October 1991

Interactive Postural Control of Articulated Geometric Figures (Dissertation)

Cary B. Phillips
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-91-82.

Interactive Postural Control of Articulated Geometric Figures (Dissertation)

Abstract

Interactive postural control is the process of interactively pushing, poking, and twisting parts of an articulated geometric figure for the express purpose of getting it into a desired posture. Many motion algorithms and computer animation techniques generate motion sequences based on starting and ending postures for geometric figures, but few of these techniques address the fundamental problem of specifying these postures. The goal of this thesis is to develop a system that allows us to specify postures of animate geometric figures in ways that suggest how we interact with real people.

The emphasis of this thesis is on real-time interactive 3D manipulation. The elements of the interaction techniques form a powerful vocabulary for describing postures and postural adjustments. The vocabulary is not a spoken or written one; rather, it includes verbs acted out by the user through the movement of input devices.

There are three major components to this work. The first component is a real-time 3D direct manipulation technique that allows the user to intuitively translate and rotate "handles" on objects using only a three button mouse as input. The second component is an inverse kinematics algorithm that uses the notion of constraints, or desired geometric relationships, to control postures of articulated figures. The inverse kinematics formulation is well suited to highly redundant figures. The final component is the system of behaviors. Behaviors provide coordination between the parts of the figure, so that when one part of a figure moves, the body reacts as a whole. One of the most important behaviors, and the one requiring the most coordination, is balance. The behaviors magnify the effect of the basic manipulation commands so that relatively few invocations of the commands are necessary to accomplish a complex positioning task.

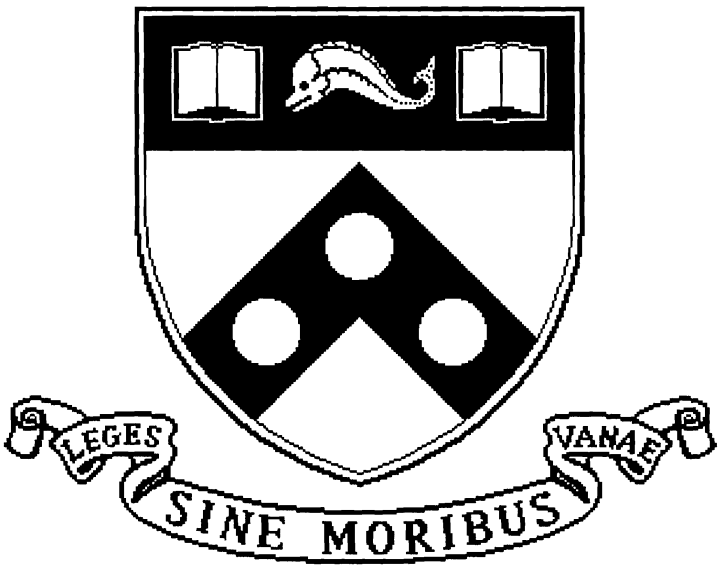
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-91-82.

Interactive Postural Control Of Articulated Geometric Figures (Dissertation)

MS-CIS-91-82
GRAPHICS LAB 45

Cary B. Phillips



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

1991

Interactive Postural Control Of Articulated Geometric Figures (Dissertation)

MS-CIS-91-82
GRAPHICS LAB 45

Cary B. Phillips

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

October 1991

Interactive Postural Control of Articulated Geometric Figures

Cary B. Phillips

Presented to the Faculties of the University of Pennsylvania in Partial Fullfillment of the Requirements for the Degree of Doctor of Philosophy

1991

To my wife, Denise

To my parents, Carl and Nina

And to the other members of my loving family, Katie, Fred, and Ethel

Acknowledgements

I owe a great debt to Dr. Norman I. Badler, who has always been a great source of support as both my advisor and supervisor. I think we have had a wonderful relationship over the last six years. Norm has always made the Computer Graphics Research Lab at Penn an exciting, energetic, and fun place to be.

I also owe a great debt to Jianmin Zhao, without whom this thesis research would not have been possible. Jianmin opened up a whole new world to me through his inverse kinematics software, and he revolutionized what the *Jack* software can do. Jianmin is also a great friend and a great student of American culture.

I would also like to thank the many people at our funding agencies, both for their financial support and for their patience in using and constructively criticizing our software:

- At the US Army Human Engineering Lab, Dr. Ben Cummings, Brenda Thein, and Bernie Corona.
- Also at the US Army Human Engineering Lab, Richard Kozycki. Rick has probably used *Jack* more than anyone, and his patience and persistence have been a source of inspiration to me. Rick, I appreciate your effort.
- At NASA Johnson Space Center, Barbara Woolford, Edmund Khouri, James Maida, Abilash Pandya, Ann Aldridge, Debbie Barella, and the late Linda Orr. NASA and Lockheed have long been consistent supporters of the research at Penn. Jim, Abilash, Ann, Debbie, and Linda were some of the ones to use *Jack*.
- At NASA Ames Research Center, James Hartzell, James Larimer, Barry Smith, Christian Neukom, Mike Prevost, and Gretchen Helms. Gretchen, Mike, and Chris have slogged through a lot of *Jack* code over the past few years.
- At FMC Corp, Ed Bellandi.
- At Deere & Company, Jerry Duncan. I always did like tractors. Thanks for your support.

This research is partially supported by Lockheed Engineering and Management Services (NASA Johnson Space Center), NASA Ames Grant NAG-2-426, NASA Goddard through University of Iowa UICR, FMC Corporation, Siemens Research, NSF CISE Grant CDA88-22719, Air Force HRL/LR ILIR-40-02 and F33615-88-C-0004 (SEI), and ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Human Engineering Laboratory, Natick Laboratory, and TACOM.

Abstract

Interactive postural control is the process of interactively pushing, poking, and twisting parts of an articulated geometric figure for the express purpose of getting it into a desired posture. Many motion algorithms and computer animation techniques generate motion sequences based on starting and ending postures for geometric figures, but few of these techniques address the fundamental problem of specifying these postures. The goal of this thesis is to develop a system that allows us to specify postures of animate geometric figures in ways that suggest how we interact with real people.

The emphasis of this thesis is on real-time interactive 3D manipulation. The elements of the interaction techniques form a powerful vocabulary for describing postures and postural adjustments. The vocabulary is not a spoken or written one; rather, it includes verbs acted out by the user through the movement of input devices.

There are three major components to this work. The first component is a real-time 3D direct manipulation technique that allows the user to intuitively translate and rotate "handles" on objects using only a three button mouse as input. The second component is an inverse kinematics algorithm that uses the notion of constraints, or desired geometric relationships, to control postures of articulated figures. The inverse kinematics formulation is well suited to highly redundant figures. The final component is the system of behaviors. Behaviors provide coordination between the parts of the figure, so that when one part of a figure moves, the body reacts as a whole. One of the most important behaviors, and the one requiring the most coordination, is balance. The behaviors magnify the effect of the basic manipulation commands so that relatively few invocations of the commands are necessary to accomplish a complex positioning task.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

James Foley and Victor Wallace eloquently described the very nature of computer graphics as a means of communication in 1974, practically before the dawn of the era of computer graphics. Their words are still insightful today:

> The dictionary provides two meanings for the world "graphic." One is "pertaining to the drawing of marks, lines or characters on a surface."[1] Although the traditional usage of the term "computer graphics" clearly derives from this meaning, it is by no means accidental that the alternate meaning is given by the phrase "clearly and vividly described." It is precisely because graphics (first definition) are graphic (second definition) that they are used as a medium of communication between man and machine [30].

This thesis, in its most basic sense, involves the communication between man and machine, particularly the communication of geometric information involving the postures of complex articulated figures.

1.1 Motivation

Computer animation has evolved greatly in recent years. There are now motion generation algorithms to simulate a wide variety of phenomena, from leaves blowing in the wind to humans walking and snakes

[1] C.L. Barnhard, Ed., *The American College Dictionary*. New York: Wise & Co., 1952.

slithering [17, 57, 84]. However, in the headlong rush to develop new techniques for generating motion, some very basic problems have been left relatively untouched. In particular, many motion algorithms accept starting and ending positions or postures for geometric figures, and possibly intermediate postures as well, and from these generate very impressive motion sequences. But very little current research addresses the fundamental problem of placing the figures in these postures to start with. With complex figure models, particularly human figures, this can be a major ordeal.

Interactive postural control is the process of interactively pushing, poking, and twisting parts of a figure for the express purpose of getting it into a desired posture. This notion of postural control has applications very far beyond key positioning for animation systems. The principal practical motivation for this research is to develop an effective software tool for human factors analysis, and many types of analyses, particularly involving reach, fit and visibility, can be performed quite well in terms of static postures alone.

This thesis involves motion, but not animation. The goal of this research is to develop a system that allows us to interact with computer models of animate geometric figures in ways that suggest how we interact with real people, particularly how we describe human postures. This means that we will be able ask the figures themselves to interact with *their* simulated environment, and they will *behave* in some sense like real people. They will respond to our inputs in a realistic manner, and the vocabulary with which we can communicate with the figures will be rich enough and intuitive enough to easily describe a wide range of postures and postural adjustments.

1.1.1 The Human Factors Problem

Traditionally, human factors engineers analyze the design of a prototype workspace by building a mock-up and using real people to perform sample tasks and report observations about the design. This is limiting for several reasons. Jerry Duncan, a human factors engineer at Deere & Company, says that once a design has progressed to the stage at which there is sufficient information for a model builder to construct the mock-up, there is usually so much inertia to the design that radical changes are difficult to incorporate due to cost and time considerations [24]. The goal of computer-simulated human factors analysis is not to replace the mock-up process altogether, but to incorporate the analysis into the early design stages so that designers can eliminate most problems before building the mock-ups.

The problems in human factors analysis that can benefit from interactive postural control of human figure models involve such geometric aspects as reach, fit, and visibility in a workspace. For example, does a human body of certain dimensions adequately fit into a given workspace? Will it fit through doors or hatches? How will certain changes in the workspace design affect the fit? What can a person see from a particular vantage point in a workspace? In a given posture, can he or she see certain key elements? The answers to these questions will either verify that the design is good or lead to changes and improvements early in the design process.

For these kinds of questions, static analysis really is more important than motion simulation. A human factors engineer is not interested in generating a detailed animation of a human subject turning around in the seat of a tractor just to see what is visible out the back window. The important element of the analysis is the final posture itself, not the motion that the figure goes through in arriving at the posture. However, if the movement of a simulated human figure in a human factors analysis system suggests how a real person moves, and the controls that the system gives over the figures suggest how we would instruct a real person to move, then the user can begin to imagine that the virtual human figure is real and can become engrossed in the process of interacting with it. No existing human factors system provides this level of interaction.

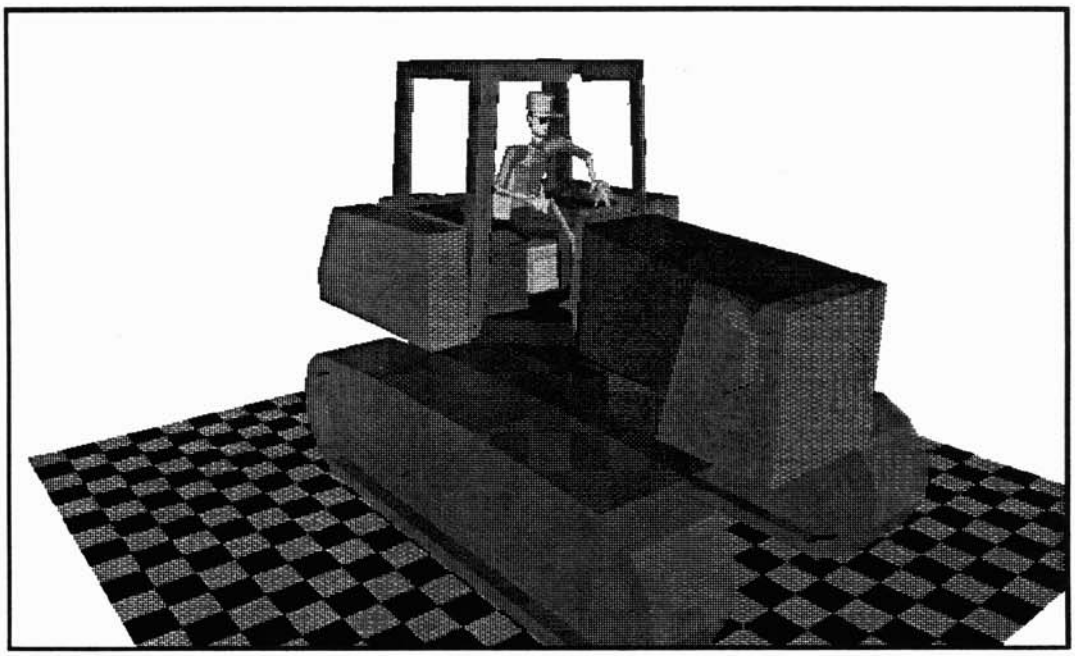


Figure 1.1: Human Factors Analysis of an Off-Road Vehicle

1.2 Manipulation, Animation, and Simulation

There are important distinctions between *manipulation*, *animation*, and *simulation*. Geometric manipulation is the process of interactive scene composition, or the interactive specification of positions and postures for geometric figures, usually on a trial and error basis. Manipulation usually involves movement of the figures, but the movement serves to assist in the control process and is generally not worth saving as a memorable motion sequence. Rather, the purpose of manipulation is to get the figures into a desired static posture, although the posture need not remain static afterwards. Manipulation is inherently real-time: objects move as a direct response to the actions of the user. In short, interactive manipulation is not necessarily choreography.

Animation, on the other hand, is choreography. In computer animation, the goal is to describe motion, and the animator usually knows what motion he or she wants before beginning the animation process. Of course, experimentation may lead to revisions, like an illustrator who erases lines in a drawing, but the computer does not serve so much to answer questions as to obey orders. Animators measure the success of a computer animation system in terms of how well it serves as a medium for expressing ideas.

Simulation is automated animation, and the concern is again with motion. The system generates the motion based on some kind of input from the user ahead of time. The input usually consists of objectives and rules for making decisions, and it is generally less specific than with animation. The user knows less about what motion should result. The job of the simulator is to predict what would happen under certain circumstances and inform the user of the results. Sometimes simulation can generate animation, as in the case of the animation of physics and natural phenomena.

Animation and simulation have been studied extensively, but manipulation of articulated figures has not received the attention it deserves. Volumes of research discuss animation techniques and simulation algorithms, but most research directed at interactive manipulation deals either with the low-level input mechanisms of describing 3D translations and rotations, or with the numerical issues of real-time dynamics. One of my main objectives is to show that interactive manipulation is a worthy problem, and that for purposes of analysis, manipulation can often serve as a very good substitute for motion and animation.

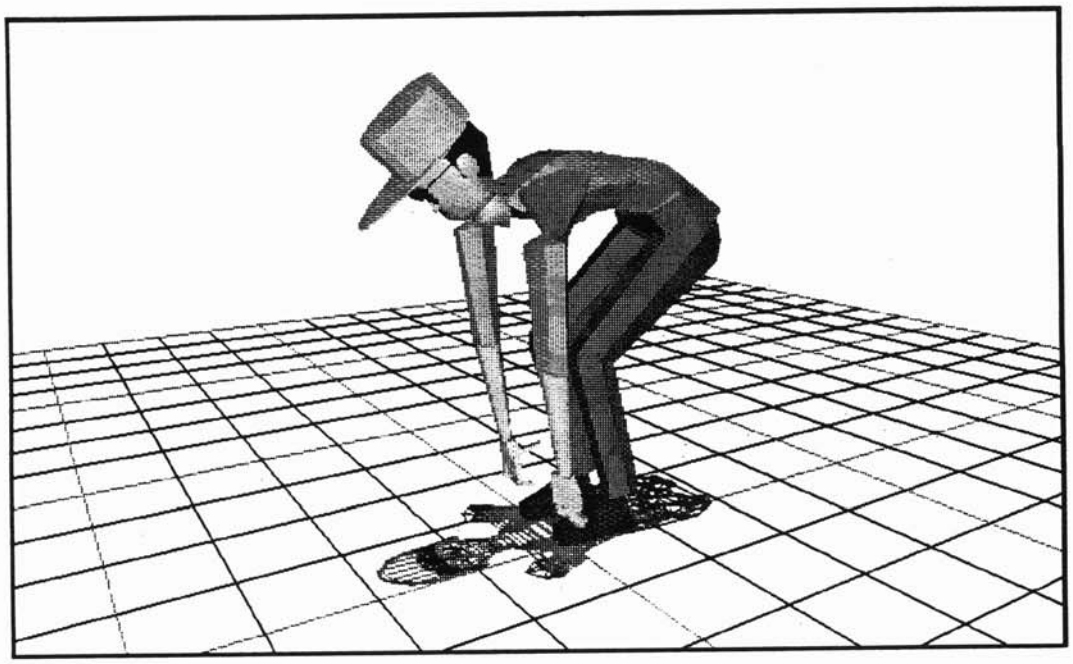


Figure 1.2: Is it the Motion or the Posture?

For example, consider the task of bending a figure over to touch its toes. Is the bending motion important, or is it just the final posture that is critical? In animation, it's the motion: the motion must look realistic. In simulation, the motion must *be* realistic. In manipulation, the finer points of the posture are critical. Is the figure balanced? How bent are the knees? Where is the head pointed? How are the feet oriented? The motion through which the manipulation system sends the figure is not important in itself. It serves only to assist the user in arriving at the posture. I have often thought that if having the user toss salt over his shoulder and bite on a lemon would help make a system easier to use, then it should be a feature. But of course, it won't. Motion, on the other hand, can help. What I mean is this: motion in a manipulation system is not there for it's own sake, it is there because it serves a purpose.

1.3 A Statement of the Problem

The goal of this work is to develop a system for interactive postural control of articulated geometric figures, particularly human figures. The emphasis is on real-time interactive manipulation. The ele-

ments of the interaction techniques form a vocabulary for describing postures and postural adjustments. The vocabulary is not a spoken or written one; rather, it includes verbs acted out by the user through the movement of input devices. The vocabulary includes some elements that allow the user to directly manipulate parts of a figure, and it includes other elements that are called *behaviors*. The behaviors characterize the figure's response to active manipulation and capture some of the coordinated characteristics of how animate geometric figures react to stimuli.

A good system for interactive postural control should make it very easy to describe simple postures and postural adjustments. But it should also only be moderately difficult to describe very complex postures. The difficult aspect of controlling human figure models is that the human body is highly coordinated. Seldom does one part of the body move without affecting the body as a whole. There are a multitude of ways of satisfying any postural instruction. For example, imagine that you are standing with your feet together, and your dance instructor tells you step slightly to the right with your right foot. What happened to your center of gravity? Did it move? Possibly, or possibly not. Both are reasonable responses, depending upon what the instructor meant to convey but left unexpressed. Any system of postural control must take into account the variability of the responses to postural instructions. A *good* system will do so without making the simple instructions cumbersome.

A subgoal of this work is to develop techniques for interaction that use only commonly available technologies, without relying on special purpose hardware such as virtual reality systems with head mounted displays and six degree of freedom input devices. These techniques should work well with the graphics facilities and input devices commonly available on computer graphics workstations.

1.4 The Approach

The approach I advocate here is interactive and graphical. Eric Bier describes "interactivity" as a measure of how well a system takes advantage of "the ability of the computer to draw pictures" [10]. I try to take this one step further and take advantage of the computer's ability to simulate movement. Although the goal here is not choreographed animation, movement can play an important role in the postural control process. As I mentioned before, if the movement of the figure suggests how a real person moves, and the controls that the system gives over the figures suggest how we would instruct a real person to move, then the user can begin to imagine that the virtual human figure is real and can become engrossed in the process of interacting with it. Interactive postural control is a trial and error process: poke at the figure and see how it responds, and continue doing so until the posture is just right. Interactive graphics gives the process a "what you see is what you get" feel.

There are three major components to this work. The first component is a real-time 3D direct manipulation technique that allows the user to intuitively translate and rotate "handles" on objects. The second component is an inverse kinematics algorithm that uses the notion of *constraints*, or desired geometric relationships, to control postures of articulated figures. The kinematic constraints are handles by which to control parts of the figure. Inverse kinematics has been used extensively in robotics, and in computer animation for the generation of motion, but it has never been applied with any rigor to the postural control of highly redundant articulated figures. The third component is a system of behaviors for controlling the inverse kinematics algorithm in a coordinated way.

This type of control is like a marionette puppet controlled by strings, except that the strings need not hang vertically, and they can twist and push as well as pull. The inverse kinematics algorithm provides the computational power behind the strings. But how many strings do we need? Where should we attach them? How do we push, pull, and twist on the strings to get the figure to move as we want? For

this, we need the notion of a *behavior.*

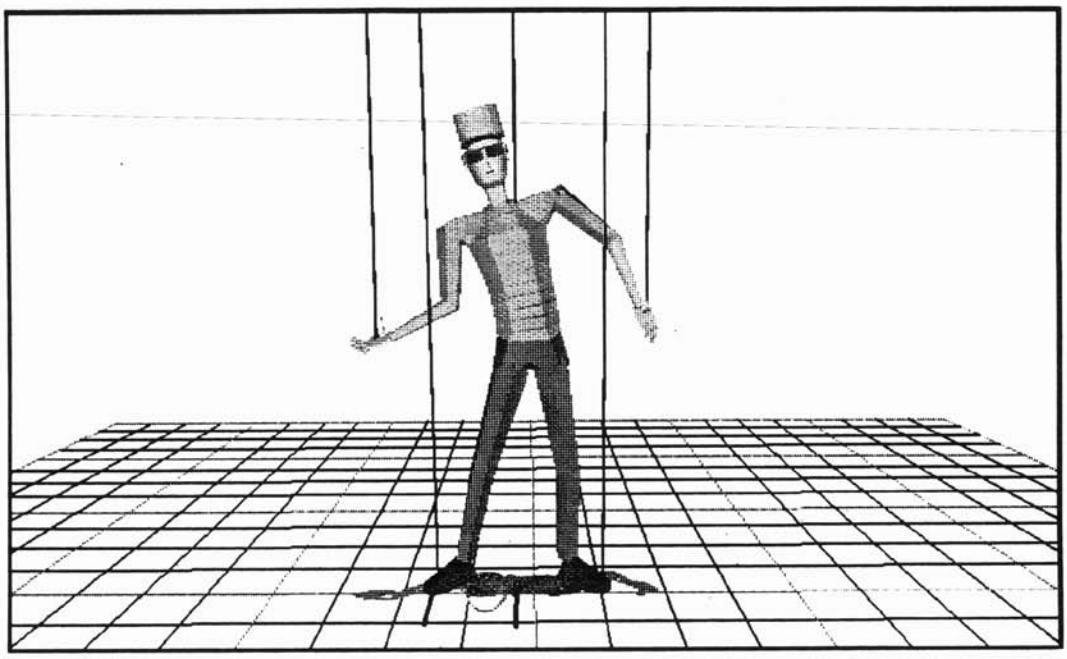


Figure 1.3: A Marionette Puppet

What exactly is a behavior? It's not the psychological or emotional connotations of the word, such as light-heartedness or sadness. Mostly, behaviors are the stimulus-response characteristics and actions and reactions that a figure makes towards its environment. Behaviors also provide coordination between the parts of the figure, so that when one part of a figure moves, the body reacts as a whole. In the puppet analogy, the behaviors coordinate the effects of the strings. The idea of a behavior has been applied to animation, but it has never really been applied to postural control.

Probably the most important behavior, and the one demanding the most coordination, is balance. The need to remain balanced dictates much of the behavior of a human figure. For example, when a standing figure bends over, its hips should shift backwards to compensate. This is truly a natural consequence, one that should not really require much intervention on the part of the user. Probably, your dance instructor will not seriously say, "Bend forward, but don't fall over." But because the center of mass of a figure is such a global property, it is very difficult to control with traditional manipulation techniques. Placing a marionette string on the center of mass and having it behave in a reasonable way encapsulates many of the subtle aspects of human postural control.

In the context of postural control, a behavior serves to predict what the user wants out of a simple instruction. Consider the example above of your dance instructor and the small step to the right. Probably what should happen is that your weight should shift along with your foot. If the system predicts this, then it's making a reasonable assumption, even if this is not what your instructor meant. If you did this in your dance class, your instructor might respond by saying, "No, please keep your weight on your left foot." The approach I advocate here is to keep the set of postural control instructions simple and intuitive but provide a broad range of behavioral parameters to fine tune the responses.

This kind of behavioral prediction would be much different if the intention were animation or simulation instead of postural manipulation. In animation and simulation, the choice of response must be "correct" or critics may complain that the movement is not valid. For postural control, there is no issue of validity outside of concerns like joint limits, balance, and strength limitations. The only real issue is expediency. If the system can anticipate your intentions, then it can possibly save you additional steps in the postural specification process. If it anticipates incorrectly, it doesn't give you bad information. It only forces you to take the additional step. This is the role of the behaviors.

The advantage of the system of behaviors is that it allows you to manipulate the posture of a figure locally, but have the effects of the manipulation telegraphed to other parts of the body in a reasonable way. This local control means that you can describe a posture incrementally, one part of the body at a time, and make subsequent adjustments as necessary, without disturbing the overall posture. For example, you may first position the legs of a figure by moving the legs into place. If you then bend the figure over, its legs may reposition themselves slightly to maintain balance, but the feet will remain as they are.

The techniques developed here are implemented in *Jack*™[†], a system for modeling, manipulating, animating, and analyzing humans and other articulated figures. *Jack* is a large software system that has many features unrelated to this work, but for the purpose of this discussion, *Jack* is the embodiment of these ideas.

1.5 Overview

This thesis describes the postural control mechanisms in a bottom-up fashion, starting with the 3D direct manipulation input technique, then the positioning system that uses inverse kinematics, then the behavioral primitives and the interface for manipulating human figures. Actually, this progression represents much the way in which these ideas, and the software implementation of them, has evolved over time. Fred Brooks, Jr. once wittily described nature of the evolution of such a system using a inside computer graphics joke, referring to his evolving system for architectural visualization:

> We are approaching our dream system in steps, following a sort of Bresenham's algorithm[‡] in which each successive version makes a step improvement in whatever aspect seems to have the widest discrepancy between where we are and the ideal system [15].

Chapter 2 reviews other areas of research that are related to this work, mostly animation and dynamic simulation systems. I also discuss existing human modeling systems used for human factors analysis. Chapter 3 describes the 3D direct manipulation technique. This mechanism provides the basic primitives for manipulating three dimensional translational and rotational quantities with a three-button mouse. Chapter 4 describes the notion of a kinematic *constraint*, and the inverse kinematics algorithm that positions the figures. The inverse kinematics primitives provide the basic elements through which the behavior of the figure can be specified. Chapter 5 describes the vocabulary for interacting with human figures, including the manipulation commands and the system for behavioral controls. Finally, Chapter 6 draws conclusions on this work and describes its contribution to the state of the art.

[†] *Jack* is a trademark of the University of Pennsylvania.

[‡] Bresenham's algorithm is a technique for drawing lines on a raster grid that keeps track of the error between the actual line and the nearest pixel.

The Appendices contain some important details not central to the ideas of this thesis, but important none the less. Appendix A describes *Jack*'s representation for articulated figures, called *peabody*. The cognitive model that *peabody* promotes is an important part of the interaction techniques. The discussion of *peabody* is mostly an argument against the use of Denevit & Hartenberg notation [23], and an argument for a more general representation than most animation systems provide. It is also an argument for why *peabody* represents geometric figures as articulated mechanisms instead of taking the physically-based modeling approach. Appendix B describes some of the implementation issues for the direct manipulation operator, as described in Chapter 3. Appendix C describes the interactive technique for controlling the viewpoint in *Jack*.

The remainder of this chapter gives a preview of the contributions of this thesis, and then gives some background information on the implementation. This implementation information provides the context for the discussion of the implementation in the remaining chapters.

Many of the ideas of this thesis have been published over the last three years. In particular, "Jack: A Toolkit for Manipulating Articulated Figures" [67] describes the basic 3D direct manipulation technique. "Software Systems for Modeling Articulated Figures" [68] describes the *Jack* system and its representation for articulated figures. "Interactive Real-time Articulated Figure Manipulation Using Multiple Kinematic Constraints" [69] describes the interactive manipulation system built on top of the inverse kinematics algorithm. "Interactive Behaviors for Bipedal Articulated Figures" [66] describes the basic idea of behavioral control for articulated figures. In addition, the *Jack 5 User's Guide* [64] describes the details of the interactive commands in *Jack*, and the programmer's documentation *Programming with Jack* [65] describes the implementation in much greater detail.

1.6 Contributions

This work is unique in its emphasis on interactive postural control of articulated figures, as opposed to many other systems that have focused primarily on the problem of motion generation and animation. The significant contributions include:

- A novel technique for 3D direct manipulation that functions well in an integrated geometric environment and integrates control of the virtual camera into the manipulation process.
- A technique for applying inverse kinematics to the positioning of highly redundant articulated figures.
- A technique for modeling balance in articulated figures using inverse kinematics.
- The notion of kinematic behaviors for postural control;
- A rich vocabulary for describing complex postures interactively and graphically.

1.7 Implementation

The representation for geometric objects in *Jack* is called *peabody*. *Peabody* uses the term *environment* to refer to the entire world of geometric objects. The environment consists of *figures*, each of which is a collection of *segments*. The segments are the basic building blocks of the environment. It represents a single physical object or part, which has shape and mass but no movable components. *Joints* connect

segments through attachment frames called *sites*. A site is a local coordinate frame relative to the coordinate frame of its segment. Each segment can have several sites. Joints connect sites on different segments within the same figure. Sites need not lie on the surface of a segment. A site is a coordinate frame that has an orientation as well as a position. Each site has a *location* that is the homogeneous transform that describes its placement relative to the base coordinate frame of its segment. Joints may have several *degrees of freedom*, which are rotational and translational axes. Each axis and its corresponding angle form a single rotation or translation, and the product of the transform at each degree of freedom defines the transform across the joint, defining the placement of the sites, and thus the segments, that the joint connects.

Jack's user interface is well integrated. While this is not really a central issue, it is important to note that the interface provides very little impediment to the implementation of the interaction mechanisms that *are* are the central issues here. There are many, many issues concerning the implementation of *Jack* that are not relevant to this discussion. Many of these issues involve choices in system design inherent in any large implementation. Some of the choices made with *Jack* are rather arbitrary. For example, *Jack*'s pop-up of menus may or may not be more effective than screen menus. In many cases, alternative system architectures would serve equally well, or even better. In its favor, the chosen architecture has proved itself to be robust and powerful over several years of use.

Jack takes a *verb-object* approach to interaction: the user first selects an action and then selects the objects upon which to act. This is different from some systems, particularly some Macintosh† applications such as MacDraw [53] that operates with a notion of the "current" object. In *Jack*, the user selects commands from the keyboard or pop-up menus. The keyboard/pop-up menu interface is well integrated: keyboard entry and menu selection can be mixed freely without transferring back and forth between modes.

Most commands in *Jack* take *arguments*. The arguments are the objects and data that the command operates on. *Jack* prompts the user to select the arguments after initiating the command. The user picks geometric objects interactively with the mouse and enters numeric and string arguments from the keyboard, or selects them through the pop-up menus.

Jack is *modal* in the sense that some commands place the system in a mode where the keys and mouse buttons perform specialized functions. This is particularly true of the 3D interaction commands that allow the user to move 3D objects. The user terminates the mode by hitting a special key, usually the **ESCAPE** key.

Jack is a window-driven system as well. It has two standard windows for information and control, a status window that also serves as an "edit buffer" for entering values from the keyboard, and a message window in which it logs informational messages for the user. *Jack* draws the geometric figures in graphics windows. By default, a single, large graphics window covers the entire screen except for the status and message windows. The user can create any number of additional windows, each of which provides a view into the world of geometric objects. The view in each window can be manipulated independently using the techniques described in the appendix in Section C.2. Objects can be manipulated in any window. Figure 1.4 shows the *Jack* screen.

†Macintosh is a registered trademark of Apple Computer.

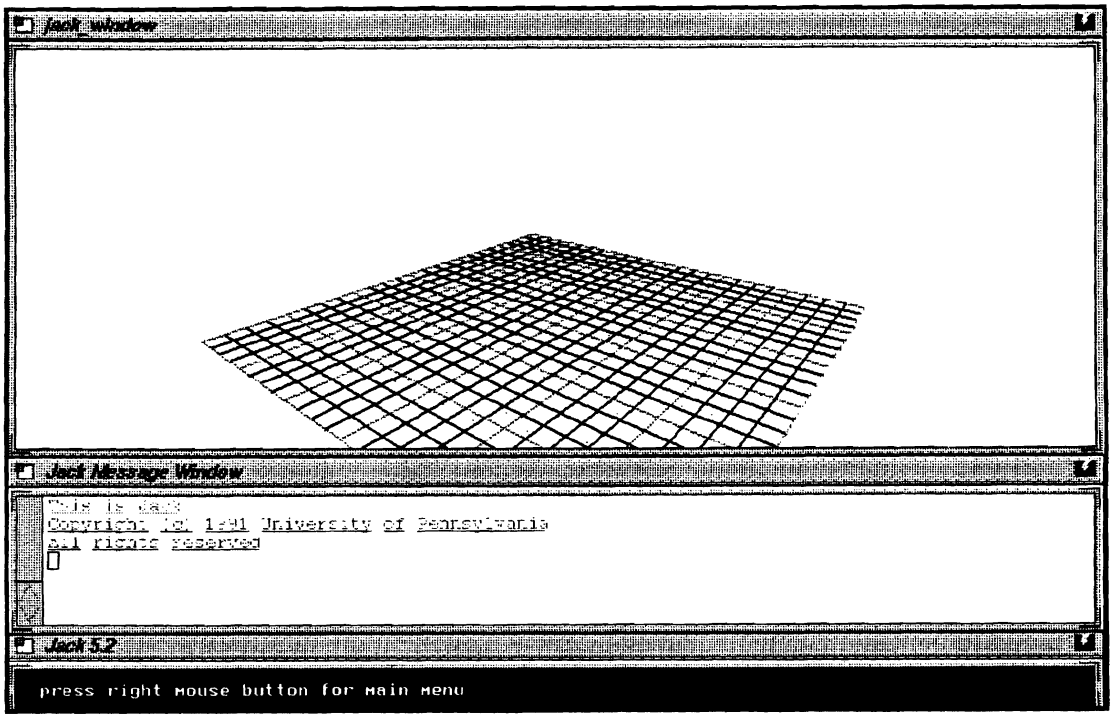


Figure 1.4: The *Jack* Screen

Chapter 2

Related Work

Very little research directly addresses the problem of interactive manipulation and postural control of articulated figures, although there are several areas of research that address it indirectly. This chapter describes these areas. The purpose of this chapter is to place my work in the proper context and to enumerate how this thesis contributes to the state of the art. The relevant work described here falls into several categories:

- interactive keyframe animation systems;
- dynamic simulation systems, including:
  - systems that focus on human figure animation;
  - general purpose dynamic simulation systems;
  - physically-based modeling systems;
- goal-directed human motion systems;
- inverse kinematics approaches to human figure movement;

- man-modeling systems for human factors analysis.

Dynamic simulation deserves special attention here, because it is such a popular technique, because it is such a conceptually pleasing approach, and because I believe it has some severe limitations when applied to interactive manipulation for postural control. Most of the systems described here are motion generation and animation systems, although some address the issues of interactive manipulation. The process of postural control has undeniably static characteristics, and so dynamics may not be necessary at all. The purpose of this discussion is to describe the capabilities of existing dynamic simulation systems and to illustrate why my system of behaviors and manipulation primitives dispenses with a formal approach to dynamics altogether.

2.1 Keyframe Animation Systems

Some of the most impressive works of computer animation were created by very talented artists using very simple computational tools. One of the simplest approaches is the keyframe animation system, such as TWIXT [36], developed by the Computer Graphics Research Group at Ohio State University, and BBOP [79, 80], developed and used at the New York Institute of Technology. Other systems described in the literature include GRAMPS [61], which was designed as a molecular modeling system. These systems allow an animator to position an object model, possibly an articulated, hierarchical model, at key poses in time, and the animation system interpolates between these poses, usually performing the interpolation of articulated models in joint space. The postures need not include all body parts; BBOP, in particular, is a *track-based* system so it can "keyframe" different parts of a figure independently.

With a keyframe animation system, the animator begins by specifying the keyframes, and this involves positioning the figures in a sequence of static postures. Different systems provide different techniques for doing this positioning, but few provide any sophisticated mechanism for manipulating a complex, coordinated figure. Interactive keyframe animation systems provide ways of adjusting the parameters of the object models interactively, either through sliders or through direct manipulation. BBOP, for example, lets the user manipulate objects with a three-axis joy stick. It provides no constraints or rules governing the model, so "when manipulating a human body model, for instance, limbs can disconnect from the body structure and joints can be bent at unrealistic angles" [80].

None of the systems just described provide the ability to manipulate more than a few degrees of freedom at a time, and usually not in any coordinated fashion. The GRAMPS system does provide a macro facility that can tie variables to input devices such as dials. The macro can reference variables in several places, so it is possible to have a single input device control several degrees of freedom in a pre-programmed manner. The examples given to illustrate the macro facility include a leg lift in which both the hip and knee rotate together, and a bend of the torso in which each vertebrae bend in unison. This is an improvement, but it still only provides a preprogrammed type of coordination. There are some exceptions, in systems that provide inverse kinematics capabilities, such as Michael Girard's PODA system [33, 34, 35]. This system allows the user to manipulate limbs of an articulated figure by dragging the end of the limb. However, Girard's emphasis is clearly on the motion generation problem. None of his work describes the manipulation features in detail.

One keyframe animation system that particularly addresses human figure animation is LifeForms [52]. Through a cut-and-paste interface, the user can assemble a sequence of postures that are interpolated as keyframes. The user can define postures in the traditional way, by adjusting the joints in the body one by one, using a combination of sliders and virtual spheres. There is no form of coordination of movement.

2.2 Dynamic Simulation

Because keyframe animation systems leave so much of the motion control process up to the user, it is very difficult to generate realistic-looking motion. Therefore, many researchers have attempted to generate motion automatically by applying the laws of physics and simulating the effect of forces and torques acting on the geometric objects. This technique first derives the equations of motion for the objects, then solves the equations for accelerations given the forces and torques acting on the objects, then integrates the accelerations to find velocities and positions. In the case of forward dynamics, the inputs to the system are the forces and torques acting on the body. In the case of inverse dynamics, the inputs are desired accelerations, and the system solves for the necessary forces and torques to generate the accelerations.

There are several formulations of the equations of motion, each offering different advantages of speed, robustness, and ease of implementation. Mark Green discusses several of the methods for solving the ODE's of motion, including Euler's method, the Runge-Kutta method, the multi-step method, and predictor-corrector methods [37]. Jane Wilhelms uses the Gibbs-Appell formulation [87]. Armstrong and Green [2] describes a recursive dynamics formulation for tree-structured mechanisms that operates in near-real-time. Witkin and Kass [92] phrase the dynamics problem as a boundary value problem by specifying values at the beginning and ending times along with an objective function, such as energy minimization. The algorithm then solves for the complete equations of motion at once.

2.2.1 Dynamics for Human Figure Animation

Much of the research in human figure animation has attempted to generate motion via dynamic simulation. Much of this work has been done by Jane Wilhelms [85, 86, 87, 88]. Armstrong and Green describe a recursive formulation of the dynamics equations and apply it to human figure models [2].

Wilhelms's *Virya* system is an interactive force and torque editor that allows the user to specify forces as functions of time [85, 87]. The forces can be global forces such as collisions and gravity, or joint spring forces that could model limits. Wilhelms states that "it was immediately obvious that controlling the motion by designing force functions was not intuitive" [85]. One reason is that there is no easy way to predict the interplay between the forces.

Wilhelms's *Kaya* system provides collision detection and response, elasticity, friction, joint limits, and joint dampening [89]. Wilhelms alludes to world-space goals implemented by spatial constraints and spring forces, but gives few details about how her system implements them. The principal articulated figure example she describes involves a human figure passively falling onto a pile of rocks. This example involves no special control on the part of the user. Wilhelms also alludes to interactive manipulation capabilities but does not describe them in detail.

Forsey and Wilhelms's *Manikin* system is an interactive physical simulator that allows the user to specify forces and torques in real time [31, 85]. The user can also specify goal positions and have the system generate appropriate forces and torques to maintain the goal. It achieves much of its interactive speed by limiting the dynamics to subsets of the body.

Forsey and Wilhelms [31] present a very good motivation for the problem of interactive manipulation. However, their recursive dynamics formulation requires that they treat all joints as 3 DOF ball joints, and this complicates the matter of kinematic constraints. *Manikin* models constraints by attaching extremely heavy masses to parts of the figure ("consider this operation as analogous to attaching a segment the mass of the Queen Elizabeth II to the figure"). The mechanism for pushing and pulling

on the figure uses inverse dynamics applied to the extra point masses. This technique makes it difficult to decouple control of position and orientation, because a massive segment necessarily has enormous rotational inertia. The approach also has problems when there are many goals. A figure with more than about 10 ocean liners attachment would probably be very difficult to manipulate.

2.2.2 Control of Dynamic Simulation

Each of the systems described above implement dynamic simulation algorithms with different capabilities. Each of these systems has problems with controlling the simulation, and none pose an adequate solution to the problem of specifying the necessary forces and torques to cause a desired motion. The examples given to illustrate each system involve relatively simplistic motions chosen to minimize the need for complex controlling forces, motions such human bodies falling on a pile of rocks, or an arm swinging in space.

Isaacs and Cohen [41] describe a technique for incorporating simple kinematic constraints into the dynamics equations, and they then extend the technique to constraints on several degrees of freedom simultaneously [42]. These "complex kinematic constraints" can simulate world-space goals for reference points on the figure. Their system runs off-line, and it is relatively slow. They present an example of a marionette puppet figure with 77 degrees of freedom, and they quote a simulation time of 69 CPU minutes per second of animation. They conclude by saying that "user interfaces need to be developed to allow easy and intuitive specification of complex kinematic constraints" [42].

Some systems have suggested better mechanisms of control. Mark Green suggests using motion verbs for specifying an animation [37]. For human figure motion, the verbs could specify reaches, walks, or dance steps. He suggests that the motion verbs could produce necessary forces and torques to drive the simulation, but he does not suggest how this could be done. Furthermore, he describes an appealing structure for modeling the behavior of the figure based on "behavioral processes," but he does not describe how these processes could be implemented. He simply states that there could be behavioral processes for "avoiding collisions, exploring the current environment, finding food, or dancing."

Armstrong, Green, and Lake describe the notion of a "motion process" [3]. Local motion processes act on individual joints. This is a dynamic version of Zeltzer's kinematic local motion processes [93]. Armstrong, Green, and Lake's motion processes can allow the joint to swing freely, to dampen motion, to maintain a particular angle, or to move to a particular angle. The global motion processes include gravity and balance. To model balance, they simply measure the difference between the positions of the upper and lower body and generate a restorative force to apply to one or more of the limbs. They present no other details of how balance can be controlled or how an animator might be able to specify different types of balance.

Lee et al [51] developed a partially interactive version of the same basic optimization apprach, called *strength guided motion*. It used a set of heuristic strategies to guide the optimization based on strength equations for human reaching tasks.

2.2.3 Dynamic Simulation for Motion Interpolation

Witkin and Kass [92] note that dynamic simulation has traditionally been an initial value problem: arrange the figure in a particular posture, apply forces to it, and watch it move. However, traditional animation largely concerns where objects end up as well as where they start out. They present a technique, called *spacetime constraints*, that solves for the motion of an object between a starting and

ending configuration subject to some global constraint such as minimum expended energy. They solve for the character's motion over the entire time interval at once, rather than by progressing through time step by step. This means that the effects of the constraints propagate both forward and backward in time over the interval, enabling the character to anticipate movements in a way not possible with a forward simulation. Their example involves a jumping Luxo lamp, jumping between a starting and ending location. The crouching in anticipation of the jump and the extension in anticipation of landing come directly out of minimizing the power consumed by the muscles, that is, its joints.

Michael Girard describes similar motion control algorithms based on optimization techniques that consider minimizing jerk and minimizing expended energy [33]. Given an end effector path with timing information, he uses a dynamic programming technique to solve for the optimal joint-space velocities that will move the end effector along the path. He describes extensions to the PODA system that allows an animator to compose limb and body trajectories [34, 35]. The trajectories are specified in terms of key postures that are splined together into a smooth path.

2.2.4 Physically-Based Modeling

Alan Barr advances the notion of "teleological modeling" [6], the idea that the modeling process should include the motion and behavioral characteristics of objects as well as their surface geometry and light reflectance properties. This is the basic idea behind physically-based modeling, that is, to include the equations of motion into the definition of the object models.

Barzel and Barr describe a modeling system based on dynamic constraints [7], emphasizing the "equivalence of modeling and animation." They treat objects as primitive Newtonian bodies, and they model the connections between them through constraints. The constraints generate necessary forces to achieve the desired connections, using inverse dynamics. They provide several types of connections, including point-to-nail, point-to-point, point-to-path, and orientation. They mention only briefly that their interface allows the user to specify time lines of events that turn constraints and forces on and off, but they give no details. Their techniques are interactive only for very simple environments of a few objects.

The examples that Barzel and Barr use to illustrate the system involve movements that need very little control. "Pandora's Chain" involves a chain hanging from a hook that grabs the handle of a trap-door and pulls it open. The "Linking Chain Between Two Towers" involves a chain whose links automatically assemble themselves and then swing naturally between the tops of two towers composed of sticks held together by constraints. Both of these motion sequences required little interaction from the user except for specifying the initial conditions, that is, the arrangement of the primitives and the definition of the constraints. In these examples, there are few natural behaviors to compare against except for gravity. The swinging motion is natural, but it's difficult to assess a natural motion for the tower.

In spite of this, Barzel and Barr's system is appealing because of its emphasis on modeling and its use of motion to assist in the modeling process. They do not intend it to be primarily an animation system. The motion it generates enhances the modeling process. The extent to which it mimics the real world only serves to make the modeling process more intuitive. This is a very appealing notion, and the only drawback of this work is that it only addresses half of the problem: this modeling system leaves the user with a very primitive set of controls, and it fails to give the user much intuition about how to use the primitives to accomplish useful work.

Witkin, Gleicher, and Welch [91] describe a formulation of constrained dynamics suited to interactive

manipulation. They describe an application involving tinkertoys that can be connected together with constraints. Their work has a flavor similar to Barzel and Barr: they do not address the issue of specifying and controlling the constraints in order to accomplish useful work.

Witkin, Fleischer, and Barr describe the use of energy functions for assembling models and controlling their movement [90]. The energy functions operate on the parameter space of the models and have zeros at points that satisfy the constraints, although the functions do not measure the actual mechanical energy of the system. The energy functions sum to an energy field, and the models move in parameter space along the gradient of this field to reach the minimum energy state. This is a powerful way of defining geometric relationships. I discuss this approach in greater detail in Chapter 4.

2.2.5 A Discussion of Dynamic Simulation

When motion is an issue, dynamics is obviously important, but for interactive postural control, it is not clear that dynamics is necessary. For example, Lee's strength guided motion technique generates dynamically appealing motions without using dynamics. For postural control and interactive manipulation, expediency is the most important criteria. How easily and quickly can a user describe postures and postural adjustments? None of the dynamics systems described here demonstrate sufficient speed and versatility to satisfy this demand. Forsey and Wilhelms's *Manikin* system comes close, but it too falls short. The shortcomings are conceptual as well as computational. No one has yet developed any kind of efficient, complete, and natural language for postures and movements that can be translated into the inputs of a dynamic simulation system. Green's system of motion verbs seems ideal, but it is by no means clear how to implement such a thing. Jugal Kalita [44] shows that many useful verbs need not involve dynamics.

In Chapter 4, I advocate the use of inverse kinematics for figure positioning. This system dispenses with most notions of masses and inertias, except for center of mass calculations which model balance. The advantages of this technique are speed and versatility. Because the inverse kinematics algorithm is computationally very fast, it is possible to simulate through kinematic descriptions many of the features which make dynamics appealing. This is discussed in greater detail in Chapter 4.

2.3 Goal-Directed Motion and Locomotion

Bruderlin and Calvert [17] describe a goal-directed system for generating human walking. The system parametrizes the walks in terms of three principal locomotion variables, which are forward velocity, step length, and step frequency. In addition, there are up to 28 other locomotion attributes, such as the width of the feet and the tilt of the pelvis. It uses a dynamic model to generate the motion of the legs. This system generates very realistic-looking human locomotion, but its techniques do not apply to other kinds of movements, even turning.

Michael Girard describes the use of an inverse kinematics algorithm for the simulation of animal locomotion in the animation system PODA [34, 35]. The gait cycle of the legs determines a world-space path for the foot, given the path along which the animal is walking or running. The inverse kinematics algorithm positions the leg so that the foot follows the computed trajectory. The motion path can be curved, but the banking of the body is precomputed. The gait generator has several parameters so it can generate several qualities of walks, but this is the only sophisticated way of specifying the foot or limb trajectories. The system is non-interactive in the sense that it computes the motion based on pre-specified inputs.

2.4 Inverse Kinematics as a Positioning Technique

Badler, Manoochehri, and Walters describe an articulated figure positioning system based on inverse kinematics [5]. The algorithm uses *balanced reach trees*, and is capable of handling multiple positional constraints, but it does not handle orientation constraints, and it doesn't operate in real time. It also yields a solution that is heavily biased by the application order of the constraints.

Girard briefly describes a feature for interactively positioned limbs using inverse kinematics in the PODA system [35, 34]. The user can rotate the joints independently, or drag the position of the end effector in space, or rotate the joints with the location of the end effector held in place. PODA catalogues the limb postures in tables for interpolation. This interactive limb positioning is clearly not a major emphasis of the PODA system.

2.5 Man-Modeling Systems for Human Factors Analysis

There are many systems for performing human factors analysis using computers and computer graphics. The ones of most interest here concentrate on the geometric elements of analysis, rather than the cognitive aspects. Karl Kroemer surveys human-machine interface models as of 1985 [50]. These include BOEMAN, developed by Boeing Company, an early system that provided the conceptual basis for many subsequent models; the CAR (Crew Assessment of Reach) model; COMBIMAN (COMputerized BIomechanical MAN), and SAMMIE (System for Aiding Man-Machine Interaction Evaluation). Also, the SAFEWORK system was developed at the Ecole Polytechnique de Montréal [32], Crew Chief was developed for the US Air Force [25], and the TEMPUS system was developed at the University of Pennsylvania [4]. TEMPUS is the predecessor of this work, and it did not have the advantage of highly interactive graphics.

Each system provides an articulated figure with various degrees of detail in the limbs and torso. Most provide a means of manipulating the figure at the joint level, with additional operations to perform reaches and reach analysis. TEMPUS uses the reach algorithm developed by James Korein [49]. Each of these systems is command driven, so the user interacts with the system by executing discrete commands. The reach features are provided as commands, so that the user first selects the parameters of the reach, and then the system computes joint angles for the arm using an inverse kinematics algorithm that places the hand at the desired reach point. Patricia Rothwell [74] surveys the techniques for modeling human figures with CAD technology, and she notes that few systems provide for multiple simultaneous reaches. Jill Easterly, one of the developers of CrewChief, notes that one particular need of such systems is "automatic task composition", by which the user could avoid specifying a reach task over and over again [25].

SAMMIE is one of the first systems designed to be interactive, that is, to attempt to shorten the cycle between user input and system response [13, 47]. The developers of SAMMIE were limited, however, by the primitive graphics and computing capability of their hardware, and they concluded that "a detailed man-model would take so long to manipulate and display that it could not be used as an interactive tool" [13].

David Beevis [8] notes that most existing man-modeling systems do not have "internal rules governing posture," and so a great degree of user skill is needed to specify complex postures. He also notes that most models are relatively static and "must be worked through a sequence of postures" to analyze a task. He states that in the human factors community this seems largely sufficient, and there is no great demand for dynamic models outside of crash studies.

2.6 Preview of Contributions

This thesis emphasizes interactive, real-time manipulation and postural control of complex articulated figures. Wilhelms, Witkin, Kass, Armstrong, Green, Girard, Bruderlin, Isaacs and Cohen have all been primarily concerned with the generation of movement sequences given pre-specified user inputs [88, 92, 3, 37, 33, 17, 41]. These systems may be interactive only in the sense that they may allow the user to input parameters through an interactive interface. They do not in general allow the user to input parameters as the motion is being generated and performed.

The systems that do use motion during the modeling and postural control process, such as Witkin, Fleischer, and Barr, Barzell and Barr, and Wilhelms [90, 7, 89] do not address the higher level issues of how a user is to compose primitive actions into useful work. In each of these works, the author simply states that the elements of control — constraints or forces — can be interactively manipulated by the user. I advocate the notion of a behavior for controlling and coordinating the manipulation process.

Chapter 3

3D Direct Manipulation

"Those who manipulate this unseen mechanism ... constitute an invisible government which is the true power of our country."

Edward L. Bernays

In recent years, there has been much interest in virtual reality systems because of the need to provide better interaction with simulated 3D environments. These systems combine multi-dimensional input devices with head-mounted stereoscopic displays to give the user the feeling of being truly inside a virtual space [29]. Some such systems also provide tactile and force feedback [16]. Such systems are the ultimate in direct manipulation since they seek to give the user the true feel of manipulating and interacting with virtual objects.

Although virtual reality is very exciting and promising, it has some inherent limitations. In addition to the high cost of the hardware for such systems [29], it is not clear whether such systems will ever be suitable for use for long periods of time on a regular basis, as is needed in the everyday problems of human factors analysis and computer animation. Even as the technology significantly improves, we cannot expect that the existence of this hardware alone will reveal the most effective paradigms for real-time interaction with geometric objects.

The goals of virtual reality serve as good objectives, although it may be possible to achieve some of these goals without relying on advanced hardware. It is worthwhile to develop input techniques that rely only on the commonly available and familiar technologies of the keyboard and mouse. At the very least, these technologies will be the dominant ones for the foreseeable future. It is therefore essential to ensure that they are used as effectively as possible.

3.1 Chapter Overview

This chapter describes *Jack*'s basic 3D input mechanism, which provides a means of translating, rotating, and viewing objects in a 3D environment, all controlled through a 3-button mouse. The basic technique is a general purpose *operator*, so it can be used to input several different types of 3D translational or rotational quantities. This technique is at the heart of *Jack*'s interactive nature. Many commands in *Jack* that require the user to specify geometric information use this operator for their input, so that different commands that perform different functions all use the same basic manipulation interface.

During the process of manipulating an object in 3D, it is absolutely essential to "see what you are doing," so the problem of positioning the viewpoint in the virtual workspace is critical to direct manipulation. *Jack*'s manipulation technique involves an automatic viewing adjustment scheme that helps significantly. Our interactive system has many other features for controlling the viewpoint as well. These are discussed in greater detail in Appendix C.

The main points of this chapter are:

- A new technique for translating objects in 3D with a 2D mouse as input.
- A new technique for rotating objects in 3D with a 2D mouse as input.
- A new technique for automatically positioning the camera to assist in the manipulation process.

The implementation details of the direct manipulation technique are given in Appendix B.

3.2 Direct Manipulation

3D direct manipulation is a technique for controlling positions and orientations of geometric objects in a 3D environment in a non-numerical, visual way. It uses the visual structure as a handle on a geometric object. Direct manipulation techniques derive their input from pointing devices and provide a good correspondence between the movement of the physical device and the resulting movement of the object that the device controls. This is *kinesthetic correspondence*. Much research demonstrates the value of kinesthetically appropriate feedback[10, 14, 75]. An example of this correspondence in a mouse-based translation operation is that if the user moves the mouse to the left, the object moves in such a way that its image on the screen moves to the left as well. The lack of kinesthetic feedback can make a manipulation system very difficult to use, akin to drawing while looking at your hand through a set of inverting mirrors. Providing this correspondence in two dimensions is fairly straightforward, but in three dimensions it is considerably more complicated.

The advantage of the direct manipulation paradigm is that it is intuitive: it should always be clear to the user how to move the input device to cause the object to move in a desired direction. It focuses the user's attention on the object, and gives the user the feeling that he is manipulating the object itself.

3.2.1 Translation

Several techniques have been developed for describing three dimensional transformations with a two dimensional input device such as a mouse or tablet. Nielson and Olson [60] describe a technique for mapping the motion of a two dimensional mouse cursor to three dimensional translations based on the orientation of the projection of a world space coordinate triad onto the screen. This technique uses a one-button mouse, and it compares the 2D displacement of the mouse cursor to the screen projection of the six world coordinate axes and causes a differential movement in world space along the axis whose projection is closest to the mouse movement. For example, if the view is positioned such that the world coordinate x axis points left, then moving the mouse to the left will cause a $+x$ translation. This provides good kinesthetic correspondence, but it has problems if two of the axes project onto the screen close to one another, since it will not be able to distinguish between the two. In other words, it is highly dependent on the view.

3.2.2 Rotation

Rotations are considerably more complex, but several techniques have been developed with varying degrees of success. The most naive technique is to simply use horizontal and vertical mouse movements to control the world space euler angles that define the orientation of an object. This technique provides little kinesthetic feedback because there is no natural correspondence between the movements of the mouse and the rotation of the object. A better approach, described by Chen et al [21], is to make the rotation angles either parallel or perpendicular to the viewing direction. This makes the object rotate relative to the graphics window, providing much greater kinesthetic feedback.

The problem with screen-space transformations is that it is impossible to make movements around either the global or local axes. In an integrated geometric environment, it is more common to move objects relative to either the global or local coordinate frame, rather than along axes aligned with the screen. For example, the simple task of raising an object vertically requires translating along the global y axis. Unless the view in the graphics window is perfectly horizontal, the vertical direction in screen coordinates is not exactly vertical. As another example, the task of moving a hand forward may require moving along an axis aligned with the body, not the screen.

Chen et al [21] also describe a technique originally developed by Evans, Tanner, and Wein [28] known commonly as the *virtual sphere*. This technique simulates the effect of a trackball centered around the objects origins. You "grab" the trackball with the mouse and rotates it much as you would rotate a physical trackball with a single finger. The virtual sphere is an efficient technique for certain operations, as Chen et al verify experimentally. However, since the rotation is not confined to a specific axis, it can be difficult to rotate around a particular axis. It is nearly impossible to make precise rotations around global coordinate axes.

Chen et al describe an experimental comparison between several techniques for 3D rotation. The subjects were asked to rotate a geometric object, in the shape of a house, to align it with a similar object in a random orientation. They were measured for both speed and accuracy. The techniques evaluated included several angle-based rotations with and without kinesthetic correspondence, and the virtual sphere. The studies generally showed that the virtual sphere was the best, out-performing the others in both precision and speed.

The virtual sphere is good at "tumbling" objects, when the path of their rotation is not important. This may be the case for objects floating in space. However, in an integrated modeling environment, the technique has some limitations because it does not allow constrained movement. Because of its free-form

nature, it is very difficult to rotate an object around a single axis at a time, global or local, which is often required. For example, to turn around a human being standing on the floor requires rotating only around the vertical axis. With the virtual sphere, it is nearly impossible to rotate precisely around only one axis at a time.

Evans, Tanner, and Wein [28] also describe a rotation technique that suggests a turntable on which objects sit. Movements of the mouse in circles around the origin of the turntable cause the turntable, and thus the object, to rotate. There must also be a way of positioning the turntable underneath the object.

3.2.3 Integrated Systems

Bier's *snap-dragging* technique [12, 11, 10] simulates gravity between objects and takes advantage of surface geometry to constrain and control object movements. The user first positions *jacks* in space using a 3D cursor called the *skitter*. The jacks are coordinate frames that serve as anchors for other operations. The skitter slides along faces and edges, controlled by the mouse or through dials. The technique determines the position and orientation of the visible surface beneath the mouse in order to control the position and orientation of the skitter. Jacks can be placed with the skitter and then used to specify rotation axes or end-points for angles †.

This technique exploits the geometric structure of the objects, but it provides little help for manipulating positions and orientations in the absence of geometry. This means that the technique does not work especially well for manipulating points in open space as is often required.

3.3 The Jack Direct Manipulation Operator

The 3D direct manipulation mechanism in *Jack* interactively describes a global homogeneous transform. Internally, this is called the *manipulation transform*. There are many different commands in *Jack* that require the user to enter three-dimensional translational and rotational quantities. Each command may interpret the transform in its own way, possibly mapping it to a local coordinate system.

The user manipulates this transform through the 3-button mouse, together with the `SHIFT` and `CONTROL` keys on the keyboard. The keys alter the interpretation of the mouse buttons. Each mouse button corresponds to an axis in space, using a mapping scheme described below. The direct manipulation mechanism can alter the manipulation transform based on the selected axis by rotating around it, translating along it, or translating in a plane perpendicular to the axis. This characterizes the three primitive types of direct manipulation: linear translation, planar translation, and rotation.

The manipulation procedure is a loop, shown in Figure 3.1, that continues until the user terminates it.

The translation and rotation operator described here was originally presented in "Jack: A Toolkit for Manipulating Articulated Figures" [67], although it has been extended since then.

---

†I owe a great debt to Eric Bier and his paper *Skitters and Jack: Interactive Positioning Tools*. This is the paper that sparked my interest in interactive manipulation. It is also the source of the name "*Jack*," although the name has long since lost its relationship to the notion in the original paper.

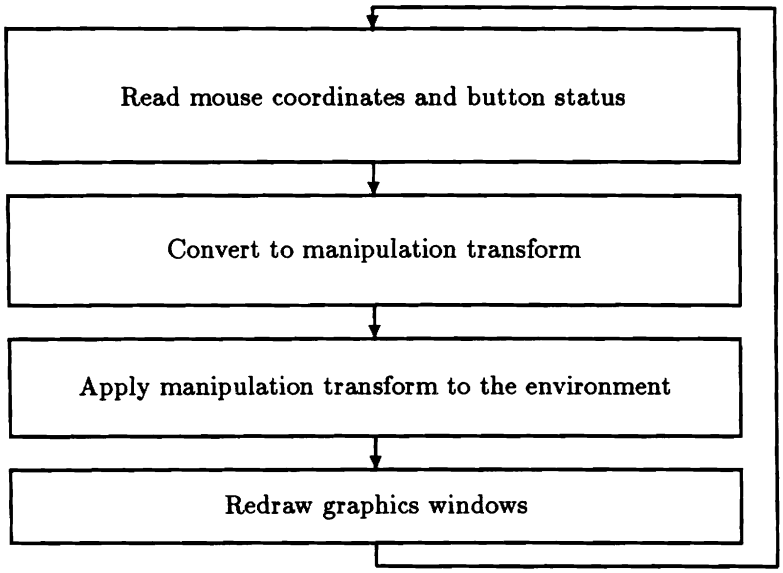


Table 3.1: The Manipulation Loop

3.3.1 Transform Coordinate System

Some 3D direct manipulation systems perform rotations in screen coordinates [21]. This means that the object always translates along and rotates around an axis parallel or perpendicular to the current line of sight. The advantage of such a system is that it gives the necessary kinesthetic feedback. It is predictable: the object never rotates around an unexpected axis. However, this technique seems useful only in relatively simplistic situations, involving rotations of rigid bodies in an uncluttered environment. In an integrated modeling environment, it is more common to move objects relative to either the global or local coordinate frame, rather than along axes aligned with the screen. For example, the simple task of raising an object vertically requires translating along the global y axis. Unless the view in the graphics window is perfectly horizontal, the vertical direction in screen coordinates is not exactly vertical. As another example, the task of moving a hand forward may require moving along an axis aligned with the body, not the screen.

The challenge in the design of a manipulation system is to properly choose the coordinate axes along which the user will be allowed to transform the object, and to adequately convey what those axes are. Also, there is a tradeoff between what is easy and what is difficult for the user to perform. Most good systems provide ways of doing many types of operations, but some operations are designed to be more readily available than others.

I feel that providing global and local rotations and translations available through the three mouse buttons and the two keys is the most appropriate choice in such an integrated environment. In addition, through the snapping mechanism described in the appendix in Section B.4, this technique allows the user to align the transform with any reference point in the environment and manipulate it with respect to that transform.

3.3.2 The User Interface

Our system is modal: each manipulation command places *Jack* in a mode where the mouse buttons and keyboard keys are interpreted as instructions to move the transform in question. How the movement is interpreted depends upon the command. This mode is terminated by hitting the **ESCAPE** key. While in the manipulation mode, the mouse buttons and keys behave as described below.

The user interface for the manipulation operation encodes by default the left, middle, and right mouse buttons to control translations along the x, y, and z axes, respectively, of the global coordinate frame. When the user presses down any mouse button, it enables translation along that axis. When the user presses two mouse buttons, translation is enabled along those two axes, i.e. in the plane spanned by those axes. With this technique, it is not possible to translate along three axes simultaneously, so pressing three buttons at once has no effect.

Rotation is signified by holding down the **CONTROL** key. In this case, the mouse buttons are interpreted as rotations around the x, y, and z axes of the global coordinate. Only one rotation button may be selected at once.

The user can change the axes of translation and rotation to the local coordinate frame of the manipulation transform by holding down the **SHIFT** key. The **CONTROL** key still signifies rotation, but the rotational axes are local to the manipulation transform instead of the global axes. Table 3.2 summarizes how the state of the keys and mouse buttons translates into the transform axis.

SHIFT	CONTROL	mouse buttons	action
		left	linear translation along global x axis
		middle	linear translation along global y axis
		right	linear translation along global y axis
		left and middle	planar translation in global xy plane
		left and right	planar translation in global xz plane
		middle and right	planar translation in global yz plane
	CONTROL	left	rotation around global y axis
	CONTROL	middle	rotation around global y axis
	CONTROL	right	rotation around global z axis
SHIFT		left	linear translation along local x axis
SHIFT		middle	linear translation along local y axis
SHIFT		right	linear translation along local y axis
SHIFT		left and middle	planar translation in local xy plane
SHIFT		left and right	planar translation in local xz plane
SHIFT		middle and right	planar translation in local yz plane
SHIFT	CONTROL	left	rotation around local y axis
SHIFT	CONTROL	middle	rotation around local y axis
SHIFT	CONTROL	right	rotation around local z axis

Table 3.2: Axis Mappings for Manipulation

Jack relies on a set of graphical icons to inform the user about the axes of translation and rotation. The manipulation transform is drawn as a labeled six-axis coordinate frame. The translation icon is

a transparent arrow. The rotation icon is a spoked wheel. Each icon is overlaid against the objects themselves, but since they are transparent, they do not intrude too severely.

3.3.3 The Mouse Line

The translation and rotation of the manipulation transform is determined interactively by the ray in the world coordinates that is cast through the location on the screen where the mouse cursor lies. This line in space is referred to internally as the *mouse line*, and it can be easily computed by an inversion of the viewing transform. The mouse line serves as a *probe* into the environment with which to move the manipulation transform. This notion of a probe is useful in describing the implementation, although it is not one that is visible to the user. The user has the feel of moving the object itself.

Linear and angular displacements are determined by intersecting the mouse line with lines and planes defined by the origin of the manipulation transform and the translation or rotation axis using a scheme described below.

3.4 Translation

3.4.1 Linear Translation

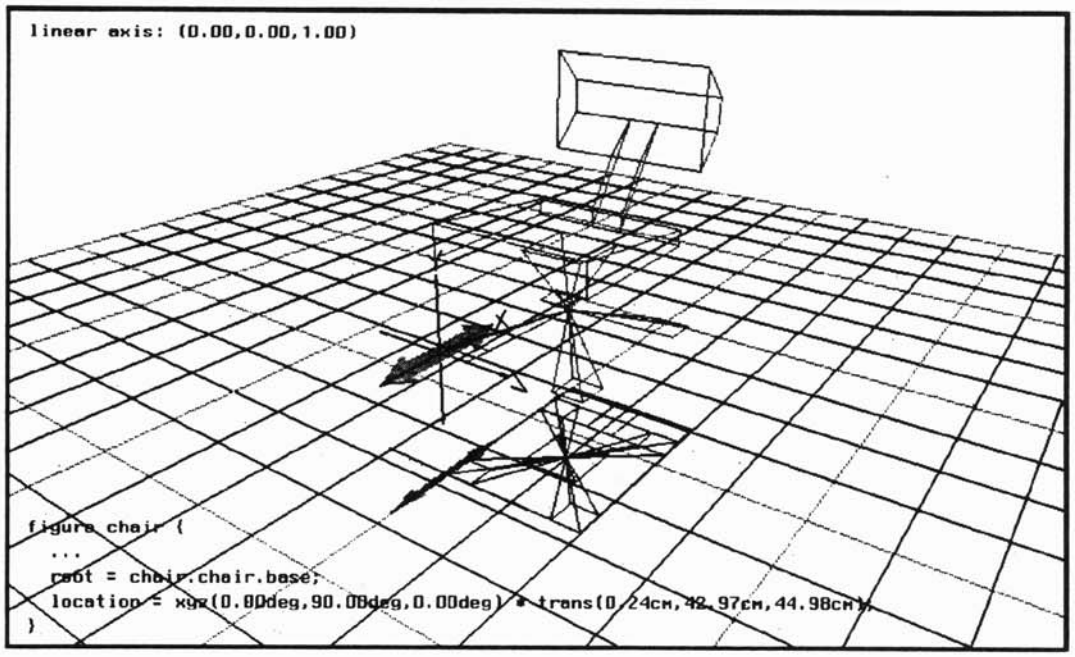


Figure 3.1: Linear Translation

As described above, linear translation takes place when the user presses one mouse button. The mouse may move anywhere on the screen, but the translation is restricted to the particular axis, determined

by which mouse button was pressed. This axis projects to a line on the screen. The translation icon illustrates this line, and it also instructs the user to move the mouse in that direction on the screen. Ideally, the user moves the mouse exactly along this line, but in practice the mouse will not follow the line precisely. The position of the manipulation transform is the point along the translational axis that is nearest to the mouse line. Figure 3.1 shows the translation icon. The details of the implementation of the linear translation technique are given in Section B.1 in the appendix.

3.4.2 Planar Translation

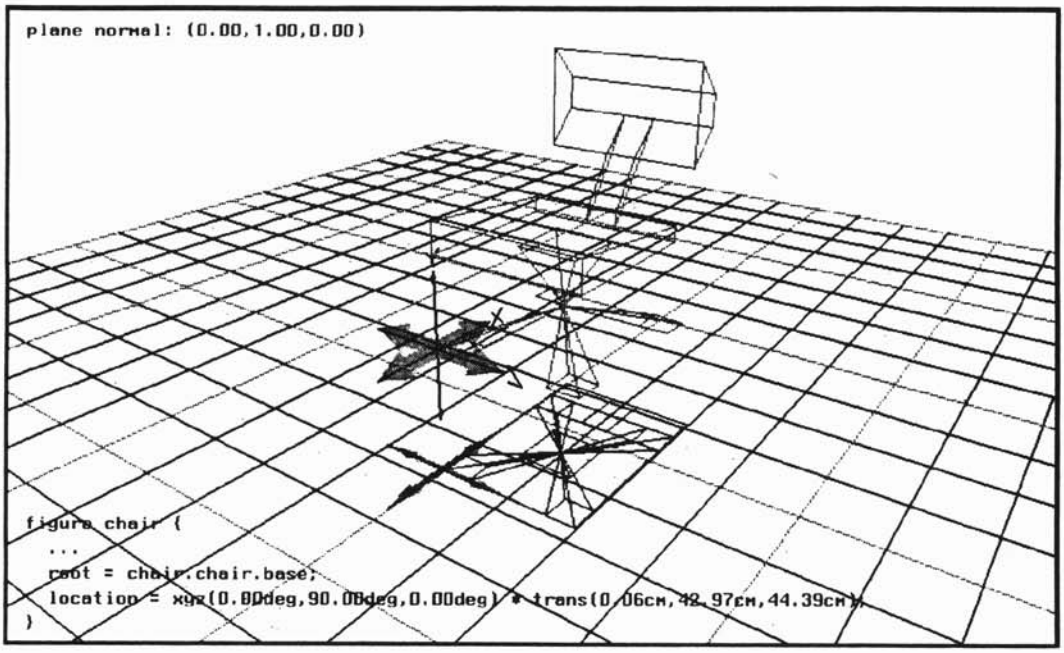


Figure 3.2: Planar Translation

Planar translation is actually somewhat simpler than linear translation because its two degrees of freedom more closely match those of the mouse. The plane of translation passes through the origin of the manipulation transform, and is spanned by the two axes defined by the selected mouse buttons. The technique is to intersect the mouse line with the plane of translation and supply the point of intersection as the origin of the manipulation transform. This means that the object automatically goes to the location in the plane that lies underneath the mouse cursor. Figure 3.2 shows the planar translation icon. The details of the implementation of the planar translation technique are givin in Section B.2 in the appendix.

The user can translate in the global or local xy, xz, or yz planes, but in practice the linear and planar translation techniques provide a comfortable pattern of use involving only planar translation in the xz plane. This is the horizontal plane, and it is the most intuitive to visualize. The user can comfortably translate objects in the horizontal plane using planar translation, and then raise and lower them using linear translation along the y axis.

3.5 Rotation

The user interface for rotation requires the user to move the mouse around in circles on its pad. This is similar in some ways to the turntable technique of [28] described earlier, except that their technique determined the rotation angle in screen coordinates, not world coordinates, making the angular displacement independent of the viewing direction. This technique provides a more direct feel over the rotation because it gives the sense of actually holding on to the wheel.

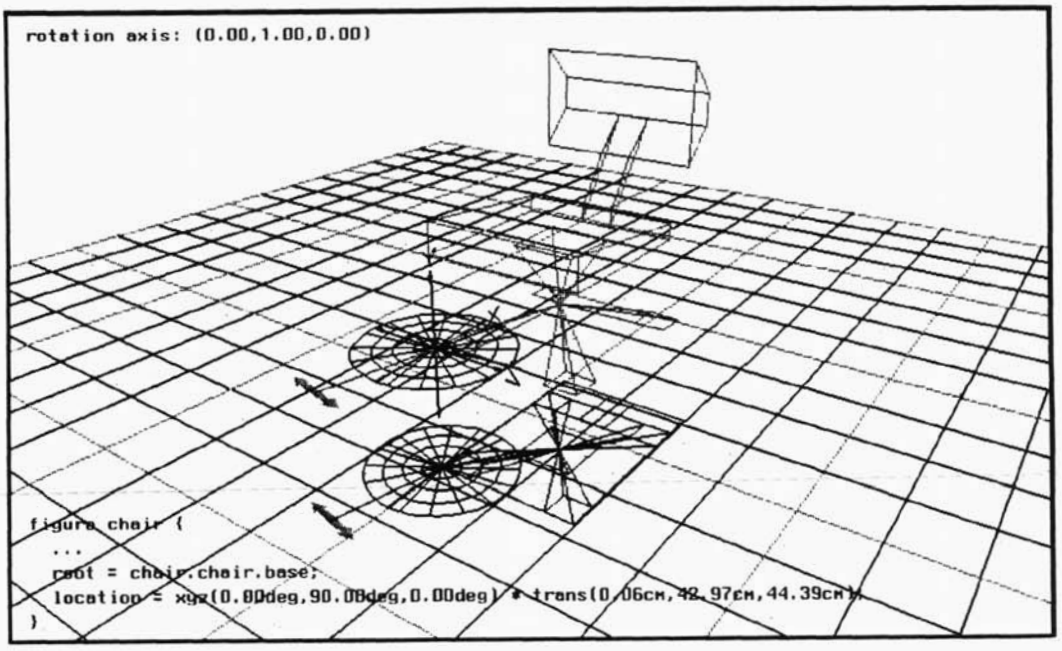


Figure 3.3: Rotation

The three mouse buttons are encoded as rotation around the x, y, and z axes. When the user presses down on a button, a wheel is displayed at the origin of the manipulation transform describing the rotational axis. This wheel lies in the plane in which the rotation is to take place, with the origin of the wheel at the rotational axis. Then a vector is drawn from the current intersection point between the plane and the mouse line. This vector forms an extended *spoke* of the wheel, and as the user moves the mouse around in this plane, *Jack* computes a new spoke and then measure the angular distance between this it and the original spoke. This determines the angle of rotation. The sensation that the user gets is one of twisting a crank by moving the mouse around in circles on the screen. Figure 3.3 shows the rotation wheel icon. The details of the implementation of the rotation technique are givin in Section B.3 in the appendix.

3.6 Discussion of the Direct Manipulation Mechanism

A seeming drawback of this technique is that the user must remember the current orientation of the x, y, and z axes. However, the translation and rotation icons give the user a simple means of determining this

information quickly. Typically, the user has a particular direction in either local or global coordinates in mind when he initiates a movement operation. Since the icons are displayed as the keys are pressed and before the motion begins, the user can easily cycle through the available axes to select the appropriate set.

For instance, the user may want to "turn the figure around," which may technically involve rotating 180° about the y axis. The user can initiate a move figure operation, press the control key to specify rotation, and press the left, middle, and right mouse buttons in turn and begin moving the mouse when the appropriate axis is displayed. This technique frees the user from having to remember that the y axis is the appropriate axis. This avoids overloading the screen with information by making it available at the user's fingertips as needed.

It can be difficult to specify precise angles and distances using this manipulation operator. One way to provide better end conditions is by *snapping* the transform to features of the geometric environment. This enables the user to position objects at precise locations and align them precisely with existing objects. Our technique for snapping is described in detail in the appendix in Section B.4.

3.7 Automatic Viewing Adjustments

A drawback of this manipulation technique is the inability to translate an object along an axis parallel to the line of sight, or to rotate around an axis perpendicular to the line of sight. In these cases, small differences in the screen coordinates of the mouse correspond to large distances in world coordinates, which means that the object may spin suddenly or zoom off to infinity. This is an intrinsic problem with viewing through a 2D projection: kinesthetic correspondence dictates that the object's image moves in coordination with the input device, but if the object's movement is parallel to the line of projection, the image doesn't actually move, it only shrinks or expands in perspective.

In the past, I adopted the view that the first prerequisite for manipulating a figure is to position the camera in a convenient view. Although the viewpoint manipulation techniques in *Jack* are quite easy to use (these are described in Appendix C), this forced the user through additional step in the manipulation process, and the user frequently moved back and forth between manipulating the object and camera.

Much of this viewing adjustment as an aid to manipulation can be automated, in which case the system automatically places the camera in a view that avoids the problems of degenerate axes. This can usually be done with a small rotation to move the camera away from the offending axis. This automatic camera rotation can even be helpful by itself, because it provides a kind of depth cue.

Jack uses Ware and Osborne's *camera in hand* metaphor[82] for the view. The geometric environment in problems in human factors analysis usually involve models of human figures in a simulated workspace. The most appropriate cognitive model to promote is one of looking in on a real person interacting with real, life-size objects. Therefore, *Jack* suggests that the controls on the viewing mechanism more or less match the controls we have as real observers: move side to side and up and down while staying focused on the same point.

To prevent degenerate movement axes from causing problems, *Jack* uses a threshold between the movement axis and the line of sight, beyond which it will not allow the user to manipulate an object. To do so would mean that small movements of the mouse would result is huge translations or rotations of the object. This value is usually 20°, implying that if the user tries to translate along an axis that is closer than 20° to the line of sight, *Jack* will respond with a message saying "can't translate along that axis from this view," and it will not allow the user to do it.

The automatic viewing adjustment invokes itself if the user selects the same axis again after getting the warning message. *Jack* will automatically rotate the camera so that its line of sight is away from the transformation axis. To do this, it orients the camera so that it focuses on the object's origin, and then rotates the camera around both a horizontal and a vertical axis, both of which pass through the object origin. The angles of rotation are computed so that the distance away from the offending axis is at least 20°.

Both automatic viewing rotations occur simultaneously, and *Jack* applies them incrementally using a number of intermediate views so the user sees a smooth transition from the original view to the new. This avoids a disconcerting snap in the view. *Jack* applies the angular changes using an ease in/ease out function that ensures that the transition is smooth.

The procedure for rotating the camera is sensitive to the interactive frame rate so that it provides relatively constant response time. If the camera adjustment were to use a constant number of intermediate frames, the response time would be either too short if the rate is fast or too long if the rate is slow. *Jack* keeps track of the frame rate using timing information available from the operating system in 1/60th's of seconds. *Jack* computes the number of necessary intermediate frames so that the automatic viewing adjustment takes about 1 second of real time.

The reason for the repeated axis selection is to ensure that the user didn't select the axis by mistake. It is common to position the view parallel to a coordinate axis to get a 2D view of an object. If the user likes this view, then it would be wrong to disturb it. For example, if the user positions the view parallel to the z axis to get a view of the xy plane, and then accidentally hits the right mouse button, the view will not automatically change unless the user confirms that this is what he or she wants to do.

Automatic view positioning also takes place when the object is not visible. This may mean that the object is not visible at all, or only that its origin is not visible. For example, a human figure may be mostly visible but with its foot off the bottom of the screen. In this case, a command to move the foot will automatically reposition the view so that the foot is visible.

3.7.1 Avoiding Viewing Obstructions

When manipulating an object using solid shaded graphics, it can be especially difficult to see what you are doing because of the inability to see through other objects. In some situations, this may be impossible to avoid, in which case the only alternative is either to proceed without good visibility or revert to a wireframe image. Frequently however, it may be possible to automatically change the view slightly so that the object is less obstructed. To do this, we borrow an approach from radiosity, the *hemicube* [22].

The hemicube determines the visibility of an entire geometric environment from a particular reference point, and it gives enough information to find an unobstructed location for the camera if one exists. The hemicube computation is centered around the origin of the object being manipulated but oriented towards the current camera location. This yields a visibility map of the entire environment, or what you would see through a fish-eye lens looking from the object's origin towards the camera. If the camera is obstructed in the visibility map, *Jack* looks in the neighborhood of the direction of the camera for an empty area in the hemicube map. This area suggests a location of the camera from which the object will be visible. From this, *Jack* computes the angles through which the camera should be rotated. It generates the hemicube map using the hardware shading and z-buffer, so its computation is quite efficient.

3.8 Direct Manipulation of Joints

The movement operations are helpful for positioning reference points in space, and the discussion in the latter chapters address the problems of how such reference points can affect the posture of an articulated figure in a general way. However, *Jack* also uses the same basic manipulation procedure to control the transform at the joints of an articulated figure. As described in Appendix A, *peabody* models joints with specific degrees of freedom, which are rotation and translation axes. To adjust a joint means to rotate it around or translate it along one of its defined axes. This can be done with the same basic manipulation tools, although they need to be packaged differently to conform to the definition of the joint.

The user interface for manipulating joints associates each axis of the joint with a mouse button. The left button controls the first DOF, the middle button controls the second if there is one, and the right button controls the third if there is one. Few joints have more than three degrees of freedom, but for these joints, the extra DOF's are activated by holding down the CONTROL key.

Before the movement operation, *Jack* must map the axis of a degree of freedom to global coordinates, which may mean transforming around previous degrees of freedom for multi-degree of freedom joints. This means that each axis is *local*. The user can transform the joint along only one axis at a time. If a degree of freedom is prismatic, its distance is manipulated with the linear translation technique. If it is angular, its angle is manipulated with the rotation technique. This mechanism does not provide the ability to manipulate multi-DOF prismatic joints using the planar translation technique, although it would be possible to do so. In practice, such joints seldom occur.

This mechanism has left unspecified how the joint displacement affects the position of the figure. When a joint is adjusted, one segment remains fixed with respect to the world frame and the other moves. With a predefined hierarchy, this is a simple matter since there is a well defined "proximal" and "distal" segment. But peabody allows the hierarchy to be inverted, so it is possible for a joint to point upwards in the tree. However, since the manipulation takes place one DOF at a time, this only involves negating the DOF axis and angle so that it appears that the joint is behaving according to its true definition.

3.8.1 Joint Limits

Peabody models joint limits as upper and lower clamps on the angles of a joint. These limits can be enforced by the manipulation procedure by simply clamping the linear or angular displacement.

The user interface for rotational joints with degrees of freedom illustrates the joint limits through colored sections of the wheel. A green section of the wheel illustrates the legal range of movement; a red section illustrates the range outside the limits. The red and green sections of the wheel illustrate legal and illegal regions for the movement of the *mouse*, not the movement of the joint itself. In this respect, it does not simulate a reach volume for the joint because this would involve knowing and visualizing the geometry around the joint as it moves. The green section appears centered around the initial spoke of the rotation wheel when the rotation begins, and it illustrates how far the spoke is allowed to rotate in each direction. For joints with a distinctive proximal and distal end, the green section of the wheel will illustrate the reach space of the DOF if the initial spoke is at the distal end of the rotating segment. Figure 3.4 shows the rotation wheel icon with joint limits.

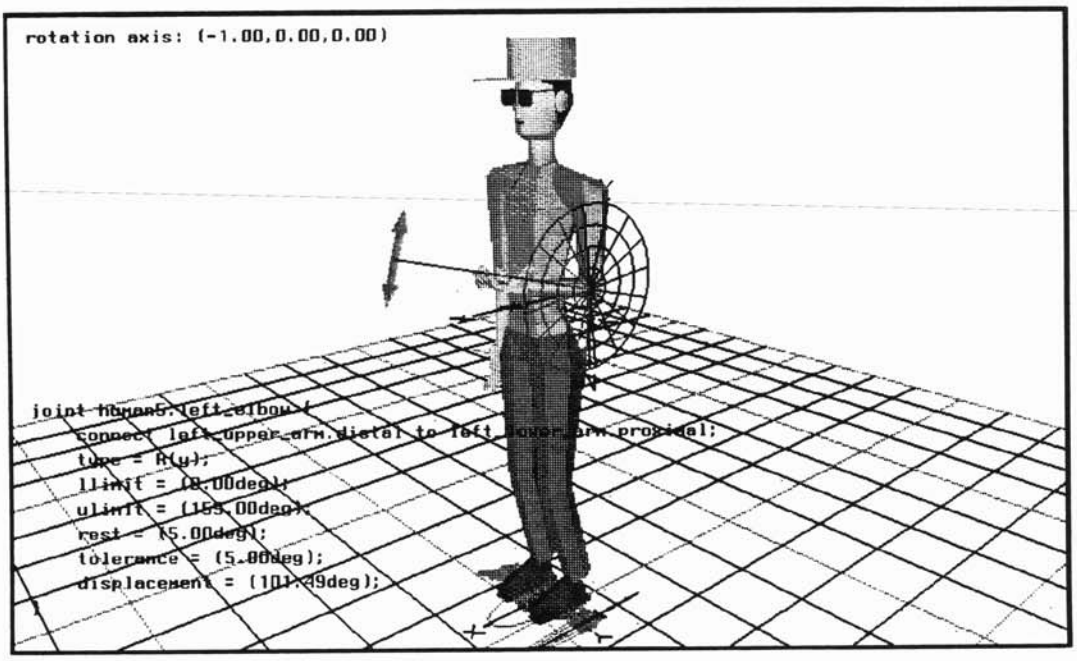


Figure 3.4: The Rotation Wheel Icon with Joint Limits

3.8.2 Drawbacks of Joint Manipulation

This joint rotation mechanism does not solve the traditional problems of gimbal lock for multi-DOF joints. This manipulation scheme allows joints to be rotated only around their defined axes, which means that in a particular configuration, it is possible for the first and third axes to have the same global orientation. This means that the user may not be able to rotate the joint in the desired direction. In this situation, the user must make the rotation in two stages. This is a dreadful situation from a user interface point of view because it may not be obvious to the user how to rotate the joint to get it into the desired position.

However, the interactive techniques described in the next chapter overcome this problem to a large degree by using the inverse kinematics algorithm to position joints. As an example, the user can manipulate the arm by dragging the position of the elbow, forearm, or hand. The inverse kinematics algorithm automatically determines the angles of the shoulder that solve the positioning problem.

3.9 Summary

Jack provides a flexible and easy-to-use interface for displaying and manipulating complex articulated figures. The movement icons provide good visual feedback on the current state of the environment and the effect of the input. Since all three dimensional position and orientation information is provided by the same operator, there is a consistent interface for all aspects of the system. Since the movement operator generates homogeneous transforms based on the screen location of the mouse cursor, it is easy

to anticipate the effect which movements of the input device will have on the objects. Since *peabody* is a very general mechanism for representing articulated structures, it is very easy to define attachment and reference points to specify transforms with respect to arbitrary frames.

The control of a virtual camera is vitally important to many techniques for 3D direct manipulation, although no one as yet has addressed the two issues in an integrated manner. The technique used in *Jack* for automatically adjusting the view in conjunction with direct manipulation is an effective addition to the manipulation process. The automatic viewing rotations are usually very small so they do not inject large changes to the user's view of the geometric environment. Since the viewing adjustments are only activated on the second attempt at movement along a degenerate axis, the adjustments are seldom invoked accidentally, minimizing the degree to which the adjustments are inappropriate.

The contributions of this chapter are:

- A new technique for translating and rotating in 3D with a 2D input device. This technique allows the user to easily translate and rotate around either global or local coordinate axes, and it provides good kinesthetic correspondence.
- An integration of the viewpoint specification problem with the direct manipulation technique.

Chapter 4

Inverse Kinematics for Postural Control

"The more constraints one imposes, the more one frees one's self."

Igor Stravinsy
quoted in NY Times 7 Apr 71

The 3D direct manipulation technique described in the previous chapter provides a primitive kind of control for interacting with articulated figures. The movement operator gives the user the ability to move the root of a figure hierarchy and to rotate the joints. The structure of *peabody* ensures that joints only rotate around the appropriate axes and that they obey the proper joint limits. However, the interaction technique really only provides control over one or two degrees of freedom at a time, and figures typically have many dozens of degrees of freedom. What is more, real figures typically don't move one joint at a time; they move in a coordinated fashion. Positioning a complex figure using such primitive tools can be very difficult, tedious, and inaccurate.

The solution to this problem is to use some type of algorithmic assistance in positioning the figures and describing their postures. But what should this algorithm do? What will its inputs be? What are

the primitives through which complex postures and motions may be described?

In this thesis, I advocate the use of inverse kinematics for describing postures of articulated figures. The basic idea behind inverse kinematics is to specify desired positions and orientations for reference points on the figures and have the algorithm automatically determine joint angles which place the figure in a posture which satisfies the positioning criteria. The principal focus of this chapter, and of this entire work as well, is not with the numerical issues of the inverse kinematics algorithm but with how such an algorithm fits into the overall picture of interactive postural control.

4.1 Chapter Overview

This chapter describes how to phrase the inverse kinematics problem for use in interactive postural control. I discuss other applications and implementations of inverse kinematics and describe how this formulation is different. This chapter defines *Jack*'s notion of a *constraint*, which is the primitive building block through which to describe postures. The term "constraint" means many things to many people in the computer graphics community, so it is important to quantify my use of the term. Finally, I describe how the manipulation system interfaces to the inverse kinematics algorithm.

The main points of this chapter are:

- a formulation of inverse kinematics well suited for interactive postural control;
- the definition of *Jack*'s notion of a "constraint";
- interactive methods for manipulating articulated figures under the influence of constraints.

4.2 Background

For an articulated mechanism, we can describe cartesian positions $\mathbf{x}$ of selected reference points in terms of joint angles Θ:

$$\mathbf{x} = \mathbf{f}(\Theta) \tag{4.1}$$

This equation can be differentiated with respect to time to determine end effector velocities in terms of joint velocities:

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\Theta} \tag{4.2}$$

where $\mathbf{J} = \partial\mathbf{f}/\partial\Theta$ is the Jacobian matrix.

4.2.1 Inverse Kinematics in Robotics

The inverse kinematics problem has been studied in great detail in robotics, and it can be posed in two ways. First, we can solve for joint coordinates in terms of cartesian coordinates:

$$\Theta = \mathbf{f}^{-1}(\mathbf{x}) \tag{4.3}$$

Alternatively, we can solve for joint velocities in terms of cartesian velocities:

$$\dot{\Theta} = \mathbf{J}^{-1}(\mathbf{x})\dot{\mathbf{x}} \tag{4.4}$$

where $\mathbf{J}^{-1}$ is the inverse Jacobian. Most robot manipulators are relatively simple, and they are designed so that these equations can be derived in closed form. Therefore, no iterative procedure is necessary, and the necessary joint positions can be computed with very few arithemetic operations.

Equation 4.4 can of course be solved only if the Jacobian $\mathbf{J}$ is invertible, which is the case only if the number number of joint angles in the mechanism matches the dimension of the end effector vector. An end effector typically has six degrees of freedom, three for position and three for orientation. If there are more degrees of freedom, then the mechanism is redundant, $\mathbf{J}$ is not square and thus not invertible, and Equation 4.4 has no closed-form solution. In this case, the inverse kinematics problem is usually formulated as

$$\dot{\Theta} = \mathbf{J}^{+}\dot{\mathbf{x}} + (\mathbf{I} - \mathbf{J}^{+}\mathbf{J})\mathbf{y} \tag{4.5}$$

where $^{+}$ denotes the pseudoinverse and the $(\mathbf{I} - \mathbf{J}^{+}\mathbf{J})$ term projects an arbitrary $n \times 1$ vector $\mathbf{y}$ onto the null space of $\mathbf{J}$. Redundant manipulators are common in robotics because they have greater dexterity. There is much literature in robotics dedicated to the inverse kinematics of redundant manipulators. Klein and Huang [48] present a summary of these techniques.

The robotics literature seldom addresses techniques for controlling massively redundant mechanisms such as human figures. Most redundant robot manipulators have 7 degrees of freedom. Some researchers have addressed the problem of coordinating two arms. Karlen et al [45] describe a robot manipulator designed with two 7-DOF arms mounted on a torso-waist assembly with 3-DOF's for a total of 17-DOF's. They address redundancies by incorporating spring-origins in the joints.

4.2.2 Inverse Kinematics in Computer Animation

Inverse kinematics techniques for redundant manipulators have been used in computer graphics for the animation of articulated figures, mostly by Michael Girard [34, 35]. Most notably, Girard's PODA system for simulating animal locomotion using a gait controller to generate motion paths for the legs. Arm trajectories can be generated by splining together end effector positions defined through key poses. PODA solves for joint-space velocities which move the end effector smoothly along the precomputed trajectory.

4.2.3 Constraints

The term "constraint" has many meanings and applications. Some researchers use it to mean a very low level concept. Isaacs and Cohen [42, 41] use the term to mean essentially any kinematic control over a joint or group of joints during dynamic simulation. The constraint removes degrees of freedom from the system in a straightforward way. Witkin and Kass [92] and Girard [33] use the term to mean a global optimization criterion, such as minimum expended energy. Sometimes, the term means an desired relationship, which in computer graphics usually implies a geometric one. This is the case of Nelson's Juno system [59]. Many researchers use the term to mean specifically a desired *spatial* relationship, that is, goals for reference points. This is usually the meaning in physically based modeling, as in the dynamic constraints of Barzel and Barr [7], as well as in inverse kinematics as in Badler, Manoochehri and Walters's POSIT system [5].

Constraints that mean geometric goals may generally have temporal components. Most constraint based systems like those just mentioned feature the ability to vary the effect of constraints over time. Given this, it is rather nebulous whether the temporal component is a part of the constraint definition.

My feeling is that it should not be, and that it is better to view a constraint instantaneously as a static entity, although its parameters can be changed over time by some means external to the constraint itself.

4.3 Inverse Kinematics for Postural Control

Most formulations of inverse kinematics in robotics and computer animation are principally concerned with motion — smooth motion. In robotics, this is necessary because jerky motion could damage a manipulator. In animation, jerky motion looks bad. For postural control, as have I described before, the motion is not so important because the static posture is the principal objective. This means that for interactive postural control, it may be possible to entertain options which are not available to robotics and animation.

In particular, the requirements for postural control are, first, that the technique accommodate massively redundant figures, and second, that it perform fast enough for interactive manipulation, even with complex, highly constrained figures. The third concern is that it generate smooth movement. Recall from the Introduction that the interactive postural control process consists of placing the figure in a sequence of postures closely spaced in time and space, giving the illusion of motion for purposes of assisting in the postural specification process.

The approach I advocate uses the technique of Jianmin Zhao [95]. It dispenses with physical interpretations and solves a minimization problem for pure functions using a non-linear programming algorithm. The approach uses a variable-metric optimization technique. The function it minimizes is a defined through a linear combination of kinematic constraints as defined below. The objective function describes positions, not velocities. This approach essentially treats the figure a purely geometric entity rather than a physical one. It does not take into account the figure's mass and inertia. Zhao's algorithm is highly efficient and is capable of controlling a complex figure model with large numbers of constraints.

4.3.1 Postures through Potential Functions

The approach that I advocate for defining postures is somewhat similar to the energy constraints of Witkin, Fleischer, and Barr [90], although there are some important distinctions. Their approach models connections between objects using energy functions. The energy functions do not measure mechanical energy, but are simply constructed to have zeros at the proper locations and have smooth gradients so that the object can follow the gradient towards the solution. This provides the animation of the solution process. The user interface of Witkin, Fleischer, and Barr's system allows the user to specify these energy functions and turn them on. This causes a sudden increase in energy and thus causes the model to begin descending in the direction of the gradient of the energy function. One drawback of this approach is that the timestep of the iterative process must be sufficiently small to ensure convergence. This is particularly a problem in the case of articulated figures. Witkin, Fleischer, and Barr's formulation of the gradient descent algorithm does not permit jointed mechanisms, so they must model joints as constraints. The joints must have a very steep energy function to ensure that they never come apart. This means that the timestep must be very small, making the system of equations very stiff.

One problem with Witkin, Fleischer, and Barr's approach, from a higher level point of view, is that the user's notion of time is embedded in the algorithm. The energy function defines the path of the end effectors towards their ultimate destinations because the gradient of the energy function determines the direction of movement. I suggest removing the notion of time from the numerical process and putting it in a higher level control scheme. This control scheme decides on discrete locations for the goals

for end effector reference points, closely spaced in space and time — the user's notion of time. The inverse kinematics algorithm solves an instantaneous positioning problem. The control scheme has two components, the interactive manipulation and the behaviors described in the next chapter.

Zhao's numerical formulation has no notion of time. It uses the joint angles of the *peabody* figures as the variables of optimization. This means that the search step can be significantly larger, and thus the convergence is faster. The control scheme guides the movement in cartesian space, rather than in the space of the energy function. It is still convenient to interpret the positioning criteria as a potential energy function, but the control scheme ensures that the figure is always very near a state of equilibrium.

Using the example of a human arm reaching for a switch, the technique of Witkin, Fleischer, and Barr would model the reach by defining an energy function with a zero when the hand is on the switch, that is, a constraint for the hand to lie at the switch. If the arm begins by the side of the body, the arm would see a sudden increase in energy when the constraint becomes active and would immediately begin to move in a way to minimize the energy. This would cause the hand to begin to move towards the switch. The velocity would be controlled through the intensity of the potential energy function.

Using Zhao's numerical formulation, the higher level control scheme that I suggest computes successive locations for the hand, starting at its current location and progressing towards the switch. The number of steps and the spacing between them is not the domain of the inverse kinematics algorithm but of the control scheme. A kinematic constraint describes the position of the hand and the algorithm's parameters at each time step. The inverse kinematics algorithm is invoked at each time step to place the hand at the desired location.

4.3.2 Constraints for Inverse Kinematics

Jack's notion of a kinematic *constraint* defines a *goal* coordinate frame, an *end effector* coordinate frame, a *set of joints* which control the end effector, and an *objective function* which measures the distance between the goal and the end effector. The set of joints is usually defined in terms of a *starting joint*, and the joint set then consists of the chain of joints between the starting joint and the end effector. We can think of individual constraints as defining a potential energy, measured by the objective function, much like that of Witkin, Fleischer, and Barr. The constraint is satisfied when the objective function is minimized, although the function need not be zero. The potential energy of several constraints is a weighted sum of the energies from the individual constraints.

4.3.3 Objective Function Types

The objective function of a constraint has separate position and orientation components. The potential energy of the constraint is a weighted combination of the two, according to the constraint's *position/orientation weight*. The weight may specify one or the other, or a blending of the two. Zhao's procedure provides the following objective types [95]:

point The point-to-point distance between the end effector and the goal.

line The point-to-line distance between the end effector and a line defined in the goal coordinate frame.

plane The point-to-plane distance between the end effector and a plane defined in the goal coordinate frame.

frame The orientational difference between the end effector frame and the goal frame. This measures all three orientational degrees of freedom.

direction The orientational difference between a vector defined in the end effector frame and a vector defined in the coordinate frame of the goal.

aim A combined position/orientation function designed to "aim" a reference vector in the coordinate frame of the end effector towards the position of the goal. This is used mostly for eye and camera positioning, but it could also be used, for example, to point a gun. This should never be done in the presence of multiple human figures, of course, particularly children.

4.4 Features of Constraints

Each constraint in *Jack* has its own set of joint variables, so the control of each constraint can be localized to particular parts of the figure. Since the joint angles are the variables of optimization, this means that the algorithm operates on chains of joints, ultimately terminated at the root of the figure hierarchy. This implies that the root of the figure remains fixed during the positioning process. This makes the placement of the figure root particularly important. One of the major components of peabody, described in Appendix A, is that the actual definition of a figure does not include the figure root. Instead, the root is a dynamic property which can be changed from time to time. Since the inverse kinematics algorithm operates on chains emanating from the root, the inverse kinematics algorithm cannot change the location of the root.

Zhao's algorithm works very well provided that it doesn't have to search too far for a solution, although it will converge from any starting point. The farther it has to search, the more likely it is to produce large changes in the joint angles. In geometric terms, this means that the goals should never be too far away from their end effectors, lest the interior segments of the joint chains move too far. This also relieves the problem of getting trapped in a local minima because hopefully the higher level control strategy which is specifying the goal positions will do so in a way to lead the figure away from such conditions.

The elegance of the potential energy approach, like that of Witkin, Fleischer, and Barr, is that the constraints can be layered on top of one another. This means that it is acceptable to overconstrain the figure. The posture which the algorithm achieves in this case yields the minimum energy state according to the weighting factors between the constraints. This provides a way of resolving the redundancies in a massively redundant figure: use lots of constraints, and don't worry too much about whether the constraints specify conflicting information.

4.5 Inverse Kinematics and the Center of Mass

The center of mass of an object is one of its most important landmarks because it defines the focal point for forces and torques acting on it. The center of mass of an articulated figure such as a human figure is particularly significant because its location relative to the feet defines the state of balance. This is critical for human figures, because so many aspects of the movement of a human figure are dictated by the need to maintain balance. The center of mass of is, of course, a dynamic property, but it is possible to manipulate it in a purely kinematic way and thus produce some of the effects of dynamic simulation without the extra cost.

4.5.1 Methods of Maintaining Balance

One approach to maintaining balance of an articulated figure is to root the figure through its center of mass. The center of mass is a dynamic feature of a figure, so rooting the figure through the center of mass means that each time the figure moves, the center of mass must be recomputed and the figure's location updated so that the center of mass remains at the same global location.

This approach works, but it does not give good control over the elevation of the center of mass, since the center of mass is effectively constrained to a constant elevation as well as location in the horizontal plane. The figure appears to dangle as if suspended from its waist with its feet reaching out for the floor. This is particularly true during an operation in which the center of mass normally goes down, such as bending over. In order for the balance behavior to function naturally, the elevation of the center of mass must be allowed to float up and down as required by the movement of the feet. This is more appropriately handled through a constraint.

4.5.2 Kinematic Constraints on the Center of Mass

The balance of a figure is a constraint on the center of mass to remain vertically above a point in the support polygon. The constraint designates a single point as the balance point rather than using the entire support polygon because this gives control over the placement of the point within the polygon. This allows the figure's weight to shift side to side or forward and backward, without moving the feet.

Jack associates the center of mass logically with the lower torso region of the human figure, and it uses this as the end effector of the constraint, with the ankle, knee, and hip joints of the dominant leg as the constraint variables. During the constraint satisfaction process at each interactive iteration, the center of mass is not recomputed. Since the center of mass belongs logically to the lower torso, its position relative to the torso remains fixed as the inverse kinematics algorithm positions the ankle, knee, and hip so that the previously computed center of mass point lies above the balance point. There are generally other constraints active at the same time, along with other postural adjustments, so that several parts of the figure assume different postures during the process.

After the constraints are solved, *Jack* recomputes the center of mass. It will generally lie in a different location because of the postural adjustments, indicating that the figure is not balanced as it should be. Therefore, the constraints must be solved again, and the the process repeated until the balance condition is satisfied. In this case the structure of the human figure helps. Most of the postural adjustments take place on the first iteration, so on subsequent iterations the changes in the center of mass relative to the rest of the body are quite minor. *Jack* measures the distance that the center of mass changes from one iteration to the next, and it accepts the posture when the change is below a certain threshold. Although it is difficult to guarantee the convergence theoretically, in practice it seldom takes more than two iterations to achieve balance.

4.6 Interactive Methodology

There are several possibilities for overcoming the problems with redundancies and local minima. One is to incorporate more information into the objective function, modeling such factors as strength, comfort, and agent preference [51]. This is an important addition, although it adds significantly to the computational complexity of the constraint solving procedure. *Jack*'s technique is to provide the positional input to the inverse kinematics algorithm with the 3D direct manipulation system. *Jack* allows the user

to interactively "drag" goal positions and have the end effector follow. In this case, the geometric information obtained by the mouse at each iteration of the manipulation process is applied to the goal position of a constraint, and the inverse kinematics algorithm is called to solve the constraints before the graphics windows are redrawn. Alternatively, the user can move a figure which has end effectors constrained to other objects. Each case causes a relative displacement between the end effector and the goal. This interactive control of inverse kinematics was originally described in "Interactive Real-time Articulated Figure Manipulation Using Multiple Kinematic Constraints" [69].

4.6.1 Interactive Dragging

This dragging mechanism is a modified version of the basic direct manipulation scheme described in Chapter 3. After selecting the parameters of the constraint, the manipulation procedure works as shown in Figure 4.1. The inverse kinematics procedure is invoked at every iteration during the interactive

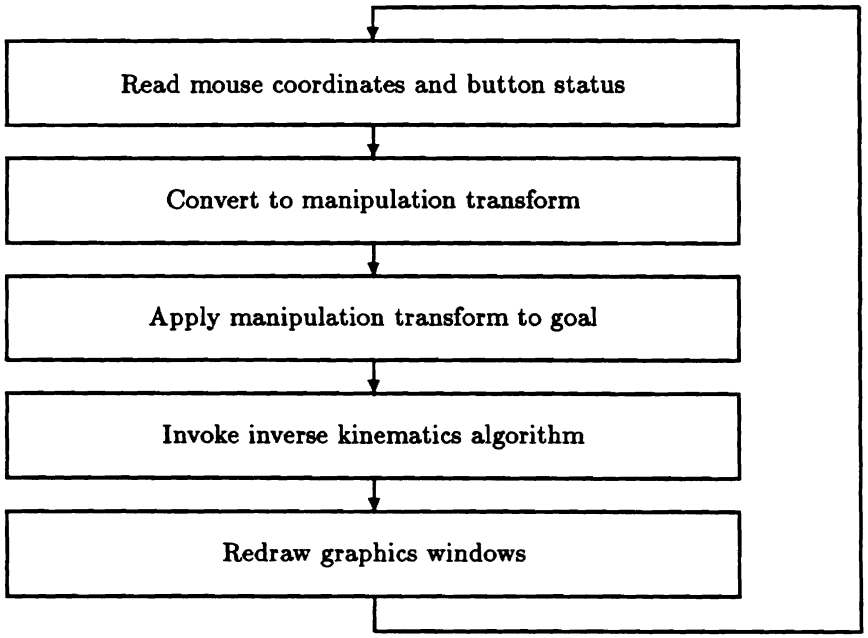


Table 4.1: Interactive Dragging

manipulation.

This is a very effective and efficient tool for manipulation for several reasons. Because of the incremental nature of the interactive manipulation process, the goals never move very far from one iteration to the next, as necessary. The algorithm still suffers from problems of local minima, but since the user can drag the end effector around in space in a well-defined and easy to control way, it is relatively easy to overcome these problems by stretching the figure into temporary intermediate configurations to get one part of the figure positioned correctly, and then dragging the end effector itself into the final desired position. This takes advantage of the user's abilities, because local minima can be easy for the user to see but difficult for the system to detect and avoid.

A common example of this dragging technique involves the elbow. The user may initially position the hand at the proper place in space but then find that the elbow is too high. If this is the case, the user can extend the hand outwards to drag the elbow into the correct general region and then drag the hand back to the proper location.

4.6.2 Interactive Twisting

Another effective feature of the direct manipulation interface is the use of orientation constraints, particularly the weighted combination of position and orientation. In this case, the orientation of the goal is significant as well as the position, so the user may manipulate segments in the interior of the joint chain by twisting the orientation of the goal and thus the end effector. This is especially helpful because of the difficulty the user encounters in visualizing and numerically describing rotations which will achieve a desired orientation. The above example of the elbow position may be handled this way, too. By twisting the desired orientation of the hand, the interior of the arm can be rotated up and down while the hand remains in the same location. This achieves in real-time a generalization of the "elbow circle" positioning scheme implemented by Korein [49].

This raises an interface issue concerning the relationship between the actual orientation of the end effector coordinate frame and the manipulation transform. The manipulation technique described in Chapter 3 allows the user to translate and rotate a 6D quantity which now guides the position and orientation of the end effector. However, as noted in Chapter 3, this technique is not so good at choreographing smooth movements through space. The movement trajectory generated by the technique consists of intermittent segments of straight-line translations and rotations. As the user translates the manipulation transform, its orientation remains fixed, and vice versa. Is this appropriate behavior for the end effector as well?

If the end effector is sensitive to orientation, then translating the manipulation transform means that the end effector will translate *but will try keep the same global orientation.* Typically, the user can quickly arrive at a goal position for the end effector which is not achievable.

Positional differences are easy to visualize; orientational differences are not. It is easy to manipulate a positional goal which is outside of the reachable space of the end effector. We can intuitively think of a spring or rubber band pulling the end effector towards the goal. Orientational differences are much harder to visualize. Even though it may be easy to conceptualize "rotational springs," in practice is it very difficult to apply that intuition to the geometric model. If the goal is very far outside of the reachable space of the end effector along *angular dimensions*, all correspondence between the orientation of the goal and the end effector gets quickly lost. *Jack* illustrates orientational differences through rotation *fans*, shown in Figure 4.1, which are icons to illustrate how an object must rotate to achieve a desired orientation, but no amount of graphical feedback can help when the differences are large. Therefore, it is absolutely essential that the orientation of the goal – the manipulation transform — not deviate too far from the end effector.

Jack solves this problem through an orientation *offset* to the goal which can be adjusted during the manipulation process. This offset is a relative transform which is applied to the manipulation transform to rotate it into the true orientation of the goal as supplied to the inverse kinematics algorithm. The end effector dragging mechanism resets this offset each time a translation or rotation is completed during the manipulation process, i.e. each time a mouse button comes up and the movement stops. This means that each time the user begins to rotate or translate the goal, the manipulation transform starts out from the current orientation of the end effector. This prevents the user from getting lost in the orientation difference.

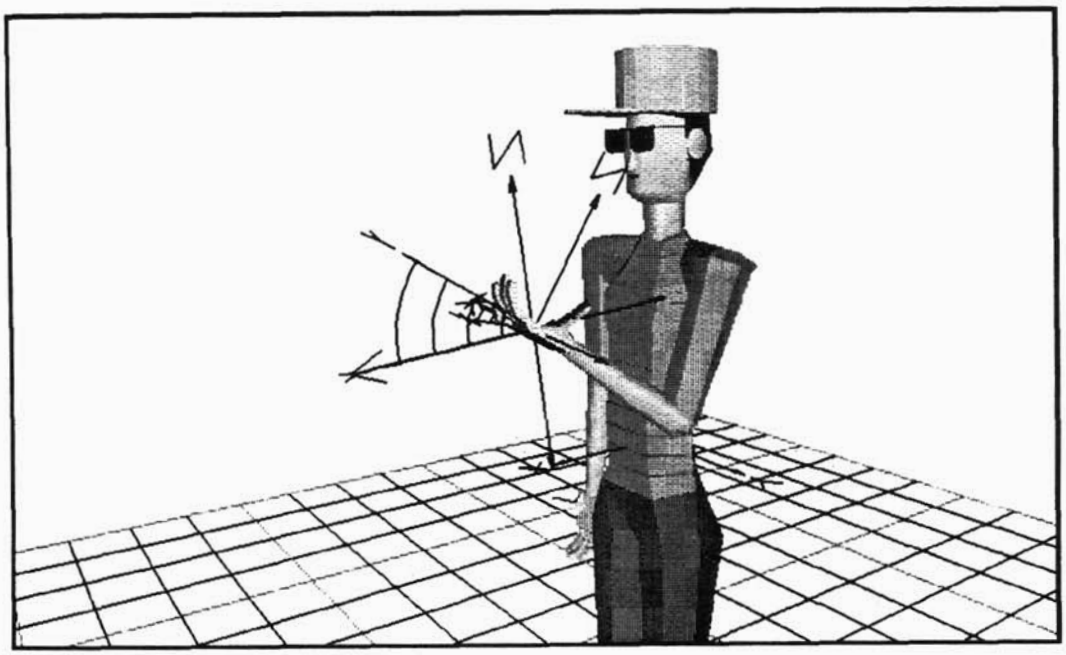


Figure 4.1: The Orientation Constraint Icon

This simulates the effect of a spring-loaded crank which applies a torque to an object, but only as long as the user holds down the mouse button. When the mouse button comes up, the torque disappears so that it doesn't continue to have a undesirable effect. This lets the user control the goal through a ratcheting technique of applying short bursts of rotation.

4.6.3 Manipulation with Constraints

The nature of the 3D direct manipulation mechanism allows the user to interactively manipulate only a single element at a time, although most positioning tasks involve several parts of the figure, such as both feet, both hands, etc. In addition to interactively dragging a single end effector, there may be any number of other kinematic constraints. These constraints are persistent relationships to be enforced as the figure is manipulated using any of the other manipulation tools. By first defining multiple constraints and then manipulating the figure, either directly or with the dragging mechanism, it is possible to control the figure's posture in a complex way.

This mechanism involves another slight modification to the direct manipulation loop, shown in Figure 4.2. Step #4 may cause the end effectors to move away from their goal positions. The inverse kinematics algorithm in step #5 repositions the joints so the goals are satisfied.

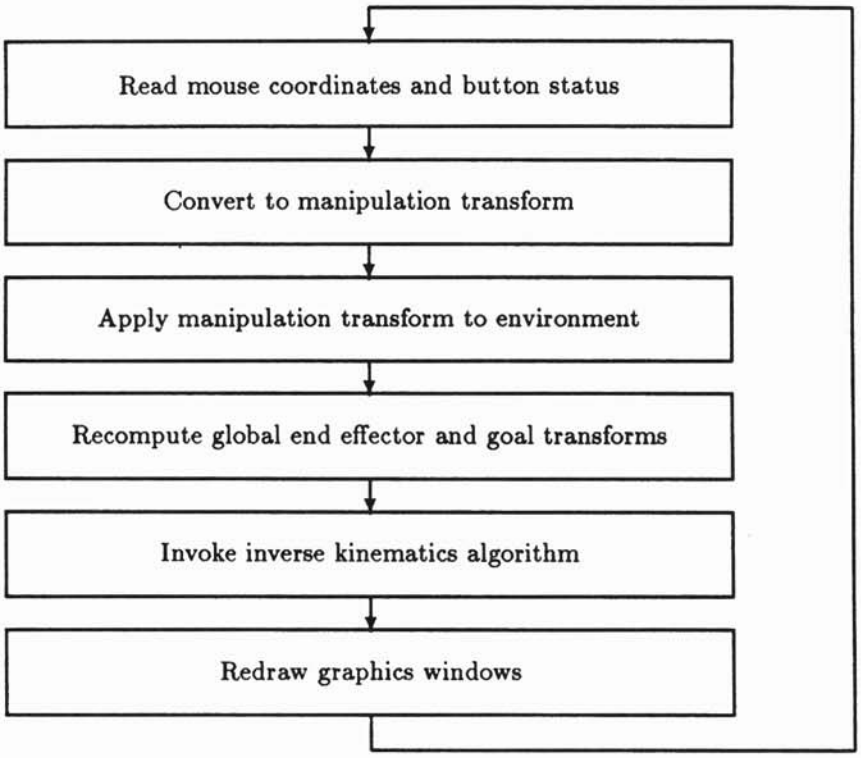


Table 4.2: Manipulation with Constraints

4.7 Interactive Performance Enhancements

I am primarily interested in the interactive nature of the system. The user must have the feeling of real-time control over the figures. A slow screen update rate is detrimental to this sense of interactive control. Unfortunately, the inverse kinematics algorithm can be fairly time consuming when there are several constraints. The lag time between the motion of the mouse and the ensuing motion of the objects makes the manipulation process difficult to control. To alleviate this problem, *Jack* makes two enhancements: it limits the distance which goals are allowed to move from one manipulation time step to the next, and it limits the amount of time which can be consumed by the inverse kinematics algorithm at each interactive iteration.

4.7.1 The Inverse Kinematics Step-Factor

The *step factor* is a clamp on the distance which a goal is allowed to move at each iteration. This ensures that the positioning algorithm never has to move the end effectors very far. Before the constraints are evaluated, *Jack* measures the distance between each end effector and its goal, using the objective function. If it is greater than some threshold, *Jack* interpolates between the current end effector position and the true goal position to determine an intermediate goal, and it supplies this intermediate goal to the inverse kinematics algorithm. The type of interpolation depends upon the objective function, so the step factor can involve position or orientation. The actual goals of the constraints keep their original values, so during the next iteration the same process will take place until all goals are reached. The default step factor is 25cm in distance units and 5° of angular displacement.

This makes the figures move sluggishly, but it guarantees that their movements are well-behaved. For example, in dragging the hand, the user can move the goal for the hand quite rapidly and see the feedback in terms of the goal position icon in real time, but the hand will drift gradually towards it. If the user holds the goal position still for several frames, the hand will eventually reach it, if it is reachable. However, the user is free to move the goal before the hand arrives. This fills the dead time, when the user is just looking at the screen, with useful computation.

4.7.2 The Inverse Kinematics Time-Limit

The inverse kinematics algorithm is iterative, and it converges monotonically, so at each internal iteration the end effectors are closer to their goals. *Jack* exploits this property and accept an intermediate solution if the entire solution cannot be computed quickly enough. Rather than limiting the number of iterations, *Jack* limits the amount of time consumed. *Jack* does this by recording the time at which the algorithm begins, and then checking the current time at the beginning of each iteration. This timing information is available from the operating system in 1/60th's of seconds. If the time limit has expired, *Jack* terminates the algorithm and accept the current configuration. The direct manipulation process then proceeds with the next interactive iteration.

With the time limit set properly, the frame rate never deteriorates beyond several frames per second even with several constraints, so the user never loses the sense of interactive control. The advantage of this technique is that it can also be sensitive to the amount of time consumed at each frame by the drawing of the graphics windows. This works well since when there are many large, complex geometric objects, much of the time consumed by the manipulation loop is spent in drawing the graphics windows. This sensitivity means that the amount of time allotted to the inverse kinematics algorithm is automatically

decreased. *Jack* implements this by keeping a record of how much time is consumed each time the screen is drawn. This timing information is only approximate.

This has an interesting effect on the "feel" of the manipulation, particularly combined with the step-factor process described above. Although the inverse kinematics algorithm is monotonically convergent, it tends to oscillate, meaning that although an end effector will be closer to its goal at each iteration, it may flap back and forth from one side to the other. The step factor calculation minimizes this effect because it means the goals are never very far away. The time limit serves to guarantee that the frame rate never degrades beyond an acceptable level for interaction.

4.8 Summary

There are two main concepts here. The first is Zhao's numerical technique for performing inverse kinematics, and the second is the setting in which this algorithm is applied. Both are equally important. The main points of this chapter are:

- The inverse kinematics algorithm computes positions, not velocities.
- The algorithm treats the figure as a purely geometric entity, not a physical one. It uses a variable-metric optimization technique which minimizes pure functions.
- Postures are defined through "constraints," which constitute potential functions similar to Witkin, Fleischer, and Barr's [90] energy constraints.
- The algorithm models balance through a kinematic constraint on the center of mass.
- The inverse kinematics algorithm has no notion of time. The notion of time in the manipulation system is handled at a higher level.
- One higher level control strategy is interactive dragging and twisting. The manipulation system provides a mechanism for negotiating the overlapping influence of position and orientation.
- The performance enhancing features ensure that interactive rate never degrades beyond an acceptable level.

Chapter 5

Behaviors and Interactive Manipulation

"Now why exactly are you behaving in this extraordinary manner?"

Tom Stoppard, 1937–
Rosencrantz and Gildenstern are Dead

The inverse kinematics system described in the previous chapter provides the computational power behind the marionette strings in the puppet analogy, but now where should the strings be attached? How should their effect be coordinated so that the figure behaves in a reasonable manner? Simply stating that positioning can be done by inverse kinematics is not sufficient. What vocabulary should the user have to control the posture of complex articulated figures?

5.1 Chapter Overview

This chapter describes the basic architecture for an interactive system for postural manipulation with behavioral controls. The details of the currently implemented behaviors, along with the structure of the standard human body used in *Jack*, are given in Appendix D. The behaviors constitute a powerful vocabulary for postural control. The manipulation commands provide the stimuli; the behaviors determine the response.

The chapter culminates with a discussion of the interactive manipulation commands, concentrating on their global effect given all of the interplay between the various behaviors that have been implemented so far. The discussion of the manipulation commands serves as a good example of the power of the entire system of postural control because it addresses the global effect of each manipulation primitive.

5.2 Behavioral Animation

The rationale for using dynamics simulation for computer animation is its economy of description: a simple input from the user can generate very complex and realistic motion. The same rationale applies to behavioral animation. By defining a simple set of rules for how objects behave, the user can control the objects through a much more intuitive and efficient language because much of the motion is generated automatically.

Several systems have used the notion of behaviors to describe and generate motion. The most prominent of this work is by Craig Reynolds, who used the notion of behavior models to generate animations of flocks of birds and schools of fish [72]. The individual birds and fish operate using a simple set of rules which tell them how to react to the movement of the neighboring animals and the features of the environment. Some global parameters also guide the movement of the entire flock. William Reeves used the same basic idea but applied it very small inanimate objects, and he dubbed the result *particle systems*[70].

Behaviors have also been applied to articulated figures. McKenna and Zeltzer [56] describe a computational environment for simulating virtual actors, principally designed to simulate an insect (a cockroach in particular) for animation purposes. Most of the action of the roach is in walking, and a gait controller generates the walking motion. *Reflexes* can modify the basic gait patterns. The stepping reflex triggers a leg to step if its upper body rotates beyond a certain angle. The load bearing reflex inhibits stepping if the leg is supporting the body. The over-reach reflex triggers a leg to move if it becomes overextended. The system uses inverse kinematics to position the legs.

5.3 Concepts of Posture from Movement Notation

Where does knowledge about human behaviors for postural control come from? Much of it seems like common sense, but common sense is difficult to quantify. In a more formal approach, much of this knowledge can come from movement notations such as Labanotation, and also from biomechanics literature.

Although our spoken language is replete with descriptions of motion and posture, a commonly accepted formal language for describing movement does not really exist. Most attempts at developing such a language have been made by choreographers, but none of the attempts at developing a notation for

dance have been as effective as in other arts such as music. The most common notations are Benesh Movement Notation [55], Eshkol-Wachmann Notation [27], and Labanotation [40]. Of the three, Labanotation is the most commonly used because it is the most complete. The syntax of Labanotation is graphical, written on a staff which looks like a vertical time line. The syntax is very complex and difficult to read, so it doesn't serve as a good medium for communication about movement, at least not for novices. Some researchers have attempted to develop animation systems based on movement notations, but none have been very successful. Calvert describes an animation system which uses Labanotation as an input, but it is rather simplistic and non-interactive[19].

Actually, movement notations are much more interesting from the point of view of their semantics, because the languages attempt to develop a set of primitives powerful enough to compose complex movements. One interesting aspect of Labanotation is that even though it is a notation for movement, much of its information is postural, that is, static. The symbols on a time line basically describe a sequence of postures.

Labanotation is interesting here because of its economy. It attempts to describe simple movements very simply, which means that it makes lots of assumptions about how movements are carried out. As long as the movements conform to these assumptions, no extra notations are needed. There are dozens of special symbols for indicating variations to the basic movement, so a complex movement has a necessarily complex syntactic representation. Still, a simple movement has a simple representation, even if it involves the whole human body.

This economy is a nice idea, and I attempt to accomplish the same thing, using some of the assumptions incorporated into Labanotation. In short, Labanotation provides a formalization for reasonable default behaviors for human movement and posture. These are described in detail below and in Appendix D.

5.4 An Interactive System for Postural Control

The human figure in its natural state has constraints on its toes, heels, knees, pelvis, center of mass, hands, elbows, head and eyes. These constraints are the marionette strings in the puppet analogy. They correspond loosely to the columns of the staff in Labanotation, which designate the different parts of the body.

The heart of interactive system is a control loop, shown in Figure 5.1. The system repeatedly evaluates the kinematic constraints and executes the *behavior functions*. It also polls the user for information, which can be geometric movements through the direct manipulation operator described in Chapter 3, or commands to execute which can change the state of the system or the parameters of the constraints. The behavior functions can also modify the state of the environment or the parameters of the constraints, as described below. Each iteration of the control loop is a time step in a simulated movement process, although this is an imaginary sense of time.

There are four categories of controls for human figures, illustrated in Table 5.1. First, there are *manipulation primitives*. These are the commands which allow the user to interactively drag or twist parts of figure, that is, pull on its marionette strings. These are the principal sources of input, the stimuli for the postural adjustments, although much of the movement during a postural adjustment usually comes from the response generated by the behavioral controls. Second, there are *behavioral parameters*. These are the parameters of the body constraints. These govern things like whether the feet should remain planted on the floor or whether they should be allowed to twist as the body moves. Third,

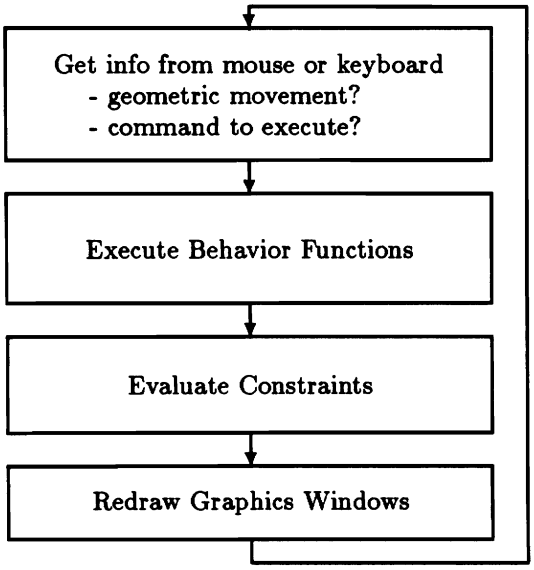


Figure 5.1: The Interactive System Architecture

there are *passive behaviors*. These behaviors express relationships between different parts of the figure. These relationships are usually more complex than can be expressed through the behavioral parameters. An example of a passive behavior is the parametrization of the distribution of weight between the feet. Finally, there are *active behaviors*. Active behaviors have a temporal component. They wait for events or conditions to occur and then fire off a response which lasts for a specified duration. An example of an active behavior is the automatic stepping action the figure takes just before it loses its balance. Table 5.1 provides a good summary of *Jack*'s current vocabulary for postural control.

5.4.1 Behavioral Parameters

The behavioral parameters are the properties of the constraints which model the posture. These are like "tendons" that connect the strings to the puppet. Mostly, these parameters describe the objective functions of the constraints, as in whether the constraint specifies position, orientation, or both. The parameters can include the goal values of the constraints as well. These are simple relationships which require no computation on the part of the system.

Section D.1 in the appendix describes *Jack*'s individual behavioral parameters in detail. To summarize, they give control over the position and orientation of the feet, the elevation of the center of mass, the global orientation of the torso, the orientation of the pelvis, and the gaze direction of the head and eyes. There are additional non-spatial controls on the knees and elbows. These controls were originally designed for the human figure in a rather ad hoc fashion, although they correspond loosely to the columns of the staff in Labanotation. Designing a set of parameters for a non-human figure would be straightforward although there is no formal procedure for doing so.

Most of *Jack*'s current behavioral parameter commands, listed in Table 5.1, allow the user to instruct the figure to maintain the current posture of the part of the body that it controls, such as the position and orientation of feet or the elevation of the center of mass. This means that this property will be

manipulation primitives
move foot
move center of mass
bend torso
rotate pelvis
move hand
move head
move eyes
behavioral parameters
set foot behavior
pivot
hold global location
hold local location
keep heel on floor
allow heel to rise
set torso behavior
keep vertical
hold global orientation
set head behavior
fixate head
fixate eyes
set hand behavior
hands on hips
hands on knees
hold global location
hold local location
hand on site
grab object
passive behaviors
balance point follows feet
foot orientation follows balance line
pelvis follows feet orientation
hands maintain consistent orientation
root through center of mass
active behaviors
take step when losing balance
take step when pelvis is twisted

Table 5.1: Behavioral Controls

maintained whenever possible even as the other parts of the figure change posture. This works well in an interactive manipulation context: the user manipulates the body part into place, and it stays there. This is the "what you see is what you get" approach. This also obviates the need for a complex syntax for describing positions and orientations grammatically, since the method of describing the location is graphical and interactive, through direct manipulation, that is, WYSIWYG.

5.4.2 Passive Behaviors

Passive behaviors can represent more complex relationships than the behavioral parameters. They are like little processes attached to each figure. The passive behavior functions are executed at each interactive iteration. A passive behavior can involve a global property of the figure such as the center of mass or the shape of the figure's support polygon. An example of this kind of behavior is the parametrization of the distribution of the weight between the figure's feet: when the feet move, the balance behavior function must compute the proper location for the balance point and register this with the constraint on the center of mass. Passive behaviors are *instantaneous* in that they explicitly define a relationship to be computed at each iteration.

Passive behaviors are easy to implement in this basic system architecture because their only job is to compute the necessary global information and supply it to the behavioral controls. Because of the general nature of the inverse kinematics constraints, the behaviors can overlap to a degree not possible with other systems, like Zeltzer's local motion processes [93].

Currently, I have implemented six basic passive behavior functions for human figures, and they illustrate a range of capabilities. The details of these behavior functions are described in Section D.2 in the appendix. They control the location of the balance point, the orientation of the feet, the orientation of the pelvis, and the orientation of the hands. The final two behaviors control the figure root.

5.4.2.1 Balance as a Passive Behavior

As I stated in the Introduction, probably the most important human behavior, and the one demanding the most coordination, is balance. The need to remain balanced dictates much of the subtle and elusive behavior of a human figure. The location of the balance point of a figure is significant in both cause and effect. The location of the balance point is dependent on other parts of the figure, namely, the feet. Also, the balance point sends information to the other parts of the body regarding the figure's state of balance. In order to handle this, any system of behaviors needs a good vocabulary for balance.

To parametrize the location of the balance point with respect to the feet, I suggest the idea of a *balance line*, which is the line between a fixed reference point in the middle of each foot. Biomechanics literature [20] states that in the standing rest position, the body's vertical line passes 2-5cm in front of the ankle joint, midway through the arch of the foot. This line between the feet divides the support polygon down the middle.

Given the location of the center of mass, the balance point parameters, call them x and z, can be determined as shown in Figure 5.2. To do this, project the balance point on the $y = 0$ plane and call the point $\hat{\mathbf{b}}$. Then find the point on the balance line closest to this point, and call it $\hat{\mathbf{p}}$. z is the distance between $\hat{\mathbf{b}}$ and $\hat{\mathbf{p}}$, that is, the balance point's distance forward from the balance line. However, it is more convenient to normalize z between 0.0 and 1.0 according to the placement of $\hat{\mathbf{b}}$ between the balance line and the front edge of the support polygon. Therefore, if $z > 1$ then the balance point lies outside the support polygon. If the balance point is behind the balance line, then let z be normalized between -1.0

and 0.0. Likewise, x is the interpolation factor which gives $\hat{\mathbf{p}}$ in terms of the left and right foot reference points, normalized between 0.0 and 1.0, with $x = 0$ being the left foot. If x is outside of the $[0, 1]$, then the balance point is to the side of the support polygon.

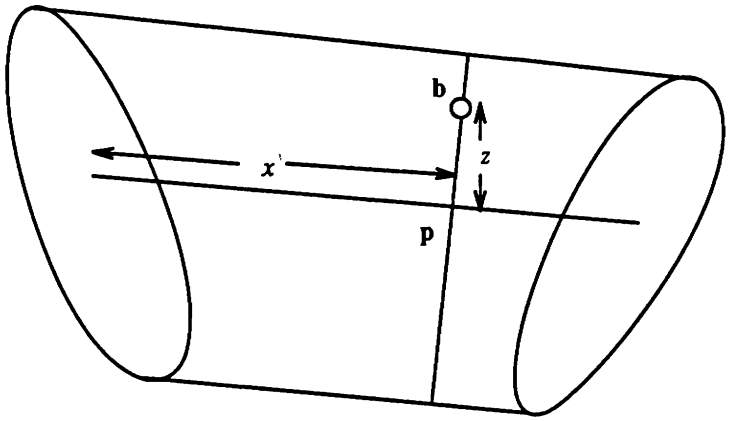


Figure 5.2: The Parametrization of the Balance Point

Once the system has the ability to measure balance, these parameters are available for the behavior functions to use. The *balance point follows feet* behavior, described in Section D.2.1, falls directly out of this parametrization. This behavior causes the distribution of weight between the feet to remain constant even as the feet move. The active stepping behavior *take step when losing balance*, described in Section D.3, uses this parameter as its trigger.

5.4.2.2 Global Effects of Local Manipulations

Another capability of the passive behaviors in this system is to telegraph changes in the posture of a local part of the figure to the rest of the figure as a whole. This can provide coordination between the different parts of the figure. The behavioral parameters as described above generally hold the different parts of the figure in place, but sometimes it is better to have them move automatically. A good example of this is the *pelvis follows foot orientation* behavior, described in Section D.2.3, in which the orientation of the pelvis automatically adjusts to the orientation of the feet. Whenever the feet change orientation, they radiate the change to the pelvis which mimics the rotational spring-like behavior of the legs.

5.4.2.3 Negotiating Position and Orientation

The passive behaviors offer a solution to the problem of negotiating the overlapping influence of position and orientation while interactively dragging part of the body. Because of the nature of the direct manipulation technique described in Chapter 3, it is not possible to rotate and translate during a single movement of the mouse. This has come up before, in Section 4.6.2: either the dragging procedure has *no* control over orientation, in which case the orientation is arbitrary and unpredictable, or the dragging procedure *does* have control over orientation, in which case the orientation remains globally fixed during spurts of translation. Fixing the orientation during translation can, for example, cause the hand to assume an awkward orientation as it is translated.

Passive behavior functions allow the direct manipulation operator to have control over the orientation and avoid awkward orientations. While the user is translating with the mouse, the behavior function can automatically determine a suitable orientation based on heuristic observations. While rotating, the user has complete control over the orientation. The heuristics can simply be embedded in the behavior functions.

The pair of behaviors *foot orientation follows balance line* and *hands maintain consistent orientation*, described in Section D.2 of the appendix, use heuristics taken from Labanotation to predict suitable orientations for the hands and the feet during their manipulation. This allows the user to position them mostly by translating them, making changes to the orientation only as necessary.

5.4.2.4 The Figure Root

One passive behavior deserves special attention: the figure root. Appendix A discusses the *peabody* object representation and the nature of its hierarchy. The principal disadvantage of modeling an articulated figure as a hierarchy is that one point on the figure must be designated as the figure root. Section 4.4 explains the effect of the figure root on the inverse kinematics algorithm: the positioning algorithm itself cannot move the figure root. It can only manipulate chains emanating from the root. Any movement of the figure root must be programmed explicitly. Therefore, a major element of *peabody* is the ability to change the setting of the figure root when necessary.

The figure "root" is an unnatural concept. It has no natural analog for a mobile figure like a human being, so it has no place in the language for controlling human figures. Since it is a necessary internal concept, can it be controlled internally as well? For certain postures of a human figure, there are distinct reference points on the figure which serve as good figure roots: the feet, the lower torso, and the center of mass. It should be possible to have the system choose these automatically and thus make the root transparent to the user.

There are several possibilities for the figure root of a human figure. Many systems which don't have the ability to change the root choose to locate it at the lower torso, like TEMPUS[4]. However, this complicates the process of moving the lower torso during balance adjustments. Using this approach, it can be very difficult to get the figure to bend over convincingly because the hips need to shift backwards and downwards in ways that are difficult to predict. However, for a seated posture, the lower torso is a good choice for the root. When a figure is standing, the feet are natural choices for the root.

The choice of the figure root can be handled by designing a behavior function which monitors the figure's posture and automatically changes the figure root when necessary to provide the best behavior. This behavior function uses the following rules:

- It roots the figure through a foot whenever the weight of the body is more than 60% on that foot. This ensures that if the figure is standing with more weight on one leg than the other, the supporting leg serves as the root. It also ensures that if the figure is standing with weight equally between the two legs but possibly swaying side to side that the root doesn't rapidly change between the legs.

- If the height of the center of mass above the feet dips below 70% of the length of the leg, then the root changes to the lower torso. This predicts that the figure is sitting down. Heuristically, this proves to be a good choice even if the figure is only squatting, because the constraint on the non-support leg tends to behave badly when both knees are bent to their extremes.

Section D.2.6 in the appendix describes an alternative rooting formulation that is based on rooting the figure through the center of mass permanently. This approach has not proved to be effective in practice.

5.4.3 Active Behaviors

Active behaviors mimic reflexive responses to certain conditions in the body. They can have temporal elements, so they can initiate movements of parts of the body which last for a certain duration. These behaviors make use of the notion of a *motion primitive*. A motion primitive has a distinct duration in terms of interactive iterations, and it typically involves a constraint which changes over this time interval. An example of this is the stepping movement of the feet which is initiated when the figure's center of mass leaves its support area. The interactive system architecture maintains a list of currently triggered active behaviors, and it advances them at each iteration until they are complete. The behaviors terminate themselves, so the duration can be explicit in terms of a number of interactive iterations, or they can continue until a certain condition is met.

Active behaviors are like motor programs, or *schemas* [46, 73, 77, 76]. Considerable physiological and psychological evidence suggests the existence of motor programs, which are preprogrammed motor response to certain conditions. The theory of schemas suggests that humans and animals have catalogs of preprogrammed motor responses that are fired off to produce coordinated movements in the body. Schemas are parametrized motor programs which can be instantiated with different settings. For some motor programs, there even seems to be very little feedback involved. Evidence of this comes from experiments which measure the excitation of the muscles of the arm during reaching exercises. The patterns of excitation remain constant even if the movement of the hand is impeded [73].

The incorporation of active behaviors into the postural control process begins to blur the distinction between motion and manipulation. The purpose of the behaviors is predictive: if the user drags the center of mass of a figure away from the support polygon, this probably means that the desired posture has the feet in a different location. The job of the active behavior is to anticipate this and hopefully perform the positioning task automatically.

I have implemented two active behaviors, both involving the placement of the feet. The *take step when losing balance* and *take step when pelvis is twisted* behaviors, described in Section D.3, automatically reposition the feet just before the figure loses its balance. They use the balance point parameters described above as their triggers. As I described in the Introduction, the purpose of these behaviors is to predict a proper posture for the figure given that its center of mass is leaving the support polygons.

Active behaviors can be used to simulate movement even in the context of postural control. As I mentioned in the Introduction, the entire process of interactive postural control can serve as a good approximation to motion anyway. Although the purpose of this work is not to address motion issues, the active behaviors provide a way in which motion primitives can be incorporated into the interactive system. To do this more effectively, the interactive system needs a more sophisticated notion of time and timed events. This work is currently being done by Jeffrey Esakov[26].

5.5 Interactive Manipulation With Behaviors

This section discusses *Jack*'s manipulation primitives, but in the process it describes the entire manipulation process, including the effect of all of the implemented behaviors. The effect of the manipulation

commands cannot be treated in isolation. In fact, the very nature of the system of behaviors implies that nothing happens in isolation. This discussion serves as a good summary of these techniques because these are the commands which the user uses the most. These are the verbs in the postural control language.

move foot
move center of mass
bend torso
rotate pelvis
move hand
move head
move eyes

Table 5.2: The Manipulation Primitives

5.5.1 The Feet

The feet can be moved with the active manipulation command move foot. This command allows the user to drag the foot interactively. This automatically transfers the support of the figure to the other foot, provided the figure is standing. The control over the position of the feet is straightforward. The manipulation operator also gives control over the orientation. However, while translating the foot, its orientation depends upon the foot orientation behavior. The default behavior maintains a constant global orientation. The *foot orientation follows balance line* behavior causes the orientation of the foot to remain fixed with respect to the balance line during translation. This means that if the foot goes forward, it automatically rotates as if the figure is turning toward the direction of the stationary foot.

The move foot command automatically causes a change in the balance point according to the *balance point follows feet* behavior, which is the default. This means that the distribution of weight between the feet will remain constant as the foot moves. The location of the balance point within the support polygon, both side to side and forwards/backwards, will remain fixed as the support polygon changes shape. This is evident in Figure 5.3. The balance point shifts along with the foot. If this behavior is disabled, the balance point will remain fixed in space.

Manipulating the feet also telegraphs a change to the pelvis according to the *pelvis follows foot orientation* behavior, which is the default. This means that as the foot rotates, the pelvis automatically rotates as well. This keeps the body turned in the direction of the feet.

5.5.2 The Center of Mass and Balance

The move center of mass command allows the user to interactively drag the balance point of the figure, shifting its weight back and forth or forward and backward. This command changes the parametrization of the balance point in terms of the feet. If the *balance point follows feet* behavior is active, then when the move center of mass command terminates, the balance point will remain at its new location relative to the support polygon.

The location of the balance point has a great effect on the feet. If the foot behavior is *pivot*, then

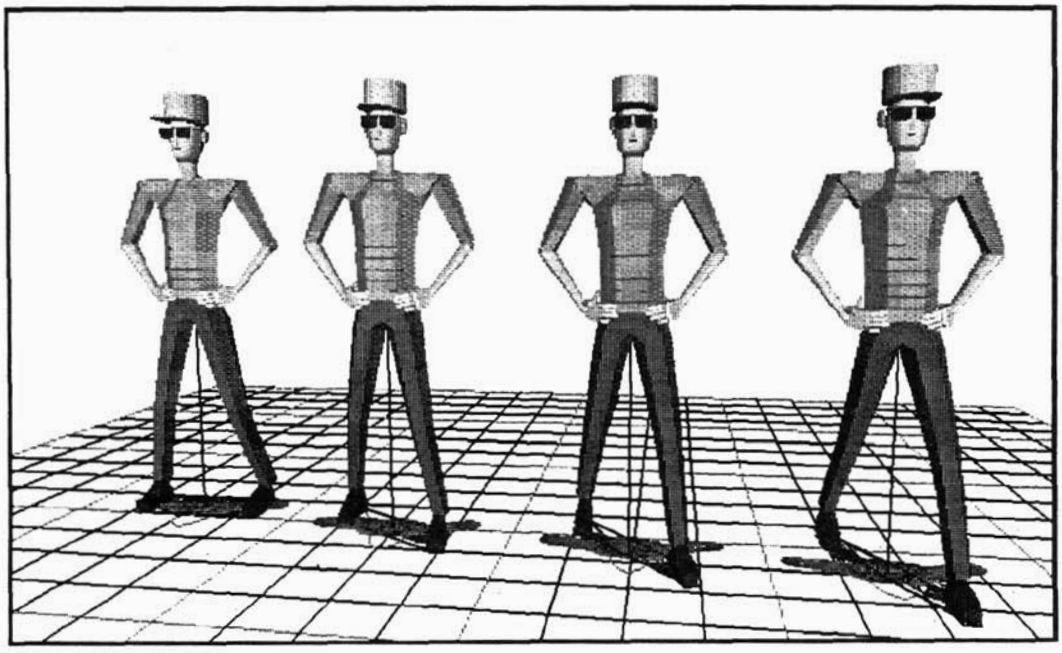


Figure 5.3: Moving the Left Foot, With Balance Point Following Feet

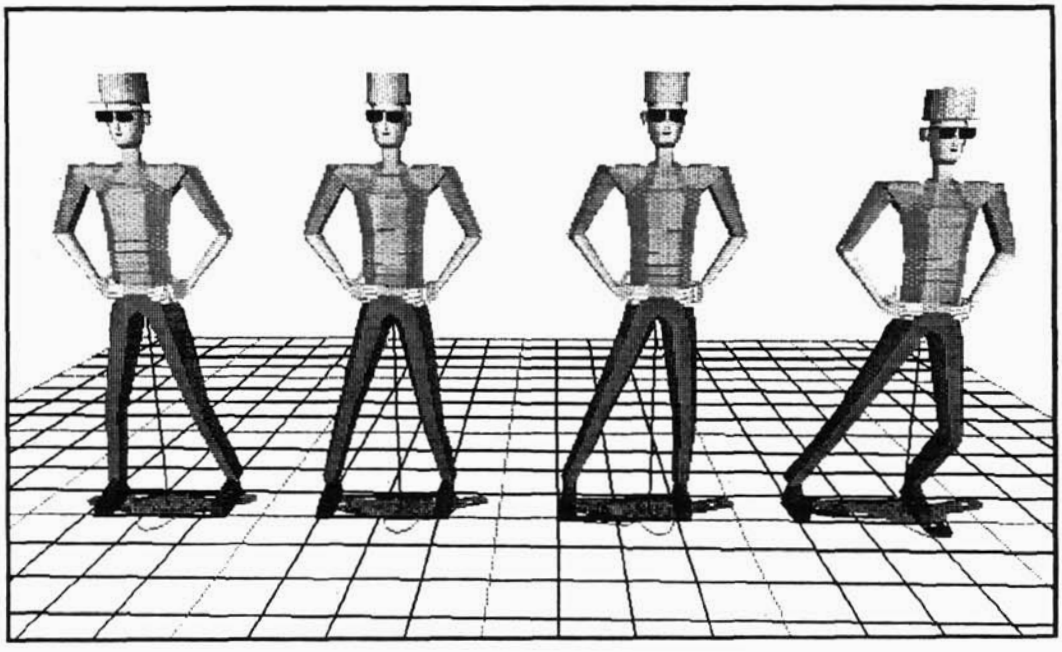


Figure 5.4: Shifting the Center of Mass

shifting the weight laterally back and forth will cause the feet to twist back and forth as well. On the other hand, if the feet do not pivot, then they remain planted, possibly inhibiting the movement of the balance point. In Figure 5.4, the feet are held in place, not pivoting.

The move center of mass command also gives control over the elevation of the center of mass. Normally, the elevation of the center of mass is not controlled explicitly, except through the *hold current elevation* behavior option to the *set balance behavior* command. The move center of mass command gives control over the elevation, so moving the center of mass up and down allows the figure to stand on its tip-toes or squat down. Figure 5.5 shows the center of mass being lowered into a swatting posture. The constraint on the pelvis ensures that the hips remain square and straight.

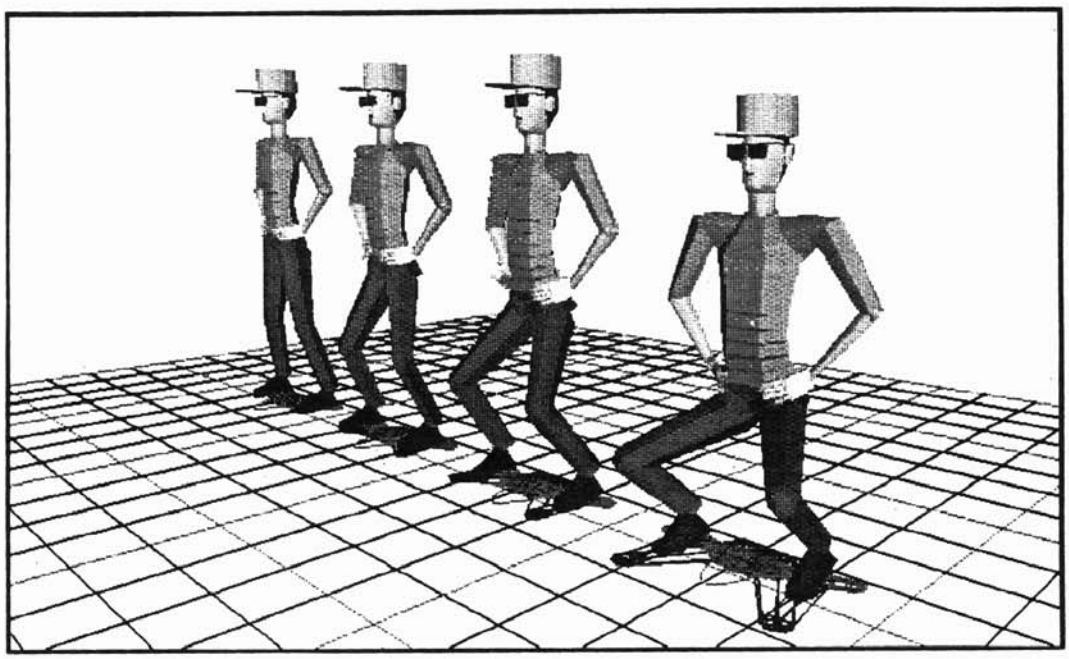


Figure 5.5: Lowering the Center of Mass

The movement of the center of mass also tends to trigger the rooting behavior. This is mostly transparent, but to the trained eye, it is apparent in the movement of the feet. The support foot (the rooted one) is always very stationary.

The manipulation of the center of mass is the main instigator of the active stepping behavior. While the stepping behavior is active, if the balance point reaches the perimeter of the support polygon, the feet are automatically repositioned by the stepping behavior. Figure 5.6 illustrates the stepping behavior as the center of mass is dragged forward.

5.5.3 The Torso

The bend torso command positions the torso using forward kinematics, without relying on a dragging mechanism. This manipulation interface was developed by Gary Monheit [58]. It consists of potentiometers which control the total bending angle along the three degrees of freedom. The command also

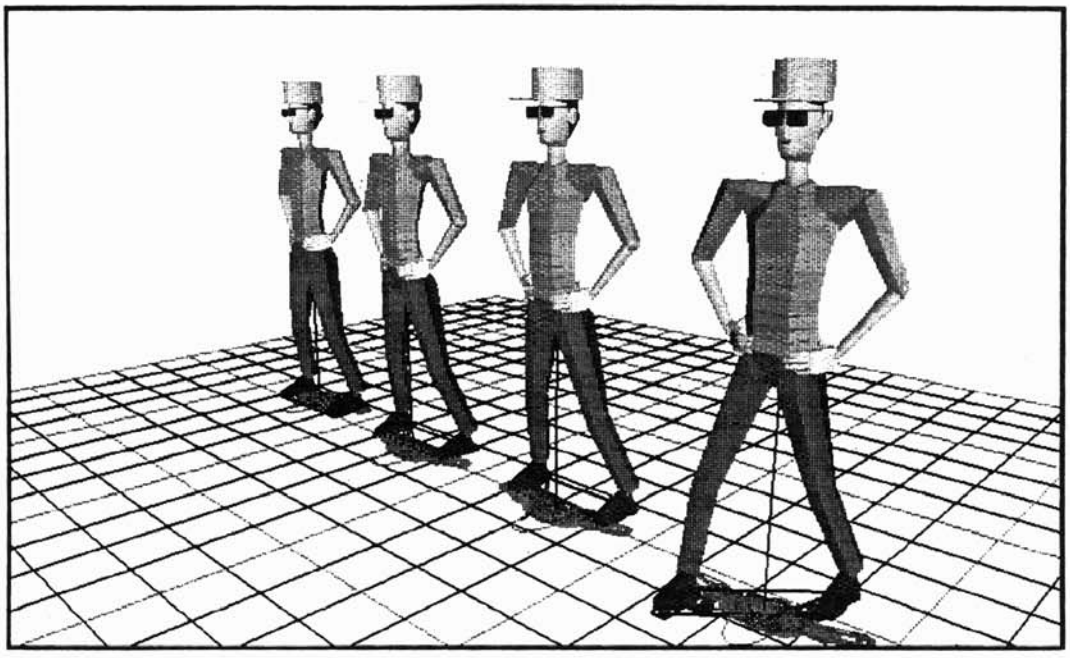


Figure 5.6: Taking a Step Before Losing Balance

prompts for the flavor of bending. These controls are the same as for the set torso behavior command described above. They include options which specify the range of motion of the spine, defined through a top and bottom joint, along with *initiator* and *resistor* joints which control the weighting between the vertebrae.

Bending the torso tends to cause large movements of the center of mass, so this process has a great effect on the posture of the figure in general, particularly the legs. For example, if the figure bends forward, the hips automatically shift backwards so that the figure remains balanced. This is illustrated in Figure 5.7.

5.5.4 The Pelvis

The rotate pelvis command changes the global orientation of the hips. This can curl the hips forwards or backwards, tilt them laterally, or twist the entire body around the vertical axis. The manipulation of the pelvis also activates the torso behavior in a pleasing way. Because of its central location, manipulations of the pelvis provide a powerful control over the general posture of a figure, especially when combined with the balance and *keep vertical* torso constraints. If the torso is kept vertical while the pelvis curls underneath it, then the torso curls to compensate for the pelvis. This is shown in Figure 5.8.

The rotate pelvis command can also trigger the active stepping behavior if the orientation reaches an extreme angle relative to the feet.

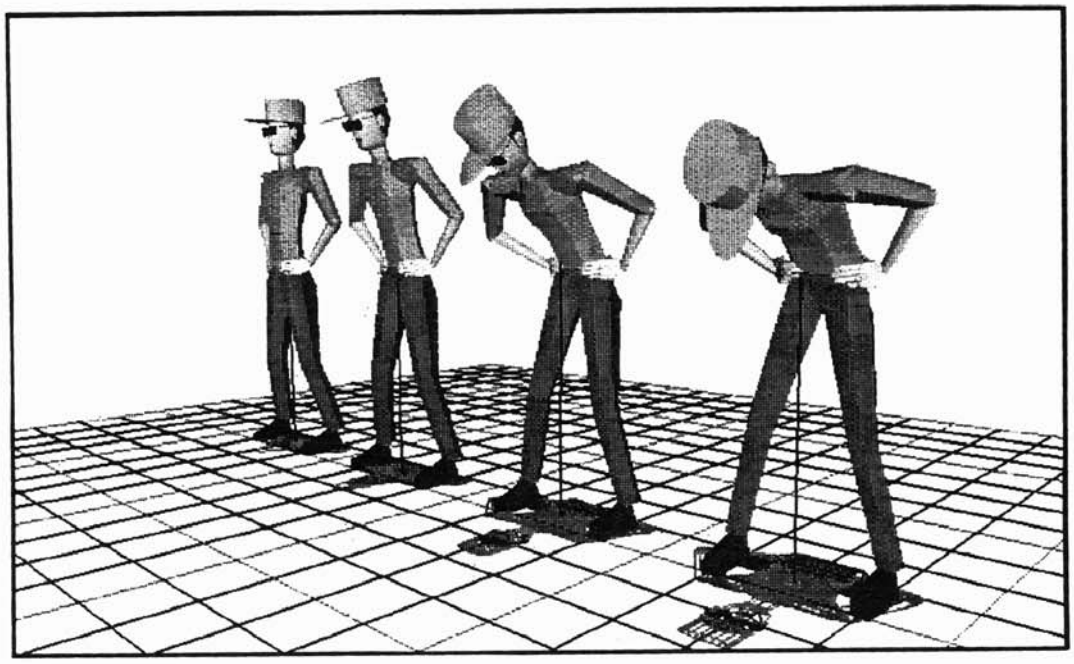


Figure 5.7: Bending the Torso While Maintaining Balance

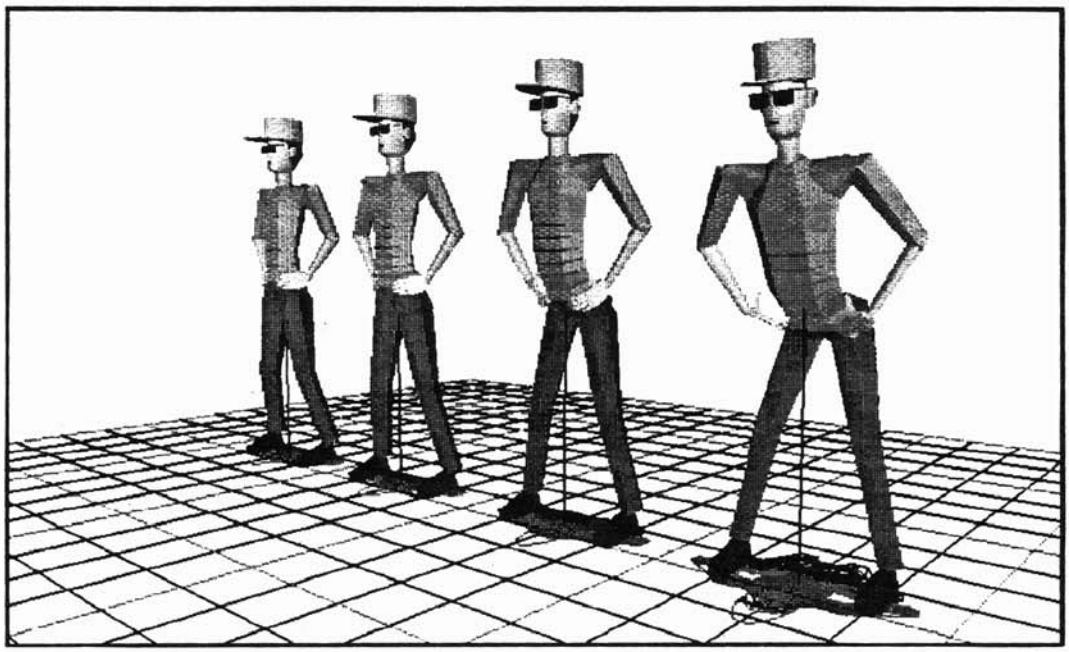


Figure 5.8: Rotating the Pelvis While Keeping the Torso Vertical

5.5.5 The Arms and Hands

The active manipulation of the arm allows the user to drag the arm around in space using the mechanism described in Section 4.6.1. The standard human figure model in *Jack* represents the arm as a two degree-of-freedom clavicle joint, a three DOF shoulder joint, a one DOF elbow, and a three DOF wrist. The movement of the clavicle and shoulder are coupled together using a mechanism developed by Jianmin Zhao [94], using biomechanics literature gathered by Ernest Otani [62]. Using Zhao's coupling scheme, the clavicle and shoulder together have a total of three degrees of freedom. Figure 5.9 shows the left hand being moved forwards.

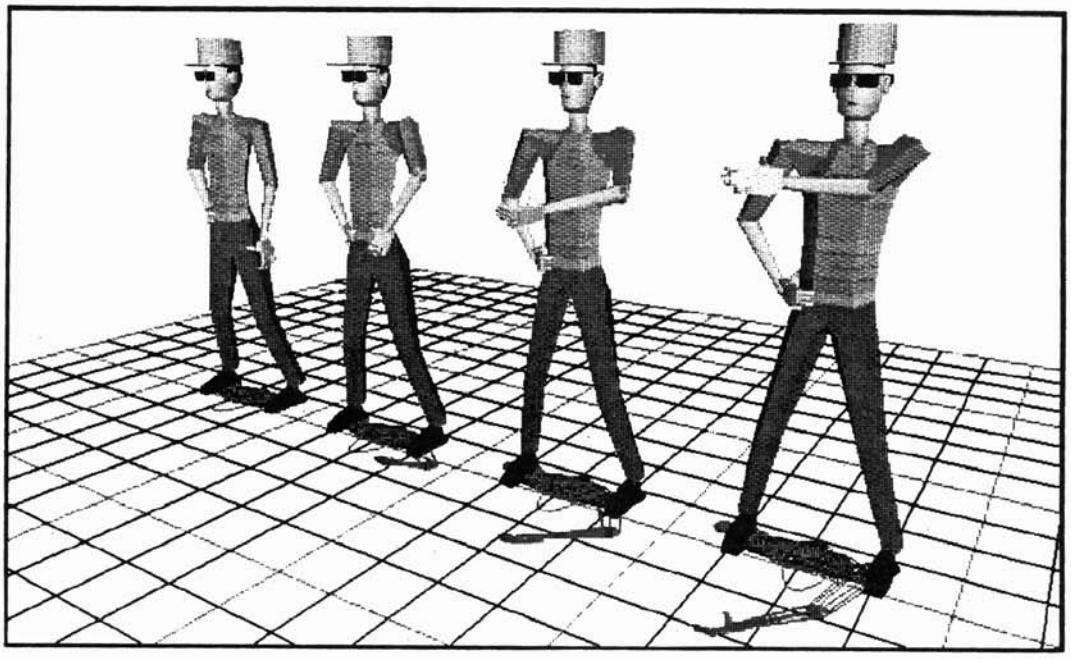


Figure 5.9: Moving the Hand

Although it seems natural to drag this limb around from the palm or fingertips, in practice this tends to yield too much movement in the wrist, and the wrist frequently gets kinked. The twisting scheme helps, but the movements to get the wrist straightened out can interfere with an acceptable position for the arm. It is much more effective to do the positioning in two steps, the first positioning the arm with the wrist fixed, and the second rotating the hand into place. Therefore, our active manipulation command for the arms can control the arm either from a reference point in the palm or from the lower end of the lower arm, just above the wrist. This process may actually simulate how humans reach for objects, for there is evidence that reaching involves two overlapping phases, the first a ballistic movement of the arm towards the required position, and the second a correcting stage in which the orientation of the hand is fine-tuned [73].

5.5.6 The Head and Eyes

The move head and move eyes commands manipulate the head and eyes, respectively by allowing the user to interactively move a fixation point. The head and eyes both automatically adjust to stay fixated on the reference point. The head and eyes rotate as described in Section D.1.4.

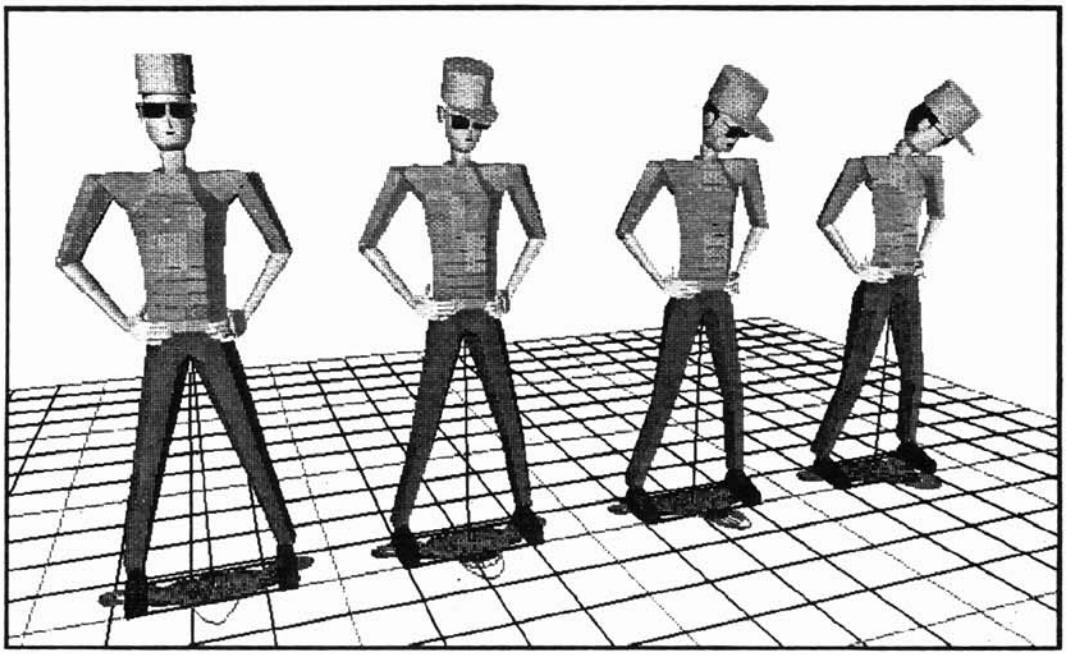


Figure 5.10: Moving the Head

5.6 Summary

The vocabulary for interactive postural control includes manipulation primitives which allow the user to push, poke, and twist parts of the body, and behaviors controls which govern the body's response. The manipulation commands are sufficiently intuitive to provide good handles on the figure, and the behavioral controls make the responses reasonable.

The structure of the behaviors for human figures did not come out of a magic hat. The rationale behind the behaviors comes partially from biomechanics and physiology literature, and partially from the semantics of movement notations such as Labanotation. Labanotion provides a good set of default values for the behaviors because it incorporates so many assumptions about normal human movement.

Chapter 6

Conclusions

James Foley and Victor Wallace eloquently expressed the notion of an appropriate *language* for computer graphics communication, and their words seem particularly relevant to the task of developing an appropriate language for interacting with articulated figures. I believe this passage is a good litmus test for any interactive system:

> It is essential also that the language be efficient, complete, and have a natural grammar. An efficient language is one which conveys ideas effectively and concisely. A complete language permits expression of any idea relevant to the domain of discourse. A natural grammar offers few awkward constraints to the expression of ideas using the fundamental devices and symbols which are available. A natural grammar also permits the system to be useful with a minimum of user training. An interaction language should allow the user to concentrate on the semantics of what he intends to express. The distractions and discontinuities injected into his conscious thought processes by the language's syntax and vocabulary must be minimal or nonexistent. Only if this goal is achieved can a user become engrossed in productive man–machine communications [30].

I believe that my system of behaviors and manipulation commands is *efficient* because it provides a small but powerful vocabulary for describing postures. The behaviors magnify the effect of the seven basic manipulation commands so that relatively few invocations of the commands are necessary to accomplish a complex positioning task. Keyframe and joint-positioning systems pale by comparison: *Jack*'s manipulation commands control many degrees of freedom simultaneously in a coordinated fashion, whereas joint-based systems require tediously manipulating each individual joint. Each manipulation command in *Jack* controls a relatively localized region of the body but transmits movement to other parts as necessary. *Jack*'s ability to define persistent geometric relationships means that once relationships are established, they are maintained automatically, minimizing the need to reposition parts of the figure which are disturbed after an initial positioning step. The passive behaviors automatically enforce complex relationships involving global properties like the center of mass. The active behaviors provide an extra level of efficiency by anticipating the action of the user and predicting an appropriate posture.

The system of behaviors is *natural* because the manipulation commands correspond to well known action verbs — move foot, bend torso, etc. — and the behaviors have intuitive interpretations that roughly correspond to a vocabulary we would expect to use for real human beings — *keep torso vertical*. Unnatural concepts like the figure root are hidden from the user and handled automatically, transparently providing the user with the best possible performance. The instructions are goal-directed, not in the vocabulary

of the implementation. Once again, joint-based manipulation systems pale in comparison because they force the user to think in joint coordinates. The direct manipulation technique makes it easy to translate and rotate in three dimensions. The "what you see is what you get" feel of the manipulation commands means that the user communicates graphically, by pointing and dragging rather than describing.

The system of behaviors in its current state is capable of describing a wide range of postures, so for its intended application, it is relatively *complete*. This is especially true compared to other techniques such as keyframe systems or the dynamic simulation systems described in Chapter 2. However, there are many other features and behaviors which could be implemented, so the completeness should be judged in terms of its *extensibility*. The basic architecture of behavioral parameters, passive behaviors, active behaviors, and manipulation primitives can accommodate many other kinds of behaviors not yet envisioned.. Because of the generality of the way in which the postures are defined through potential functions, constructing sets of constraints which give desired behaviors, and designing behaviors functions to control them, is quite straightforward now that the basic architecture is in place.

6.1 Contributions

This work is unique in its detailed focus on interactive postural control. Although many researchers have addressed the issue of motion control for animation, most seem to have regarded the problem of describing basic postures as trivial. For simple object models, indeed it may be trivial, but for complex models such as human figures, the complexity can overwhelm a naive approach to manipulation.

The problem of postural specification is ubiquitous in computer animation. Every keyframe animation system must have some mechanism of defining the key poses. Many of the very best motion generation techniques, like Witkin and Kass's spacetime constraints[92], rely on static postures for the definition of the movement, but the techniques themselves do not help in the posture specification process. My research does not *compete* with these techniques because it focuses on static positioning instead of motion description. It does, however, complement them.

As I stated in the Introduction, one of the principal applications of this research is in human factors analysis, and for this problem, interactive postural control is particularly well suited. Because many types of human factors analyses — reach, fit, and visibility — involve only static postures, motion is really not necessary. Certainly, a good human factors analysis system will not concentrate on animation capabilities to the exclusion of interaction techniques. I believe that *Jack* demonstrates that if the postural control process is effective enough, it can substitute for motion generation in many situations.

Systems which have addressed interactive manipulation such as the interactive dynamics system of Witkin, Gleicher, and Welch [91], and the *Manikin* system of Forsey and Wilhelms [31] have not developed their techniques beyond the ability to specify world-space goals. Witkin et al's technique does not apply to articulated figures.

Physically-based modeling systems address only half of the positioning problem. They present nice numerical primitives, but they fail to suggest how the primitives can be assembled to perform useful work. For simple positioning tasks involving only several objects, this is not a problem, but for articulated figures with many degrees of freedom, the numerical tools begin to suffer from the computational load, and the conceptual gap between the task which the user wants to accomplish and the tools which are available grows very wide. This is particularly true with human figures because as observers of human movement and posture, our eyes are very keen. If a human figure is standing awkwardly, the fault is immediately obvious. It strikes me as much more difficult to find fault with the posture of a collection

of sticks.

My system of behaviors provides the necessary higher level of control, and thus suggests to the user in a much more natural vocabulary how to accomplish a positioning task. Although the notion of behaviors and behavior functions has been applied to animation, it has never been applied to the postural control process.

I believe that my work demonstrates that inverse kinematics is a much more powerful tool for postural control than has ever been realized before. Although inverse kinematics has been used to position limbs, as in Michael Girard's PODA [35, 34], it has never before been applied to the positioning of highly redundant systems like human figure models as a whole.

This work is original in its interactive control of the center of mass through inverse kinematics. This technique is critical because it models balance, and balance has an overwhelming influence on both posture and movement. My experience with the interactive balance behaviors has convinced me that many of the subtle and elusive aspects of human movement fall naturally out of the balance constraint.

Concerning the basic 3D interaction technique, most discussions of 3D direct manipulation have not addressed the special issues of articulated figures. This is true of Evans, Tanner, and Wein [28], Nielson and Olson [60], Chen et al [21], and Bier [11, 10, 12]. Although my technique has a similar flavor to these other systems, I believe I have applied it with much greater rigor in an integrated geometric setting. Each of these manipulation techniques is also sensitive to the viewing angle, but none have integrated the viewing controls into the manipulation process. I believe that my automatic viewing technique could augment these other techniques.

Human factors analysis systems have failed to achieve any great degree of interactivity. Few of these systems outlined in Chapter 2 even provide much direct manipulation. As Beevis notes, no existing system sufficiently incorporates internal rules governing posture [8]. I believe that my system demonstrates that it is possible to do so.

In summary, the major contributions of this work are:

- A novel technique for 3D direct manipulation that functions well in an integrated geometric environment and integrates control of the virtual camera into the manipulation process.
- A technique for applying inverse kinematics to the positioning of highly redundant articulated figures.
- An technique for modeling balance in articulated figures using inverse kinematics.
- The notion of kinematic behaviors for postural control;
- A rich vocabulary for describing complex postures interactively and graphically.

6.2 Future Work

This system of behaviors implemented so far is designed mostly for standing and sitting figures. I have not addressed postures involving more complex supports, such as lying down or kneeling. It should be relatively straightforward to do this, however. I think that it would only require generalizing the parametrization of the balance point in terms of the support polygon in order to accomodate extra contact points.

Jack's system of behaviors and manipulation commands is geared directly to human figures because it is designed to be a human factors analysis tool. However, I believe the basic idea of manipulation and behavior functions for postural control easily applies to other kinds of figures as well, like dogs or horses. It may be more difficult to design behavior functions for animals because I think we have less intuition about how dogs and horses move than we do about humans, but they probably tend to behave in a similar manner. Doing this for four-legged animals would obviously require extending the balance mechanism as I just mentioned.

This work has concentrated on interactive postural control, which is a basically a static problem, but I believe that the basic architecture of the interactive system based on behaviors can be extended to motion as well. As described in Section 5.4.3, the active behaviors can be used to simulate movement even in the context of postural control. The active behaviors provide a way in which motion primitives can be incorporated into the interactive system. To do this more effectively, the interactive system needs a more sophisticated notion of time and timed events. This work is currently being done by Jeffrey Esakov[26]. This could provide the foundation of a robust implementation of Kalita's motion verbs [44]. It also provides the necessary foundation for natural language-based task level animation specification, like that of Moon Jung [43]. Given basic postural control through behaviors, the vocabulary through which the natural language system can describe movements is much more intuitive and robust.

Appendix A

A Representation for Articulated Figures

"O give me new figures!"

Thomas Haynes Bayly
Quadrille a la Mode

This chapter describes the geometric foundation for interaction and manipulation mechanisms, called *peabody*. The term *peabody* applies to the data structure itself, to the library of routines that operate on the data structure, and the language for describing the objects and storing them in files.

The purpose of this chapter is to explain the rationale behind the kinematic representation used in *peabody*, and to describe how and why it differs from other representations. The details of the implementation of the data structure and library are given in *Programming with Jack* [65]. The syntax of the *peabody* language is described in detail in the *Jack 5 User's Guide* [64].

A.1 Introduction

The increasing interest in recent years in object-oriented systems is largely due to the realization that the design of a system must begin with a deep understanding of the objects it manipulates. This seems particularly true in a geometric modeling system, where the word "object" takes on many of its less abstract connotations. It has long been an adage in the user interface software community that a system with a poorly designed basic structure cannot be repaired by improving the interface, and likewise that a well designed system lends itself easily to an elegant interface.

Peabody represents articulated *figures* composed of *segments* connected by *joints*. The *peabody* data structure has a companion language and an interactive interface in *Jack* for specifying and creating articulated figures. The data structure itself maintains geometric information about segment dimensions and joint angles, but it also provides a highly efficient mechanism for computing, storing, and accessing

various kinds of geometric information. One of the principal tasks requested of *peabody* is to map segment dimensions and joint angles into global coordinates for end effectors.

Peabody was designed with several criteria in mind:

- It should be general purpose. It should be able to represent many types of figures of tree-structured topology. It should not be hard coded to represent a specific type of figure, such as a human figure or a particular robot manipulator.
- It should have a well developed notion of articulation. Rather than concentrating on representations for primitive geometric shapes, *peabody* addresses how such shapes can be connected together and how they behave relative to each other.
- It should represent tree-structured objects through a hierarchy. The inverse kinematics positioning algorithm can calculate and maintain the information necessary to simulate closed loops.
- It should be easy to use. The external user view of the figures should be logical, clear, and easy to understand. Understanding the figures should not require any knowledge of the internal implementation, and it should not require any advanced knowledge of robotics or mechanics.

A.2 Background

This section outlines some approaches to the syntax and semantics of representations for articulated mechanisms, in both robotics and computer graphics.

A.2.1 Kinematic Notations

The most common kinematic representation in robotics is the notation of Denavit and Hartenberg [63, 23]. This representation derives a set of parameters for describing a linkage based on measurements between the axes of a robot manipulator. The notation defines four parameters that measure the offset between subsequent coordinate frames embedded in the links, or segments: 1) the angle of rotation for a rotational joint or distance of translation for a prismatic joint; 2) the length of the link, or the distance between the axes at each end of a link along the common normal; 3) the lateral offset of the link, or the distance along the length of the axis between subsequent common normals; and 4) the twist of the link, or the angle between neighboring axes. The notation prescribes a formal procedure for assigning the coordinate systems to the links in a unique way.

Berthold K.P. Horn outlines several things that a kinematic notation must provide [39]:

- A standard coordinate system within each link of the chain.
- The means for determining the transformations between these coordinate systems
- Methods for using these transformations in the analysis of kinematics, statics, and dynamics.

Horn further points out that there are several arbitrary choices that must be made in assigning the coordinate systems in the Denevit & Hartenberg notation when certain types of axis alignments occur. He derives a modified form of the notation that eliminates these choices. There are alternative formulations

that provide greater parametric continuity by allowing more flexibility in assigning the zeros of the joint rotations [96].

The objective behind these kinematic notations in robotics is to develop a standard representation that all researchers can use in the analysis and description of manipulators. There are several types of manipulators, namely the Puma arm and the Stanford arm, that are extremely common in the robotics research community. The adoption of a standard representation would greatly simplify the process of analyzing and implementing robotics algorithms since so many algorithms are described in the literature using these manipulators.

A.2.2 Drawbacks of Kinematic Notation

In computer graphics, there is no great need for standardization and formality, since there is seldom any need to describe mechanisms in literature, and there are no commonly used mechanisms with which most researchers are familiar. Object modeling in computer graphics is a process of synthesis as well as analysis. Therefore, the main criteria for a kinematic notation involve two factors:

- how easy is it to describe a mechanism in the system?
- how efficient it is for the system to perform geometric operations on the mechanism?

The process of object modeling is that of deriving representations from scratch, so this should be as straight forward as possible. For this purpose, Denevit & Hartenberg notation is not well suited since the parameters can be very difficult to interpret. In fact, Horn points out that mistakes in the application of Denevit & Hartenberg notation are quite common [39].

The second factor deals with computational efficiency. Much work in robotics deals with developing efficient representations for rotations. In most situations, a matrix representation for rotations would not be a good choice because matrices are more expensive to multiply. However, computer graphics software generally needs the matrix representation because many graphics workstations and graphics libraries are matrix-based. This is true of the Silicon Graphics IRIS Workstation and its Graphics Library, which is the primary platform for the implementation of *Jack*. Since the IRIS workstation requires the matrices for use with the drawing routines, any computational advantage in performing multiplications in some other representation would likely be lost in the conversion to matrices. In this case, the matrix stack on the IRIS workstation can perform the matrix multiplications efficiently. Internally, the *Jack* can perform certain operations, such as interpolation, in other representations such as quaternions [78], but they are not a part of the representation.

A.2.3 Animation Systems

Computer graphics and animation literature seldom addresses syntactic, or even semantic, issues in representations for mechanisms, except as background for some other discussion of an animation technique or system.

Most interactive animation systems such as GRAMPS [61] TWIXT [36], and BBOP [80, 79], and commercial animation packages such as Alias [1] and Wavefront [83] only provide a mechanism of attaching one object to another. In this way, the user can construct hierarchies. When the user manipulates one object, its child objects follow, but there is no real notion of articulation. The attachments simply state that the origin of the child object is relative to the origin of the parent.

Many animation systems are non-interactive and are based on scripts that provide a hierarchy only through a programming language interface. Examples of such systems are ANIMA-II [38], ASAS [71] and MIRA-3D [81]. In this kind of system, the hierarchy is hard-coded into the script, possibly through an iteration loop. A hierarchy designed in this way is very limited, except in the hands of a talented programmer/animator who can write into the animation a notion of behavior.

A.2.4 The Physically-Based Modeling Approach

Physically based modeling systems such as that of Witkin, Fleisher, and Barr [90] and Barzel and Barr [7] adopt a conceptually pleasing view of objects and constraints. Constraints connected objects together through desired geometric relationships. Constraints can hold objects together or can hold them in place. Otherwise, they float in space under the appropriate laws of physics. There is no notion of articulation other than constraints. This forces the burden of maintaining object positions entirely to the algorithms that do the positioning. For simple objects, this is conceptually pleasing, although for complex objects it is computationally difficult. If a system represents joints like the elbow as a constraint, the constraint must have a very high weighting factor in order to ensure that it never separates, requiring very small time steps in the simulation. This may also complicate the user's view of objects such as robots or human figures, which are inherently articulated. I believe it is important to make a distinction between the type of relationship at the human elbow and the relationship between a hand and a steering wheel.

A.3 The Terminology of Peabody

Peabody uses the term *environment* to refer to the entire world of geometric objects. The environment consists of individual *figures*, each of which is a collection of *segments*. The segments are the basic building blocks of the environment. Each segment has a geometry. It represents a single physical object or part, which has shape and mass but no movable components. The geometry of each segment is represented by a *psurf*, which is generally a polyhedron or a polygonal mesh but can be of a more general nature.

The term *figure* applies not only to articulated, jointed figures such as a human body: any single "object" is a figure. It need not have moving parts. A figure may have only a single segment, such as a coffee cup, or it may be composed of several segments connected by *joints*, such as a robot. Sometimes the term "object" denotes any part of the *peabody* environment.

Joints connect segments through attachment frames called *sites*. A site is a local coordinate frame relative to the coordinate frame of its segment. Each segment can have several sites. Joints connect sites on different segments within the same figure. Sites need not lie on the surface of a segment. A site is a coordinate frame that has an orientation as well as a position. Each site has a *location* that is the homogeneous transform that describes its placement relative to the base coordinate frame of its segment.

Segments do not have specific dimensions, such as the length, offset, and twist of Denevit & Hartenberg notation, because the origin can lie anywhere on the segment. The location of the axes of the joints that connect the segment are phrased in terms of this origin, rather than the other way around. The measurement of quantities such as length is complicated, because segments may have several joints connected to them, and none of these joints is designated in the definition as the "parent."

Joints may have several *degrees of freedom*, which are rotational and translational axes. Each axis

and its corresponding angle form a single rotation or translation, and the product of the transform at each degree of freedom defines the transform across the joint, defining the placement of the sites, and thus the segments, that the joint connects.

The directional of a joint is important because it defines the order in which the degree of freedom transforms are concatenated. Because these transforms are not commutative, it is essential that the order is well-defined. This is an especially important feature of *peabody*, since it is sometimes convenient to define the direction of the joint in a way different from the way the joint occurs in the figure hierarchy. An example of this is the human knee. Although it may be useful to structure the hierarchy of a human body with the root at the foot, it is also appealing to have the joints at both knees defined in the same manner.

The *peabody* language for describing a four-legged table figure is shown in Figure A.1. The syntax has a straightforward block structure. Normally, it is not necessary for the interactive user to have any understanding of the language. The syntax of the *peabody* language is described in detail in the *Jack 5 User's Guide* [64].

A.3.1 A Metaphor for Peabody

Peabody promotes a certain cognitive model of the objects and the relationship between them. This cognitive model is the user's mental representation of the objects, and it keeps the following metaphors in mind.

To visualize the *peabody* environment, imagine the process of building a folding table out of wood. The blueprints give the specifications for the dimensions of the top and the legs, and information about how to assemble the parts together using hinges. Each part has no moving components.

We begin by fashioning each part from our stock material, doing so based on the measurements given in the design plans. These plans give the dimensions of the part measured from a local origin, probably a corner or the center. We do so once for each part, and the specification and fabrication of each part is independent of the others. The dimensions of each part are measured from its own local origin, not in any global coordinate system.

To assemble the articulated structure out of these parts, we connect hinges to them. We connect the hinges one half at a time, with the hinge pin removed. The placement of each half of a hinge on the object is defined in the object's own coordinate system according to the design specification, just like the dimensions. This enables us to drill holes to screw the hinge plate into place. The process of connecting the hinge plates proceeds independently for each part, since the placement of a hinge on one of the parts is unaffected by the other parts. Then we move the parts together so that the hinges meet, and we slide in the pin. Once we have done this for each part, our table is assembled.

We need not think of the segments and joints in our table has having a strict hierarchy. The joints simply connect segments and hold it together. In order to define the placement of the table in the room, we must designate a reference point on one of the table parts the origin of the table. This is the figure's *root*. We can then give the position and orientation for this reference and uniquely define the table's exact location. It may be convenient to change the reference point later on, in order to position the table in a different way.

When we bend a hinge on one of the legs of the table, usually the top will remain fixed and the leg will move. However, if the table lies upside down on its top, the opposite will happen. Which side moves and which remains fixed depends on how the figure is rooted: the ones on the rooted site remain fixed.

```
figure table {
    segment leg1 {
        psurf = "leg.pss";
        site base->location = trans(0.00cm,0.00cm,0.00cm);
        site top->location = trans(7.5cm,75.00cm,7.5cm);
    }
    segment leg2 {
        psurf = "leg.pss";
        site base->location = trans(0.00cm,0.00cm,0.00cm);
        site top->location = trans(7.5cm,75.00cm,7.5cm);
    }
    segment leg3 {
        psurf = "leg.pss";
        site base->location = trans(0.00cm,0.00cm,0.00cm);
        site top->location = trans(7.5cm,75.00cm,7.5cm);
    }
    segment leg4 {
        psurf = "leg.pss";
        site base->location = trans(0.00cm,0.00cm,0.00cm);
        site top->location = trans(7.5cm,75.00cm,7.5cm);
    }
    segment top {
        psurf = "top.pss";
        site base->location = trans(0.00cm,0.00cm,0.00cm);
        site leg1->location = trans(-25cm,0.00cm,-25cm);
        site leg2->location = trans(-25cm,0.00cm,25cm);
        site leg3->location = trans(25cm,0.00cm,-25cm);
        site leg4->location = trans(25cm,0.00cm,25cm);
    }
    joint leg1joint {
        connect top.leg1 to leg1.top;
        type = R(z);
    }
    joint leg2joint {
        connect top.leg2 to leg2.top;
        type = R(z);
    }
    joint leg3joint {
        connect top.leg3 to leg3.top;
        type = R(z);
    }
    joint leg4joint {
        connect top.leg4 to leg4.top;
        type = R(z);
    }
    root = top.base;
    location = trans(0.00cm,75.00cm,0.00cm);
}
```

Figure A.1: A Four-legged Table Figure in the *Peabody* Language

A.3.2 Constructing a Peabody Figure

With the four-legged table example mind, the process of constructing an articulated figure in *peabody* is as follows:

1. Construct the geometric representations for the segments in any manner whatsoever. Choose whatever local coordinate frame is convenient in the design of the geometry.
2. Specify site locations with respect to the coordinate origin of each segment. Place one site on each segment for each joint that attaches to the segment.
3. For each joint, choose a local coordinate axis (x, y or z, positive or negative) for the joint to rotate around, and orient each of the sites that the joint connects so that the axis lies in the desired direction relative to the segment's coordinate origin. The joint may have several degrees of freedom.

Note that there are lots of arbitrary choices in this process, so there is great freedom in the way in which the base coordinate frames of each segment may be defined.

A.4 The Peabody Hierarchy

Peabody avoids imposing a predefined hierarchy on the figures by encouraging the user to think of figures as collections of segments and joints, none with special importance. However, there must exist an underlying hierarchy because *peabody* is not equipped to handle closed-loop mechanisms. The structure of the *peabody* tree is defined by designating one site on the figure as the *root*. The root site roughly corresponds to the origin of the figure, and it provides a handle by which to specify the location of the figure. Viewing the figure as a tree, the root of the figure is the root of the tree. The root site of a figure may change from time to time, depending upon the desired behavior of the figure.

This means there are two representations for the hierarchy, one internal and one external. There are many advantages to having a dual representation of the hierarchy. First, it allows the hierarchy to be inverted on the fly. Most models have a natural order to their hierarchy, emanating from a logical origin, but this hierarchy and origin may or may not correspond to how a model is placed in the environment and used.

The choice of the figure root is particularly important to the inverse kinematics algorithm, since the algorithm operates on chains of joints within the figure. At least one point on the figure must remain fixed in space. Because the internal representation of the hierarchy is separate, the user maintains a consistent view of the transform across a joint, regardless of how the figure is rooted.

A.4.1 Computing Global Transforms

The root site for the figure is the one at the top of the tree, and its global location is taken as given, that is, not dependent on any other element of the environment. From this, taken together with the site locations and joint displacements, the global location of every site and segment in the tree is uniquely determined, in terms of a product of transforms from the root downward.

The computation of the coordinate transforms for each segment and site in the downward traversal of the tree requires inverting the site locations that connect the segment to other segments lower in the

tree. It may also require inverting joint displacements if the joint is oriented upwards in the tree. Computationally, this is not expensive because the inverse of a homogeneous transform is easy to compute, through a transpose and a dot product.

A.4.2 Traversing Upwards in the Hierarchy

It is frequently necessary to traverse upwards in the tree, from a particular reference site towards the root. In order to do this, *peabody* maintains a set of *root* fields with each element of the data structure. Each segment has a *rootsite* field that points to the site at the "top" of the segment. Each site has a *rootjoint* field that points to the joint that connects it to another site in the tree, if there is one. Most sites do not have a root joint, so their pointer is nil. Only the sites that are at the roots of segments have a root joint. Finally, each joint has a root site, which points to the one of the two sites that the joint connects that is higher in the tree. Using this set of fields, upwards traverse proceeds recursively from any site, segment, or joint towards the root of the tree, according to the following algorithm:

1. from any site, go to its segment
2. from the segment, go to its root site.
3. if the root site doesn't have a root joint, then this is the root of the tree, at the figure root. Otherwise...
4. from the root site, go to its root joint.
5. from the joint, go to its root site.
6. recurse to step #1.

The root fields themselves must be set any time the tree is restructured. This is done by simply traversing the figure in a depth-first fashion and setting the pointers upwards as the traversal proceeds downwards.

A.5 Summary

The principal features of the *peabody* object representation are:

- It avoids the rigid structure of Denevit & Hartenberg and similar kinematic representations. The parameters are all easy to interpret because they are general homogeneous transforms, and the origin of each segment can be located at any convenient point.
- It has a well developed notion of articulation. The articulation is a part of the data structure itself and is not dependent on the algorithms that operate on it, unlike physically-based modeling representations.
- The hierarchy can be inverted "on the fly."
- The representation is object-oriented. The internal data structures and the external user's view of the models are very similar.

Appendix B

Implementation of Direct Manipulation

B.1 The Equation for Linear Translation

This section describes the details of the implementation of the linear translation manipulation, described in Section 3.4.1 Figure B.1 shows the coordinate systems of the linear translation process. The figure

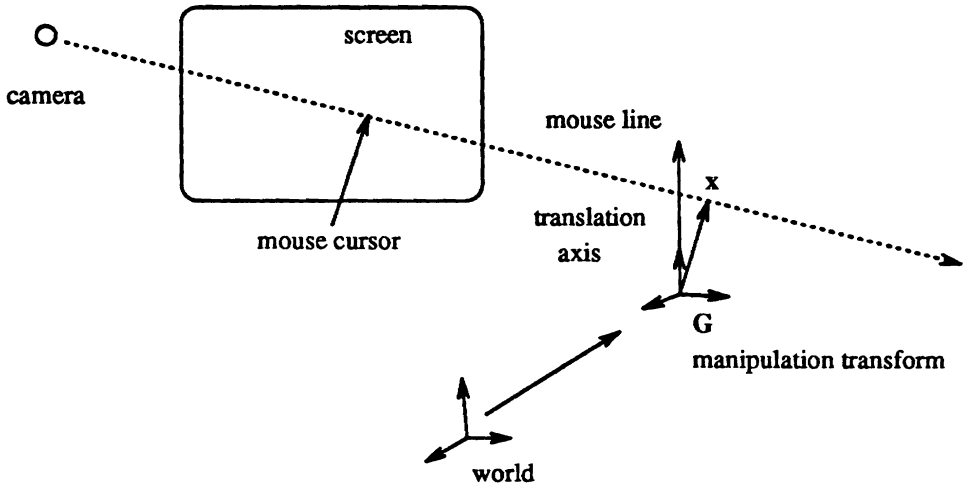


Figure B.1: The Coordinate System for Linear Translation

shows a tilted view of the graphics window as a portal to the geometric environment. The mouse line originates at the center of projection, i.e. the camera location, and passes through the current mouse cursor position in the window. G is the current manipulation transform. The translation axis passes through the origin of G. If the translation is local, the axis will be aligned with one of the axes of G. This axis projected onto the graphics window gives the direction of the translation icon. The diagram shows the location of the mouse cursor deviating slightly from this line, which means that the mouse

line does not intersect the translation axis: they are skew. $\hat{\mathbf{x}}$ is the point on the translation axis which is closest to the mouse line. This is the global position of the manipulation transform in the next iteration.

B.2 The Equation for Planar Translation

This section describes the details of the implementation of the planar translation manipulation, described in Section 3.4.2 Figure B.2 illustrates the process of planar translation. The figure shows a tilted view of

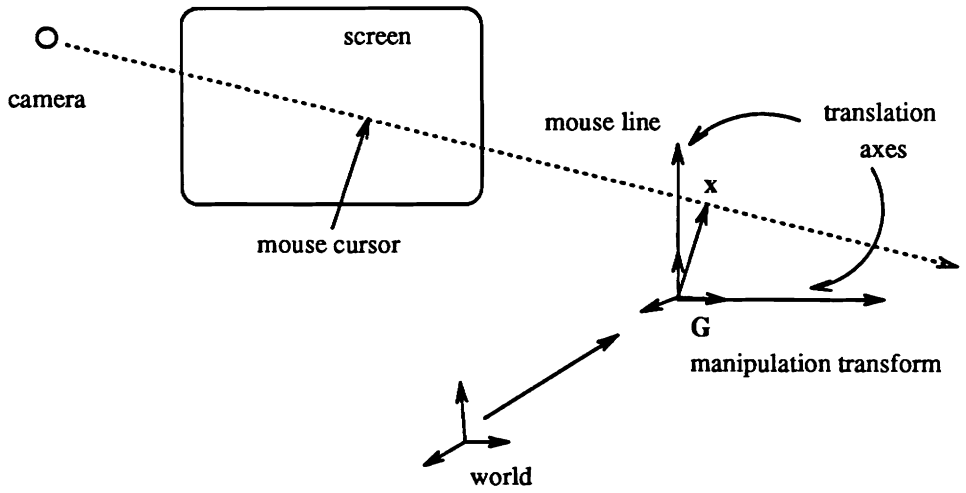


Figure B.2: The Coordinate System for Planar Translation

the graphics window as a portal to the geometric environment. The mouse line originates at the center of projection, i.e. the camera location, and passes through the current mouse cursor position in the window. $\mathbf{G}$ is the current manipulation transform. The translation plane passes through the origin of $\mathbf{G}$. If the translation is local, the plane will be aligned with two of the axes of $\mathbf{G}$. $\hat{\mathbf{x}}$ is the intersection of the mouse line with the plane of translation. This is the global position of the manipulation transform in the next iteration.

B.3 The Equation for Rotation

This section describes the details of the implementation of the rotation manipulation, described in Section 3.5 Figure B.3 illustrates the coordinate systems involved in the rotation process. The figure shows a tilted view of the graphics window as a portal to the geometric environment. The mouse line originates at the center of projection, i.e. the camera location, and passes through the current mouse cursor position in the window. $\mathbf{G}$ is the current manipulation transform. The rotation axis $\hat{\mathbf{a}}$ passes through the origin of $\mathbf{G}$. If the rotation is local, the axis will be aligned with one of the axes of $\mathbf{G}$. The rotation wheel lies in the plane perpendicular to the axis. Its projection onto the viewing plane forms an spoked ellipse. The initial intersection between the mouse line and the rotation plane is labeled $\hat{\mathbf{y}}$. Subsequent intersection points are labeled $\hat{\mathbf{x}}$. The angle formed by $\hat{\mathbf{x}}$, $\mathbf{G}_o$, and $\hat{\mathbf{y}}$, labeled θ, measures the angle of rotation. The incremental rotation matrix $\mathbf{R}$ is the rotation of θ around the axis $\hat{\mathbf{a}}$ mapped

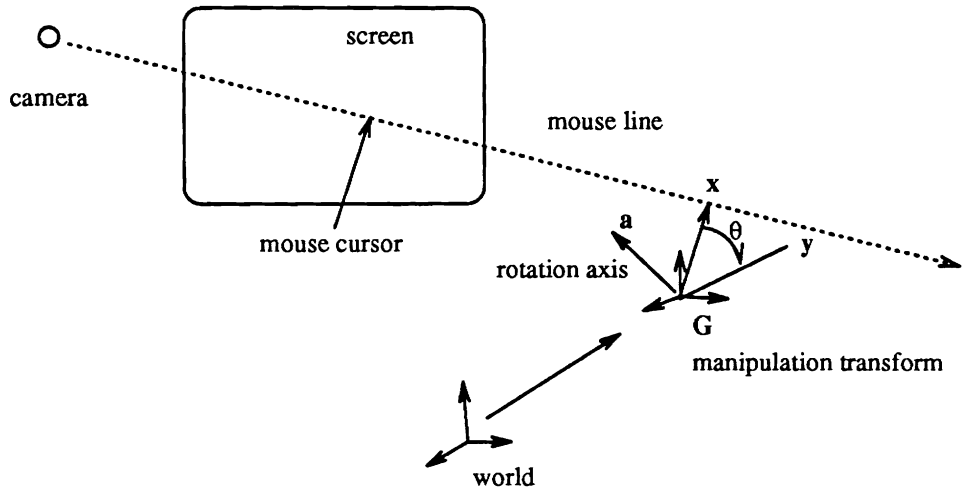


Figure B.3: The Coordinate Systems for Rotation

to the coordinate system of **G**. Then

$$\mathbf{G}' = \mathbf{R}\mathbf{G}$$

B.4 Snapping

The 3D movement operator provides a flexible way of manipulating transforms interactively, but it can be difficult to specify precise angles and distances in this way. One way to provide better end conditions is by *snapping* the transform to features of the geometric environment. This enables the user to position objects at precise locations and align them precisely with existing objects.

This technique is similar to the *snap-dragging* technique of [11], although in this context it is not feasible to do the snapping automatically and continuously. In Bier's technique the gravity which attracts the cursor to geometric features is always active. This means that if an object gets close to another object, it automatically snaps to it. In the complex geometric environments the computation required to perform the gravity continuously would make the system excessively slow. In *Jack*, gravity is only an option of the manipulation operator. The option is called *snapping*.

The snapping can involve position or orientation. Position snapping is straightforward, and amounts to computing a new location based on a selected reference point and then moving the manipulation transform there. Orientation snapping can involve making the coordinate frames of the manipulation transform and the target transform match exactly, or aligning each axis of one frame with any axis of the other. This latter type of alignment means that the relative transform between the two transforms consists only of 90° rotations around coordinate axes. The advantage of this type of alignment comes from the fact that in an integrated modeling environment, there are many coordinate frames (*sites* in peabody terminology) defining many points of interest, each one having a specific orientation. It is sometimes convenient to align transforms squarely but not such that their coordinate frames are perfectly coincident. For example, the coordinate frame in the hand of *Jack*'s standard human figure is oriented such that the $+z$ axis points out of the palm, the $+y$ axis points towards the finger tips, and the $+x$

axis points out the side of the hand. If a coordinate frame on a table happens to be configured with the y axis oriented vertically, then to place the hand on the table requires aligning the $+z$ axis of the hand with the $-y$ axis of the table.

Snapping is implemented in a way similar to Bier's gravity for orientation. When the user requests an orientation snap, it aligns each axis of the manipulation transform with the axis of the target transform which is currently closest to it, in angular displacement. This means that to achieve a particular orientation, the user first moves the transform approximately to the correct location and then uses the snapping mechanism to square up the alignment.

The features to which objects can be snapped are:

site This snaps to both the position and orientation of a site.

site position This snaps to only the position of a site.

site orientation This snaps to only the orientation of a site.

node position This snaps to the position of a node.

edge position This snaps to the position of an edge. The snapping point is the point on the edge which is closest to the current manipulation transform.

edge line This snaps the position of the manipulation transform to the line of an edge. The snapping point is the point on the line which is closest to the current manipulation transform, which may not lie between the endpoints of the edge.

edge orientation This rotates the manipulation transform through the smallest angle so that one of its axes has the same orientation as the edge.

face position This snaps the position to the face. The snapping point is the point on the face which is closest to the manipulation transform.

face center This snaps the position to the center of the face.

face plane This snaps the position to the plane in which the face lies. The snapping point may not lie within the boundary of the face.

face orientation This rotates the orientation of the manipulation transform through the smallest angle such that one of its axes is aligned with the normal to the face.

Appendix C

Interactive Viewing Control

The computer graphics workstation provides a view into a virtual 3D world. It is natural to think of a graphics window as the lens of a camera, so the process of manipulating the viewpoint is analogous to moving a camera through space. Viewing rotation has been described as the single most effective depth cue, even better than stereoscopy [28]. In order for an interactive modeling system to give the user a good sense of the three-dimensionality of the objects, it is essential that the system provide a good means of manipulating the viewpoint.

C.1 Controlling the View

As described in [82], the process of moving objects in front of a fixed camera is isomorphic to the process of moving a camera around fixed objects, although the cognitive model which each suggests is quite different and yields a vastly different feel to the user interface. Because of this, it is important to carefully consider the objectives of the view manipulation process. There are several objectives:

- to position the camera relative to the objects in order to provide a good view for subsequent manipulation, i.e. "to be able see what you are doing." The end resulting view is the most important part of the positioning process, not the camera movements.
- to get a better sense of the shape or configuration of the objects by interactively moving the camera. This provides the dynamic depth cue. In this case, the objective does not involve a specific end resulting view.
- to navigate through a large space, such as a building or landscape.
- to design a motion path for a camera during an animated sequence. Ware calls this "making movies" [82].

C.1.1 Metaphors for Viewing

[82] describes three different metaphors for exploring virtual worlds: *eyeball-in-hand, scene-in-hand,* and the *flying vehicle.* Each metaphor promotes a unique cognitive model of the virtual environment and our

relationship to it in the viewpoint manipulation process. The eyeball-in-hand metaphor suggests that the user controls the location of the eyeball or camera which moves around a fixed 3D environment. The scene-in-hand metaphor suggests that the entire 3D scene is in the user's hand and he is free to turn it around to look at it from different points of view. The flying vehicle metaphor suggests the view from a flying airplane or helicopter. This metaphor usually means that the user controls directly the velocity, not the position, of the camera.

The results of the experiments in [82] show that each metaphor is good for certain types of control tasks and not good for others, with no one metaphor being absolutely better or worse than the others for all circumstances.

C.1.2 Camera Controls in Jack

The geometric environment in problems in human factors analysis usually involve models of human figures in a simulated workspace. The most appropriate cognitive model to promote is one of looking in on a real person interacting with real, life-size objects. Therefore, *Jack* suggests that the controls on the viewing mechanism more or less match the controls we have as real observers: move side to side and up and down while staying focused on the same point. The scene-in-hand metaphor is therefore not appropriate to expect the user to imagine picking up a human being and its working environment and moving it in his hand. And even though the workspace may be large in human terms, it is still typically localized. *Jack* is not, for example, principally concerned with simulating people walking through a building or over a landscape. Therefore, there is no great need to navigate through the space as would be suggested by the flying vehicle approach. *Jack* is also not concerned with choreographing camera movements for movies. Therefore, *Jack* concentrates mostly on the first two viewing objectives: the need to position the view during the geometric manipulation process, and viewing as an aid to visualization.

Therefore, the eyeball-in-hand metaphor is best suited to the tasks required by *Jack*. In a structured 3D environment, the camera commonly obeys certain simple constraints which restrict its motion and make it easier to control. Although there are six degrees of freedom associated with the camera, *Jack* restricts its movements to make it more predictable.

The characteristics of camera movements which *Jack* attempts to model are:

- The geometric environment has an intrinsic vertical axis. Since the articulated figures are usually human figures, we expect that they have a specific vertical orientation, even in times which they are not oriented that way. The view manipulation procedure should respect this axis.

  This is not universally true in computer graphics. An example of an environment in which this is not true may be molecular modeling because molecules don't have an intrinsic vertical axis.

- Horizontal movements of the camera give a distinctly different feel and effect from vertical movements. Because of the sacred nature of the vertical axis, we tend to differentiate between vertical and horizontal movements and we wish to be able to control them separately.

- The camera is usually focused on a particular object, the object of interest. We want to be able to easily direct the camera to a particular object and then manipulate the camera around the object.

The camera controls described here were originally presented in [67], although they have been extended since then.

Jack models the viewing transform through a geometric object, thought of as a camera. The view in the corresponding graphics window is defined by the placement and direction of the camera's lens. The process of changing the view then amounts to moving the global location of the camera figure. The location of the camera can be changed only in a controlled way:

- by rotating the camera horizontally and vertically around a fixed view reference point, or point of interest. *Jack* calls this *sweeping*.
- by pivoting the camera around its own origin, thus moving the point of interest. *Jack* calls this *panning*.
- by translating the camera along the line of sight. *Jack* calls this *zooming*. Zooming can either maintain the same distance between the camera and the view reference point, or it can maintain the view reference point in the same global location.
- by selecting a new view reference point and having the camera automatically adjust to focus on it. This is controlled panning, since the camera still pivots around its origin. *Jack* calls this *snapping*.
- by having the camera location and the view reference point automatically shift to give a better view of an object. This automatic camera adjustment technique plays an important role in the manipulation mechanism described in Chapter 3.

Jack intentionally prohibits *twisting* along the line of sight. Although cameras do occasionally twist, they do so infrequently and usually only temporarily. Twisting has its place in animation because it can help give a sense of flying, like the banking of an aircraft. But in the visualization and manipulation process, it does not help because it disturbs the user's sense of the vertical axis. After a twist is applied to the view, the user's notion of horizontal and vertical does not correspond to the world coordinate frame.

Jack therefore assumes that the x axis of the viewing transform is always perpendicular to the global y axis. These two axes provide the handles through which to move the camera.

C.1.3 The Representation of the View

The location of the camera is defined through the homogeneous transform describing the location of the camera figure. If we view this transform as a 4×4 matrix, then the position vector of the matrix is the origin of the camera, the $-z$ axis is the line of sight, with the $+x$ axis extending to the right, and the $+y$ axis extending upwards.

The view matrix as required by the Graphics Library for drawing purposes is the inverse of the global location of the camera figure. To see that this is true, note that the screen coordinates for the viewing pipeline have $+x$ horizontal from left to right, $+y$ vertical from bottom to top, and $+z$ coming out of the screen. If we perform no viewing operation, but transform all objects in the environment by the inverse of the relationship between the world and the camera, then the placement of all objects relative to the camera remains the same. Note that the *projection* matrix is entirely different from the *viewing* matrix. The projection matrix models the perspective transform.

Jack does not have an explicit representation for the view reference point. Instead, it models the view reference point implicitly through a *view reference point distance*. The view reference point lies along the line of sight this distance from the camera's origin. This means that instead of determining the

viewing matrix from the view reference point, the view reference point is determined from the viewing matrix.

C.2 Changing the View

Jack allows the user to change the view through the interactive command change view. This command follows the behavior described in Section 1.7. It places the system in view change mode and it remains there until the user terminates the mode by hitting the ESCAPE key. Also, the ^C key aborts the operation and returns to the previous view. This mechanism can be invoked in *Jack* at any point by hitting the ^V key, including during the manipulation of some other object. When the change view command finishes, it returns to the previous context.

Jack takes an incremental approach to specifying the viewing transform. Rather than constructing the viewing transform from scratch based in the input parameters, the parameters describe an incremental rotation and translation which is applied to the current view. This incremental approach avoids the singularity which exists in some viewing systems. For example, the SGI Graphics Library routine `lookat` takes a camera location and reference point, and a twist angle that is implicitly measured from the $+y$ axis, which means that a view parallel to the y axis has an undefined twist. In *Jack*, the "view-up" vector is always defined consistently because it travels along with the camera location.

Within the view change mode, *Jack* has several ways of changing the view, based on the operations of *sweeping, panning, zooming,* and *snapping.* The sweep operation sweeps the camera around horizontally and vertically on a virtual circular track, keeping it focused at the same reference point. The pan operation does the opposite, changing the orientation of the camera but keeping it at a fixed position. The zoom operation translates the camera along its line of sight. Zoom can be performed simultaneously with either the horizontal or vertical sweep. Snapping is automatic panning.

C.2.1 The Sweep Operation

The user interface for the change view command provides the sweep operation as the default. The left mouse button ties horizontal movement of the mouse to a rotation of the camera around the global vertical axis, passing through the view reference point. The camera remains at the same global elevation. The middle mouse button ties vertical movement of the mouse to a rotation of the camera around its local horizontal axis, but passing through the view reference point as well. The camera remains at the same longitude but the elevation rises or falls. The right mouse button ties vertical movement of the mouse to translation of the camera along the line of sight. In each of these cases, the view reference point remains fixed, so the focus remains on the same object.

The user can select any two mouse buttons simultaneously to get two effects at once. This works well in the case of horizontal and vertical rotation, which simulates the camera moving around on the surface of a sphere centered at the view reference point. It also works well for horizontal rotation and zoom, in which case the camera moves back and forth and in and out simultaneously. However, in order to combine vertical rotation with zoom, the zoom must be tied to the horizontal mouse movements, so that the two operations are not controlled by the same valuator. This, however, is not a pleasing kind of motion to perform anyway, because it does not correspond to the way cameras or eyeballs tend to move.

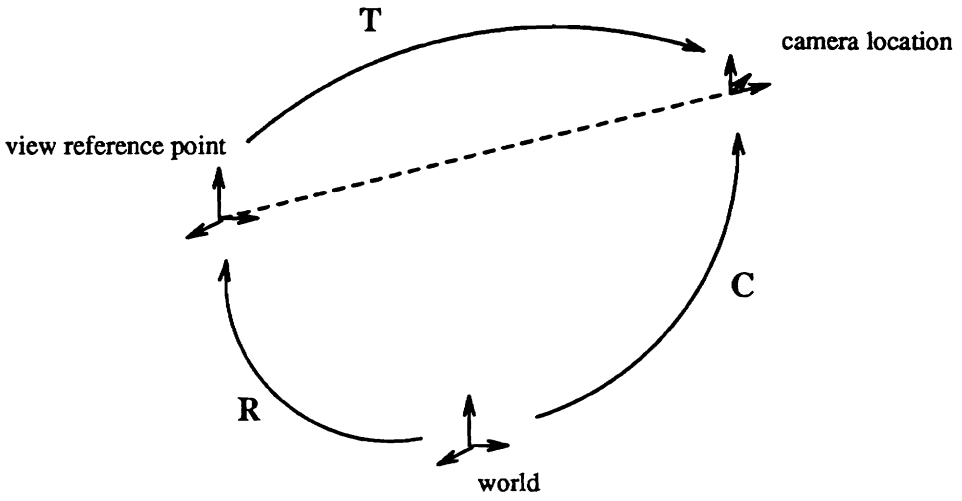


Figure C.1: The View Change Coordinate System

C.2.1.1 The Usefulness of Sweeping

The sweep operation is most useful for visualization, for dynamically rotating the view back and forth to get a better sense of the shape of an object or the posture of a figure. Since the operation moves the camera around a sphere centered at the view reference point, it embodies the eyeball-in-hand metaphor directly. It can, of course, also be used to position the view during manipulation.

C.2.1.2 The Equation for Sweeping

The sweep procedure first determines horizontal and vertical changes in the mouse location and maps them to angles according to mouse *gain* parameters. It then determines which mouse buttons are pressed in order to determine which rotation to apply.

Figure C.1 illustrates the transforms in involved. **C** is the current global camera transform. **R** is the global location of the view reference point, which is determined from the camera location, line of sight, and view reference distance. **R** is always aligned with the world coordinate frame. **T** is the transform from **R** to **C**. **T** is easily computed:†

$$\mathbf{T} = \mathbf{C}\mathbf{R}^{-1}$$

We want to compute $\mathbf{C}'$, a new global transform for the camera. Let the horizontal and vertical angular increments be θ_h. We can express $\mathbf{C}'$ as

$$\mathbf{C}' = \mathbf{T}\mathbf{D}\mathbf{R}$$

where **D** is an incremental rotation.

†*Jack* represents homogeneous transforms as matrices with the position component in the bottom row, so that premultiplication is a local transformation.

For horizontal movement, $\mathbf{D}$ is $R_y(\theta_h)$, a rotation of θ_h around the global y axis. For vertical movement, $\mathbf{D}$ is $R_{\mathbf{C}_x}(\theta_v)$, a rotation of θ_v around the local axis of the camera $\mathbf{C}_x$.[†]

C.2.2 The Pan Operation

The *pan* operation is the complement of the sweep; the location of camera remains fixed but it pivots around its horizontal and vertical axes. This simulates the effect of the view reference point rotating around the camera. The user interface for the viewing routine allows the user to select the pan operation by holding down the `CONTROL`. The left mouse button ties horizontal movement of the mouse to a rotation of the camera around the global vertical axis. This corresponds to the view reference point circling the camera horizontally at a constant elevation. The middle mouse button ties vertical movement of the mouse to a rotation of the camera around its local x axis, which is horizontal. This corresponds to a rotation of the view reference point vertically over the camera. And as with the sweep operation, the right mouse button ties vertical movement of the mouse to translation of the camera along the line of sight.

C.2.2.1 The Usefulness of Panning

Panning is useful to change the location of the view reference point, to locate it near an object for subsequent sweeping. It is not very good as a visualization tool itself, except to get a panoramic view of an entire environment, but because *Jack*'s environment is localized, this is not very important. This can be useful if there is a distant object which is not visible but which needs to be located.

The panning operation is not very good at positioning the view reference point precisely. This led to the development of the snapping mechanism described later.

C.2.2.2 The Equation for Panning

The pan procedure first determines horizontal and vertical changes in the mouse location and maps them to angles according to mouse gain parameters. It then determines which mouse buttons are pressed in order to determine which rotation to apply.

Figure C.1 illustrates the transforms involved. Again, we want to compute $\mathbf{C}'$, a new global camera transform. The location of the view reference point $\mathbf{R}$ is not relevant to these calculations, since the camera rotates around its own origin. The transform $\mathbf{D}$ is the increment applied to the camera location $\mathbf{C}$.

$$\mathbf{C}' = \mathbf{RD}$$

For horizontal movement, $\mathbf{D}$ is $R_y(\theta_h)$, a rotation of θ_h around the global y axis, mapped to the coordinate frame of the camera. For vertical movement, $\mathbf{D}$ is $R_{\mathbf{C}_x}(\theta_v)$, a rotation of θ_v around the local x axis of the camera.

[†]I use the notation $\mathbf{C}_x$ to denote the x axis of transform $\mathbf{C}$.

C.2.3 The Zoom Operation

The zoom operation translates the camera along the line of sight. Normally, this changes the view reference distance by a proportional amount so that the view reference point remains at the same point in space.

C.2.3.1 The Usefulness of Zooming

The zoom operation is equally important to view positioning and dynamic depth cuing. Its importance for positioning is obvious: it is the mechanism by which the camera can be brought towards an object, or backed away. The zoom is particularly effective for this in conjunction with horizontal sweeping. This causes the camera to swing around side to side and in and out. If the camera starts out at about eye level, it maintains the same approximate elevation.

C.2.3.2 The Equation for Zooming

The zoom operation is very simple since it only involves translating the camera along the line of sight. The orientation of the camera remains fixed and only the location changes. We need only change the location of the camera, which is the position vector of the global transform of the camera. It is shifted by a distance θ_v along the z axis of the camera transform, which is the line of sight. θ_v is modulated by an additional gain parameter which measures linear distances.

$$\mathbf{C}'_p = \mathbf{C}_p + \theta_v \mathbf{C}_z$$

C.3 Snapping the View

The sweeping mechanism is a good way of viewing an object provided that the view reference point is located near the object's center. Although the pan operation allows the user to move the view reference point, it is a poor way of positioning it precisely. Typically, it is convenient to place the view reference point exactly at a particular feature, such as a joint center or figure origin. Therefore, *Jack* provides a way of automatically shifting the view reference point to selected features of the geometric environment. This mechanism is called *snapping*. It is in some ways similar to the point-of-interest control described in [54], except that *Jack* activates it only on demand, rather than continuously.

To snap the view, the user first selects the snap option, which in the user interface for the viewing operation is bound to the ^S key. The user then selects a feature point, and this defines the new location for the view reference point. *Jack* then *animates* the change of the view reference point by generating in-between views. The interpolation technique rotates the camera simultaneously around the global vertical axis and the local horizontal axis until the line of sight is directed towards the feature point. Both of these rotation angles are easy to compute.

The animation causes the view to change gradually and continuously. The in-betweening function incorporates an ease-in/ease-out factor to provide continuity in the view. A sudden and discontinuous shift in the view can be disorienting and the user may have to study the image to regain his perspective. By shifting the view smoothly, the user never loses the frame of reference.

This in-betweening technique might not work so well when the interactive update rate of the workstation is very slow, because it might take more time than the user wants. Therefore, instead of using a constant number of in-between frames, *Jack* adopts a constant *total animation time*, and compute the necessary number of in-betweens based on the time it takes to draw the screen. This data is available from the operating system in 1/60th's of seconds. It is measured in *Jack* by simply sampling the time counter before and after each screen redraw. The necessary number of frames comes from dividing the number time of redraw by the total animation time, which *Jack* arbitrarily set at one second. This means that the snapping will never take longer than one second. If the drawing rate is very slow, then the screen may be redrawn only once at the final location. On the other hand, if the snapping distance is very small, then there is no need to consume a full second with it, so there is also a minimum angle through which the camera can rotate at each iteration.

Appendix D

Behavioral Controls for Human Figures

This appendix elaborates on the behaviors for human figures which have currently been implemented in *Jack*. Most of these were mentioned in Chapter 5. This appendix gives the details.

D.1 Behavioral Parameters

The behavioral parameters are the properties of the constraints which model the posture of the human figure. These are like "tendons" that connect the strings to the puppet. Mostly, these parameters describe the objective functions of the constraints, as in whether the constraint specifies position, orientation, or both. The parameters can include the goal values of the constraints as well. These are simple relationships which require no computation on the part of the system.

D.1.1 The Position and Orientation of the Feet

set foot behavior
pivot
hold global location
hold local location
keep heel on floor
allow heel to rise

Table D.1: Foot Behaviors

The foot behaviors are shown in Table D.1. These are options to the set foot behavior command. The standard human figure model in *Jack* has a foot with two segments connected by a single toe joint, and two natural constraints, one on the toes and another on the heel. The toe constraint keeps the toes on the floor; the heel constraint can keep the heel on the floor or allow it to rise if necessary. For a standing

figure, the pair of behaviors *keep heel on floor* and *allow heel to rise* control the height of the heel. The *pivot* behavior instructs the toes to maintain the same position, and to maintain an orientation flat on the floor while allowing them to rotate through a vertical axis. The *hold global location* behavior disables the *pivot* behavior and fixes the toe orientation in space. This is the appropriate behavior when the foot is not on the floor. The *hold local location* behavior attaches the foot to an object such as a pedal. If the object moves, the foot will follow it and maintain the same relative displacement from it. If the figure is seated, then the heel behaviors and the *pivot* behavior have no effect, and the *hold* behaviors control the position and orientation of the heel instead of the toes.

The behavior of the feet is usually activated by the manipulation of some other part of the figure, such as the center of mass or the pelvis. A good example of the *pivot* behavior is when the center of mass is dragged towards one foot: should the other foot pivot in order to extend the leg, or should it remain planted and inhibit the movement of the center of mass? The behaviors say which should occur.

D.1.2 The Elevation of the Center of Mass

The horizontal location of the center of mass is a passive behavior which determines balance, as described below. The elevation of the center of mass is more straightforward. This concept has a direct analog in Labanotation: the *level of support* [40, page 31]. A middle level of support is a natural standing posture, a low level of support is a squat, and a high level of support is standing on the tip-toes. The *hold current elevation* behavior is an option of the set balance behavior command. It instructs the figure to maintain the current elevation of the center of mass. This behavior is off by default. This behavior is necessary because under normal circumstances, the center of mass of the figure is free to rise and fall as necessary to meet the requirements of the feet. After adjusting the center of mass to an appropriate level, if no control holds it there, it may rise or fall unintentionally.

D.1.3 The Global Orientation of the Torso

set torso behavior
keep vertical
hold global orientation

Table D.2: Torso Behaviors

The torso behaviors are listed in Table D.2. The default behavior is *keep vertical*, which causes the torso to maintain a vertical orientation. Biomechanics research tells us that one of the most constant elements in simple human locomotor tasks is the global orientation of the head. One theory explaining this suggests that the head is the principle sensor of stability[9]. The *keep vertical* behavior mimics this nicely through a directional constraint on the chest to remain vertical, while not affecting its vertical rotation. This means that as the pelvis of the figure rotates forward, backward, or side to side, the torso will automatically compensate to keep the head up. Since the constraint is on the upper torso, not the head, the neck is free to move in order for the figure to look at certain reference points, as described below with the head and eye behaviors.

The *hold global orientation* behavior involves all three degrees of freedom of the torso. This allows other parts of the body to be adjusted while the head and chest stay relatively fixed. This is particularly

important in making adjustments to the pelvis and legs after positioning the torso acceptably. This behavior does not involve position, because it is usually acceptable to have the position float with the rest of the body.

The standard human figure model in *Jack* has 36 joints with a total of 88 degrees of freedom, excluding the hands and fingers. It has a torso consisting of 17 segments and 18 vertebral joints, designed by Gary Monheit [58]. Each vertebral joint has three degrees of freedom, each of which has a very small range of motion, with joint limits based on biomedical literature. The spine has a total of 54 degrees of freedom, although for realistic-looking postures, there is considerable coupling between the joints.

Monheit [58] also developed a computational model for describing movements of the spine in terms of total bending angles in the forward, lateral, and axial directions. The technique uses weighting factors that distribute the total bending angle to the individual vertebrae in such a way that respects the proper coupling between the joints. Different weight distributions generate bends of different flavors, such as neck curls or bends confined to the lower back. These parameters are options to the torso behavior controls through the set torso behavior command because they govern how the torso behaves as it bends to maintain the proper orientation. The user can select one of the standard *curl from neck* or *bend from waist* options, or alternatively input the range of motion of the spine by selecting a top and bottom joint, and *initiator* and *resistor* joints which control the weighting between the vertebrae.

D.1.4 The Fixation Point for the Head and Eyes

The head and eyes can be controlled by specifying a fixation point, modeled through aiming constraints which orient them in the proper direction. The constraint on the head operates on a reference point between the eyes, oriented forwards of the head. The head constraint positions only the head, using the neck. The constraint on the eyes rotates only the eyeballs. The eyeballs rotate side to side and up and down in their sockets. The behavioral parameters control the head and eyes independently during postural adjustments. The active behaviors described below simulate the coupling between head and eye movement.

The *fixate head* behavior option of the set head behavior command allows the user to select a fixation point for the head. The *fixate eyes* behavior does the same for the eyes. When these behaviors are active, the head and eyes will automatically adjust to remain focused on the fixation point as the body moves.

set head behavior
fixate head
fixate eyes
release head
release eyes

Table D.3: Head Behaviors

D.1.5 The Position and Orientation of the Hands

The principal control for the hands involves holding them at particular points in space as the body moves. Postural control of the arms and hands is usually a two step process. First, get the hands into

set hand behavior
hands on hips
hands on knees
hold global location
hold local location
release hands
hand on site
grab object

Table D.4: Hand Behaviors

position using the active manipulation facilities, and second, set a control to keep them there as some other part of the body moves. The *hold global location* and *hold local location* behaviors serve much the same for the hands as their counterparts for the feet. The desired geometric positions and orientations are either global or local to some other object. The set hand behavior command provides several standard postures. For a standing figure, a pleasing reference point is the figure's hips. The hands rest on the hips with the elbows out to the side, the arms akimbo posture.† For a seated position, a pleasing reference point is the figure's knees. The *site* behavior moves the hand to a particular site, in both position and orientation. This simulates a reaching movement, but its real purpose is to hold the hand there once it reaches the site.

The controls for the hands are invoked whenever the body moves or whenever an object to which the hand is constrained changes location. A good analogy is holding onto an object such as a doorknob. If the door closes, the arm goes with it. Likewise, if the body bends over, the door stays fixed and the arm adjusts accordingly.

Conversely, it is possible in *Jack* to define a type of relationship which is more suited to the way a person holds a screwdriver. A screwdriver is controlled completely by the hand and is not fixed in space in the same sense as the door. If the body bends over, the screwdriver should move along with the arm, not remain in place like the doorknob. This type of a relationship comes from the *grab object* behavior. "Grab" in this context does not mean "grasp." It doesn't mean the fingers will wrap around the object. It means that the object will subsequently be attached to the figure's hand, as if the figure grabbed it. Once again, the process is a WYSIWYG two-step: first, position the hand the object relative to each other, then specify the *grab* behavior to hold it there.

Jack cannot currently model a two-way relationship between a hand and an object such as a doorknob. The relationship described above is distinctly one-way in the sense that the hand is constrained *to* the doorknob, but not vice versa. Movement of the door affects the hand, but movement of the hand cannot affect the door. The reason for this is that the behaviors for the hands maintain the same fixed relationship to selected objects, as the objects move or as the body moves. In the current *Jack* architecture, there is no way of telegraphing information from the hand, or any other body part for that matter, to another object, as would be necessary to cause the door to close.

The constraints on the hands are logically separate from the other constraints on the body. *Jack* eval-

†This posture has been the one I have used for most of the recent demonstrations of the *Jack* software. I always felt that it looked nice. I was quite horrified to read not long ago in Peter Bull's *Posture and Gesture* [18], a fascinating discussion of experiments in the role of posture in non-verbal human communication, that the arms akimbo posture almost universally sends a negative impression to the observer.

uates the hand constraints *after* the other constraints, not simultaneously. The reasons for structuring the arm behavior this way are partly practical and partly philosophical.

In practice, the inverse kinematics algorithm does not perform well when the hand constraint is considered collectively with the other body constraints. With the human figure rooted through the toes, there are too many degrees of freedom between the toes and the hands to be controlled effectively. Even though the constraints on the pelvis, center of mass, and opposite foot help to resolve this redundancy, if the hand constraint is on equal par with the other parts of the body, the hand constraint can frequently cause the other constraints to be pulled away from their goals. Recall from Chapter 4 that the potential energy function describing the equilibrium state for the figure is a weighted combination of all of the constraints. To avoid having the hand pull the body off balance, the center of mass and pelvis constraints must have significantly higher weight. It has been difficult to arrive at a set of weights which give the right behavior. It has been much easier to simply localize the movement of the arms and isolate it from the rest of the body.

Philosophically, it is acceptable to consider the arm movements independently as well. Normally, a reaching task does not initiate much movement of the lower body, unless there are explicit instructions to do so. For example, consider what happens when a human being reaches for a nearby object such as a doorknob. If the door is near enough, this won't involve any bending of the waist, but if the door is farther away, it may be necessary to bend the waist. If the door is farther away still, it may be necessary to squat or counter-balance by raising a leg backwards. This system of behaviors requires that the user explicitly control which approach the figure takes. If the torso must bend or the center of mass must shift in order to perform a reaching task, then the user, or some higher level behavior function, must initiate it.

D.1.6 The Knees and Elbows

The knees and elbows require special care to prevent them from becoming locked at full extension. The fully extended position not only appears awkward, but it tends to cause the inverse kinematics algorithm to get trapped in a local minimum. Because the algorithm uses a gradient descent approach, if an elbow or knee reaches its limit, it has a tendency to stay there. To prevent this, *Jack* uses *limit spring* constraints to discourage the knees and elbows from reaching their limiting value. The springs give a high energy level to the fully extended angle. The springs can be tuned to any angle, but the default is 10°, and in practice, this tends to work well.

This gives the figure in its natural standing position a more pleasing posture, more like "at ease" than "attention". Biomechanics literature describes this in terms of the stresses on the muscles [20]. Labanotation uses this posture as the default. The elbows are "neither bent nor stretched" [40, page 116]. In the middle level of support, the knees are "straight but not taut" [40, page 31].

D.1.7 The Pelvis

The pelvis and torso are intricatedly related. The torso includes all joints in the spine from the waist to the neck, and rotating these joints allows the figure to bend over. However, when human beings bend over, they generally bend their pelvis as well as their torso. This means manipulating the two hip joints as a unit, which can be a problem for a computer model because there is no single fixed point below the hips from which to rotate. However, this problem is easy to handle by controlling the orientation of the pelvis through a constraint.

D.2 Passive Behaviors

Passive behaviors are like little processes attached to each figure. The passive behavior functions are executed at each interactive iteration. Passive behaviors are *instantaneous* in that they explicitly define a relationship to be computed at each iteration. Each of these behaviors has a distinct behavior function that computes a global property and transmits it to the relevant part of the figure.

D.2.1 Balance Point Follows Feet

Labanotation has a notion for the distribution of the weight between the feet and the shifting of the weight back and forth [40, page72]. This notion is well-defined regardless of the position of the feet: after specifying the distribution of weight between the feet, this proportion should remain fixed even if the placement of the feet need adjustment during a postural manipulation. This is the job of the *balance point follows feet* behavior.

Given these two parameters, a new balance point can be computed based on any new position of the feet. Holding these parameters fixed as the feet move ensures that the balance point maintains the same relationship to the feet, both laterally and in the forward/backward direction.

D.2.2 Foot Orientation Follows Balance Line

During the active manipulation of the feet with the move foot command, the user can intersperse translations and rotations of the feet, centered around the toes. Because of the nature of the direct manipulation technique described in Chapter 3, it is not possible to rotate and translate during a single movement. This has come up before, in Section 4.6.2: either the dragging procedure has *no* control over orientation, in which case the orientation is arbitrary and unpredictable, or the dragging procedure *does* have control over orientation, in which case the orientation remains globally fixed during spurts of translation. The *foot orientation follows balance line* behavior offers a convenient alternative.

The solution which the behavior offers is to predict the proper orientation of the foot based on the balance line and adjust the orientation automatically as the foot is translated with the move foot command. The balance line, as described above, is an imaginary line between the middle of the feet. Actually, this rule fixes the orientation of the foot with respect to the balance line. As the foot translates, the balance line changes, and the orientation of the foot changes to keep the same relative orientation. This behavior is particularly appropriate when the figure is taking a step forward with the intention of turning to the side.

D.2.3 Pelvis Follows Feet Orientation

The muscles in the leg make the leg act like a rotational spring. The hip and ankle joints provide only a little more than 90° of rotation in the leg around the vertical axis. This means that the orientation of the feet and the orientation of the pelvis are linked together. If the orientation of the feet are fixed, the orientation of the pelvis is severely limited. What is more, the extreme limits of pelvis orientation place an uncomfortable twist on the legs. If the legs are rotational springs, then the "middle" orientation of the pelvis can be determined by simply averaging the orientation of the feet. This seems to be in fact what happens when a person stands naturally: the pelvis is oriented to relieve stress on the legs. The *pelvis follows feet orientation* behavior simulates this.

D.2.4 Hands Maintain Consistent Orientation

The same problem with the orientation of the feet during the move foot command occurs with the hands with the move hand command. In fact, the problem is more intricate because the hands have a much greater range of movement than the feet. How is the orientation of the hand related to its position? How can this be determined automatically in order to predict reasonable postures when moving the hands?

Labanotation suggests an answer. Labanotation has a detailed system for describing arm postures and gestures, but what is most interesting here is what the notation does *not* say. To simplify the syntax, Labanotation has a standard set of orientations for the palms when the arms are in specific positions. Notations need be made only when the orientations differ from these defaults. The rules are [40, pages 129ff]:

- When the arms hang by the side of the body, the palms face in.
- When the arms are raised forward or upward, the palms face towards each other.
- When the arms are raised outward to the side, the palms face forward.
- When the arms cross the body, the palms face backward.

These rules are useful as defaults, but of course they do not dictate absolute behavior. These rules govern the orientation of the hands when the user translates them from one area to another without specifying any orientational change. These rules only take effect when the hand moves from one region to another.

D.2.5 Root Through Center of Mass

Most of the behaviors described so far are only appropriate for standing figures, which of course means that they are also only appropriate for earth-bound figures. But what about figures in zero-gravity space? This is actually quite easy to simulate by rooting the figure through the center of mass and disabling all other behaviors. The one constant element of zero-gravity is the center of mass. When the figure is rooted through the center of mass, the global location of the center of mass remains fixed as the figure moves.

D.2.6 A Rooting Approach That Doesn't Work

When the figure is rooted through one of its feet, the center of mass can be regarded as an end effector for the constraint that models balance. Using this approach, the elevation of the center of mass automatically fluctuates as necessary. Without this fluctuation, the figure appears to be suspended in space. Michael Girard's PODA system [35, 34] computes trajectories for the center of mass which rise and fall appropriately based on the gait of the feet in running and walking, but this doesn't work for general postural adjustments. My experience has shown that this fluctuation is difficult to model with any other approach.

One alternative formulation is to root the figure through a dummy segment connected to the center of mass through a joint which can translate and rotate through three degrees of freedom. This approach attempts to use the inverse kinematics algorithm to automatically determine the elevation of the center of mass and the orientation of the body while maintaining the elegance of locating the root at the center

of mass. The horizontal placement of the pseudo-segment defines balance point, and the constraints on the feet keep the feet on the floor, possibly by pulling the center of mass downward or causing the entire body to rotate. This approach does give only part of the desired behavior, and it is computationally more expensive so the interaction is not nearly as fluid. In this case, the constraints on the feet have 11 degrees of freedom each (three for the ankle, one for the knee, three for the hip, one for the elevation of the center of mass, and three for the orientation of the body).

A more disconcerting aspect of this approach comes from the oscillation of the inverse kinematics algorithm. In order to ensure interactive update rates, it is sometimes necessary to accept intermediate solutions from the inverse kinematics algorithm, as described in Section 4.7, and this yields some slight oscillation. This is not normally a problem for postural control, especially if it occurs around the waist of a figure. It sometimes even looks like the figure is swaying slightly. But if the oscillation happens around the feet, it looks very disconcerting.

D.3 Active Behaviors

Active behaviors mimic reflexive responses to certain conditions in the body. They generally have temporal elements, so they can initiate movements of parts of the body which last for a certain duration. The interactive system architecture maintains a list of currently triggered active behaviors, and it advances them at each iteration until they are complete. The behaviors terminate themselves, so the duration can be explicit in terms of a number of interactive iterations, or they can continue until a certain condition is met.

D.3.1 Take Step When Losing Balance

This behavior fires a stepping response to the loss of balance of a figure. When this behavior is active, it monitors the parametrization of the balance point of the figure as described with the *balance point follows feet* behavior. If the balance point leaves the support polygon to the front or back, the behavior moves the non-support foot forward or backward to compensate. The non-support foot in this case is the one which bears less weight. The behavior computes the new foot location such that the current balance point will lie in the middle of the new support polygon once the foot arrives there. If the balance point leaves the support polygon to the side, the stepping motion moves the support foot instead. In this case, the support foot is the only one which can be repositioned in order to maintain balance.

D.3.2 Take Step When Pelvis Is Twisted

The discussion of the *pelvis follows feet orientation* behavior above described the relationship between the global orientations of the feet and pelvis, particularly in terms of determining an orientation for the pelvis from the orientation of the feet. The opposite relationship is possible as well. Consider standing with your feet slightly apart, and then begin to twist your body to the right. After about 45° of rotation, your legs will not be able to rotate any more. In order to continue rotating, you will be forced to take a step, a circular step with either your left or right foot.

The *take step when pelvis twisted* behavior mimics this. When it senses that the orientation of the pelvis is near its limit relative to the feet, it repositions the non-support foot in an arc in front of or behind the other foot, twisted 90°.

Bibliography

[1] *ALIAS V3.0 Reference Manual.* Alias Research, Inc., 1990.

[2] William W. Armstrong and Mark Green. The Dynamics of Articulated Rigid Bodies for Purposes of Animation. *The Visual Computer*, 1(4), 1985.

[3] William W. Armstrong, Mark Green, and Robert Lake. Near-Real-Time Control of Human Figure Models. *IEEE Computer Graphics and Applications*, 7(6), June 1987.

[4] Norman I. Badler, Jonathan Korein, James U. Korein, Gerald Radack, and Lynne Brotman. Positioning and Animating Human Figures in a Task-Oriented Environment. *The Visual Computer*, 1(4), 1985.

[5] Norman I. Badler, Kamran H. Manoochehri, and Graham Walters. Articulated Figure Positioning By Multiple Constraints. *IEEE Computer Graphics and Applications*, 7(6), 1987.

[6] Alan H. Barr. Teleological Modeling. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann Publishers, Inc., 1991.

[7] Ronen Barzel and Alan Barr. A Modeling System Based on Dynamic Constraints. *Computer Graphics*, 22(4), 1988.

[8] David Beevis. Introduction: Worspace Design – Anthropometrical and Biomechanical Aspects. In Grant R. McMillan, David Beevis, Eduardo Salas, Michael H. Strub, Robert Sutton, and Leo Van Breda, editors, *Applications of Human Performance Models to System Design*, Plenum Press, 1989.

[9] Alain Berthoz and Thierry Pozzo. Intermittent Head Stabilization During Postural and Locomotory Tasks in Humans. In B. Amblard, A. Berthoz, and F. Clarac, editors, *Posture and Gait: Development, Adaptation, and Modulation*, Excerpta Medica, 1988.

[10] Eric Allan Bier. Skitters and Jacks: Interactive Positioning Tools. In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, ACM, Chapel Hill, NC, October 1987.

[11] Eric Allan Bier. Snap-Dragging. *Computer Graphics*, 20(3), 1986.

[12] Eric Allan Bier. Snap-Dragging in Three Dimensions. *Computer Graphics*, 24(4), 1990.

[13] Maurice Bonney, Keith Case, and Mark Porter. Applications of SAMMIE and the Development of Man Modelling. In Grant R. McMillan, David Beevis, Eduardo Salas, Michael H. Strub, Robert Sutton, and Leo Van Breda, editors, *Applications of Human Performance Models to System Design*, Plenum Press, 1989.

[14] Edward G. Britton, James S. Lipscomb, and Michael E. Pique. Making Nested Rotations Convenient for the User. *Computer Graphics*, 12(3), 1978.

[15] Fredrick P. Brooks, Jr. Walkthrough – A Dynamic Graphics System for Simulating Virtual Buildings. In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, ACM, Chapel Hill, NC, October 1987.

[16] Fredrick P. Brooks, Jr., Ming Ouh-Young, James J. Batter, and P. Jerome Kilpatrick. Project GROPE – Haptic Displays for Scientific Visualization. *Computer Graphics*, 24(4), 1990.

[17] Armin Bruderlin and T. W. Calvert. Goal-directed, Dynamic Animation of Human Walking. *Computer Graphics*, 23(43), 1989.

[18] Peter E. Bull. *Posture and Gesture.* Pergamon Press, 1987.

[19] T. W. Calvert, J. Chapman, and A. Patla. Aspects of the Kinematic Simulation of Human Movement. *IEEE Computer Graphics and Applications*, 2(9), November 1982.

[20] Sven Carlsoo. *How Man Moves.* William Heinemann Ltd, 1972.

[21] Michael Chen, S. Joy Mountford, and Abigail Sellen. A Study in Interactive 3-D Rotation using 2-D Control Devices. *Computer Graphics*, 22(4), 1988.

[22] Michael F. Cohen and Donald P. Greenberg. The Hemi-Cube: A Radiosity Solution for Complex Environments. *Computer Graphics*, 19(3), 1985.

[23] Jacques Denavit and Richard Hartenberg. A Kinematic Notation for Lower Pair Mechanisms Based on Matrices. *Journal of Applied Mechanics*, 23, 1955.

[24] Jerry Duncan. personal communication. 1990.

[25] Jill Easterly and John D. Ianni. Crew Chief: Present and Future. In Edward Boylee, John Ianni, Jill Easterly, Susan Harper, and Medhat Korna, editors, *Human-Centered Technology for Maintainability: Workshop Proceedings*, Wright-Patterson Air Force Base, Armstrong Laboratory, June 1991.

[26] Jeffrey Esakov. *An Architecture for Human Task Performance Analysis.* PhD thesis, University of Pennslvania, 1992. *to appear.*

[27] Noa Eshkol and Abraham Wachmann. *Movement Notation.* Weidenfeld and Nicolson, 1958.

[28] Kenneth B. Evans, Peter Tanner, and Marceli Wein. Tablet Based Valuators that Provide One, Two or Three Degrees of Freedom. *Computer Graphics*, 15(3), 1981.

[29] S.S. Fisher, M. McGreevy, J. Humphries, and W. Robinett. Virtual Environment Display System. In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, ACM, Chapel Hill, NC, October 1987.

[30] James D. Foley and Victor L. Wallace. The Art of Natural Graphic Man-Machine Conversation. In *Proceedings of IEEE*, IEEE, April 1974.

[31] David R. Forsey and Jane Wilhelms. Techniques for Interactive Manipulation of Articulated Bodies Using Dynamic Analysis. In *Proceedings of Graphics Interface '88*, 1988.

[32] Robert Gilbert, Robert Carrier, Jean Schiettekatte, Christian Fortin, Bernard Dechamplain, H.N. Cheng, Alain Savard, Claude Benoit, and Marc Lachapelle. SAFEWORK: Sofware to Analyse and Design Workspaces. In Grant R. McMillan, David Beevis, Eduardo Salas, Michael H. Strub, Robert Sutton, and Leo Van Breda, editors, *Applications of Human Performance Models to System Design*, Plenum Press, 1989.

[33] Michael Girard. Constrained Optimization of Articulated Animal Movement in Computer Animation. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann Publishers, Inc., 1991.

[34] Michael Girard. Interactive Design of 3-D Computer Animated Legged Animal Motion. *IEEE Computer Graphics and Applications*, 7(6), 1987.

[35] Michael Girard and Anthony A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. *Computer Graphics*, 19(3), 1985.

[36] Julian Gomez. TWIXT: A 3D Animation System. In *Proceedings of Eurographics '84*, Copenhagen, Denmark, September 1984.

[37] Mark Green. Using Dynamics in Computer Animation: Control and Solution Issues. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann Publishers, Inc., 1991.

[38] Ronald J. Hackthorn. Anima II: A 3-D Color Animation System. *Computer Graphics*, 11(4), 1977.

[39] Berthold K.P. Horn. New Notation for Serial Kinematic Chains. May 1987. (unpublished report).

[40] Ann Hutchinson. *Labanotation.* Theatre Arts Books, 1970.

[41] Paul M. Isaacs and Michael F. Cohen. Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions, and Inverse Dynamics. *Computer Graphics*, 21(4), 1987.

[42] Paul M. Isaacs and Michael F. Cohen. Mixed Methods for Complex Kinematic Constraints in Dynamic Figure Animation. *The Visual Computer*, 4(6), 1988.

[43] Moon Jung. *Human Body Motion Planning in Work Spaces.* PhD thesis, University of Pennslvania, 1992. *to appear.*

[44] Jugal Kumar Kalita. *Natural Language Control of Animation of Task Performance in a Physical Domain.* PhD thesis, University of Pennsylvania, 1990.

[45] James P. Karlen, Jack M Thompson, Havard I. Vold, James D. Farrell, and Paul H. Eismann. A Dual-Arm Desterous Manipulator System with Anthropomorphic Kinematics. In *IEEE International Conference on Robotics and Automation*, 1990.

[46] Steve W. Keele. *Human Motor Behavior*, chapter Learning and Control of Coordinated Motor Patters: The Programming Perspective. Lawrence Erlbaum Associates, 1982.

[47] E.C. Kingsley, N.A. Schofield, and K. Case. SAMMIE: A Computer Aid for Man Machine Modelling. *Computer Graphics*, 15(3), 1981.

[48] C.A. Klein and C.H. Huang. Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(2), 1983.

[49] James U. Korein. *A Geometric Investigation of Reach.* MIT Press, Cambridge, MA, 1985.

[50] Karl H.E. Kroemer. A Survey of Ergonomic Models of Anthropometry, Human Biomechanics, and Operator-Equipment Interfaces. In Grant R. McMillan, David Beevis, Eduardo Salas, Michael H. Strub, Robert Sutton, and Leo Van Breda, editors, *Applications of Human Performance Models to System Design*, Plenum Press, 1989.

[51] Philip Lee, Susanna Wei, Jianmin Zhao, and Norman I. Badler. Strength Guided Motion. *Computer Graphics*, 24(4), 1990.

[52] *Life Forms User Manual.* Kinetic Effects, Inc., Seattle, WA, 1991.

[53] *CLARIS MacDraw II.* Claris Corp., 1988.

[54] Jock D. Mackinlay, Stuart K. Card, and George G. Robinson. Rapid and Controlled Movement Through a Virtual 3D Workspace. *Computer Graphics*, 24(4), 1990.

[55] Julia McGuinness-Scott. *Movement Study and Benesh Movement Notation.* Oxford University Press, 1983.

[56] Michael McKenna, Steve Pieper, and David Zeltzer. Control of a Virtual Actor: The Roach. *Computer Graphics*, 24(2), 1990.

[57] Gavin Miller. Goal-directed Animation of Tubular Articulated Figures or How Snakes Play Golf. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann Publishers, Inc., 1991.

[58] Gary Monheit and Norman I. Badler. A Kinematic Model of the Human Spine and Torso. *IEEE Computer Graphics and Applications*, 11(2), 1991.

[59] Greg Nelson. Juno, a Constraint-based Graphics System. *Computer Graphics*, 19(3):235–243, 1985.

[60] Gregory Nielson and Dan Olsen Jr. Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices. In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, ACM, Chapel Hill, NC, October 1987.

[61] T.J. O'Donnell and Arthur J. Olson. GRAMPS – A Graphics Language Interpreter for Real-Time, Interactive, Three-Dimensional Picture Editing and Animation. *Computer Graphics*, 15(3), 1981.

[62] Ernest Otani. *Software Tools for Dynamic and Kinematic Modeling of Human Motion.* Master's thesis, University of Pennsylvania, 1989.

[63] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, Cambridge, MA, 1981.

[64] Cary B. Phillips. *Jack 5 User's Guide.* Technical Report MS-CIS-91-78, Department of Computer and Information Science, University of Pennsylvania, 1991.

[65] Cary B. Phillips. *Programming with Jack.* Technical Report MS-CIS-91-19, Department of Computer and Information Science, University of Pennsylvania, 1991.

[66] Cary B. Phillips and Norman I. Badler. Interactive Behaviors for Bipedal Articulated Figures. *Computer Graphics*, 25(4), 1991.

[67] Cary B. Phillips and Norman I. Badler. JACK: A Toolkit for Manipulating Articulated Figures. In *Proceedings of ACM SIGGRAPH Symposium on User Interface Software*, Banff, Alberta, Canada, 1988.

[68] Cary B. Phillips and Norman I. Badler. Software Systems for Modeling Articulated Figures. In Sandy Griffin, editor, *Graphics Technology and Space Applications (GTSA 1989)*, Johnson Space Center, Houston, Texas, 1989.

[69] Cary B. Phillips, Jianmin Zhao, and Norman I. Badler. Interactive Real-time Articulated Figure Manipulation Using Multiple Kinematic Constraints. *Computer Graphics*, 24(2), 1990.

[70] William T. Reeves. Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *Computer Graphics*, 17(3), 1983.

[71] Craig W. Reynolds. Computer Animation with Scripts and Actors. *Computer Graphics*, 16(4), 1982.

[72] Craig W. Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics*, 21(4), 1987.

[73] David A. Rosenbaum. *Human Motor Control.* Academic Press, 1991.

[74] Patricia L. Rothwell. Representation of Man Using CAD Technology: User Beware. In Grant R. McMillan, David Beevis, Eduardo Salas, Michael H. Strub, Robert Sutton, and Leo Van Breda, editors, *Applications of Human Performance Models to System Design*, Plenum Press, 1989.

[75] Christopher Schmandt. Spatial Input/Display Correspondence In a Stereoscopic Computer Graphics Workstation. *Computer Graphics*, 17(3), 1983.

[76] Richard Schmidt. *Human Motor Behavior*, chapter More on Motor Programs. Lawrence Erlbaum Associates, 1982.

[77] Richard Schmidt. *Human Motor Behavior*, chapter The Schema Concept. Lawrence Erlbaum Associates, 1982.

[78] Ken Shoemake. Animation Rotations with Quaternion Curves. *Computer Graphics*, 19(3):245–254, 1985.

[79] G. Stern. BBOP – A Program for 3-Dimensional Animation. In *Nicograph '83*, Tokyo, Japan, 1983.

[80] David Sturman. Interactive Keyframe Animation of 3D Articulated Models. In *Proceedings of Graphics Interface '84*, Ottawa, Canada, 1984.

[81] Nadia Magnenet Thalmann and Daniel Thalmann. *Computer Animation: Theory and Practice.* Springer Verlag, 1985.

[82] Colin Ware and Steven Osborne. Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. *Computer Graphics*, 24(4), 1990.

[83] *MODEL User's Manual Version 6.0.* Wavefront Technologies, 1989.

[84] Jakub Wejchert and David Haumann. Animation Aerodynamics. *Computer Graphics*, 25(4), 1991.

[85] Jane Wilhelms. Dynamic Experiences. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann Publishers, Inc., 1991.

[86] Jane Wilhelms. Toward Automatic Motion Control. *IEEE Computer Graphics and Applications*, 7(6), 1987.

[87] Jane Wilhelms. Using Dynamic Analysis for Realistic Animation of Articulated Bodies. *IEEE Computer Graphics and Applications*, 7(6), 1987.

[88] Jane Wilhelms. Using Dynamics for the Animation of Articulated Bodies Such as Humans and Robots. In *Proceedings of Graphics Interface '85*, 1985.

[89] Jane Wilhelms, Matthew Moore, and Robert Skinner. Dynamic Animation: Interaction and Control. *The Visual Computer*, 4(6), 1988.

[90] Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy Constraints on Parametrized Models. *Computer Graphics*, 21(4), 1987.

[91] Andrew Witkin, Michael Gleicher, and William Welch. Interactive Dynamics. *Computer Graphics*, 24(2), 1990.

[92] Andrew Witkin and Michael Kass. Spacetime Constraints. *Computer Graphics*, 22(4), 1988.

[93] David Zeltzer. *Representation and Control of Three Dimensional Computer Animated Figures.* PhD thesis, The Ohio State University, 1984.

[94] Jianmin Zhao. *Reaching While Preserving Integrity of Groups of Joints.* Technical Report (unpublished report), Department of Computer and Information Science, University of Pennsylvania, 1991.

[95] Jianmin Zhao and Norman I. Badler. *Real Time Inverse Kinematics with Joint Limits and Spatial Constraints.* Technical Report MS-CIS-89-09, Department of Computer and Information Science, University of Pennsylvania, 1989.

[96] Hanqi Zhuang, Zvi S. Roth, and Fumio Hamano. A Complete and Parametrically Continuous Kinematic Model for Robot Manipulators. In *IEEE International Conference on Robotics and Automation*, 1990.