13	0.188
r_thigh3	6.79	0.0999	10.9	0.187
r_thigh2	0.00985	0.000145	10.9	0
r_thigh1	0.00985	0.000145	10.9	0
l_toe5.2	0.00283	4.17e-05	0.00283	0.007
l_toe5.1	0.00153	2.24e-05	0.00436	0.00445
l_phal5.2	0.0116	0.000171	0.016	0.0185
l_phal5.1	2.96e-06	4.36e-08	0.016	0.00335
l_metat5	0.0239	0.000352	0.0399	0.0304
l_toe4.2	0.00324	4.76e-05	0.00324	0.008
l_toe4.1	0.00189	2.77e-05	0.00512	0.0055
l_phal4.2	0.0133	0.000196	0.0184	0.0179
l_phal4.1	2.96e-06	4.36e-08	0.0184	0.00335
l_metat4	0.0338	0.000497	0.0522	0.0334

Table 10: Mass parameters of the body parts in the human figure model.

Body Name	*Mass (kg)*	*Mass/ BodyMass*	*Distal Mass (including self) (kg)*	*COM dist. from proximal (m)*
l_cuboid	0.0564	0.000829	0.149	0.013
l_toe3.2	0.00392	5.77e-05	0.00392	0.009
l_toe3.1	0.00243	3.57e-05	0.00635	0.0065
l_phal3.2	0.0133	0.000195	0.0196	0.0185
l_phal3.1	2.96e-06	4.36e-08	0.0196	0.00335
l_metat3	0.0472	0.000694	0.0668	0.0414
l_toe2.2	0.00392	5.77e-05	0.00392	0.009
l_toe2.1	0.00318	4.67e-05	0.0071	0.0085
l_phal2.2	0.0216	0.000317	0.0287	0.0188
l_phal2.1	6.94e-06	1.02e-07	0.0287	0.00445
l_metat2	0.0794	0.00117	0.108	0.0485
l_toe1	0.0212	0.000312	0.0212	0.016
l_phal1.2	0.0427	0.000628	0.0639	0.0218
l_phal1.1	4.03e-05	5.93e-07	0.064	0.008
l_metat1	0.113	0.00166	0.177	0.0456
l_nav	0.0505	0.000743	0.402	0.011
l_hindfoot	0.23	0.00339	0.781	0.0327
l_talus	0.204	0.003	0.985	0.0234
l_shank	3.15	0.0463	4.13	0.188
l_thigh3	6.79	0.0999	10.9	0.187
l_thigh2	0.00985	0.000145	10.9	0
l_thigh1	0.00985	0.000145	10.9	0
pelvis3	11.1	0.163	33	0.145
pelvis2	9.85e-06	1.45e-07	33	0
pelvis1	9.85e-06	1.45e-07	33	0
head3	4.71	0.0692	4.71	0.116
head2	9.85e-06	1.45e-07	4.71	0
head1	9.85e-06	1.45e-07	4.71	0

Table 10: Mass parameters of the body parts in the human figure model.

Body Name	*Mass (kg)*	*Mass/ BodyMass*	*Distal Mass (including self) (kg)*	*COM dist. from proximal (m)*
abdomen	23.5	0.346	68	0.0372

Table 11: Joint angles, damping constants, and exponential spring parameters, calibrated to a standing posture.

Body Name	*Joint Angle - q*	*Damper - b*	*Spring Linear Stiffness- α*	*Spring Exp. Stiffness - β*	*Spring Rest Angle - q_{target}*
r_hand3	1	0.1	1	10	1
r_hand2	0.3	0.1	1	10	0.303
r_hand1	0.2	0.1	1	10	0.19
r_forearm	-0.35	1	1	10	-0.417
r_humerus3	-1	1	1	10	-1.01
r_humerus2	-0.1	1	1	10	-0.203
r_humerus1	0.05	1	1	10	0.0882
l_hand3	-1	0.1	1	10	-1
l_hand2	-0.3	0.1	1	10	-0.303
l_hand1	0.2	0.1	1	10	0.19
l_forearm	-0.35	1	1	10	-0.417
l_humerus3	1	1	1	10	1.01
l_humerus2	0.1	1	1	10	0.203
l_humerus1	0.05	1	1	10	0.0882
r_toe5.2	0.3	0.0001	0.005	10	0.491
r_toe5.1	0.3	0.0001	0.005	10	0.541
r_phal5.2	-0.4	0.003	0.25	10	-0.355
r_phal5.1	0	0.003	0.05	10	-0.00255
r_metat5	0.14	0.2	5	10	0.146
r_toe4.2	0.3	0.0001	0.005	10	0.63
r_toe4.1	0.3	0.0001	0.005	10	0.689
r_phal4.2	-0.38	0.003	0.25	10	-0.257
r_phal4.1	0	0.003	0.05	10	0.0118
r_metat4	0.12	0.2	5	10	0.149
r_cuboid	0	0.5	5	10	0.0087
r_toe3.2	0.3	0.0001	0.005	10	0.761
r_toe3.1	0.3	0.0001	0.005	10	0.826

Table 11: Joint angles, damping constants, and exponential spring parameters, calibrated to a standing posture.

Body Name	*Joint Angle - q*	*Damper - b*	*Spring Linear Stiffness- α*	*Spring Exp. Stiffness - β*	*Spring Rest Angle - q_{target}*
r_phal3.2	-0.27	0.003	0.25	10	-0.0358
r_phal3.1	0	0.003	0.05	10	-0.0101
r_metat3	0.06	0.2	5	10	0.135
r_toe2.2	0.3	0.0001	0.005	10	0.788
r_toe2.1	0.3	0.0001	0.005	10	0.866
r_phal2.2	-0.23	0.003	0.25	10	0.0372
r_phal2.1	0	0.003	0.05	10	-0.0223
r_metat2	0.03	0.2	5	10	0.144
r_toe1	0.4	0.000333	0.015	10	0.834
r_phal1.2	-0.1	0.01	0.75	10	0.0635
r_phal1.1	0	0.01	0.15	10	-0.0176
r_metat1	0	0.4	5	10	0.0838
r_nav	0	0.5	5	10	0.0619
r_hindfoot	-0.04	1	62	10	-0.0246
r_talus	-0.07	1	62	10	-0.0361
r_shank	0	10	62	10	0.0166
r_thigh3	0	10	62	10	-0.00193
r_thigh2	0.05	10	62	10	0.0335
r_thigh1	0.09	10	62	10	0.0864
l_toe5.2	0.3	0.0001	0.005	10	0.491
l_toe5.1	0.3	0.0001	0.005	10	0.541
l_phal5.2	-0.4	0.003	0.25	10	-0.355
l_phal5.1	0	0.003	0.05	10	0.00255
l_metat5	0.14	0.2	5	10	0.146
l_toe4.2	0.3	0.0001	0.005	10	0.63
l_toe4.1	0.3	0.0001	0.005	10	0.689
l_phal4.2	-0.38	0.003	0.25	10	-0.257

Table 11: Joint angles, damping constants, and exponential spring parameters, calibrated to a standing posture.

Body Name	*Joint Angle - q*	*Damper - b*	*Spring Linear Stiffness- α*	*Spring Exp. Stiffness - β*	*Spring Rest Angle - q_{target}*
l_phal4.1	0	0.003	0.05	10	-0.0118
l_metat4	0.12	0.2	5	10	0.149
l_cuboid	0	0.5	5	10	-0.0087
l_toe3.2	0.3	0.0001	0.005	10	0.761
l_toe3.1	0.3	0.0001	0.005	10	0.826
l_phal3.2	-0.27	0.003	0.25	10	-0.0358
l_phal3.1	0	0.003	0.05	10	0.0101
l_metat3	0.06	0.2	5	10	0.135
l_toe2.2	0.3	0.0001	0.005	10	0.788
l_toe2.1	0.3	0.0001	0.005	10	0.866
l_phal2.2	-0.23	0.003	0.25	10	0.0372
l_phal2.1	0	0.003	0.05	10	0.0223
l_metat2	0.03	0.2	5	10	0.144
l_toe1	0.4	0.000333	0.015	10	0.834
l_phal1.2	-0.1	0.01	0.75	10	0.0635
l_phal1.1	0	0.01	0.15	10	0.0176
l_metat1	0	0.4	5	10	0.0838
l_nav	0	0.5	5	10	-0.0619
l_hindfoot	-0.04	1	62	10	-0.0246
l_talus	-0.07	1	62	10	-0.0361
l_shank	0	10	62	10	0.0166
l_thigh3	0	10	62	10	0.00193
l_thigh2	-0.05	10	62	10	-0.0335
l_thigh1	0.09	10	62	10	0.0864
pelvis3	0	10	62	10	0
pelvis2	0	10	62	10	0
pelvis1	0	10	62	10	0.0183

Table 11: Joint angles, damping constants, and exponential spring parameters, calibrated to a standing posture.

Body Name	*Joint Angle - q*	*Damper - b*	*Spring Linear Stiffness-* α	*Spring Exp. Stiffness -* β	*Spring Rest Angle -* q_{target}
head3	0	1	62	1	0
head2	0	1	62	1	0
head1	0	1	62	1	-0.0172

List of Terms

General Notations

$\hat{a}$, $\hat{A}$: the carat ("^ ")above a quantity denotes that it is in spatial notation. [Featherstone 1983; Featherstone 1987] Lowercase letters in spatial notation ($\hat{a}$) refer to a six dimensional spatial vector. Capital letters with a carat above them ($\hat{A}$) refer to 6×6 spatial matrixes.

$\hat{a}_i$, $\hat{A}_i$: a spatial quantity associated with body *i*.

Operators

$a\times$: cross operator, see Eq. 11, page 79. from a 3×1 to a 3×3.

$\hat{a}\hat{\times}$: spatial cross operator, Eq. 12, page 79. From 6×1 to 6×6.

$\hat{a}_i^S$, $\hat{A}_i^S$: spatial transpose of a spatial vector or matrix, see Eq. 13 and Eq. 14, page 80.

Terms

$\hat{a}$: spatial acceleration, see Eq. 7, page 78.

b: generally used as a damping constant, a scalar.

e: generally used as a coefficient of restitution, a scalar.

f: 3 dimensional linear force vector, see Eq. 9, page 79.

$\hat{f}$: spatial force.

g: gravitational constant: -9.81 meters/second^2.

i: the index number of a body within an articulated figure, from 1 to *n*.

$\hat{I}$: spatial inertia tensor, a 6×6 spatial matrix, see Eq. 15, page 80.

$\hat{I}^A$: articulated body inertia, a 6×6 spatial matrix.

m: mass, a scalar.

n: the number of bodies in an articulated figure.

$\hat{p}^v$: bias force.

$\hat{p}_i$: the bias force of body i. This includes the $\hat{p}_v$ bias force, as well as other components.

q: joint position, in meters or radians, a scalar.

$\dot{q}$: joint velocity, in meters/second or radians/second, a scalar.

$\ddot{q}$: joint acceleration, in meters/second^2 or radians/second^2, a scalar.

Q: joint force, a scalar, either a linear force or torque, depending on the joint type.

$\hat{s}_i$: the spatial joint axis for body i.

t: time.

$\hat{v}$: spatial velocity, see Eq. 4, page 77.

v_O: the linear velocity at the origin of the coordinate frame.

${}_i\hat{X}_j$: the spatial transformation, which transforms values from the coordinate system of body j, to the coordinate system of body i (or, more generally, from coordinate frame j to frame i).

α: the stiffness constant which multiplies linearly with an exponential term. See Eq. 81, page 99.

β: the stiffness constant in the exponential term of an exponential spring. See Eq. 81, page 99.

γ: the coefficient of friction, a scalar.

τ: 3 dimensional torque vector.

ω: 3 dimensional angular velocity.

$\dot{\omega}$: 3 dimensional angular acceleration.

Glossary

abduct: to move a limb in a direction away from the median axis of the body ("outwards" to the side); to separate. See *adduct.*

ABM: see *Articulated Body Method.*

adduct: to move a limb towards the median axis of the body ("inwards" from the side); to bring together. See *abduct.*

active control: a mechanism used to regulate or guide the operation of a machine, apparatus, or system. In this context *active control* refers more specifically to the variation of the actuator parameters in order to control motion.

actuator: a mechanism, such as a muscle or motor, used for moving or controlling a system. In this context: a force-producing agent which operates at the joint of an articulated figure to control motion.

Articulated Body Method (ABM): an efficient dynamic simulation method, developed by Roy Featherstone, for branching articulated figures, comprised of rigid bodies. This recursive algorithm has a computational expense of *O(n)* where *n* is the number of joints (a linear computational expense). The algorithm is developed in [Featherstone 1983; Featherstone 1987].

articulated body: see *articulated figure.*

articulated figure: A set of bodies, connected by joints, which creates an overall bodily shape or form, such as a person or mechanism.

articulated structure: see *articulated figure.*

biomechanics: the study of biological motions, and the forces and energies which create them.

body: in this context: an individual rigid object, which can be free to move independently, or can be connected to other bodies via joints to form an *articulated figure.* "Body" is usually used in this context to refer to a simulated body, as part of a dynamic simulation system. Bodies have properties such as shape, mass, density, center of mass, etc.

center of gravity (COG): See *center of mass*.

center of mass (COM): The center of mass of a body (or a set of bodies considered as a whole) is the mean point of the collective mass. A linear force applied at this point will induce no rotational acceleration. In order to remain statically stable, an articulated figure must keep its body's center of mass within the region formed by its supporting feet. The terms "center of gravity" and "center of mass" are often used interchangeably, although a distinction exists. "Center of gravity" refers to a body's center of mass in one dimension only, in the vertical, gravity-defined direction. [Winter 1990]

COG: see *center of gravity.*

COM: see *center of mass.*

corpus: a computer program developed by the author for the dynamic simulation and control of articulated figures.

degree of freedom (DOF): "one of a limited number of ways in which a point or a body may move or in which a dynamic system may change, each way being expressed by an independent variable and all requiring to be specified if the physical state of the body or system is to be completely defined." [Webster's] A DOF is an unrestricted or unconstrained direction in which motion (as in a joint) is possible. An unconstrained body, free to move in space, has 6 DOFs, 3 translating and 3 rotating. A joint can provide from 1 to 6 DOFs, depending on its nature. A single joint in *corpus* provides 1 DOF. Simply because a DOF is present does not mean that movement is completely unconstrained in that direction, as there may be forces inhibiting such movement, just as the human elbow has a limited range of motion.

determinants of gait: a set of motion characteristics of human walking put forth by Saunders, Inman, and Eberhart, 1953. These six determinants describe the motions of the limbs, and together, they capture the major types of motions of the lower body which occur during walking.

distal: located away from the center of the body. The forearm is more distal than the upper arm. See *proximal.* [Webster's]

DOF: see *degree of freedom.*

dynamics: the study of forces and their relationship to the motions of bodies of matter.

end effector: a body which lies at a peripheral terminus of an articulated figure.

equilibrium position hypothesis: a theory of biological motion control, which states that controlled movements are the result of shifts of the equilibrium, or *postural*, state of the motor system, developing a relationship between posture and movement. [Bizzi 1982; Bizzi 1984]

extension: the bending of a joint, such that the angle between the two adjacent limbs is increased, in order to straighten out a flexed limb.

figure: see *articulated figure.*

flexion: the bending of a joint, such that the angle between the two adjacent limbs is diminished.

force: a strength or energy brought to bear; an agency or influence that results in the acceleration of a free body. [Webster's] Used in this text in a general sense, to include both linear forces and rotational forces, or *torques.*

forward control: the calculation of an output force from specified actuator control parameters.

forward dynamics: the calculation of the motion (acceleration) of a body (or articulated figure), based on the applied force.

forward kinematics: the calculation of the positions and orientations of the bodies in an articulated figure, based on the joint positions and/or angles.

gait: a manner of walking or moving on foot. [Webster's]

hybrid dynamics: a combination of forward and inverse dynamics; the calculation of the unresolved accelerations and forces within an articulated figure, based on a complementary set of specified forces and accelerations.

inverse control: the calculation of the control parameters required to achieve a specified force.

inverse dynamics: the calculation of the force required to achieve a specified acceleration of a body (or articulated figure).

inverse kinematics: the calculation of the joint positions and/or angles within an articulated figure in order to achieve specified kinematic goals, such as the cartesian position of an end effector.

kinematics: the study of motions, apart from considerations of mass or force. [Webster's]

kinetics: the study of the forces which give rise to motions, and their resulting energetics.

medial: of or relating to the plane which divides a bilateral animal into left and right halves. [Webster's]

motor program: a theorized underlying representation for the production of movements in biological systems, based largely on centrally "stored" sequences of muscular activation. In this context: a mechanism which varies actuator control parameters in order to achieve a specific motion.

passive control: a means of producing motion which does not employ *active control*— the motions are accomplished without variation of the control parameters. Gravity, inertia, and the mechanical properties of the system govern the motion.

parser: in *corpus*, the program sub-system which takes text input and converts, or "parses," it into functions which are then executed.

physically based model: a computational model of a natural phenomena, using the laws of physics as the basis of the simulation.

posture: a kinematic configuration of an articulated figure; the relative positions of the limbs with respect to each other.

proximal: located toward the center of the body. [Webster's] The upper arm is more proximal than the forearm. See *distal.*

rigid body: A rigid body is a body which does not undergo any internal deformations. The shape of the body is constant, and it perfectly transmits forces through itself. No real-world objects are completely rigid, and simulations which use See *body.*

rotation: 1) an angular displacement about an axis 2) the turning of a limb about its long axis, as if on a pivot. [Webster's]

rotoscoping: the technique of mimicking the real-world motions of humans, animals or some other moving system, through film or a similar imaging medium. When rotoscoping, artists design their animated drawings, copying from the recorded moving image. Rotoscoping techniques are also used with computer models, as artists create motions for their articulated figures, "copying" from real-world moving images.

sagittal: of, relating to, or situated in the median plane, or any plane parallel thereto. [Webster's]

script: in *corpus*, a script is an ACSII text file, which contains functions and commands which are interpreted and executed by the *corpus* parser. Scripts (and typed input) are the means by which *corpus* is controlled.

spatial algebra: the algebra of *spatial notation*, using the standard operations of matrix arithmetic, with the primary exception of a unique transposition operator.

spatial notation: a mathematical representation which combines the linear and angular components of physical quantities into 6 dimensional vectors and 6x6 matrixes, which allows for fewer and simpler quantities and equations. Quantities in spatial notation are denoted with a carat ('^'), as in $\hat{a}$.

stability margin: (or *static stability margin*) the distance from the vertical projection of the center of gravity onto the support surface to the closest point of the boundary of support polygon. [Messuri; McGhee]

stiff, stiffness: there are several related meanings for "stiff." A "stiff" spring is one that strongly resists deviating from its rest position. During integration, "stiff" refers to the "difficulty" in getting accurate results. The stiffer the system, the smaller the integrator time-steps must be, which means more computation must be taken to cover the same time interval. Stiff springs will lead to a stiff system (with stiff equations of motion) because the springs' feedback loops (from position to force to acceleration and back to velocity and position) becomes tighter, and typically more integrator samples are needed to accurately follow the variations in spring force.

support region: the convex hull area formed by the points of contact between a figure and the support surface.

support polygon: (or *support pattern*) the boundary of the support region.

timestep: the amount of simulation time, in seconds, which passes between each call to the dynamics simulator. Acceleration, velocity, and other values in the simulator are integrated over this discrete time period, *dt*.

torque: a rotational force; an unopposed torque applied to a body will induce a rotational acceleration (or "angular acceleration"). The term *force* is used in this text in a general sense, including torques as well as linear forces.

virtual environment: an interactive computer model which represents an "environment," which is based on a set of rules, and typically represents some aspect(s) of the real-world.

Index of References

Index

Note

Page numbers in italics refer to illustrations.