Guided Control of Intelligent Virtual Puppets

By

Daniel Alexander Taranovsky

A thesis submitted in conformity with the requirements

for the degree of Master of Science.

Graduate Department of Computer Science

University of Toronto

Abstract

Guided Control of Intelligent Virtual Puppets

Daniel Alexander Taranovsky
Master of Science, 2001

Graduate Department of Computer Science, University of Toronto

Controlling the motion of virtual characters with many degrees of freedom can be difficult and time consuming. For some applications, complete control over all joints at every time step is not necessary and actually hinders the creative process. However, endowing the character with autonomous behaviour and decision-making capabilities completely absolves the user of clearly specifying his intentions. In many circumstances the ideal level of control allows the user to specify motion in terms of high-level tasks with timing and stylistic parameters. The user is not encumbered by low-level details, while retaining complete control over the motion's semantic interpretation. This relatively unexplored level of motion specification is termed "guided control", and is the focus of our work. We present the issues and results encountered from implementing a prototype animation system with guided control of a virtual puppet.

Acknowledgements

```
Knowlege:  Ye euery man whan ye to deth shall go
           But not yet for no maner of daunger.
Eueryman:  Gramercy, Knowlege, with all my herte.
Knowlege:  Nay, yet I wyll not from hens departe
           Tyll I se where ye shall become.
```

Everyman, Scene 17. John Skot (1521-1537?)

I am most thankful for my mother, father, and brother, whose unfaltering support and encouragement has made everything possible. This thesis is dedicated to them.

I am privileged to have worked among the talented, dedicated people at the Dynamic Graphics Project. I thank my supervisor Michiel van de Panne for his patient and knowledgeable advice. Professor van de Panne's comments and constructive scrutiny greatly improved the work. My second reader, James Stewart, also diligently read my thesis. I am grateful to both of these men for their guidance and suggestions. Petros Faloutsos and Joe Laszlo were always generous with their time and deserve special thanks for their technical help.

I acknowledge the financial support of the Government of Ontario, the University of Toronto, and the Department of Computer Science.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer animation has become prominent in developed societies around the world. It has affected the way we are entertained and communicate information in a variety of domains. Medicine, engineering, and education have adopted computer animation as an indispensable medium for modeling data and simulating virtual environments.

Computer animated films, such as Pixar's "A Bug's Life", use modern technology to tell a feature length story that appeals to people on an emotional level, rather than being of interest solely from a technical perspective. Blue Sky Studio's "Bunny" is a short computer animated film that has been awarded an Academy Award in recognition of its artistic qualities. This contrasts to an era where computer graphics and animation resided on the fringes of the scientific and artistic community. Interactive video games provide people with a form of entertainment unlike any other, and have become as widespread and popular as film. Virtual reality simulators allow people to train themselves in dangerous or rare circumstances that may not be feasible to practice in real-life. Pilots, emergency staff, and medical doctors have all benefited from the use of computer animation in virtual simulators. Finally, engineers use computer animation to design better aircraft or office space more efficiently and reliably.

The work presented in this thesis is of interest to society because of its contribution to the state of the art in computer animation. We consider a new way of controlling virtual puppets previously unexplored termed *guided control*, which allows the user to generate figure animation with relative small effort while maintaining task-level control over the motion.

1.1 Motivation

In virtually any discipline, ideas are conveyed using some level of abstraction to avoid tedious, redundant details. A script in theatre or movies directs the actions of a character in high-level terms.

DECIUS BRUTUS Great Caesar,--
CAESAR Doth not Brutus bootless kneel?
CASCA Speak, hands for me!
[CASCA first, then the other Conspirators and BRUTUS stab CAESAR]
CAESAR Et tu, Brute! Then fall, Caesar.
[Dies]

Figure 1.1 Julius Caesar, Act III Scene I. William Shakespeare (1564-1616).

The author of the play does not specify the joint angles of the character's arm, or worry about the actor remaining upright after translating the actor's centre of mass.

A musical score will specify the timing and intonation of the notes.

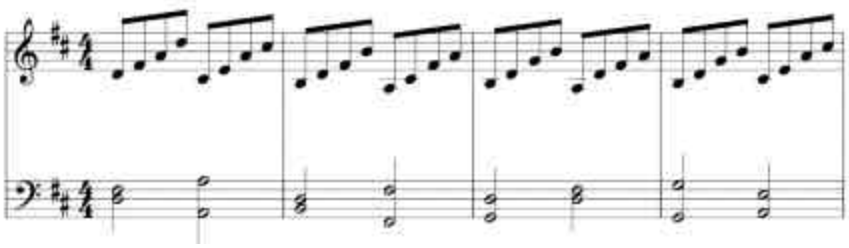

Figure 1.2 Canon in D Major, Johann Pachelbel (1653-1706).

The composer does not have to include precise information about the pianist's hand position, or the force applied to keys when producing a particular sound. Such details are avoided for several reasons. First, writing inspiring music is difficult enough without having to specify

every parameter in the environment. Second, there is an assumption that the pianist already knows how to produce notes on his instrument. The pianist is skilled enough to know proper hand position, and the appropriate joint rotation and angular velocity of the fingers to produce sound from his instrument.

Much like composing music, timing the actions of characters to produce realistic motion while instilling an emotional response from the audience is difficult. Just as a composer is not hindered by having to over-specify the state of the environment, the user interacting with the virtual puppet may want to be relieved of over-specification as well. Despite avoiding low-level details, composers have control over the critical aspects of the music, such as pitch and duration of notes. Users controlling a virtual puppet may want to influence similar characteristics of motion as well. Specifying the velocity over time or the perceived emotion associated with a movement are two examples. Unfortunately, there do not exist many tools to provide users with such a level of abstraction.

Keyframing tools require low-level specification of the figure state at several critical points in time. Keyframe animation is analogous to specifying the pianist's hand position and state of each finger joint throughout the performance. Behavioural animation endows the characters with autonomous decision-making capabilities, and does not allow the user to specify a desired sequence of tasks over time. The musical counterpart of behavioural animation is having a skilled pianist and telling him or her the style, era, and general stylistic description of the music one wants to hear without making any reference to notes or specific musical pieces. Motion data processing tools can generate motion that is limited by the properties of the original signal. Motion capture techniques are similar to making a tape recording of all individual notes in the musical scales, and replaying each note in a specific sequence to generate music. Clearly there is room in the realm of animation for generating motion in a level of abstraction that can take advantage of the animator's skill and creativity, while not overburdening him with excessive parameter specification.

A composer is given the luxury of having skilled musicians throughout the world who can make music according to his specifications. To implement a similar level of abstraction for specifying character animation, we must endow the synthetic puppet with the ability to interpret and execute task-level commands. Consider an animator who wants a character to grasp a coffee cup with the left hand. To successfully perform this task, the virtual puppet

must acquire some knowledge about the environment and its own structure. First, the character should be aware of at least one coffee cup, and knows which cup the animator is referring to. Second, the character must know how to move in a natural way to approach the cup and to pick it up. These points imply that there is some knowledge-based motion demanded of the puppet.

A musician is equally fortunate to have composers who are able to write beautiful, inspiring musical scores. Musicians depend on the composer's ability to sequence notes in a way that generates music, rather than random noise. In the same context, the motion of a virtual character is determined by the user's ability to sequence tasks and motion primitives. The user's ability to specify motion is further constrained by the interface and level of control offered. A user cannot direct a puppet to jump if there is no input that semantically maps to this command. Likewise, a composer cannot include explosions or popping noises in the piece, since these sounds do not correspond to notes. If one is interested is generating task-level motions with user specified primitives, then the interface should accommodate this level of control. In the context of guided control, the animator needs an interface that maps a level of abstraction consistent with unambiguously specifying objects, points in space, and timing parameters. The interface and level of control should be powerful enough to respect the user's intentions, yet not be overly cumbersome.

1.2 Thesis Contributions

The goal of this work is to develop an intelligent animation tool with interactive, guided control. We attempt to give enough control over the character to respect the animator's intentions and creativity, while not overburdening him or her with over-specification. "Guided control" implies autonomous resolution of some parameters when generating the final motion sequence. "Intelligent" animation implies system interpretation of ambiguous, high-level user input.

Guided control can be characterized by user specification of velocity and stylistic parameters, but avoids low-level joint state specification. Our system's intelligence stems from three features.

- Interpret animator's commands in context with the current state of the environment.
- Identify and reasonably respond to conflicting or impossible commands from the user.

- The puppet should execute user commands with humanly natural, realistic motion. The state of the environment and task difficulty should be immaterial to the realism of the final animation.

The scope of the system implemented is simplified in several respects to achieve the above objectives. The most important simplifications are the torso puppet model illustrated in Figure 3.8, and the absence of dynamic stability in the motion model described in Section 4.5.

1.3 Potential Applications

Guided control can be applied to any scenario where mid-level task specification is useful. Production animation can benefit from guided control by generating complex motion sequences quickly and efficiently. Animating a scene such as a factory assembly line requires a distinct sequence of object manipulation tasks to be performed by a large group of characters. For such an example, the animator may wish to give up some control for convenience and less cost, especially if the motion is applied to characters in the background. Production costs of foreground character animation can be lowered by quickly generating an initial motion sequence to be later refined and improved.

Guided control can be applied to adventure games where the user commands object manipulation tasks, or demands a lower-level control than typical captured motion sequences offer. Finally, the level of control presented in this thesis can be applied to virtual characters presenting information where body movements and hand gestures are fundamental to conveying the message. Virtual weathermen and repair instructional videos are two examples.

1.4 Thesis Organization

Chapter 2 presents a survey of literature relevant to our topic. An overview of animation techniques is presented, and recent research results are summarized to place the work in context. Chapter 3 presents an overview of our prototype animation system. Chapter 4 describes a methodology for scheduling the execution of tasks. Chapter 5 presents an algorithm for positioning the puppet's limbs according to user input. Our method for

interpreting the user's intentions is presented in Chapter 6. Chapter 7 describes the prototype animation system's implementation, and the animations successfully produced. Chapter 8 summarizes the work and proposes outstanding problems for further research.

1.5 Summary

This chapter introduced the inspiration and vision of a guided control animation system. An abstract analogy with similar artistic expression was presented in Section 1.1. Section 1.2 gave a more specific description of guided control, and Section 1.3 proposes potential applications. The remainder of the thesis was outlined in Section 1.4.

Chapter 2

Literature Survey

This chapter introduces principal literature and research in computer animation. An overview of animation techniques is presented to place our work in context with existing methods. Biomechanics and ergonomics literature is summarized to give the reader a background in relevant issues and invite future work. Specifically, a background in biomechanics and ergonomics helps assess the motion and posture models to be discussed in Section 4.5 and 5.7 respectively. The biomechanics literature surveyed focuses on kinematic models of realistic human motion and is directly applicable to our topic. Some biomechanics theory presented has been modeled by similar animation systems, such as Badler's Jack system discussed in Section 2.4 and 2.8 [BPW93]. The ergonomics literature discusses empirical notions of comfort and preferred postures, which is important to our model discussed in Chapter 5. Section 2.1 introduces metrics for characterizing animation techniques. Kinematic and dynamic methods are surveyed in Sections 2.2 to 2.6. Behavioural techniques are introduced in Section 2.7. Interactive methods cover techniques with more abstract control over the character's motion, and are presented in Section 2.8. Results from ergonomics and biomechanics research are presented in Section 2.9 and 2.10 respectively.

2.1 Characterizing Animation Techniques

Independent of the technique used to position the character over a sequence of frames, there is an issue of how much animator control is warranted by the system. A *parameter* refers to any information from the animator used to influence the final motion signal. Every animation technique has a unique level of parameterization in three respects. The *degree* of parameterization refers to the number of parameters the animator must resolve for the system. The parameter's level of *abstraction* indicates how much interpretation or autonomous reasoning from the animation system is required. Finally, the *interactivity* of the system indicates how often parameters are specified.

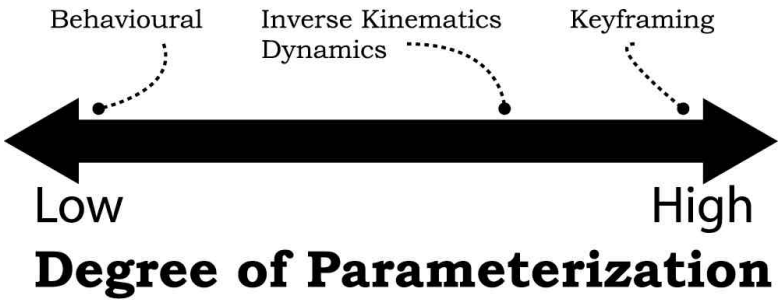

Figure 2.1 Relative number of parameters to be resolved.

In general, high degrees of parameterization and low levels of abstraction require skill and considerable effort from the animator. The extreme case is keyframing techniques discussed in Section 2.3, where all degrees of freedom in the figure are specified explicitly in at least two frames. Low degrees of parameterization and high levels of abstraction relieve the animator of having to specify too many parameters, but provides reduced control over the animation. Behavioural techniques and autonomous agents introduced in Section 2.7 are examples of low parameterization and high abstraction. Systems that offer low abstraction and low parameterization, or high parameterization and high abstraction are rare. A reason for their scarcity is that most systems aim to maximize control or minimize effort, while these two parameter mixes offer neither scenario. An example of low abstraction and low parameterization is the animator specifying the angle of the elbow and nothing else. Conversely, a system where the animator provides volumes of complex psychological information about the characters and how they relate to other characters and the environment

is an example of very high abstraction and parameterization. Figure 2.1 and 2.2 present a relative comparison of animation techniques in terms of parameterization and abstraction.

The ideal mix of parameter control depends on the task. Production animators demand complete control over characters in every frame. Virtual reality simulators and interactive autonomous agents require characters to animate autonomously and reason about appropriate future behaviour. An interesting and relatively unexplored level of parametric control is *guided* control, where the animator is able to specify task-level commands along with a set of motion primitives. Task-level commands specify motion goals such as “walk to door”, or “pickup object”. Motion primitives modify some property of the motion signal, such as the velocity or perceived emotion associated with the movement. Guided control is part of the novelty of our system, which is described in detail in subsequent chapters. A comparison of applications with respect to interactivity and abstraction is presented in Figure 2.3.

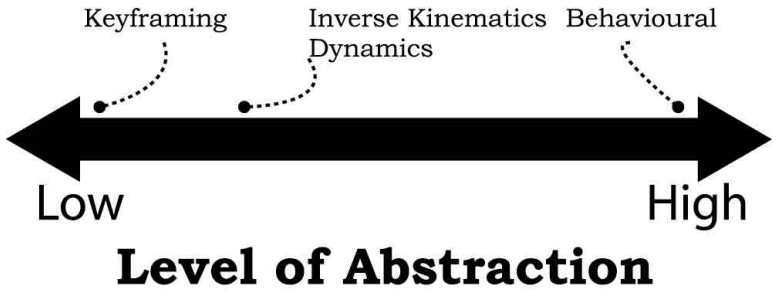

Figure 2.2 Relative ambiguity of user motion directives.

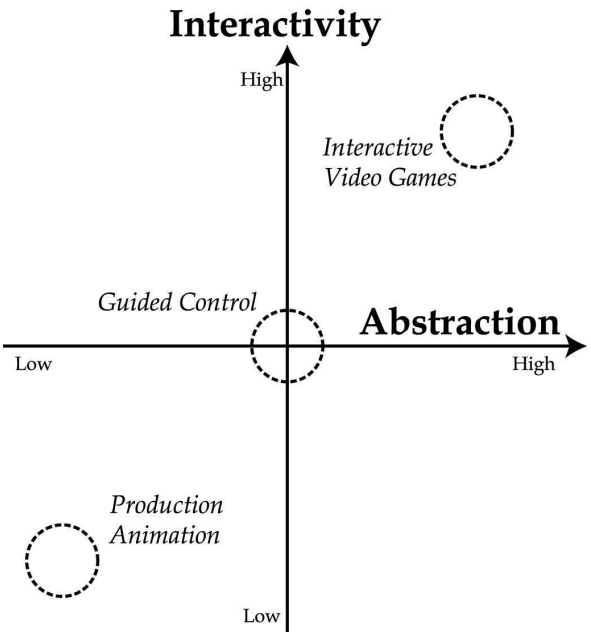

Figure 2.3 Comparison of animation applications.

High interactivity, high-levels of abstraction, and low parameterization is common in interactive video games. Action video games commonly have only a few high-level commands available to the user, such as “pass ball”, “jump”, or “pickup object”. However, the nature of user interactivity implies a degree of randomness or unpredictability in the control. For the animation to appear realistic the system must correctly interpret and respond in context with the environment. For example, if a user instructs a character to pickup an object, the character must move towards the object before attempting to grasp it. Once near the object, the character must delegate joint rotations to reach the object naturally. This level of interaction and interpretation with the system is complex, primarily because the character must reason about his surroundings and have some knowledge about its own kinesiology. This is one of the objectives of guided control, and requires collaboration between computer science, biomechanics, and psychology.

Low-levels of interactivity implies the animator resolves the value of input parameters few times relative to the length of the animation. Script-based animation allows the user to specify system parameters at the beginning of execution. The system then computes and outputs motion data without further interactivity with the user. The quantity and type of parameters the animator can specify in the script depends on the degree of

parameterization and level of abstraction. The disadvantage of this level of interactivity is the animator is given no visual feedback as the animation progresses. The advantage is that there are limitless degrees of freedom the animator can effectively control. The animator is also given the opportunity to pause and reflect. Real-time animation assumes the animator knows what the character ought to do at all times. Replaying motion capture data is a script-based animation technique with low interactivity, low abstraction, and high parameterization. Perlin's "Improv" system [PG96] is script-based with low interactivity, high abstraction, and low parameterization.

Figure 2.4 Relative user input feedback.

Interactive real-time animation, also termed *performance animation* or *digital puppetry* is the highest level of interactivity one can achieve, where the animator can specify parameters continuously over time with immediate feedback. It is attractive because it allows a fluidity of expression and spontaneous improvisation difficult to synthesize any other way. Performance animation is commonly associated with low abstraction and high parameterization, and aims to maximize the animator's control over limbs, facial expressions, moving eyes, and other degrees of freedom in real-time. This leads to demands for novel input devices such as joysticks, optical body suits, and cyber-gloves. Not only do these input devices make performance animation possible, but make animation accessible to people otherwise unskilled in animating characters. The interface allows real-life actors, dancers, and other people trained in using their bodies in expressive ways to control the virtual characters. The disadvantage with such an interface is that the level of control is limited by the independent degrees of freedom of the input device. Designing an interface that would allow an animator to control many degrees of freedom, including facial expressions and a number of high-level commands is difficult. If one were to have a character with more

degrees of freedom than the human body, one would not be able to effectively control it in real-time. In fact, one could argue the number of degrees of freedom humans can effectively control is much less than this number, independent of the input device. To overcome this, some puppetry systems will use multiple puppeteers to control a single virtual character. Another solution is to animate the character in several passes. Each pass will control a sub-set of the degrees of freedom, and the final result will combine the result of each pass in a single animation.

Beyond the quantity, abstraction, and frequency of parameter specification is a question of how the parameters will be specified. There is a range of input devices available, ranging from standard keyboards to eye-tracking mechanisms, each with their own qualities and limitations. An in-depth discussion on this topic is firmly in the realm of human-computer interaction, and is beyond the scope of this thesis.

2.2 Kinematic Specification of Articulated Figures

The motion model presented in this thesis is kinematics-based, which expresses motion in terms of joint angles, coordinates, and orientation over time. There is no influence of mass or force involved in determining the figure's differential state. Methods that consider the system's physical properties are referred to as *dynamic methods* and are discussed in Section 2.6. Virtual characters' bodies are typically modeled by articulated figures, which are a set of rigid links adjoined by joints. The orientation and position of figure's rigid links determines the figure's state. Each variable to be resolved in determining a rigid link's state is a *degree of freedom*, and the number of degrees of freedom in the structure is dependent on the number of movable joints and the dimensionality of the space. For example, a single rigid link in 3D space is characterized by six degrees of freedom since a vector of six elements specifies its position and orientation. The state of a static single rigid link can be specified as $\{x_0, y_0, z_0, f, g, l\}$, where x_0, y_0, z_0 denotes the position of the link in space and f, g, l refers to the object's local x-axis, y-axis, and z-axis rotations, which deterministically specifies the link's orientation. This concept is illustrated in Figure 2.5.

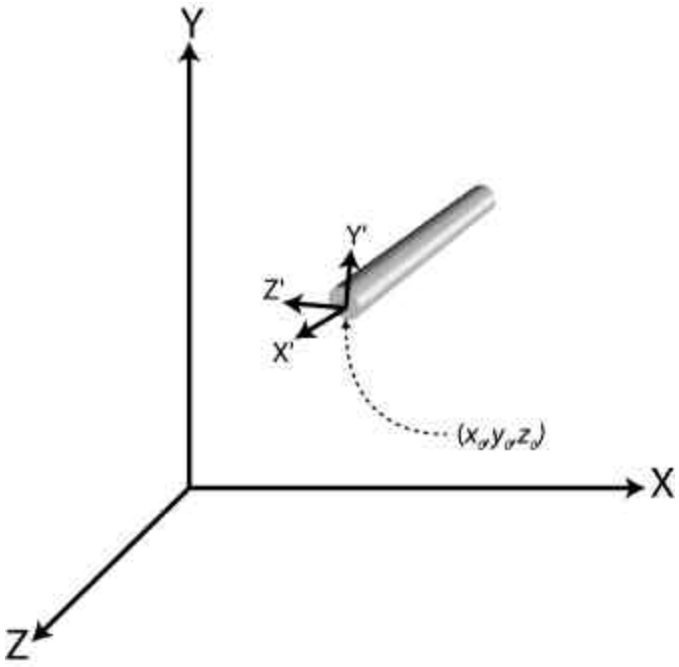

Figure 2.5 State specification of a single rigid link

An articulated figure is typically specified kinematically by the position of the figure in world coordinates, and the orientation of every joint in the figure hierarchy. An arbitrary 3D figure consisting of N links is characterized by $6N - C$ degrees of freedom, where C represents constraints imposed by the joints that remove degrees of freedom in the articulated figure. Let us consider a simple two-dimensional model of a human arm with four degrees of freedom. The shoulder's position and orientation is specified by a three dimensional vector $\{x, y, q\}$. Since the adjacent links in the arm are adjoined at their respective joints, the figure's state is deterministically specified by another local rotation at the elbow denoted b. The state of the system is expressed as an eight-dimensional vector $\{x, y, q, b, dx, dy, dq, db\}$, where dx, dy, dq, db specifies the differential coordinates over time. To effectively animate the arm, the animator must control the evolution of this vector over time. The vector's elements can be specified with either forward or inverse kinematics, which are introduced below.

Forward kinematics deterministically specifies the state of all degrees of freedom. The animator determines the value of the state vector's elements with no abstraction. That is, postures are expressed in the same terms as the low-level internal representation of the

articulated figure. Figure 2.6 illustrates the concept of forward kinematic specification with our simple two-dimensional human arm. The articulated figure is denoted F, where $F = \{J_0, J_1\}$, $J_0 = \{x_0, y_0, q\}$, $J_1 = \{b\}$.

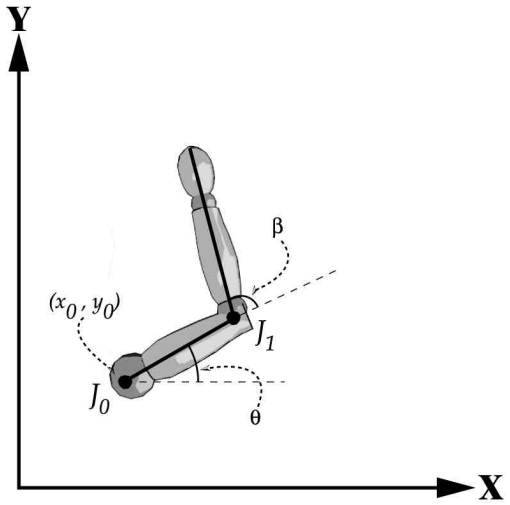

Figure 2.6 Forward kinematic specification of articulated figure.

Inverse kinematics provides the animator with a higher level of abstraction for specifying the figure state vector. The animator specifies the position of the figure's root joint and the desired position and orientation of one or more joints in world coordinates. The joints that are given desired position and orientation are referred to as *end effectors*, and constrain the system to a particular configuration. Typically the posture specification is *redundant*, or *under-constrained*. These terms refer to circumstances where there exists more than one figure posture that satisfies the end effector constraints. Solving for joint parameters given an inverse kinematic specification is a difficult problem, and developing effective algorithms for use in animation is an on-going research topic. Despite its difficulties, inverse kinematics provides the animator with a convenient level of abstraction for specifying figure postures. Ultimately, the inverse kinematics algorithm must resolve all ambiguities and define the state of the figure in forward kinematics. In the example of the two-dimensional human arm, the position of the shoulder $\{x_0, y_0\}$ and desired position of the tip of the hand $\{x_e, y_e\}$ can specify the figure's configuration. This specification is under-constrained with

two possible configurations, as illustrated in Figure 2.7. The figure F is specified as $F = \{J_0, e\}$, $J_0 = \{x_0, y_0\}$, $e = \{x_e, y_e\}$.

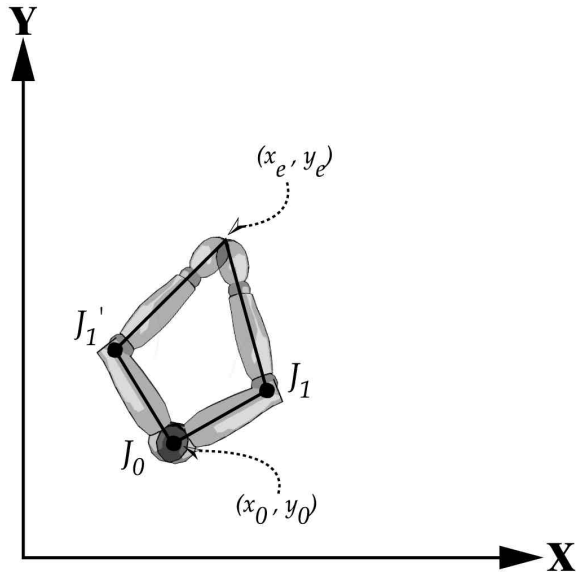

Figure 2.7 Inverse kinematic specification of articulated figure.

Keyframing and motion signal processing are examples of forward kinematics applied to the animation of articulated figures. Keyframing is a technique discussed in Section 2.3 where the animator explicitly specifies the state vector of the figure. Motion signal processing manipulates a pre-computed sequence of state vectors to generate new figure motion, and is introduced in Section 2.5. Inverse kinematics specifies the state of the figure in a higher level of abstraction than forward kinematics, and is a useful technique for generating character postures. Inverse kinematics in terms of its application to posture design and positioning of human figures is discussed in Section 2.4. An introduction of techniques for solving inverse kinematics is presented in Section 5.1.

2.3 Keyframing Techniques

Keyframing is the most common method of developing production animation. The process involves rendering animation sequences frame by frame, and gives the animator limitless control over the character's movement [FvDFH90]. Alternatively, critical frames can be drawn with a time reference for each frame. The in-between frames, in a process called

inbetweening, interpolate between critical frames. Each frame is redisplayed in sequence and the resulting image appears dynamic over time. For the resulting image to appear fluid without “flicker”, a high frame display rate is required. The particular technique employed to display the images as a contiguous animated sequence depends on whether the motion is viewed on film, computer screen, or video.

Techniques in keyframing stem from traditional two-dimensional drawing animation developed in the 1930s. To date it remains the predominant method of character animation for foreground characters in feature film production. Walt Disney studios pioneered animation as an art form with such classics as "The Three Little Pigs". More modern production animation such as Pixar's "Toy Story", "Luxo Jr.", and Blue Sky Studio's "Bunny" are examples of keyframed three-dimensional computer animation.

Despite the impressive results of keyframing for production animation, there are serious drawbacks associated with this method. Positioning characters manually is a tedious, challenging task even for experienced animators. Conveying emotional behaviours and modelling physical phenomena effectively is an art form reserved for those with the prerequisite talent. A few seconds of animation can potentially take a meticulous animator several days to produce.

Specifying animation in terms of joint angles or constraints over time requires significant involvement by the animator. The animator must typically specify postures with forward or inverse kinematics in a number of frames to ensure natural, smooth motion. Inbetweening is performed to relieve the animator of specifying every frame. Unfortunately, interpolating between user-specified keyframes does not always result in appropriate motion. First, interpolating from one posture to another with human realism is challenging. While the user specified postures in select keyframes may be biomechanically and physically plausible, the intermediate frames resulting from naïve interpolation may not be. Furthermore, even if physical laws are respected, the motion may still look unnatural. Humans have grown accustomed to observing human locomotion and are sensitive to subtle discrepancies between natural and synthesized human motion. Second, humans instinctively adjust and readjust their motion over time to avoid collisions and maintain balance [Lat93]. Interpolation is not always possible if the motion requires careful navigation around

obstacles. For these reasons, production animation is synonymous with extensive animator control over the figure's position at every time step.

Reeves introduced the Coons patch algorithm, the Miura algorithm, and the Cubic metric space algorithm for inbetweening keyframes [Ree81]. Among the three, the Coons patch algorithm performs best overall in terms of generality, smoothness, and computational efficiency. Steketee et al. give an analysis of keyframe interpolation requirements [SB85]. A prototype interpolation system is presented with second-order continuity, user control over motion parameters, and seamless blending of successive motion sequences. Cubic B-splines are proposed as a suitable interpolation function.

Lasseter discusses the principles of traditional hand drawn animation as they pertain to three-dimensional computer animation [Las87]. Subtle characteristics of motion, such as anticipation and exaggeration, have been conveyed in two-dimensional hand drawn animation for many years. Lasseter claims the methods of conveying these properties are just as applicable to three-dimensional computer animation as two-dimensional drawing animation.

2.4 Positioning Articulated Figures with Inverse Kinematics

The position of body segments in space is critical to the accomplishment of reaching and manipulation tasks in constrained environments. Inverse kinematics provides a useful abstraction for generating postures that incorporates one or more constraints on the figure's position. The user gives up explicit control over the state of all joints in favour of a convenient level of abstraction for specifying postures. However, positioning the character with constraints does not intuitively lend to generating postures for all tasks. The literature discussed in this section focuses on the application of inverse kinematics to controlling virtual human characters. Inverse kinematics is also thoroughly researched in robotics for effectively controlling manipulators [Pau81]. Solving inverse kinematics is introduced in Section 5.1.

Inverse kinematic solutions are keyframes satisfying posture constraints imposed by the animator. How one chooses to interpolate between these keyframes is an inbetweening problem, and the generated critical frames must accommodate the inbetweening algorithm being used. However, instead of employing an inbetweening algorithm the animator may

express the motion as a trajectory over time. Every time step along the trajectory defines desired end effector coordinates, so one can position the figure with an inverse kinematic solver. This approach will be problematic, however, since inverse kinematic solutions return the final end effector position rather than the path taken to arrive at a solution. Since inverse kinematic solutions are underconstrained, the resulting motion curves over time may not be continous. This will result in "jittering" or sudden posture jerking between consecutive frames. Lee et al. resolved this problem with a multi-level B-spline fitting technique [LS99]. In fact, any curve fitting technique for scattered data interpolation can be used, and the specific formulation selected is dependent on the desired properties of the final motion curve.

Badler from the University of Pennsylvania has developed a powerful experimental test-bed called “Jack” for researching algorithms in inverse kinematics, ergonomic analysis, and motion planning [BPW93]. Phillips, Zhao, and Badler developed an interface to interactively position an articulated figure by directly manipulating the state of specific joints [PZB90]. The user is also able to interactively specify an end effector goal position, and an inverse kinematics algorithm automatically generates the appropriate figure posture. The constraints are specified by mouse input converted to geometric transformations. Phillips and Badler provide a mechanism of interactively controlling figure postures by specifying various geometric constraints [PB91]. Passive constraints are implemented to satisfy the user’s intentions while ensuring the figure posture appears balanced. This is accomplished by positioning the centre of mass as an implicit hard constraint. Zhao’s dissertation provides an in-depth overview of satisfying geometric constraints with numerical and iterative optimization techniques [Zha96]. Although the research is interesting from a posture control perspective, Badler notes that figure motion can not intuitively be specified in terms of posture constraints.

Badler’s Jack system addresses many of the issues in this thesis involving posture design and task-level control of articulated figures. The user is able to specify postures by imposing multiple geometric constraints, such as end effector position and orientation, or restricting an end effector’s position to a plane, line, or region in space. The Jack system can generate postures depending on the strength model of the figure [LWZB90], as well as the geometric constraints imposed by the user [PB91]. The posture generator described in Chapter 5 differs from this approach in terms of the degree of user specification. We attempt

to automatically position figures by modeling a bias towards ergonomically correct postures, with geometric constraints being applied by the system based on the position of impenetrable surfaces and objects in the environment. Although this model does not compensate for the relative weight of objects during lifting motions, it provides a general model that can be applied to a wide range of motions described in Section 3.6.1. We attempt to position the figure from a knowledge-based approach with a notion of perceived naturalness.

Kondo developed an inverse kinematics algorithm for generating humanly natural arm positions [Kon94]. The algorithm involves an initial estimate based on results from neurophysiology research [SF89a][SF89b]. The same results are used in our algorithm presented in Chapter 5. Hand position and orientation is then satisfied with a constrained optimization algorithm. Loftin et al. developed an algorithm that originates from a similar neurophysiological relationship between the orientation of the wrist and the height of the elbow [LMY97]. This algorithm compensates posture naturalness in favour of increased performance.

Siciliano et al. proposes a recursive algorithm for overlapping constraints of multiple end effectors [SS91]. The algorithm is applied to a chain with arbitrary number of rigid links. Baerlocher and Boulic present a similar inverse kinematics formulation for satisfying multiple tasks of arbitrary priority [BB98]. Baerlocher et al. consider obstacle avoidance, centre of mass positioning, and end effector control as three potential tasks to be satisfied simultaneously. Multiple instances of these tasks were overlapped to achieve a particular posture. The algorithm was extended to specifying multiple soft and hard constraints for full body posture design [BB00]. Boulic and Thalmann developed an algorithm for generating postures that incorporate physical support for some portion of the human body [BT97].

Paul introduces an algebraic and inverse Jacobian solution to satisfying orientation and positional objective functions respectively [Pau81]. Welman applied two inverse kinematics algorithms to a human figure [Wel93]. Various bending and reaching motions were achieved, and the performance of each algorithm was recorded. The posture generator introduced in Chapter 5 uses a derivative of the cyclic-descent method discussed in this thesis.

2.5 Motion Capture and Motion Processing

Rotoscoping is perhaps the oldest means of duplicating the motion of a human subject. A human is filmed using film, video, or strobe photography while performing some desired motion [FvDFH90]. Joint angles are measured from the individual frames and are applied to a virtual character to be animated. Differences in the body dimensions of the human subject and the target character are problematic, as is the camera projection, which changes over time for a stationary camera and a moving subject.

Motion capture is a modern approach to rotoscoping [Stu94] [Mai96]. Specialized optical, infrared, electromagnetic, or mechanical indicators are placed on the subject to record its motion over time. The results are realistic since joint rotations of the live subject are directly applied to the virtual character. The motion data is recorded in real-time, and can involve multiple subjects performing complex cooperative motion sequences. This is particularly useful for interactive video games where the user commands a limited number of sophisticated motions, such as pitching a baseball or swinging a racquet.

There are several serious drawbacks to this technique, however. The hardware required for capturing motion can be costly and difficult to calibrate. Once the data is captured, noise must be removed from the signal, and optical sensor occlusions must be dealt with. The motion data recorded is specific to characters with the same dimensions and structure as the subject. The strength and skill of the subject limit the generated motions resulting from this technique. This can be seen as an advantage as well, since the animator does not have to be concerned with ensuring that biomechanical properties are respected.

Modifying or retargetting motion capture data for variable environments and character dimensions is an on-going research area [Gle98]. Concatenating independently sampled motion sequences into a fluid, continuous animation can also be difficult [RGBC96]. Each motion sequence is recorded with specific initial and final states that are unlikely to coincide with those of other motion sequences, leading to motion discontinuities that must be resolved using other methods.

The animator may also wish to adjust the nature of the data by instilling emotion, or modify stylistic characteristics of the motion. Applying these transformations is an on-going research area [UAT95]. Recent attempts have manipulated the motion signal itself by varying the amplitude of a frequency band, or scaling motion amplitudes at specific points in the

signal. For example, let us assume we have motion data of a man walking into a room and placing a book on a table with neutral emotion. How the motion can be modified so the character will appear angry as he walks to the table and slams the book down is not clear. A general method for computing such a transformation has not yet been demonstrated.

Witkin et al. modified motion capture data by specifying constraints as keyframes at certain points in time [WP95]. The constraints "warp" the original motion curve after interpolating the data points with an arbitrary interpolating spline. Fine details of the motion are preserved, and constraints are satisfied. Witkin adapted a regular walking gait to step over and duck under blocks inserted in its path. Lamouret and van de Panne propose a method of adapting existing motion data to fit arbitrary situations [LvP96]. A database of possible motions is kept, and the best-fit motion is selected and adapted for the current situation. The technique was used to navigate a Luxo character across a two-dimensional terrain map.

Lee et al. represents modifications to the motion as a set of constraints [LS99]. The constraints were satisfied for each frame and the resulting data points interpolated with a multi-level B-spline fitting technique. This method was used to modify walking postures, apply the motion to rough terrain, and apply data to variable character dimensions. Popovic et al. modified motion data according to variations in the physical properties of the characters and environment [PW99]. A weightless moon-walk and limping motion were successfully produced.

Brudelin et al. used traditional signal processing techniques on motion capture data [BW95]. Interactive motion multiresolution filtering was implemented to create a graphical motion equalizer. Motion interpolation is used to blend multiple motion signals together. Waveshaping was used to impose joint limits. Blending two signals and applying local displacement to a single signal created new walking and waving motions. Similar work was done by Unuma et al. using Fourier series expansions [UAT95]. Existing motions were interpolated and smooth transitions between motions were achieved. Most interestingly, emotional characteristics of motion were extracted to create "tired" and "brisk" variations of a captured walking motion. Rose et al. used space-time constraints and inverse kinematics to make seamless transitions between motions [RGBC96]. Data can be placed arbitrarily in time and the system computes smooth transitions among all motion sequences. Rose also spliced cyclic data, such as walking motions, to produce motion sequences of arbitrary length.

Gleicher studied ways of applying motion data to characters of similar structure but variable dimensions [Gle98]. Gleicher represents desired motion characteristics as a constrained optimization problem, such as keeping at least one foot on the ground while walking. Walking motion sequences were adapted to satisfy imposed geometric constraints while minimizing signal differences over time.

2.6 Dynamics Techniques

Dynamic simulation provides a technique for creating animation consistent with the laws of physics. Objects and figure body segments are modeled as having mass and inertia. Physical simulation will consider gravity, friction, wind, collisions, and any other external forces acting upon entities in the system. Internal forces model torque exerted in joints by *actuator motors* in the articulated figure, or can be modeled more explicitly to mimic the biomechanics of real human and animal musculoskeletal structures. Actuator motors are the "muscles" of the articulated figure, controlled manually by the animator or automatically by dynamic controllers. It is the summation of internal, external, and reactive forces over time that determines the motion. The benefit of this class of techniques is the realism of the resulting movement. Subtleties of natural motion are difficult to achieve with kinematic methods, particularly because of the number of interactive forces to consider at every time step. The disadvantage of this technique is computational cost and control difficulties.

Controlling figures and objects to perform a desired motion is difficult. Given some arbitrary figure, it is not intuitive what internal torque is required to move the character subject to arbitrary external forces. For example, it is not obvious what torque should be applied by the left and right hip, knee, and ankle to produced a balanced walk of 1 m/s given the mass, centre of mass, and moment of inertia of each segment in the character's body. To relieve the animator of such tedious calculation, *dynamic controllers* are implemented. Dynamic controllers allow the animator to specify more abstract motion goals that are then fulfilled by the controllers.

Dynamic controllers are often developed manually by trial and error for a specific figure in a certain environment, and are sensitive to initial conditions. Developing robust controllers for even simple figures to remain balanced is tedious. Algorithms that

automatically generate dynamic controllers is an on-going research area, although recent results have been successful for simple figures and motions in a set environment. Extending controllers to figures of variable mass and dimensions, or adjusting a controller to compensate for different external forces is an open research problem.

Brotman suggests considering classical mechanics when computing inbetween frames [BN88]. By respecting laws of physics when interpolating motion parameters, the result appears more natural than standard interpolation techniques. The technique is restricted to linear dynamic systems, and was used to animate a truck and aircraft given the initial and final position, velocity, and orientation.

Raibert et al. hand crafted a number of dynamic controllers [RH91]. Quadruped creatures, biped creatures, and virtual kangaroos were modeled. Controllers for biped running and galloping, quadruped trotting, bounding, and galloping, and kangaroo hopping were implemented. Speed and posture control systems are discussed in the paper. Controllers were scaled to accommodate models of variable size. Hodgins et al. presents a set of hand-designed controllers for running, bicycling, and gymnastic behaviours such as vaulting and balancing [HWBO95]. Group behaviours were incorporated to avoid collisions within large groups of cyclists and runners. Secondary animations such as cloth simulation were also incorporated. van de Panne and Fiume propose a controller for simple structured figures called “sensor-actuator networks” [vPF93]. Articulated creatures are equipped with sensors to provide feedback on the state of the figure with respect to its environment. Sensors activate actuators through a network of weighted transitions. A variety of locomotion styles are achieved by varying the value of the transition weights.

Bruderlin et al. proposes a hybrid kinematic/dynamic model for generating human walking [BC89]. The technique allows the user to specify three locomotion parameters: forward velocity, step frequency, and step length. These parameters will effectively define a walking gait. The forces and torques required to swing the legs according this walking gait are computed by iteratively approximating various values and applying them until the timing and trajectory of the walking gait is satisfied. Laszlo, van de Panne, and Fiume present a technique for dynamic control of periodic, unstable motions such as walking [LvPF96]. Laszlo proposes a method to add closed-loop feedback to an open-loop stepping motion to

obtain stable walking gaits. Various open-loop controllers were used to generate stylistic walks, such as walking into a head-wind.

Hodgins et al. adapted existing controllers for running and cycling behaviours to figures of variable dimensions and masses [HP97]. Initial controllers were initially tuned for a standard adult male character. The controllers were successfully adapted to adult female and child figures in two stages. First, a knowledge-based adaptation of the controller based on geometric and mass scaling of the target character was performed. The scaled controller was further tuned with an automatic search of the local parameter space.

van de Panne, Kim, and Fiume explored ways of automatically generating dynamic controllers for arbitrary articulated figures [vPKF94]. The synthesized controllers used cyclic finite-state machines to produce periodic motions such as running and swinging. A genetic algorithm that iteratively tests and optimizes randomly generated gaits synthesized the cyclic pose control graphs. Grzeszczuk, Terzopoulos, and Hinton developed a neural-network technique for developing dynamic controllers [GTH98]. Through observation and multiple trials a neural-network is trained to control a specific figure. Examples of motion are evaluated according to prescribed animation goals, and successful motions are integrated into the transition weights of the neural network. This technique was used to successfully teach a truck to park in position, land a space module on the moon, and a dolphin to swim.

Torkos and van de Panne introduce a novel motion specification technique for quadrupeds using footprints [NvP98]. Each footprint specifies timing and orientation information. The footprints implicitly define the motion trajectory. A second pass optimizes the trajectory to accommodate a simplified dynamic model and comfort heuristic.

Huang and van de Panne propose a search method to achieve complex dynamic motions such as flips and cart-wheels [HvP96]. The optimal sequence of movements is selected according to an evaluation function. A search of all possible motions at every decision-making opportunity in the animation is considered. The search for the best motion sequence is similar to techniques in chess programs for searching optimal sequences of moves.

2.7 Behavioural Techniques

Animation controlled by very high levels of control abstraction is characteristic of behavioural animation. Research in the area is considered interdisciplinary since the characters exhibit autonomous intelligent behaviour. The characters are endowed with preferences and decision-making capabilities to appropriately react to initial environmental conditions. The characters will select their own high-level behaviours depending on the animator's parameter specifications and the character's own behavioural model. Generally, the animator will not interact with the characters, but rather the animation is allowed to take its course with continuous closed-loop feedback of environmental stimulus and agent reactions. This method is considered by some to be on the fringe of computer animation and artificial life, and is more than simply a technique with very high-level control.

Reynolds modeled flocking, herding, and schooling behaviours of animals [Rey87]. Each animal in the group is given individual priorities and basic decision making capabilities. Remarkably, the collective behaviour of the group exhibits complex behaviours, while only rudimentary intelligence is distributed throughout the group.

Tu and Terzopoulos created artificial fish with sophisticated locomotion, sensory, and behavioural models [TT94]. The fish displayed behaviours such as mating rituals, predator avoidance, and schooling instincts. The fish were modeled with independent behaviour routines, and the underlying motion primitives were dynamically based. Thalmann proposes an approach to implementing virtual actors that autonomously solve problems in path searching, obstacle avoidance, and game playing [Tha99]. A model for learning and forgetting behaviours is implemented. A sensor-based virtual tennis player was implemented, where selected behaviours are heavily influenced by visual stimuli.

Blumberg et al. propose a system that endows the animator with multi-level control [BG95]. Behavioural, motivational, and motor-level commands are all integrated in the control mechanism. The motivation behind the system is to demonstrate that behavioural and task-level animation is not mutually exclusive. The user is able to direct the characters at various levels of interaction. Mateas proposes a system for interactive drama [Mat99]. The "Oz" system in Carnegie-Mellon University allows the animator to direct characters with high-level primitives at critical points in the story.

Funge, Tu, and Terzopoulos considered a layer of abstraction above behavioural models [FTT99]. While behaviour techniques plan and choose goals according to knowledge bestowed and acquired, Funge et al. studied *cognitive modeling* which govern how knowledge is acquired, referenced, and internally represented. Beyond autonomously choosing among a set of goals, cognitive modeling provides the possibility of characters discovering new behaviours previously unimagined.

2.8 Interactive Control

Specifying animation with a higher level of abstraction than kinematic and dynamic methods while endowing the user with control over a subset of motion directives is a powerful way of controlling virtual characters. As described in Chapter 1, task-level abstraction is a convenient means of specifying motion for certain applications. However, realistic animation of articulated figures is complex, and determining parameters worth abstracting to a higher level is task specific. The research presented in this section propose animation techniques that provide the animator with abstraction beyond the low-level kinematic and dynamic details, while allowing the user to specify his intentions more specifically than behavioural techniques.

An early attempt to implement task-level control of articulated figures is proposed by Korein and Badler [KB82]. The paper was published in 1982, and addresses the problem in terms of effective inverse kinematics and interpolation algorithms for controlling a primitive six-link, two-dimensional chain. In 1985, Zeltzer published a paper discussing animation abstraction in terms of a three-level hierarchy [Zel85]. “Guided motion” in this paper is the lowest level of abstraction and would be identical to forward kinematic techniques described earlier. “Animator-level” systems program animation much the same way one uses standard programming languages today. “Task-level” systems incorporate knowledge of the environment necessary to execute animator-level motor programs. Task-level abstraction is proposed as a future goal beyond what the technology is capable of. In 1986, Ridsdale et al. published a paper describing the recent results of Simon Fraser’s Figure Animation Project [RHC86]. The author states, “After about ten years of continuous research in this area we are still not sure whether truly convincing animation of the full range of human movement is feasible”. There are other interesting papers from a historical perspective, such as

[Zel82][AGL86][CS86][Stu84][MFV84]. These papers place the sophistication of today's systems in perspective, as well as describe the origins of task-level control problems that persist today.

Lee et al. presented an algorithm to compute the trajectory of the arm when lifting objects of variable weight [LWZB90]. A strength model is implemented to calculate a biomechanically realistic trajectory of joint angles and end effector positions from initial to final state. Initial optimal trajectories are computed. A comfort level is calculated at every time step as a ratio of the current exerted torque divided by the maximum possible torque. If the comfort level degenerates below a certain level, the trajectory constraints are relaxed. The sophistication of the biomechanical model compromises its general application to arbitrary tasks.

Levison, Badler, Geib, and Moore implemented a system that translates high-level directives into task-level commands [LB94][GLM94]. "SodaJack" is a Prolog-like predicate language, and "OSR" is LISP-like in syntax. The user specifies his intentions in higher level terms than guided control, but more specifically than behavioural techniques. The commands are parsed, and the feasibility of each goal at an instance in time is assessed. The problem is approached from a planning perspective, where a feasible order of motions is sought. The issues discussed in these papers are resolved in Chapter 6 as they apply to our work.

The research effort most related to guided control of virtual articulated figures is Badler's Jack system [BPW93]. There are important similarities and differences between our prototype and Jack. The similarities serve as appropriate alternative solutions to key problems in task-level control, while the differences illustrate the breadth of the problem this work attempts to cover. Three important aspects of Jack's functionality are posture design for ergonomic analysis of human factors, path planning, and natural language specification of task-level commands. Much of the work done in posture design seeks to measure human exertion and comfort performing object manipulation tasks in constrained environments [LWZB90]. The models proposed by Badler are more sophisticated for generating specific, constrained motions, which is ideal for human factors research. The model proposed in Chapter 5 generates postures for the 3D puppet model presented in Figure 3.8 without user specification of constraints or explicit goals. Our model aims to generate appropriate postures in the full range of task space. The posture planning algorithms developed for the Jack

system incorporate collision avoidance and strength factors when determining an appropriate trajectory for a task [BBGW94]. Our system does not consider path planning or collisions throughout the motion signal, but ensures that collisions are avoided when the task is accomplished. Collisions and external constraints on the puppet's postures are discussed in Section 5.6. The task specification of Jack focuses on decomposing high level natural language into lower-level motion primitives [LB94][GLM94], as mentioned in the previous paragraph. The fundamental difference between guided control and the Jack system is the level of animator control over the timing and sequence of motion. The level of specification and system interpretation described in this thesis is more intuitive for controlling a virtual puppet performing arbitrary tasks, while the sophistication of Jack is appropriate for studying human factor and task resolution issues.

The motion resulting from performance animation is based on the animator's skill in controlling multiple degrees of freedom in real-time. Felix the Cat by deGraf/Wahrman Inc. is an example of a kinematic performance animation system that uses multiple passes to control all the degrees of freedom [Sor89]. The first pass controls the head of Felix; the second pass controls the facial expression and eyes; the third pass implements a lip-synchronization. As one employee states, "More things are going on with Felix than are humanly possible for one puppeteer to do at once". Sturman wrote a paper discussing appropriate input devices for performance animation [Stu98]. Kalra et al. describe modeling issues associated with performance animation. Virtual tennis and dancing environments were developed [KMMS98]. Laszlo, van de Panne, and Fiume researched real-time interactive physically-based animations [LvP00]. Using simple input devices such as a keyboard and mouse, the animator is able to perform sophisticated motions with simple two-dimensional characters. The novelty of this approach is that the animator is specifying parameters for dynamic motion in real-time rather than joint angles.

Koga et al. developed a motion planning algorithm for object manipulation with two human arms [KKKL94]. The motion planner considers legal object grips and arm rotations to find a path from the initial state to the goal state. The arms are positioned with a novel inverse kinematics algorithm, and the motion between arm postures is kinematically based. To achieve the goal state, several grasping and regrasping motions may take place. The motion is computed autonomously given the goal state. The approach used to generate

humanly natural postures is similar to the methods discussed in Chapter 5. Kalisiak and van de Panne consider animation as a lower level motion-planning problem [KvP00]. The environment is a two-dimensional terrain map with specified grip points to position end effectors. The planner is equipped with a preferred mode of locomotion and stride length. Inverse kinematics is used to tailor the preferred posture to accommodate the grip points and terrain dimensions. The algorithm is based on a gradient descent process, which is subject to local minima. Further reading on the classical planning problem can be found in [FN71].

Rijpkema et al. developed a knowledge-based approach for animating grasping motions of the human hand [RG91]. An initial posture from a database is selected based on the geometry of the object, and the grasp is finished with a kinematic clasping method. The user can interactively control the hand by specifying the position of a single finger, a group of fingers, or selecting a hand posture from the database.

Perlin worked on virtual puppetry using a number of predefined dance movements [Per97]. Each movement is defined in terms of initial and final joint angles for all degrees of freedom in the figure. Every degree of freedom interpolates in one of four phases of sine and cosine functions. By structuring motions in this way, one can effectively combine motions that use independent sets of joints. Transitions from one motion to another are graceful, since successive motions are phased-in by applying a decaying weight coefficient to the current motion. Perlin's research group also built a scripted animation system called "Improv" [PG96][PG99]. This system is a layer of abstraction built on top of Perlin's virtual puppetry system. Users can write scripts to specify which motions to execute at a particular time. Virtual actors in the system can interact with one another. A behavioural model is implemented by executing motion scripts if environmental variables exceed some predefined threshold.

Bruderlin et al. developed an interactive tool for generating running motions [BC93][BC96]. The system incorporates biomechanical models to calculate appropriate limb trajectories. The user can customize cyclic running gaits by adjusting the velocity, step length, step frequency, and height of the stride. Arm, torso, and pelvis movement in the running stride are interactive parameters adjustable by the user.

Several papers were written proposing specific motion representations or discussing the characteristics important to any motion abstraction. Badler believes the notation used to

express movement should incorporate varying levels of abstraction, from task-level to kinematic specification [Bad86]. Badler considers Labanotation as a possible representation for computer animated motion. However, Badler notes that this abstraction does not include facial expressions, dynamic properties, or muscular contractions. Badler categorizes motion into four groups: rotations and translations, end effector goals, end effector trajectories, and dynamic forces that controls a motion.

Drewery and Tsotsos propose an animation system that gives animator-level control over characters [DT86]. The definition of animator-level control is identical to [Zel85], where the motion and environment are specified similar to a high-level programming language. Movements, objects, and articulated figures are instances of data structures. Low-level interaction with the figure is possible through directly referencing variables in the figure's data structure. Task-level motions are essentially procedures that reference one or more data structures, and provide a level of abstraction to compute trajectories and object positions. What is unique here is the notion of data structures or “frames” describing the state of objects, articulated figures, and motions. Motions are expressed as objects in the same context as object oriented languages, with local data and procedures referencing object and figure data structures. This is similar to the models proposed in Sections 3.4, 3.5, and 3.6.

Morawetz et al. propose an animation scripting language syntactically similar to BASIC [MC90]. Task-level commands are specified as function calls with parameters for the speed and start times of motions. Some motion functions take directional parameters, such as “Look right fast”. The animation specification is compiled to generate an animation script that outputs a series of keyframes. Overlapping motions that do not cause conflicting goals for figure body segments are possible. The most novel aspect of this work is the notion of cyclic motions. Animators can specify walking or waving motions that repeat until halted or interrupted by conflicting motion. Some similarities can be drawn between the level of user control and the motion models described in Sections 3.3 and 3.6.

Zeltzer developed an alternative method of modeling task-level commands that differs from the compiled script mode of interaction of Morawetz and Drewery in [Zel85]. Zeltzer proposes the notion of a “movement queue” that contains task-level directives input by the user. The proposal is of interest because the “task manager” acts as a simple operating system, scheduling processes in a first-come, first-serve basis. The user specifies parameters

to the task manager, which are passed to the mid-level motor programs and low-level local motor programs. The task manager's function is similar to the motion scheduler introduced in Section 3.2.2 and discussed in Chapter 4.

2.9 Ergonomics

The ergonomics literature is relevant to posture design since it studies comfort and preferred positions in some environmental context. Despite extended studies and surveys of comfortable and uncomfortable seated positions, there are few rules without exception that constitute a comfortable upper body position. The healthiness or unhealthiness of a posture is determined by physiological variables, such as oxygen consumption and volume of fluid at certain extremities. Some notions of comfort can be determined by postures that result in injury over a long-term. This section surveys some research in ergonomics relevant to the seated puppet scenario proposed in Chapter 3, and is pertinent to the discussion in Section 5.7.

There are several general recommendations given by Tichauer to maximize comfort in a seated position [Tis78]. Tichauer suggests that comfort is maximized when the elbows are kept down, moments on the spine are minimized, and forward reaches are kept short. Naderi et al. defines the general principles for seated workplace design [NA89]. Naderi recommends that workers seated at a table avoid extreme positions of the joints, avoid unnatural postures, and maintain proper elbow height in relation to the work surface height.

Corlett proposes other principles of interest to seated work environments [CB76]. Torso twisting is undesirable due to the stresses imparted on the spine. The worker should be able to maintain an upright and forward-facing posture during work. Chaffin estimates that a bend forward in excess of twenty degrees can result in increased fatigue and serious injury [Cha87]. Corlett also suggests that work activities should be performed with the joints at about the midpoint of their range of movement, particularly the torso, head, and upper limbs.

Ostrom developed a checklist of principles for ergonomically proper seated positions [OGH92]. The checklist suggested that desirable seated positions include the spine slightly arched and leaning forward. The elbow joints should remain low and to the side of the worker. Twisting of the head and trunk should be avoided. Further reading that supports these principles can be found in [AM91].

One can reasonably assume that postures that are comfortable appear natural and are preferred over uncomfortable postures. Postures that cause injury or stress contort the body unnaturally, and provide us with more evidence of what is considered a natural seated working posture. Given the above principles from ergonomic study, our postures should avoid several characteristics: unnecessarily raising the elbow, acute twisting or bending of the torso, and extreme positions of the joints.

2.10 Biomechanics

Modeling human joint movement is an on-going research area in biomechanics. The state of the art is attempting to model single joint motion for isolated, artificially simplified movements. Unfortunately, these results are unlikely to be directly applicable to natural, unrestrained, multi-joint motion. Latash claims that analysis of single joint movements provides a basic framework for studies in multi-joint tasks [Lat93]. Section 4.5 describes how we naively apply single joint motion models to multi-joint tasks, which is a simplification of the complex relationship between task and performance parameters. In the context of this work, task parameters include positional goals for the hand under strict temporal constraints. The performance parameters correspond to joint position and velocity curves to perform the task. Unfortunately, the biomechanical and neurophysiology aspects of the tasks in our scenario are remarkably complex, and are not fully understood by researchers. A function to model multi-joint movements with velocity and positional parameters is not known, and many researchers believe that a comprehensive model does not exist. Research in developing human kinematic models of motion is directly applicable to our work, and is presented below to invite future research and put the models presented in Chapter 4 and Chapter 5 in context. Results of single-joint experiments are presented first, followed by studies in motion variability and multi-joint tasks.

Implementing guided control of a character with human anatomy and realistic motion in response to arbitrary task-level commands reduces to solving some fundamental problems in biomechanics [Lat93]. Nicholai Bernstein is regarded as a pioneer in the study of human motor control, and has provided unique insight into the problem of modeling human motion. Bernstein was the first to express human motor control as a function of input parameters. The human's intentions with respect to placement of a hand, speed of the motion, or accuracy in

the movement's trajectory are all considered an input parameter to the motor control function. The output of this function is the kinematics of the resulting motion. Defining the function that maps all factors influencing human movement to a kinematic representation of the motion has been a fundemental problem researched in biomechanics and neuroscience for almost a century.

Fast movements of a single joint to a target position exhibit a smooth, bell-shaped velocity curve during the range of motion, with symmetric acceleration and deceleration phases. Certain motion can exhibit asymmetric acceleration and deceleration phases. As the joint approaches the final position small fluctuations in acceleration are common for fast and slow single joint movements. Decreasing the speed of motion typically results in more variability in trajectory and less smooth velocity curves. A flat region in the middle of the velocity curve characterizes slow movements, and is illustrated in Figure 2.8.

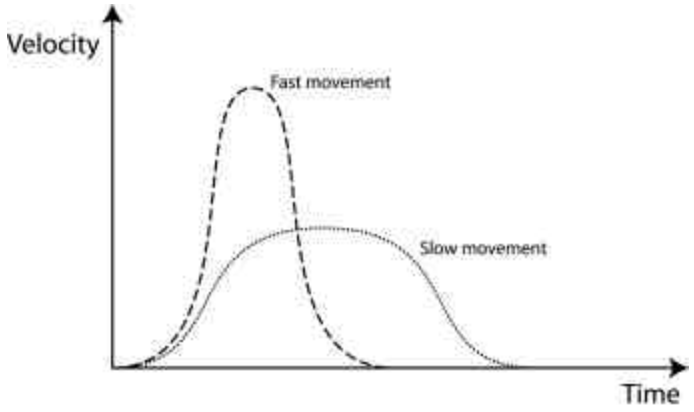

Figure 2.8 Form of velocity curves for slow and fast movement.

The variability of a single joint's velocity is modeled by variations in the target size and position. Fitts suggested a logarithmic relation between the target position, allowed margin of error, and total movement time. The relationship is known as Fitt's Law, and is defined as

$$T = a + b \cdot \log_2 \frac{2D}{W}, \qquad (2.1)$$

where T is the movement time, D is the distance of the target, W is the target width, a and b are experimentally observed constants. This relationship characterizes non-repetitive arm movements with visual recognition of explicitly presented targets. Fitt's Law considers the time required to correct the trajectory of the end effector during the motion. This implies that visual perception of the target must be maintained throughout the movement, and that there is sufficient time to correct the trajectory of the end effector.

Experimental data suggests that movements demanding accurate placement of end effectors are composed of several submovements. The trajectory of the hand is corrected with every submovement, and the number of submovements in the overall trajectory depends on whether the person perceives a correction to be required. Increasing the number of submovements lends to irregularities in the trajectory of the hand. Milner and Ijaz have found a correlation between the number of submovements and the difficulty of the task performed [Lat93]. As the required accuracy of the task increases, the duration of the movement's deceleration phase increases along with the number of submovements during the deceleration phase. A velocity curve with extended deceleration phase is presented in Figure 2.11. Impulsive or very fast motions, and motions without any visual feedback for corrections are typically composed of a single submovement. Fitt's Law has been reformulated for tasks requiring two submovements.

$$T = a + b \cdot \sqrt{\frac{D}{W}}. \qquad (2.2)$$

Determining relationships between task and performance parameters in multi-joint motor control is beyond the current state of neuroscience research. Bernstein reformulated the multi-joint control problem, known as "Bernstein's Problem", as the process of control for overcoming the ambiguity caused by redundant degrees of freedom. Some research models the joint coordinates with respect to the position of the end effector, while other research explores the trajectory of the end effector itself.

Experimental observations of the end effector's trajectory during planar movements suggest preference for certain motion properties. For example, translating the end effector on a plane results in nearly straight trajectories of the end effector. The end effector's

displacement over time is characterized by a bell-shaped velocity curve. However, Latash remarks that joint rotations in the human body that position the hand in space over time are characterized by a "more complex, multi-phasic form" that depends on the position of the end effector with respect to the body [Lat93]. As the speed of the planar motion increases, the trajectory remains a straight line. The shape of the velocity curve is also preserved, with a simple scaling procedure applied. However, the trajectory of the individual joints through their respective joint space is not preserved as the speed of the end effector increases. Soechting and Lacquaniti have shown for planar movements of a two-joint system that their maximum speed will be achieved at approximately the same time [Lat93].

The movement time for curvilinear motions is dependent on the radius of the trajectory. The relationship is given below.

$$V = b \cdot r^{2/3}, \qquad (2.3)$$

where V is the velocity of the end effector, b is a constant, and r is the radius of the motion trajectory. Tasks with complex trajectories are segmented into primitive units, such as straight lines and simple curves. Inconsistent with (2.3), the time to perform drawing and writing motions is proportional to the number of primitive units in the motion, rather than the length or radius of the primitive units. In other words, drawing smaller or larger shapes and characters does not affect the total movement time, but drawing shapes with more edges will linearly increase the time to execute the motion. Movement speed decreases at the primitive units' points of intersection.

The above results for planar motions are all applicable to three-dimensional unrestrained multi-joint movements. However, additional characteristics of three-dimensional movements have been experimentally observed. The end effector trajectory exhibits segmentation along particular planes in space. This implies that humans prefer to perform motion of complex trajectories as a sequence of planar movements.

Bernstein's problem subsumes the problem of effectively generating natural human postures. Bernstein's problem not only considers the final posture assumed in performing a task, but also the path through joint space for all joints participating in the movement. We would like an algorithm that will determine each joint's rotation throughout the performance

of the task. As mentioned previously, the relationship between the end point trajectory and the individual joints' rotation over time is not clear, and some scientists believe that a significant relationship does not exist. Two relevant experimental observations are a result of analyzing the distance between the joint centre of rotation and the end effector position in world coordinates. Let us denote the distance between joint i and the end effector as d_i. Kaminski and Gentile claim that joints with centre of rotation further from the end effector will begin rotating towards the target before more proximal joints [Lat93]. This observation is expanded to a model for solving Bernstein's problem. The Berkinblit-Gelfand-Feldman model proposes that the angular velocity of joint i is proportional to the distance d_i [BGF86]. That is, more proximal joints will rotate slower than joints with larger amplitude of motion. Joints that can contribute significantly to the displacement of the end effector towards the goal will move sooner and faster than joints will smaller d_i, which is characteristic of joints that cannot move the end effector towards the goal as readily. This model does not explain all experimental observations of voluntary human movement, but does model the rotation of joints in terms of the end effector position, which supports experimental observation [Lat93].

Models of human voluntary movement have been proposed by neuroscientists to correlate with observed experimental data. The models attempt to map task parameters to velocity curves for all joints in the human body, which determine the kinematic state of the figure over time. Cruse and Brewer assumed that there exists a joint position of maximum comfort [Lat93]. Their experimental data identified the positions of optimal comfort for the wrist, shoulder, and elbow. For these joints, comfort was maximized in the middle of the physiological range of motion. The degree of discomfort is measured as a parabolic function about the joint's optimal position, and the total degree of discomfort is measured as a sum of all joints. Voluntary movements were modeled by minimizing the cost function. Hogan attempted to minimize the amount of *jerk* when executing a motion [Lat93]. Jerk refers to the motion's acceleration time derivative, and can be correlated with the amount of stress and wear on the joints. More complex cost functions consider the weighted sum of several optimization parameters, such as effort, discomfort, expended energy, and jerk. Seif-Naraghi and Winters modeled human voluntary movements by combining several optimization

functions that consider kinematic, dynamic, and electromyographic patterns from experimental observation of human subjects [Lat93].

Adamovich and Feldman suggest that joint movements of varying amplitude will preserve properties of the velocity curve [AF84]. In particular, the rotation of a single joint of *x* degrees will be performed over some preferred time and bell-shaped velocity curve. Motions of *2x* degrees for the same joint results in a motion of longer duration, with the shape of the velocity curve preserved. Adamovich et al. claim that this process is performed by the nervous system by superimposing two motions of *x* degrees after applying a scaling factor. This study supports the idea of a primitive motion program from which all other motions are generated by applying a scaling procedure.

Ostry et al. demonstrated several properties of velocity curves for single joint elbow movements [OCM87]. Other research supports the claim that motions of variable amplitude and duration preserve the shape of their velocity curve. However, this study observed small differences in the velocity curve for motions of variable duration. As the duration of a motion increases, the duration of the deceleration phase increases, as illustrated in Figure 2.11. This result was reproduced in [Nag89]. The velocity curves of continuous motions were compared with discrete movements. Continuous motions were defined as a repetitive, continuous series of movements, and discrete movements positioned the elbow in a specific pre-defined position. Continuous movements were characterized by less sharp acceleration and deceleration phases illustrated in Figure 2.9. This research also demonstrates that the preservation of the velocity curve form is characteristic of the arm, but does not apply to other joints, such as the jaw.

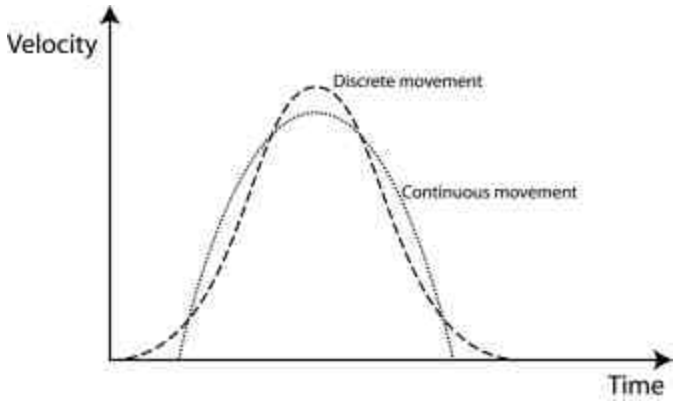

Figure 2.9 Velocity curves of discrete and continuous movements.

Gottlieb et al. studied the affects of practice on the velocity and accuracy of elbow rotations [GCJA88]. The results indicate a remarkable improvement in the accuracy and execution time over a ten-day period. This would suggest the constants *a* and *b* in Fitt's Law are dependent on the amount of practice and initial familiarity with the movement. It is not obvious how these results can be applied to more complex tasks requiring multiple limbs and degrees of freedom.

Mustard et al. studied the kinematic and electromyographic properties of wrist movements in variable velocities, amplitudes, and external loads [ML87]. As mentioned previously, velocity curves for voluntary movements of the arm are characterized by a bell-shaped form with some oscillations at the end of the motion. These oscillations have been observed to increase as the level of demanded accuracy increases. The study by Mustard et al. demonstrates the widely documented observation of large oscillations as the velocity of the movement increases and the demanded accuracy remains constant. As the velocity of the motion increases, the velocity curve degenerates to an N-shaped curve instead of a smooth bell-shaped form as shown in Figure 2.10. This is a result of a large correction at the end of the movement to position the end effector at the desired position.

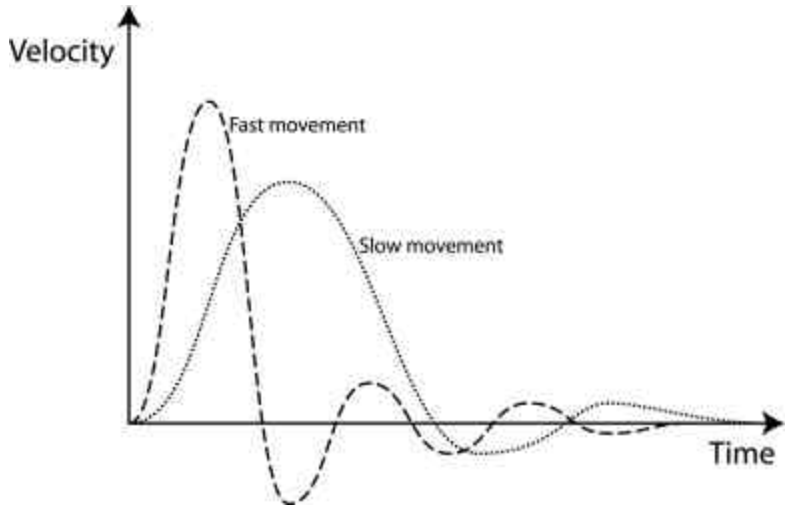

Figure 2.10 Velocity curve oscillations in fast and slow movement.

Nagasaki studied joint trajectories of skilled movements [Nag89]. Skilled movements are characterized by remarkably consistent joint trajectories when performed in variable circumstances. The research presented supports minimizing a jerk function as a suitable model for controlling human arm movements. It is suggested that skilled motions will minimize arm jerk over time, and subsequently exhibit smooth velocity curves.

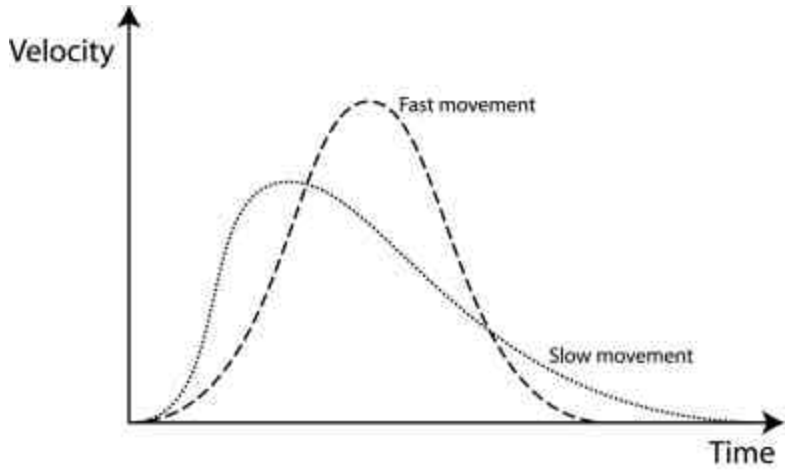

Figure 2.11 Symmetric and asymmetric velocity curves.

The research of Nelson, Ostry, and Nagasaki support the notion of an invariable relationship between the maximum velocity and average velocity during a motion [OCM87][Nag89]. The relatively consistent form of the velocity curves is a result of the following ratio.

$$c = \frac{V_{\max}}{V_{avg}} \ , \qquad (2.4)$$

where $V_{\max}$ is the maximum velocity during the motion, V_{avg} is the average velocity throughout the motion, and c is a constant. This ratio remains constant for motions of similar velocity independent of the amplitude, duration, or inertial load of the motion. Nagasaki demonstrates that the value of c depends on whether the motion's velocity can be characterized as slow, medium, or ballistic. Ratio values for such movements are

approximately 1.85, 1.95, and 2.05 respectively. Medium velocity is loosely defined as any movement with duration of approximately 0.5 seconds.

Researchers have supported the notion of modeling skilled human motions of medium velocity by an optimization function that minimizes the trajectory jerk. The motion's total observed jerk J is calculated by the following equation.

$$J = \frac{1}{2} \cdot \int_0^T \left(\frac{d(a(t))}{dt} \right)^2 \cdot dt, \qquad (2.5)$$

where T is the movement time, and $a(t)$ is acceleration. This optimization function generates smooth, symmetric velocity curves but does not explain experimental observation of asymmetric velocity curves. Nagasaki proposed an alternative model that constrains the amount of jerk at various stages in the movement [Nag89]. This modified model obtained asymmetry of the velocity curves that match observed data when executing certain motions.

van der Meulen et al. studied kinematic properties of fast, goal-directed arm movements in men [vMGvGG90]. Their results suggest that fast, goal-directed motion is characterized by highly variable initial accelerations, and numerous corrections during the motion to continuously maximize accuracy. Hoffman et al. demonstrated kinematic profiles of wrist velocity subject to variable amplitudes and intended speed [HS86]. Cordo discovered remarkable accuracy in coordinating multi-joint movements [Cor90].

2.11 Summary

This chapter presented an overview of concepts and literature relevant to generating a kinematics-based animation tool with guided control over a human character. Section 2.1 described the differences and similarities of prominent animation techniques in terms of parameterization, abstraction, and interactivity. Section 2.2 introduced kinematic specification of articulated figures, which is fundamental when discussing kinematic motion. Section 2.3 to 2.7 introduced well-documented approaches to generating computer animation, and serve to position our work in context with current trends in research. Section 2.8 described animation techniques closely related to the work presented in this thesis. Section 2.9 and 2.10 give the reader a background in ergonomics and biomechanics to help evaluate the validity of models proposed in subsequent chapters. The literature surveyed in biomechanics focuses on kinematic analysis of arm motions. This section presented many well-documented models of human motion that are directly applicable to implementing a kinematics-based animation system for generating realistic human motion.

Chapter 3

System Overview

This chapter introduces the models and modules designed to implement a guided control animation system. The system implemented controls a virtual puppet seated at a table performing tasks that may involve objects in the environment. Having a seated character implies we do not have to animate the lower body, but collisions with the table can occur. A diagram of the table scenario is presented in Figure 3.1.

In addition to selecting an environment for our prototype, we must also consider the types of tasks to be performed by the puppet. Guided control allows the user to specify a task and associated motion primitives to modify its execution. For example, the user can direct the puppet to yawn and specify its duration, velocity curve, or any other parameter that modifies the characteristics of the motion but not its semantic interpretation. The set of tasks available to the user in our prototype can be loosely categorized as object manipulation tasks and tasks independent of the environment. Object manipulation is an important subset of tasks in our work since it requires the puppet to perceive the current state of the environment and react in context. This is one of the most challenging and interesting features of guided control, since the resulting motion will vary according to the position of the object. The specific tasks and motion primitives implemented in our system are introduced in Section 3.3 and further discussed in Chapter 7.

Figure 3.1 The table scenario.

Guided control of a puppet is best described by an example. Assume we would like to command the puppet to eat. A guided control specification would involve instructing the puppet to grasp the fork with the left hand, take food from the plate, and move the fork to the mouth. Each of the three subtasks' execution can be modified by user-specified stylistic parameters. The richness and variety of the motion primitives available to the user is specific to the application's implementation.

The architecture and data flow of the system is presented in Figure 3.2 along with a subsequent explanation of the annotations. Section 3.2 gives a functional description of the three modules implemented in our prototype. In any virtual scenario, one must design a model of the motion being executed, the synthetic puppets performing the motion, and the environment with which the characters interact. These models are described in Sections 3.6, 3.5, and 3.4 respectively. The parameter space controlled by the user models his intentions, and is introduced in Section 3.3.

3.1 System Architecture

The functional architecture of the system consists of several models and modules. The models are simplified representations of some real-world phenomena. In our scenario, the puppet, the motion, the environment, and the user's intentions are represented by models with simplified properties. The modules process the models' interaction.

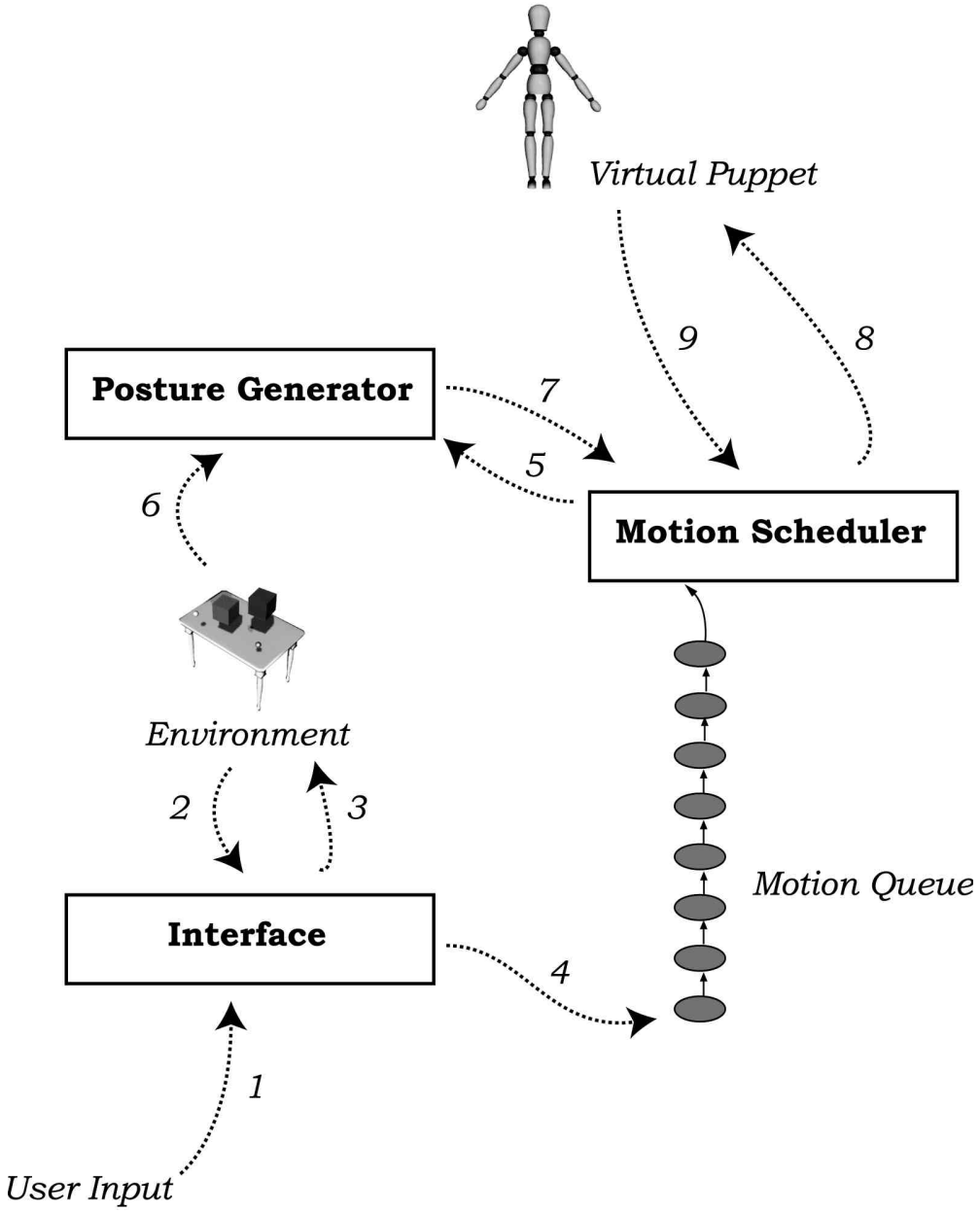

Figure 3.2 Diagram of system architecture.

The progression of the user's intentions to the final executed animation is illustrated with arrows in the above diagram. The arrows are referenced below with annotations.

1. The user specifies his intentions to the interface.
2. The interface module checks the environment state to ensure the user's intentions are plausible.
3. The system updates the environment if the task modifies its state. Subsequent input will be interpreted based on the new environment state, even though the current task may not have executed to completion. This step is justified since the motion queue is processed in first-come, first-serve order, and tasks may be input faster than they can be executed.
4. The interface interprets the task in context with the current state of the environment. The positional goals of the puppet's body segments to accomplish the task are determined. For example, a reaching motion will require the hand to be placed at a particular object. The interface will determine the current position of the object, and reformulate the task in terms of coordinates. A motion model is created that encapsulates the system's interpretation of the user's intentions.
5. The motion scheduler processes the motion queue in first-come, first-serve order. The positional goals of the next motion model in line are expressed as an inverse kinematics problem.
6. In addition to the geometric constraints imposed by the task, the table imposes additional constraints on the inverse kinematics solution. In general terms, the extra constraints imposed by the table ensures that the puppet's limbs do not penetrate the table while executing the task.
7. A forward kinematic specification of the puppet posture required to accomplish the task is output. This posture is referred to as the task's *goal state.*
8. The puppet animates from its current state to the goal state.
9. When the puppet is finished animating, the motion scheduler is signaled to consider the next task in the motion queue.

3.2 System Modules

There are three modules that process the user's input at various stages in the animation pipeline. The modules are labeled by boxes in Figure 3.2. This section presents the function of each module.

3.2.1 Interface

The interface processes input directives from the user. The operation of the interface is presented in Chapter 6. The module first interprets the user input in context with the current state of the environment. Non-executable tasks are rejected by the system. The interface will then reformulate the task in terms of the environment's configuration. For example, assume the user inputs a task to reach for some particular object. The interface will specify the task as reaching for a particular point in space, where the space corresponds to the current position of the object. Finally, the interface creates a motion model for the task input, along with the motion primitives required to animate the puppet while executing the task. Motion models are introduced in Section 3.6.

The interface accepts the current state of the environment and a task with its associated primitives specified by the user. The interface will output an updated environment state if the task moves objects or modifies the environment in any way. A motion model is output and added to the end of the motion queue. We can model the interface module as a function f_i.

$$(e', m_k) \leftarrow f_i(e, t)., \qquad (3.1)$$

where e and e' are the current and updated environment state respectively, t is the task specified by the user, and m_k is the output motion model, where k refers to the number of entities in the motion queue.

3.2.2 Motion Scheduler

The motion scheduler processes the motion models in the motion queue in a first-come, first-serve order. Each motion model in the queue has positional constraints according to the task being executed and the state of the environment. The motion scheduler will specify these constraints as an inverse kinematics problem K. The operation of the motion scheduler is described in Chapter 4.

The motion scheduler takes tasks from the motion queue as input. The output is a kinematic chain with end effector positional and orientation goals. We can model the motion scheduler as a function f_m.

$$K \leftarrow f_m(\{m_0, m_1, \ldots, m_k\}).\,, \tag{3.2}$$

where $\{m_0, m_1, \ldots, m_k\}$ is the contents of the motion queue. $K = \left(\{j_i,\ j_{i+1}, \ldots\}, e_p, e_o\right)$, where $\{j_i,\ j_{i+1}, \ldots\}$ is a subset of the puppet's joints, e_p is the goal position of the end effector in world space, and e_o is the goal orientation of the end effector in world space.

3.2.3 Posture Generator

The posture generator will solve inverse kinematics problems subject to constraints imposed by the environment. The function of the posture generator is to position the puppet with human realism to accomplish some task. The input of the posture generator is a task expressed as an inverse kinematics problem from the motion scheduler. The output is a forward kinematic representation of the puppet's posture. The output set of joint angles for every degree of freedom in the kinematic chain is termed the *goal posture* or *goal state.* The function of the posture generator is modeled as a function f_p.

$$J \leftarrow f_p(K).\,, \tag{3.3}$$

where $K = \left(\{j_i,\ j_{i+1}, \ldots\}, e_p, e_o\right)$ and $J = \{j_i,\ j_{i+1}, \ldots\}$ is a forward kinematic specification of the task's goal posture.

3.3 User Input

The user specifies motion primitives and high-level tasks. The motion primitives are esthetic qualities of motion that do not modify the semantic interpretation of the task. High-level tasks can be categorized as those that interact with the environment, and those that do not. When processing tasks that interact with the environment, the state of the puppet and objects determine how the user's intentions will be interpreted by the system. For example, sliding the puppet hand upward will result in motion that is dependent on the current state of the puppet. The specifics of any reaching action depend on the current positions of the target objects. Examples of tasks that do not interact with the environment are scratching the head, looking at one's watch, or waving. The goal posture for these tasks is determined independent of the puppet or objects' current state. The specific primitives and tasks implemented in our system are presented in Chapter 7. An overview of the parameters input by the user is given below.

The user can modify the motion's speed, interpolation function, scheduling parameter, and grip. The speed determines the duration of the animated motion in frames. The interpolation function will determine the motion's velocity over time. The scheduling parameter dictates to the motion scheduler if the task can execute concurrently with other tasks. The motion scheduler's operation is discussed in detail in Chapter 4. The grip specifies the desired position and orientation of the hand with respect to the entity being reached, and is further discussed in Section 6.1.1. The value of all these parameters is maintained by the system, and can be modified interactively by the user as described in Section 7.1 and 7.2.

Not all user-specified parameters are relevant to all tasks. For example, it does not make sense to specify a hand grip for a scratching motion. Grip parameters are only relevant to tasks that reach for objects or points in space, as explained in Section 3.6.1. The system simply ignores the value of motion primitives that do not have relevance in the semantic interpretation of the motion. In addition to motion primitives, some parts of the animation pipeline are not relevant to all tasks. For example, a yawning posture can not intuitively be computed with an inverse kinematics solver. In these circumstances, the posture generator is not helpful. We overcome this in our implementation by hard-coding certain postures in the system. These motions are called *general motions* and are introduced in Section 3.6.1.

The motion models in the motion queue each represent a task and its associated motion primitives. The motion model is created in the motion interface, and encapsulates the system's interpretation of the user's intentions. The model is placed in the motion queue to be processed by the motion scheduler introduced in Chapter 4. The model's primitives are initialized by copying the parameters' default value maintained by the system. The user does not have to specify all motion primitives for all input tasks. Instead, the user can simply modify the system's default value, and all subsequent motion models will copy these values. For example, if the user sets the speed parameter to some value, all subsequent motions input by the user will animate at this speed. Modifying the value of these parameters only affects new tasks input by the user. The models in the motion queue store the value of the parameters present at the time of their creation.

3.4 Environment

The environment consists of objects, space, and the table. Objects refer to the entities that the puppet can manipulate. Space refers to some point in world coordinates where objects or end effectors can be placed. The position of objects and space will dictate how the interface interprets the user's input in Section 6.1.1. The position of the table will influence how the posture generator computes the puppet's postures in Section 5.6. The environment in relation to the puppet is illustrated in Figure 3.1.

Objects are modeled as cubic volumes in space. The centre, width, and space and object relationships defines the state of the object. The centre coordinates correspond to the object's centre in world coordinates. The width spans perpendicularly from the centre to the volume surface. The space relationship is a reference to a space entity. The entity referenced by the space relationship is the current space occupied by the object. The object relationship references another object sharing the same space. The object referenced is stacked on top of the original object. A diagram of three objects sharing the same space is shown in Figure 6.8.

Space entities are modeled as points in world space. The state of each space entity is determined by a position expressed in world coordinates, and an object relationship. The object relation references one object occupying the space, and the referenced object is the bottom-most object in the stack. Some space entities are fixed in space, while others are dynamic and can be repositioned. Setting the position of dynamic space entities is a result of certain user-specified tasks, discussed in Section 6.3.

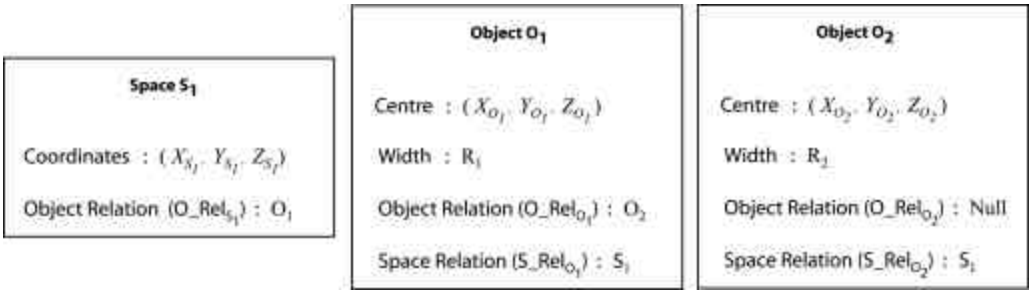

Figure 3.3 Space and Object Entities.

Consider an example where objects O_1 and O_2 share the same space S_1. O_2 is stacked on top of O_1, and O_1 is the bottom-most object. All the objects sharing S_1 can be determined by constructing a list of object references originating from the space. Both objects make reference to the same space entity S_1. The state of the space and object entities is illustrated in Figure 3.3, 3.4, 3.5, and 3.6.

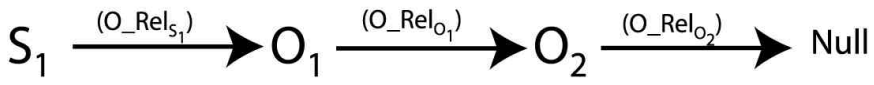

Figure 3.4 Entity object relationships.

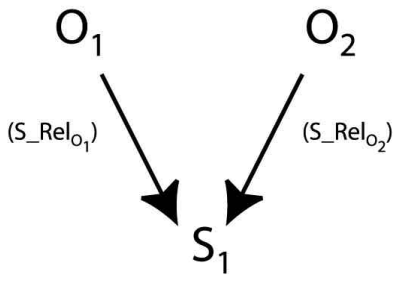

Figure 3.5 Object space relationships.

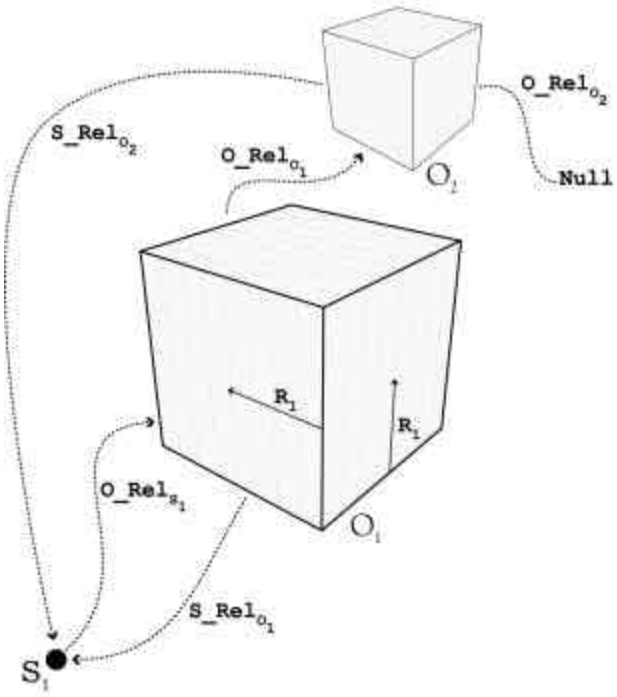

Figure 3.6 Relational diagram of entities.

The table is modeled as a two-dimensional plane in world space. Certain body parts are not permitted to penetrate the table. In this respect, the table imposes constraints on the state of the puppet discussed in Section 5.6. The table is defined by its height, front, and skew. The height value is expressed along the y-axis in world coordinates. The front of the table is expressed as a distance along the positive z-axis in world coordinates. The skew of the table indicates how far the table is shifted along the positive or negative x-axis. Figure 3.7 shows the position of the table relative to the puppet.

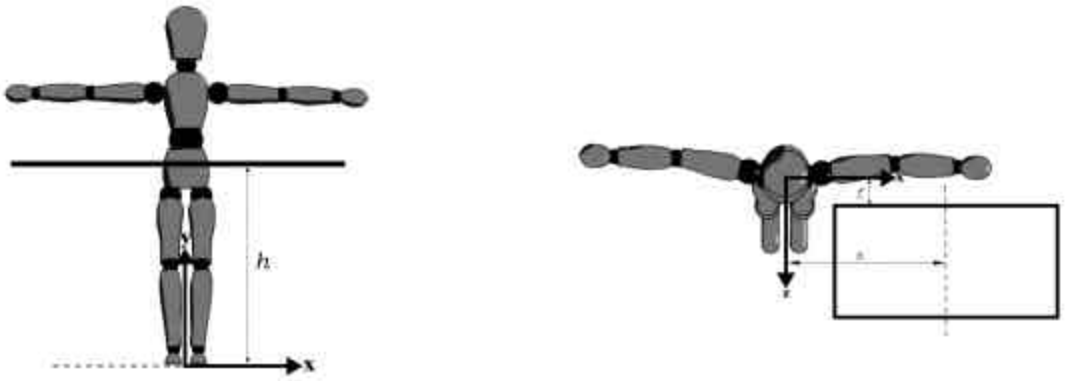

Figure 3.7 Table model dimensions (Height = h, Front = f, Skew = s).

3.5 Virtual Puppet

The puppet interacts with the environment by reaching, grasping, and moving object and space entities. The motion resulting from user input modifies the state of the puppet.

The puppet hierarchy has twenty-seven degrees of freedom. The figure is composed of nine body segments, each having a corresponding proximal ball joint which serves to attach it to it's respective parent link. The abdomen, chest, head, left shoulder, left elbow, left wrist, right shoulder, right elbow, and right wrist are the body segments considered in our model. The lower extremities are not part of the animatable portion of the puppet. Each ball joint can rotate in the local x-axis, y-axis, and z-axis, although not necessarily in that order. Range of motion limits, current orientation, and joint length model each ball joint. Joint limits are modeled from human biomechanical data as the minimum and maximum rotations with respect to a particular axis. Current orientation refers to the transformation matrix of the joint's local coordinate frame. Joint length is calculated as the length from the joint's centre of rotation to the adjacent joint's centre lower in the hierarchy. Figures 3.8, 3.9 illustrate our puppet model. Figure 3.10 and Table 3.1 present specific model dimensions. Certain degrees of freedom in the elbow are eliminated by setting the upper and lower range of motion limits to zero. Specifically, the elbow's local z-axis is restricted from rotating. Both the hand and elbow have a twist component, which are redundant degrees of freedom. However, these redundancies help the IK solver converge on a solution in step 6 of the algorithm in Figure

5.17. Redundant degrees of freedom are required since both orientation and positional goals must be accommodated while maximizing the perceived naturalness of the puppet's posture.

Our puppet model is a simplification of a full anatomical representation of the human body. The lower extremities remain in a fixed seated position. The puppet has no face, avoiding the difficult task of animating facial expression. The joint centres of rotation are positioned at the tip of the adjacent joint's geometry. This avoids mesh deformation issues when rotating joints. The puppet has simple hand geometry, avoiding the complicated task of finger positioning in grasping motions.

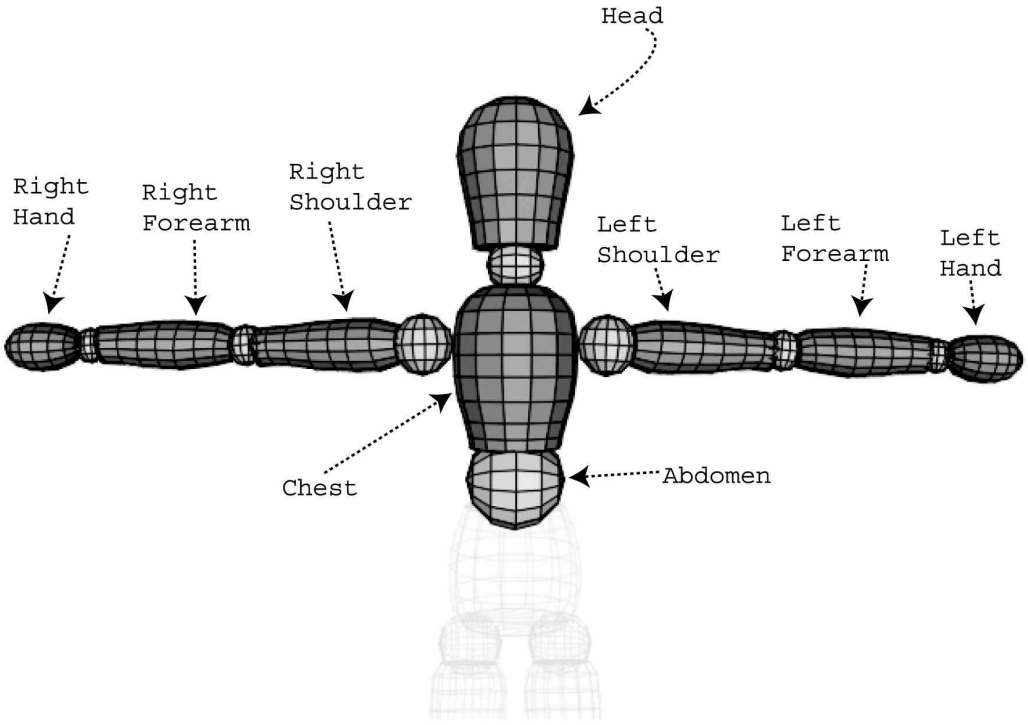

Figure 3.8 Puppet model body segments

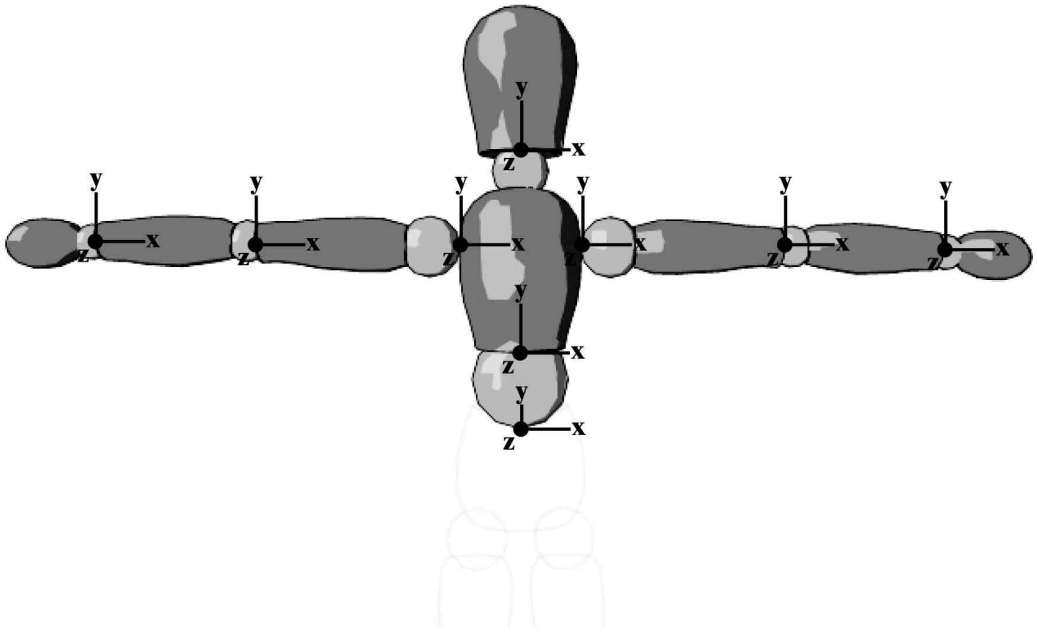

Figure 3.9 Puppet model local coordinate frames.

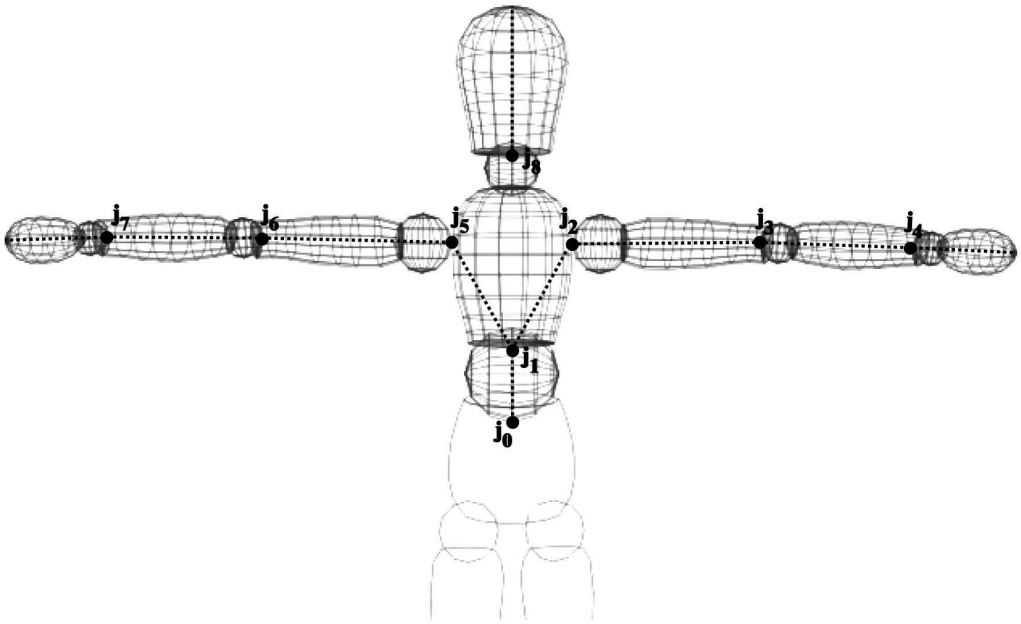

Figure 3.10 Joint centres of rotation

Every joint's axis of rotation maintains its own respective state information. To animate the puppet from its initial state to the final goal state, we store data to compute the joint's angle of rotation over time. The start angle and goal angle are updated at the beginning of the motion, and the joint's current angle is updated at every time step. The ball joint's state information is illustrated in Figure 3.11. Some reference to the desired interpolation function of the joint must be maintained, along with the speed at which the joint moves from its initial state to the goal state. The evolution of a joint degree of freedom from its start angle to the goal angle is presented in Figure 4.18.

Joint	Adjacent Body Segment		Order of Rotations	Min Limit	Max Limit	Joint Centre (Zero Posture)
j_0	Abdomen	1st	X	-7	45	(0, 0.609, 0)
		2nd	Z	-10	10	
		3rd	Y	-30	30	
j_1	Chest	1st	X	-7	45	(0, 0.690, 0)
		2nd	Z	-10	10	
		3rd	Y	-30	30	
j_2	Left Shoulder	1st	Z	-80	90	(0.063, 0.794, 0)
		2nd	Y	-130	30	
		3rd	X	-90	90	
j_3	Left Forearm	1st	Y	-180	0	(0.305, 0.791, -0.033)
		2nd	Z	0	0	
		3rd	X	-90	90	
j_4	Left Hand	1st	Z	-90	90	(0.412, 0.789, -0.033)
		2nd	Y	-90	90	
		3rd	X	-180	120	
j_5	Right Shoulder	1st	Z	-90	80	(-0.063, 0.794, 0)
		2nd	Y	-30	130	
		3rd	X	-90	90	
j_6	Right Forearm	1st	Y	0	180	(-0.305, 0.791, -0.033)
		2nd	Z	0	0	
		3rd	X	-90	90	
j_7	Right Hand	1st	Z	-90	90	(-0.412, 0.789, -0.033)
		2nd	Y	-90	90	
		3rd	X	-120	180	
j_8	Head	1st	X	-80	90	(0, 0.888, 0)
		2nd	Z	-130	30	
		3rd	Y	-90	90	

Table 3.1 Joint specifications.

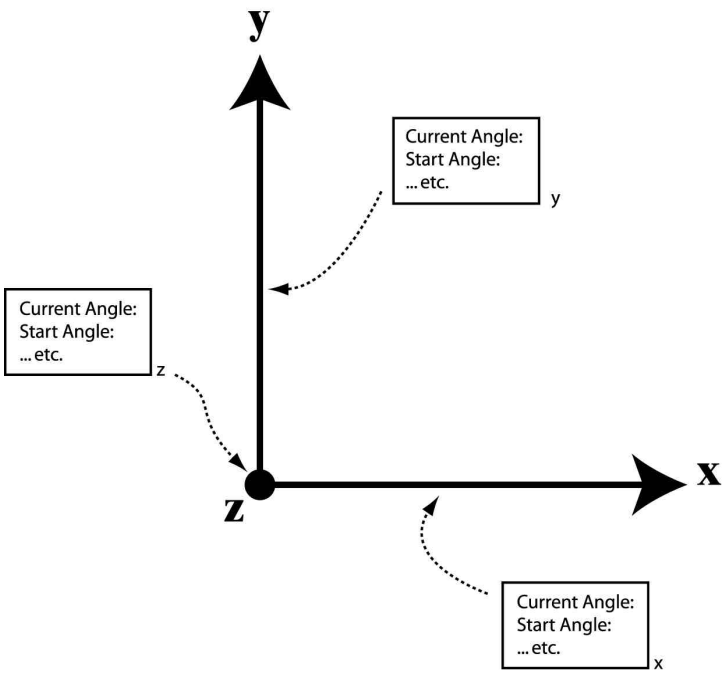

Figure 3.11 Ball joint axis state information.

3.6 Motion Queue

Motion is modeled in our system in terms of tasks. The user specifies the task to be executed along with its associated motion primitives, and the system will animate the puppet accordingly.

The interface allows the user to specify tasks and modify primitives by keyboard or script. Details of system interaction are presented in Section 7.1 and 7.2. When the user inputs a task, the system processes the input and begins animating the task as soon as possible. However, while the system is computing and animating the puppet, the user may have input more tasks into the system pipeline. This is particularly true when interacting with the system by script. The script can specify many tasks in a fraction of the time it takes to complete one motion. To deal with such cases, we store the input in a motion queue. The tasks will then be processed in first-come, first-serve order.

Maintaining a motion queue also allows the puppet to perform multiple tasks at the same time. We refer to this concept as *motion concurrency* or *motion blending*. Concurrent execution of tasks is a novel feature of our system, and its operation is explained in detail in

Chapter 4. If more than one task is in the motion queue, then the motion scheduler will consider how the multiple tasks can be executed concurrently. The user can control the concurrency of task execution by modifying a parameter called the *scheduling parameter*. The user can specify certain tasks to be executed independently, while others are blended together in a single motion.

In addition to adding tasks to the motion queue, the user can modify its contents. The user can freeze the motion of the puppet by deleted all motion models from the queue. When the queue is empty the puppet will stop animating immediately. The user can also interrupt the current motion by replacing the entire motion queue with a single new task. The puppet will stop executing the current motion and begin the new task. This feature is further discussed in Section 4.2, and its specification is presented in Table 7.1.

3.6.1 Motion Building Blocks

The motions in our system are categorized by reaching motions that interact with the environment, general motions that are performed independent of the state of the environment, and sliding motions that positions the puppet relative to its current position.

Reaching motions include tasks that place the hand at an object or space entity. The grip, speed, and interpolation function are user-specified primitives that modify the nature of the motion. The grip parameter determines the orientation and position of the hand relative to the entity to be reached, as explained in Section 6.1.1. The speed of the motion determines over how many frames the motion will interpolate. The interpolation function associated with a motion determines the velocity over time. Applying speed and interpolation functions to the puppet's motion is further discussed in Section 4.5.

Sliding motions will position the hand relative to its current position. The motion is defined by a translation vector that will the position of the hand relative to its current position in world coordinates, and a matrix that defines the final orientation of the hand. Further discussion of the translation vector and orientation matrix can be found in Section 6.1.2. The user specifies the translation vector, orientation matrix, speed, and interpolation function associated with the motion. The translation vector and orientation matrix are not considered among the set of user specified motion primitives since they define the semantic

interpretation of the sliding tasks. Specifying the translation vector and orientation matrix is discussed in Sections 7.1 and 7.2.

General tasks are independent of the environment, and can not intuitively be expressed as an inverse kinematics problem. For example, waving or gesticulating postures are not appropriate for inverse kinematics solvers. Instead of computing postures, these motions' postures are hard-coded in our system. The speed and interpolation functions are user-specified motion primitives applicable to general tasks. Further discussion on general tasks can be found in Section 6.1.3.

The three motion classifications above provide the building blocks for animating a rich variety of tasks. Sequencing the above motion building blocks can generate many object manipulation motions. For example, a drinking task can be accomplished by first reaching for the cup, then bringing the cup to the mouth and tilting the head. Grasping the cup is a reaching motion, while drinking from the cup can be a general task hard-coded in the system. Two reaching motions and two sliding motions can open a jar. A sequence of motions to accomplish this task is presented below.

1. Grasp the jar. (Reaching motion)
2. Move the jar close to the body. (Sliding or Reaching motion)
3. Place the opposite hand on top of the jar. (Reaching motion)
4. Adjust the orientation of the hand to twist the jar lid. (Sliding motion)

The second motion can be defined as either a sliding motion or a reaching motion. If the jar is moved to point closer to the body by some relative measurement, then it is a sliding motion. If the jar is positioned at an absolute point in space closer to the body, then it is a reaching motion. Either type of motion can move the jar closer to the body.

There are wide variety of tasks that can be accomplished with an appropriate sequence of reaching, sliding, and general movements. Playing musical instruments, handling cutlery and tools, and other object manipulation tasks are conducive to this classification of motion. Walking, gesticulating, and other motions that do not manipulate objects can be integrated in our system as general tasks, although much of the work has been focused on environment dependant motion.

3.6.2 Cyclic Motion

Motions in our system are further categorized as *cyclic* or *non-cyclic*. Traditionally, the term "cyclic" refers to tasks such as scratching one's head that requires several movements to complete, and these movements repeat themselves. Looking at one's watch is non-cyclic, and is completed once the wrist and head are in a position where the time can be observed. Consider drinking water, which requires one to reach for a glass, then bring the glass to the mouth and tilt the head. This task requires multiple postures to complete, although it is not cyclic in the traditional sense. The term "cyclic" can be applied to walking or scratching your head, but does not apply to higher level tasks that require several distinct motions to complete. In either case, they are treated the same in our system. Cyclic motion in the context of our classification refers more generally to tasks that involve multiple movements, whether or not the movements actually cycle.

The three basic tasks described in Section 3.6.1 can be grouped together to form higher level tasks. Scratching the head can be implemented by cycling between two general tasks. The first task is a downward stroke on the head, and the second task is an upward stroke. By repeatedly executing this pair of tasks, we can effectively generate a scratching motion. Grouping together basic motions as the examples described in Section 3.6.1 can generate tasks such as opening a jar or drinking from a cup.

Higher level tasks are implemented in our system as well as the motion building blocks described in Section 3.6.1. If the user commands a higher level task, then a group of primitive motions is placed in the motion queue as a cyclic task. The group of primitive motions forms a list of postures that define the cyclic motion. All tasks in the cyclic motion model with the exception of the last frame have a pointer to the successor in the cyclic motion. A frame's successor is referred to as the *couple motion*. Non-cyclic motions are modeled as a single frame with no couple motions. This distinction is illustrated in Figure 3.12. Two examples of cyclic motion implemented in our system are drinking from a cup and turning on a table lamp. Specifying these motions is presented in Table 7.1.

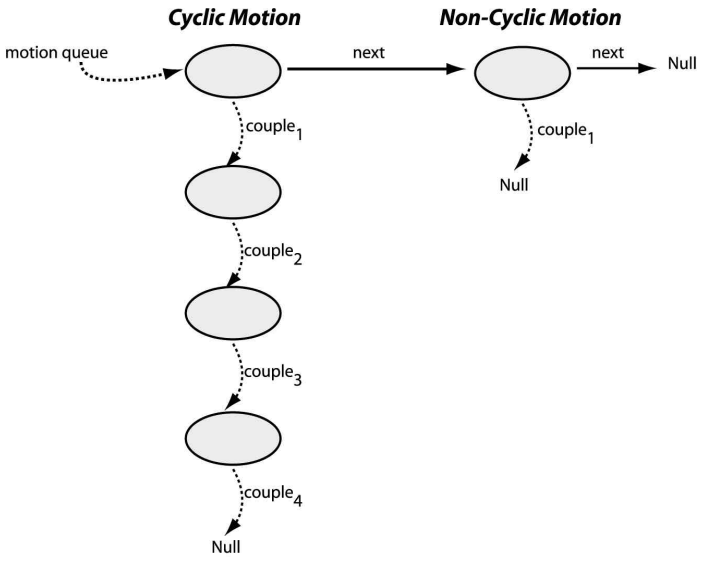

Figure 3.12 Motion Queue with one cyclic motion frame and one non-cyclic motion frame.

3.6.3 Critical body segments

All motions are classified according to which body segments are critical to the execution of the motion. We created six groupings of body segments according to the critical body segments found in many motions. The six groupings are:

- Left hand, left forearm, left shoulder.
- Left hand, left forearm, left shoulder, head.
- Left hand, left forearm, left shoulder, chest, abdomen.
- Right hand, right forearm, right shoulder.
- Right hand, right forearm, right shoulder, head.
- Right hand, right forearm, right shoulder, chest, abdomen.

The motion associated with a task is not exclusive to the critical body segments. For example, when one looks at his watch, he uses his left arm to position the watch and simultaneously positions his head to see the watch on his wrist. However, he must be doing something with his right hand as well, which will be referred to as a *secondary motion*. He may have the hand in his pocket, resting on a table, placed on his hip, or resting on his side.

Secondary motions refer to the movement of body segments that are not critical to accomplishing the task at hand, and *primary motion* refers to the movement of the critical body segments. *Secondary body segments* refer to all body segments that are not critical body segments.

How one handles non-critical body segments is an issue to be resolved. We can fix the right hand in a pre-defined position while he is looking at his watch. For example, we can place the hand on the table. Alternatively, we can implement a variety of secondary motions that may be performed concurrently with looking at the watch. In this case, the user explicitly specifies the motion of the right hand to accompany the primary motion of looking at his watch. Our system incorporates both solutions. There are default secondary motions implemented that can be overridden by a user-specified motion.

The system default secondary motions can be overridden in several ways. The user can specify a new default motion by *locking* the position of a hand in its current position. This concept is discussed in Section 6.2. The user can also blend multiple tasks together. The idea is that the secondary body segments of one task may be the critical segments of another. By overriding secondary motion with the critical body segments' goals from another task, we can concurrently execute two tasks. Chapter 4 describes this concept in detail.

3.7 Summary

This chapter introduced the integration of the models and modules used to implement a system with guided control of a puppet seated at a table. An overview of the system's dataflow was introduced in Section 3.1. Each module interacts with the system's models to effectively control the motion and state of the system. The functional operation of each module was presented in Section 3.2. The design of each model was discussed in subsequent sections. Sections 3.3, 3.4, 3.5, and 3.6 described the user input, the environment, the puppet, and the motion models respectively.

Chapter 4

Motion Scheduling

The motion scheduler is responsible for arbitrating among the tasks in the motion queue. Some commands require inverse kinematics to compute the appropriate goal posture for accomplishing the task's objectives, while others have appropriate joint angles predefined in the system. The motion scheduler will consider the body segments instrumental in accomplishing the task and the target hand position determined by the interface to formulate an inverse kinematics problem. The problem is input to the posture generator to compute the joint angles required to accomplish the task. Once a complete forward kinematic specification of the puppet's position is computed, the puppet is animated from its initial position to the goal position. The speed and function to interpolate the puppet from its initial position to the goal position is included in the task specification.

This chapter describes the issues we encountered in scheduling motion independently and concurrently from a sequential set of tasks. Section 4.1 introduces some of the issues in implementing motion concurrency. Section 4.2 explains how the user interactively controls the concurrent execution of multiple tasks. Section 4.3 describes our algorithm for multi-tasking the virtual puppet. Section 4.4 discusses how tasks are completed and removed from the motion queue. Finally, Section 4.5 describes how the puppet animates from its initial position to the goal position. Figure 4.1 shows the functional operation of the motion queue.

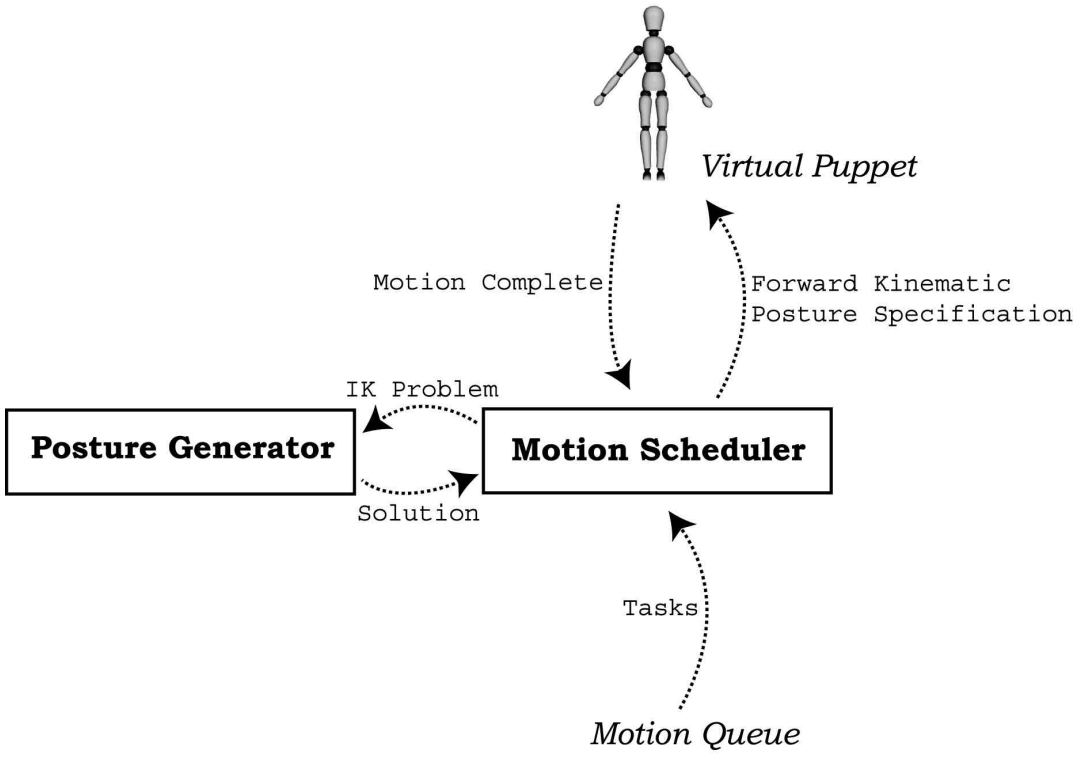

Figure 4.1 Functional diagram of the motion scheduler.

4.1 Introducing Motion Concurrency

The motion scheduler's function and operation is similar to an operating system managing processes and resources. The puppet's body segments act as resources that are assigned to particular motion models. The motion models are analogous to processes by demanding body segments as resources and having specific execution times. To complete its task, each motion model requires a certain set of body segments to execute for a specified duration of time. This set of body segments is referred to as the *critical body segments* described in Section 3.6.3. Several subtle issues arise when trying to manage an arbitrary set of motions with variable resource demands and priority.

We refer to *conflicting motion* as two motion models that specify two distinct goals for the same body segment. The problem of effectively scheduling motions that may or may not be conflicting with variable execution times can be approached in a number of ways. One possible method is to execute each motion exclusively and in succession. This method does not allow the user to combine two independent motions, such as scratching one's head and

walking. On the other extreme, we can try to multi-task the puppet at all times, and schedule as many motions to execute concurrently as possible. This does not allow the animator to separate the execution of walking and scratching one's head into two distinct motions if he wishes. Furthermore, body segments need not be mutually exclusive to one task. To concurrently execute multiple tasks, the puppet's body segments can contribute to satisfying one task's constraints, or may be shared among multiple tasks. One novel feature of our system is the user specification of how potentially concurrent motions are handled.

Motion associated with a task is comprised of two movements referred to as *primary motion* and *secondary motion* introduced in Section 3.6.3. Primary motion refers to the movement of the task's critical segments from their initial position to the goal position. To accomplish a task, the critical segments must achieve the goal position. This is analogous to stating that a task is complete once the primary motion has finished executing. Secondary motion refers to the motion of the non-critical body segments and is not crucial to accomplishing a task. The system has default secondary motion implemented that can be overridden as discussed in Section 6.2. The example below illustrates the concept of motion concurrency.

Task 1	"Reach for object *A* with left hand"
Critical Body Segments	Abdomen, Chest, Left arm
Secondary Body Segments	Head, Right arm

Task 2	"Scratch head with right hand"
Critical Body Segments	Right arm
Secondary Body Segments	Head, Abdomen, Chest, Left arm

Table 4.1 Two non-conflicting tasks.

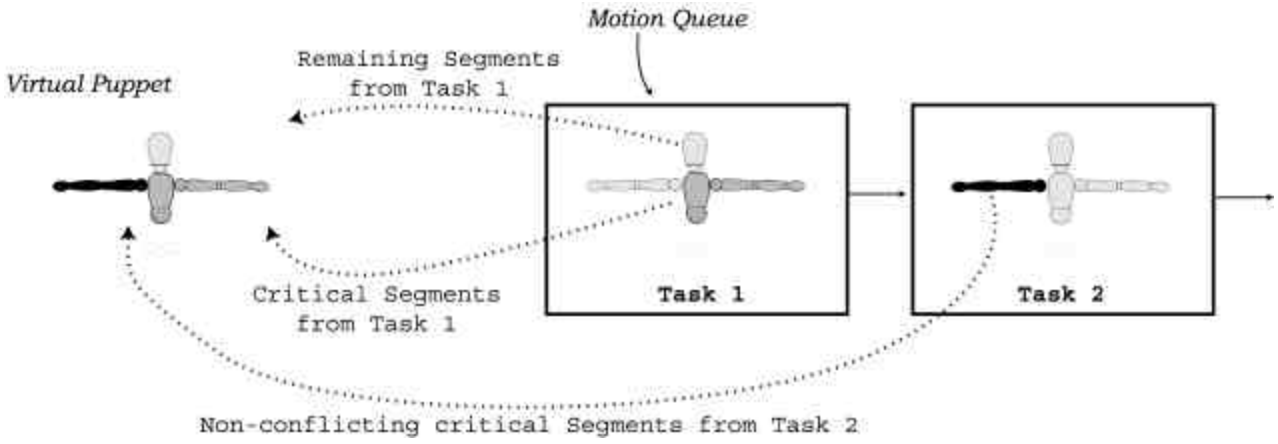

Figure 4.2 Example of concurrent execution of tasks.

Overlapping Task 1 and Task 2 results in the body and left arm reaching for object *A*, and the right arm scratching the head. The head is assigned default secondary motion. Motion concurrency is possible when the critical body segments in one task correspond to the secondary body segments in another. Our system only considers blending at most two motions at the same time. This simplification is justified since the puppet has only two hands. The head could potentially execute a third task, but presumably should be positioned to observe the motion of the hands.

4.2 Motion Scheduler Operation

The user controls the concurrent execution of tasks by setting the value of the scheduling parameter introduced in Section 3.3 and 3.6. The scheduling parameter for every task is a user- specified parameter. The value of the scheduling parameter is set to *no overlap*, *partial overlap*, and *full overlap*. Specifying the scheduling parameter is presented in Table 7.1. A more precise description of the motion scheduler's operation is presented in Appendix A.

We use the term *scheduling motion* in the functional description of the motion scheduler presented below. Scheduling motion refers to the assignment of positional goals from a motion model to a subset of body segments in the virtual puppet's body. When positional goals are scheduled from a particular motion model, we need a forward kinematic specification of the puppet's goal state. If the motion model is a general task, then the joint angles of the puppet's critical segments are predefined. For reaching and sliding motions, the

puppet's posture is specified as an inverse kinematics problem as in equation (3.2). The kinematic chain $\{j_i, j_{i+1}, \ldots\}$ corresponds to the group of body segments scheduled from the particular motion model. The end effector position and orientation corresponds to the geometric constraints of the task determined Section 6.1. When secondary motion is scheduled, the inverse kinematics problem is specified the same way, except the end effector position and orientation is determined from Section 6.2. The first and second task refers to the motion models at the head of the queue and immediately following the head, respectively.

The motion scheduler determines if motion blending can occur by scanning the first two motion models in the motion queue. If there is only one task in the motion queue, then no motion overlapping occurs. If there is more than one task in the queue, then the value of the second motion model's scheduling parameter is referenced to determine if any concurrent execution of tasks should occur. Pseudo-code for this operation is presented in Figure 4.3.

```
If( Motion Queue is empty )
      -Stop animating puppet
Else if ( Only one model in the motion queue )
      -Schedule motion for all body segments from the first motion model
Else if ( Second motion model's scheduling parameter == No overlap)
      -Schedule motion for all body segments from the first motion model
Else
      -Blend the motion of the first two tasks in the motion queue if
possible.
      The algorithm to blend motions is presented in Figure 4.6.
```

Figure 4.3 Pseudocode of motion scheduler operation

The above algorithm is executed every time a motion model is removed from the queue, or a motion model is added while there is less than two models currently in the motion queue. Freezing the puppet's motion is implemented by deleting all models from the motion queue. The algorithm above will be invoked, and the motion of the puppet will stop. Interrupting motions is implemented by deleting all models from the motion queue and placing a single task in the motion queue. The algorithm above will interrupt the puppet's current motion with the new task. Alternatively, a more sophisticated interrupt scheme could place a task at

the head of the queue and re-invoke the above algorithm. How one chooses to implement an interrupt scheme depends on the semantic interpretation of the command. While the system is in interrupt mode, all new input tasks will modify the motion queue as defined above. Input corresponding to freezing the motion and shifting the system state to interrupt mode is given in Table 7.1.

Determining whether two tasks can be blended together, one must consider if there are any conflicts in critical segments. However, disallowing two tasks to execute concurrently due to conflicts in critical segments is an over-simplification of the problem. From observation of human movement, one can notice humans performing multiple tasks in spite of conflicting body segments. The issue to be resolved when combining conflicting motions is determining which body segments can be shared between two tasks, and the motion resulting from blending the tasks. An example of such movements that applies directly to our work is reaching tasks involving both hands.

When reaching for an object, humans will use the abdomen, chest, and arm to place the hand at the desired position. Three subjects were observed reaching for a cup with the left hand, right hand, and two cups simultaneously. Photos of the people performing these tasks is presented in Figure 4.4. The bottom row of photos shows how two independent reaching tasks can be blended into a single motion. When reaching for both cups, the torso is in conflict between the two tasks but successfully performs the motion by compensating between the left and right goal. However, two independent left handed reaching tasks cannot be combined into a single motion. These two tasks cannot be blended because the left hand cannot be placed in two positions at once. Extending this concept to the scenario below, none of the subjects can grasp both cups with the left hand at the same time. Although the head is not an end effector in the traditional sense, it can be thought of as positioning the human's gaze, which must be focused on a single subject. In this context, the head cannot be shared between two tasks. This implies that while some conflicts are unresolvable, others can be overcome by sharing the conflicting body segments. Figure 4.4 is also of interest because it illustrates how people perform tasks in remarkably similar ways. This supports the notion of a "preferred posture", which the posture generator in Chapter 5 attempts to model.

Figure 4.4 Three people performing simple reaching tasks.

When animating two tasks being combined, one must also consider the motion rather than strictly the final posture. The final posture of two combined tasks must accomplish the semantic interpretation of both tasks. For example, blending two reaching motions must place each respective hand at its goal. However, the final posture can be assumed by moving both arms simultaneously towards their goal, or by sequentially moving each hand into position. Consider the photo sequence of two blended reaching motions presented in Figure 4.5. The first subject places the left hand at its goal first, then moves the right hand into position. The second subject simultaneously moves the hands into their respective goal positions. The first subject is executing the two tasks independently with some anticipation of the second task. From observation of Figures 4.4 and 4.5 we can make several generalizations with respect to how humans combine motion. These generalizations are the basis on which our algorithm for combining motion is designed.

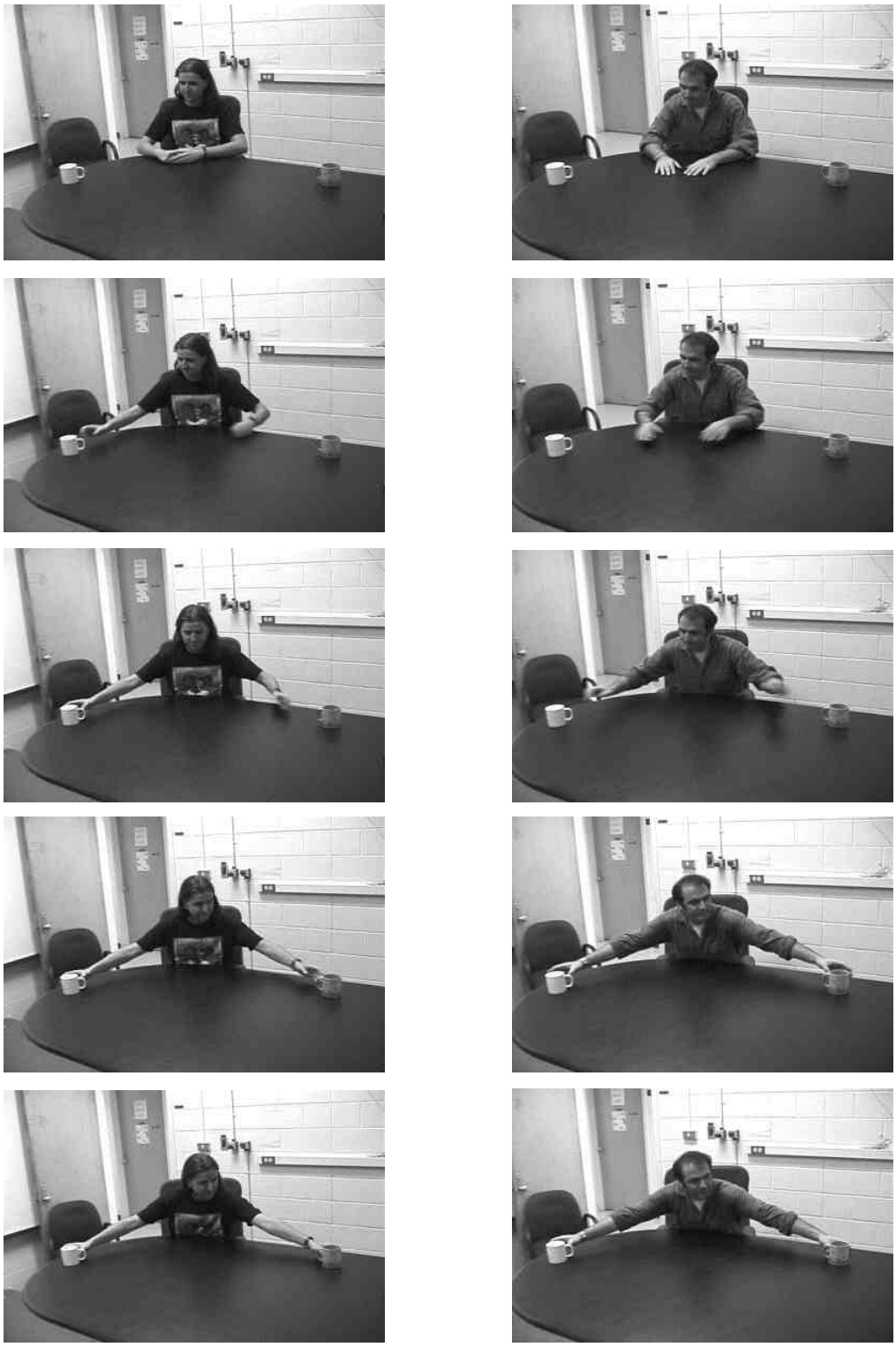

Figure 4.5 Partial and full overlap of two reaching tasks.

The top two rows of photos in Figure 4.4 illustrate reaching tasks being executed with no overlap. Figure 4.5 are examples of how two distinct reaching motions can be combined. The left column of photos is a human example of the partial overlap scheme implemented in our system. The right column of photos corresponds to the full overlap scheme. Notice that the final postures in the bottom row of Figure 4.5 corresponds to the photos presented in the bottom row of Figure 4.4.

As previously mentioned, end effectors cannot be shared between two independent tasks. In our system, the left and right hand are the only end effectors that can be positioned, as introduced in Section 3.6.1. This implies that the arms are dedicated to at most one task at any time. Humans can also look at only one object at a time. The photos of the combined reaching motions in Figure 4.4 illustrate this concept. All three subjects are looking at one object, rather than trying to somehow observe both objects, or interpolating the position of the head to look in between the objects. These two conclusions eliminate all possibility of blending tasks with conflicting segments with the exception of motions that share the abdomen and chest as critical segments.

The execution of multiple tasks sequentially or simultaneously is more complex than simply deciding which tasks to execute at the same time and which to perform in sequence. Task anticipation is important for creating lifelike, realistic motion and is introduced as a fundamental principle of animation by Lasseter [Las87]. From a physiological and psychological standpoint, anticipation can be described qualitatively as preparing the body for a particular task. Three examples of anticipatory motion are presented with a brief example.

First, positioning the head to look at an object before reaching for it can be described as the result of positioning the body to acquire sensory data. Second, consider moving the hand slightly towards a target with one hand while the other hand is executing a task, then finishing the movement once the head is moved to observe the target. The left column in Figure 4.5 shows an example of this. The second grasping motion is delaying to acquire sensory data on the cup's position. However, the hand begins to move in the cup's general direction in anticipation of the second grasping movement. The third example of motion resulting from anticipation of future tasks is extending muscles before contracting. Or, alternatively, to establish the necessary momentum for accomplishing the motion. For

example, moving a foot backward before kicking a soccer ball prepares the body to effectively perform the task [Las87]. Our system attempts to model anticipatory motion as a form of motion concurrency. While only some tasks can be fused into a single motion, many tasks can influence the current movement in the form of anticipatory motion, as illustrated in Figure 4.5.

In our system, motion blending is implemented by not only combining tasks that have no conflicting critical segments, but also by sharing certain critical segments among two tasks. The user can also schedule a subset of a task's critical segments to begin moving as a form of anticipatory motion. Handling multiple tasks simultaneously has been explored in robotics literature for highly redundant manipulators [SS91][BB98]. Tasks have also been combined by computing a weighted sum of the task's goal postures [BPW93][Per97].

4.3 Motion Concurrency Algorithm

Our system blends motions depending on the user's input and the specifics of the tasks to be combined. A functional description of the algorithm is presented below followed by a high-level description of how the various blending schemes compare to our observations in Section 4.2. An example of partial and full overlap of a reaching task is presented. The first and second motion models refer to the model at the head of the motion queue and the model immediately following the head, respectively.

```
If ( Second motion model's scheduling parameter == Partial overlap )
      -Schedule all of the first motion model's critical segments
      -Schedule the second motion model's non-conflicting critical
segments
      -Schedule secondary motion from the first model for the rest of the
      body segments
Else If ( Second motion model's scheduling parameter == Full overlap )
      -Schedule all critical segments from first and second motion model
      If there are conflicting critical segments between the first and
      second motion model, then resolve them as follows:
            -Share the conflicting critical segments that can be shared.
            -Schedule the critical segments that cannot be shared from the
            first motion model.
```

Figure 4.6 Pseudocode for blending two tasks' position goals.

Partial overlap mimics anticipation of the proceeding task by starting the motion of the second motion model's unconflicting critical segments. The right hand begins its motion while the first task is executing. Once the left hand has reached its target, the right hand's motion executes to completion. Full overlap implies sharing body segments where applicable. If the conflicting body segments cannot be shared, then the full overlap scheme performs identically to the partial overlap scheme. The implementation of sharing body segments is presented in Section 5.7.5.

Blending tasks to effectively model the user's intentions is further complicated by the variable speed and velocity functions over time of the two tasks. For example, if the puppet's body segments' positional goals are scheduled from both the first and second task, we would like to respect the user- specified motion primitives from both tasks. The general algorithm used to determine the body segments' velocity over time is presented below.

```
If ( No conflicting critical body segments between the first and second
task )
      -Schedule the first task's speed and function primitives for the
      first task's critical segments.
      -Schedule the second task's speed and function primitives for the
      second task's critical segments.
      -Schedule the first task's speed and function primitives for the
      remaining body segments.
Else
      -Schedule the first task's speed and function primitives for all
      body segments
```

Figure 4.7 Pseudocode for scheduling speed and interpolation functions.

The user controls whether the first and second task in the motion queue is executed by the full, partial, or no overlap scheme by modifying the scheduling parameter before inputting the task to the system. Specifying motion parameters is discussed in 3.2. Consider a single left and right handed reaching motion. The task models are presented below, and the three blending schemes' operation is illustrated.

If the second motion model's scheduling parameter specifies no overlapping of the two tasks, then all of the first frame's critical body segments are scheduled positional goals from the first frame. Non-critical body segments are scheduled the first task's default secondary motion. All body segments move according the first frame's speed and interpolation function values. This blending scheme is illustrated in Figure 4.8. In this example the left arm and torso correspond to the first task's critical segments. No overlap implies that the first task in the motion queue is executed independently, and the task's primary and secondary motion will execute to completion.

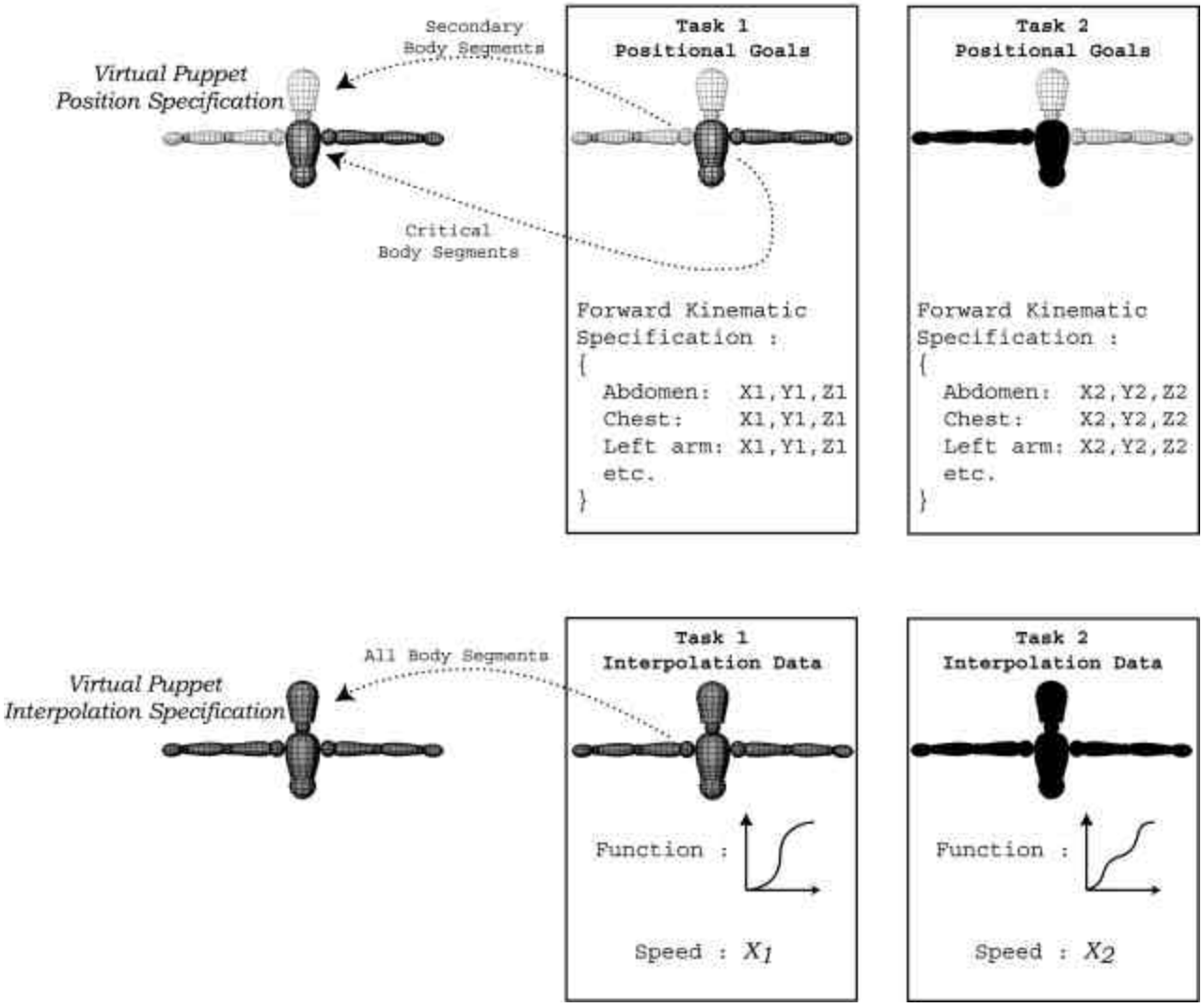

Figure 4.8 No overlap of two tasks.

In the above example, the puppet's left arm and torso is scheduled according to the first task, and the remainder of the segments is scheduled secondary motion from the system. In Figure 4.8 and all subsequent diagrams, one should assume the puppet is facing the reader.

If the second motion model's scheduling parameter specifies partial overlapping of the two tasks, then all of the first frame's critical body segments are scheduled positional goals from the first frame. The second frame's non-conflicting critical body segments are scheduled positional goals from the second task. The remainder of the body segments are

scheduled the first frame's secondary motion. If the two tasks have no conflicting critical body segments, then the critical segments are scheduled speed and interpolation data from their respective tasks, as illustrated in Figure 4.9. Otherwise, all body segments animate according to the first task's speed and interpolation function. This case is presented in Figure 4.12. In Figure 4.9, the left arm and torso is scheduled according to the first task, and the right arm is scheduled from the second task. The head is given the Task 1 secondary motion. All body segments animate according to the first frame's speed and trajectory function.

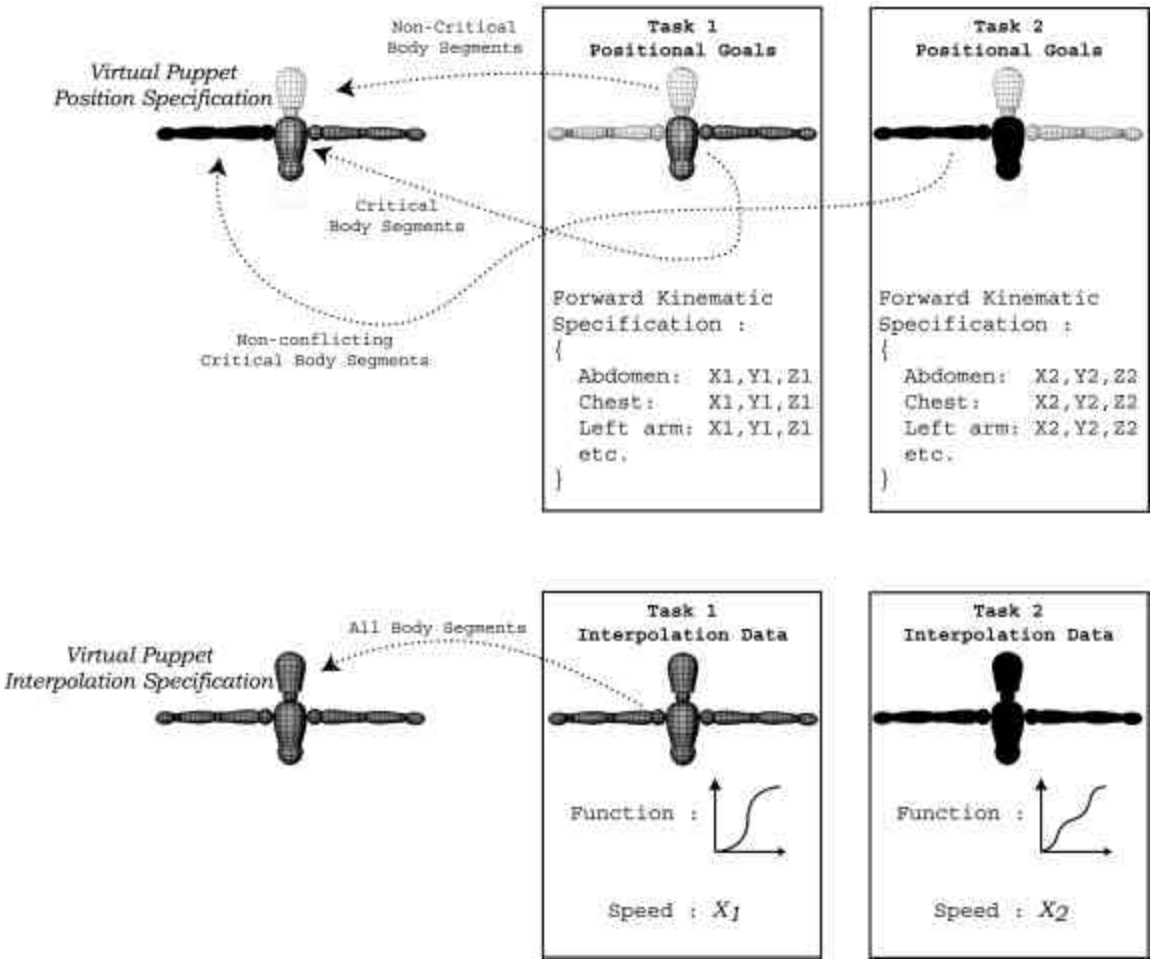

Figure 4.9 Partial overlap of two tasks.

If the second motion model's scheduling parameter specifies a full overlap of the two tasks, then sharing conflicting body segments between the two tasks is possible. If the conflicting body segments cannot be shared, then they are scheduled positional goals from the first task. If a conflict in critical body segments has occurred, then all body segments will animate according to the first task's speed and interpolation parameters. Otherwise, the critical segments animate according to their respective tasks' specifications. Figure 4.10 illustrates an example where Task 1 and Task 2 share conflicting critical segments. Figure 4.12 illustrate an example of two tasks with no conflicting critical segments.

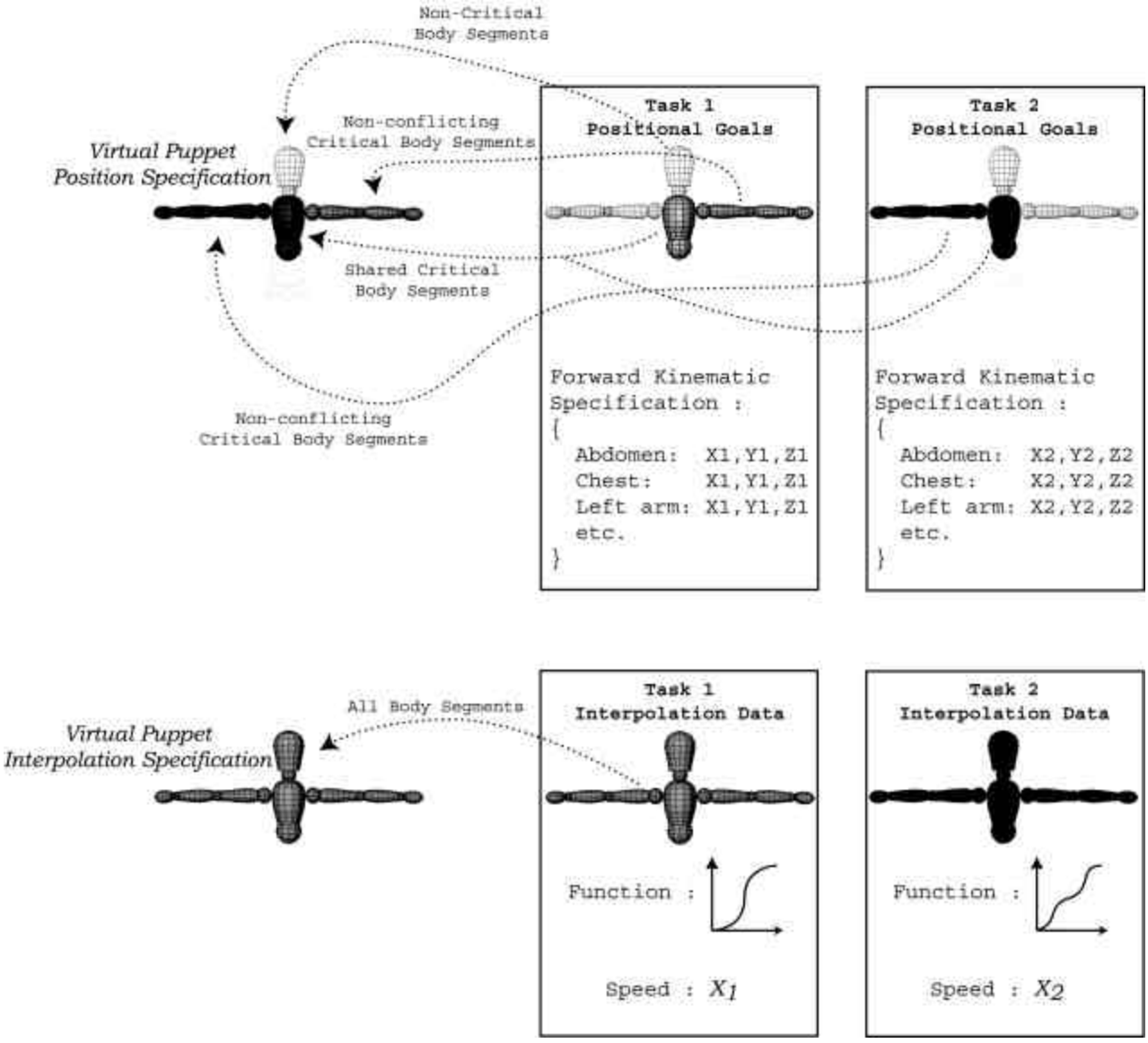

Figure 4.10 Full overlap of two tasks.

4.4 Removing Tasks

Aside from scheduling motion from the motion queue, there is an issue of determining when a motion has ended and a new motion frame should be scheduled. A task is considered complete once all its critical frames have executed to completion. If the tasks are scheduled, executed, and completed in order and succession, then the removal of motion models from the queue becomes trivial. Tasks are removed when finished and a new task begins execution. However, several anomalies can arise if tasks are naively removed from the motion queue when multiple tasks are executing concurrently. These problems are a result of modeling motion as a series of self-contained tasks as described in Section 3.6. This model is conducive to guided control since the user's input is inherently task-level, but is subject to the problems described in this Section when trying to generate motion from several tasks.

Consider the case where two tasks are executing concurrently. The first task is reaching for an object with the abdomen, chest, and left arm. The second task is scratching the head with the right arm. There are no conflicts in critical segments, so each task executes from the model's own respective speed and interpolation function. Assume the reaching task is to execute for 2.0 seconds, and the scratching task executes for 1.8 seconds. Figure 4.11 and Figure 4.12 illustrates this scenario.

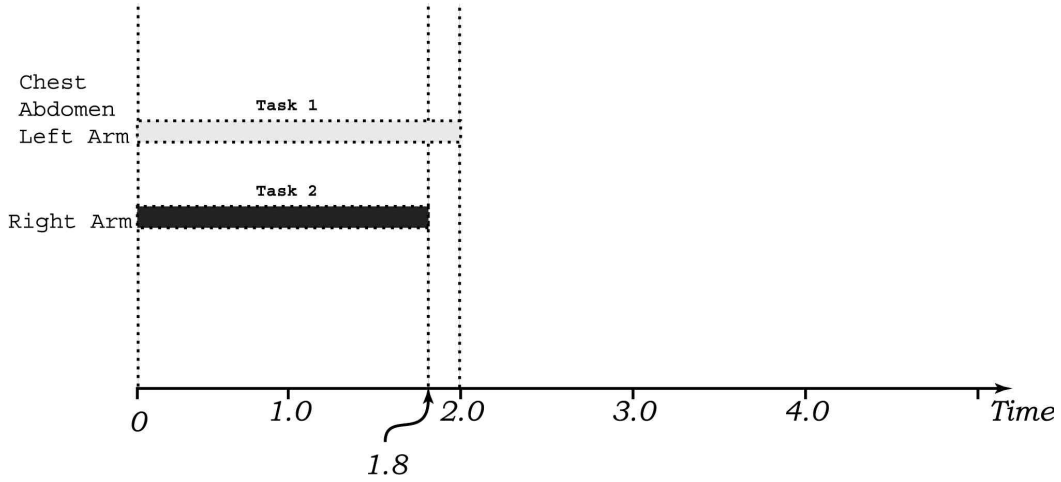

Figure 4.11 Task execution timeline.

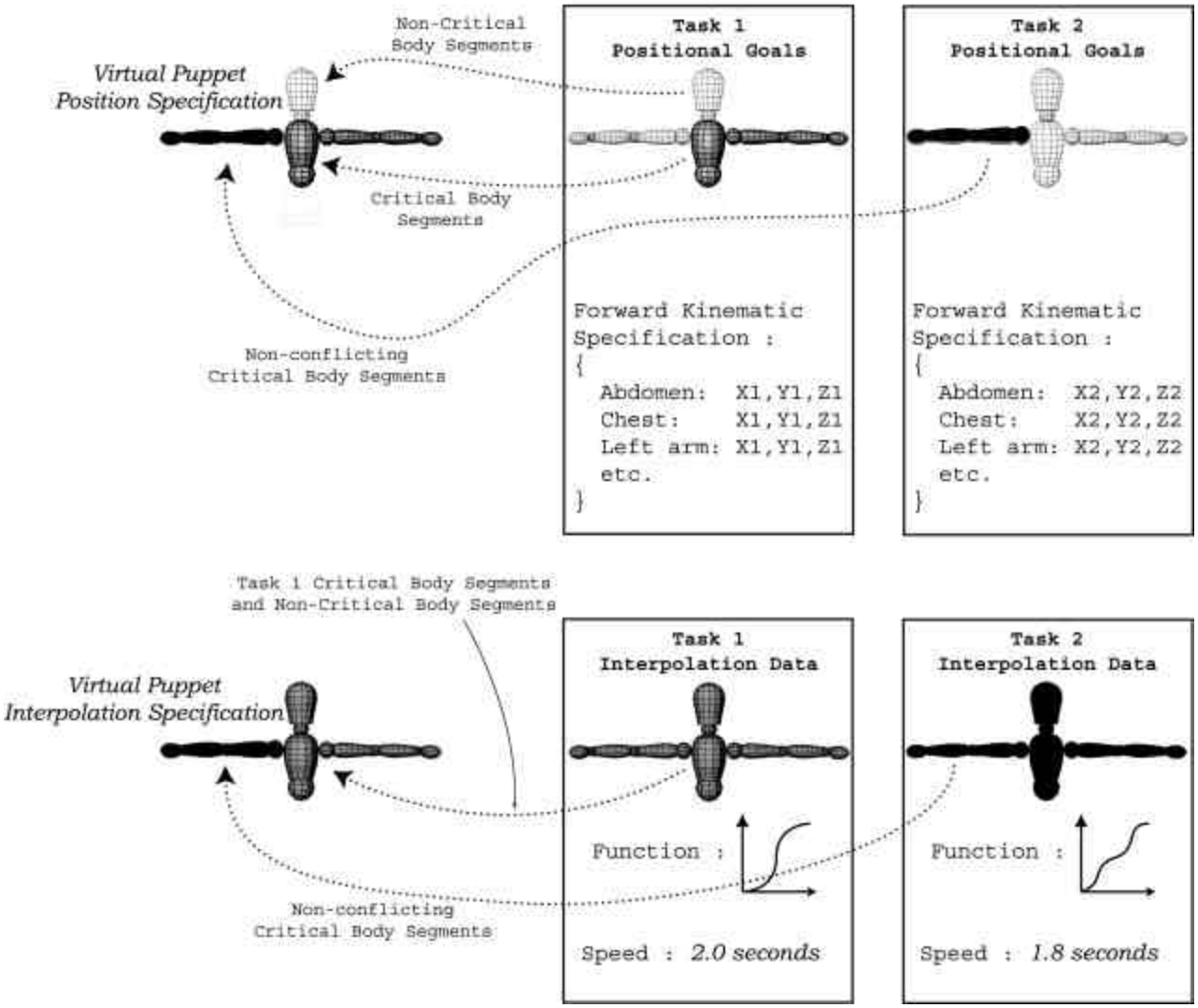

Figure 4.12 Overlap of two tasks with no conflicting critical segments.

The scratching task will end after 1.8 seconds. The motion scheduler will remove the scratching task from the motion queue since all its critical segments have finished executing. The motion scheduler will now scan the motion queue to determine which tasks should execute next. Since the left handed reaching motion model is at the head of the queue, the motion scheduler is guaranteed to schedule all the left-handed tasks' critical segments. This is a result of the algorithm presented in Figures 4.3 and 4.6. However, the left-handed reaching motion has already begun animating. Rescheduling the task from scratch will

animate the reaching motion from its current state to the goal state for another 2.0 seconds. The timeline for this task execution is illustrated in Figure 4.13.

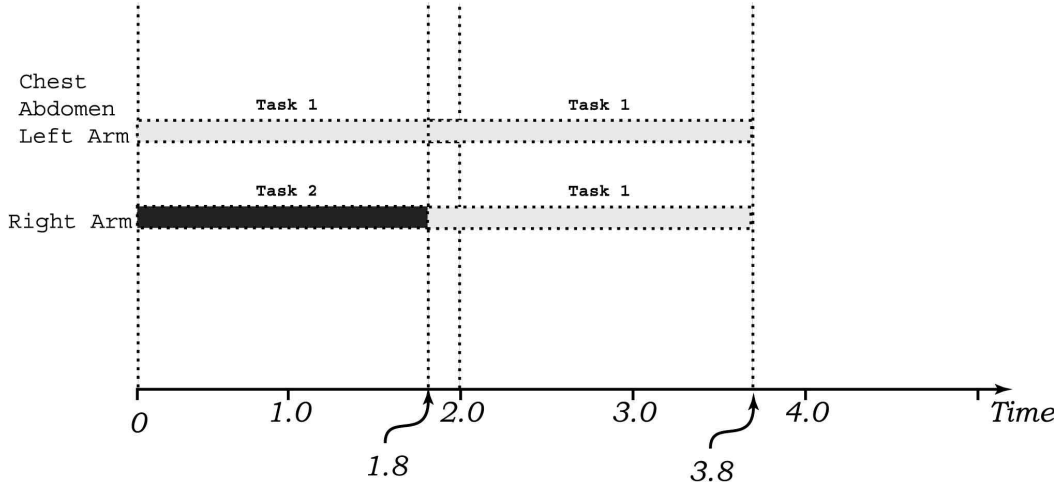

Figure 4.13 Rescheduling Task 1.

This is clearly not what the user had intended. We must respect the user's specification of time with respect to the start and finishing time of tasks. To avoid this anomaly, we apply the following general rule when scheduling motions from the motion queue.

- *If all the critical segments in a motion model have begun animating, do not reschedule the motion model.*

This rule implies that when a set of body segments is scheduled positional data from a motion model, and this set of body segments correspond to the task's set of critical body segments, then the body segments should animate to completion.

While the above anomaly seems intuitive enough to assume without explicit discussion, its implementation leads to another problem which requires overwriting some motion models' positional goals to overcome. Consider the state of the system when the right hand scratching motion has completed in Figure 4.11. It is not clear what the right hand's motion should be during the last 0.2 seconds of the left hand-reaching task. There are two cases to consider.

1) The next task in the motion queue has the right arm among its set of critical segments, and the motion model's scheduling parameter is set to partial or full overlap. In this case, the right arm's positional goals for the task are scheduled, and new motion for the right arm begins.
2) There are no more frames in the queue, the next task does not have the right arm among its set of critical body segments, or the next task has the scheduling parameter set to no overlap. In this case, no useful motion for the right arm can be scheduled.

In the second case, it is not clear what positional goals should be assigned to the right hand. There are three reasonable alternatives to consider.

The first option is to schedule the first task's secondary motion for the right hand. Let us assume the secondary motion places the right hand on the table. The speed and interpolation function with which we animate the right hand is questionable. We can animate the right hand to finish its motion at the same time as the left hand's reaching task. This option considers the user's timing directives for the first task as a hard constraint, and moves the hand from its current position to the top of the table in 0.2 seconds, which will look completely unnatural. However, this option ensures that the time to execute the motion associated with blending the two tasks is bounded above by the longest task's execution time.

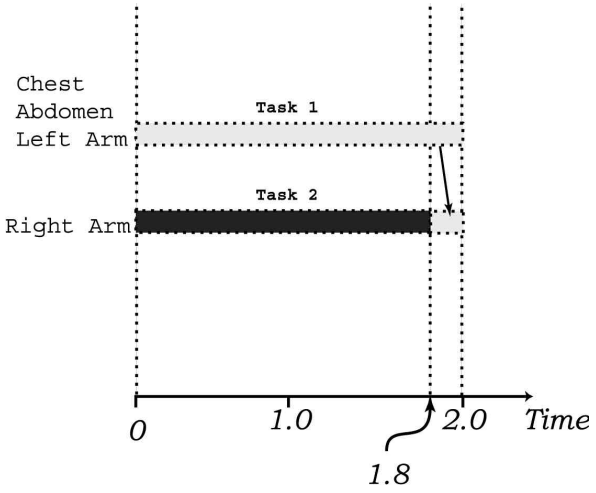

Figure 4.14 Scheduling Task 1 secondary goals for 0.2 seconds.

Alternatively, we can move the right hand to the table at the same speed as the reaching task, or limit the speed of the motion to some reasonable maximum limit. This will look more natural, but the entire animation sequence will extend beyond 2.0 seconds.

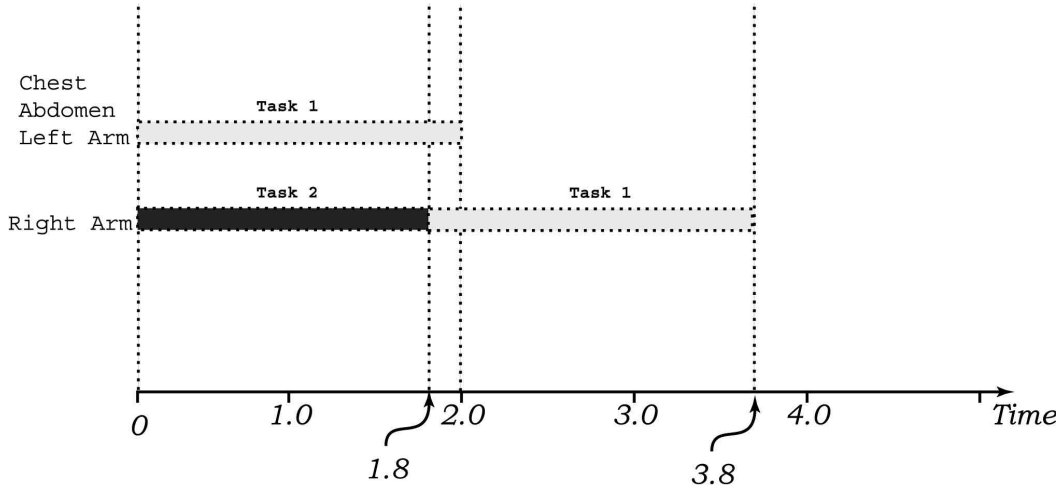

Figure 4.15 Scheduling Task 1 secondary goals for 2.0 seconds.

Our prototype system implements the final option, which is to *migrate goals*. This concept implies storing positional data of the right hand in the left hand's motion model. In other words, upon completion of the scratching motion the position of the right hand overrides the left-hand reaching motion's secondary motion. The result is that no motion of the right hand will occur following the scratching task. This concept is illustrated in Figure 4.16 and 4.17.

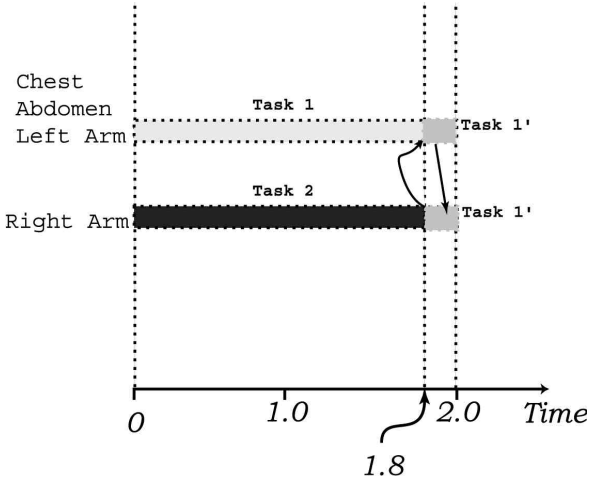

Figure 4.16 Scheduling right arm goals from Task 1'.

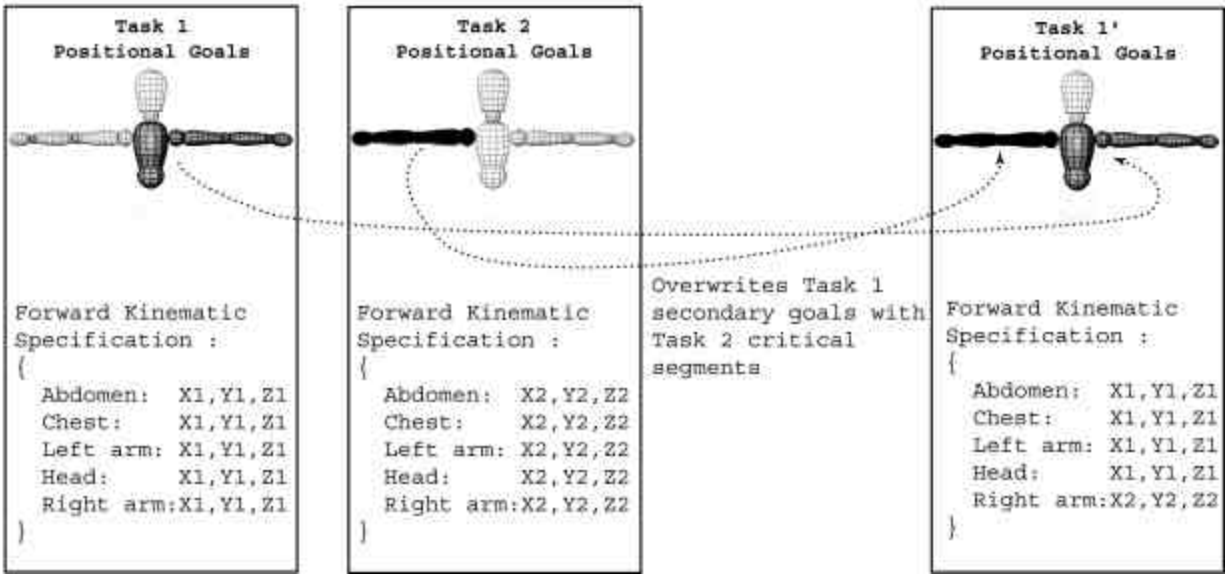

Figure 4.17 Modifying task 1 by migrating goals.

4.5 Animating the Virtual Puppet

The puppet can animate once every degree of freedom is scheduled goal states and interpolation information from the motion queue. A schematic of the state of every degree of freedom in every ball joint is shown in Figure 3.11. At the start of a new motion, the start angle is set to the current angle, and the goal angle is updated according to the scheduled positional data from the motion queue. The current step is a measure of the number of time steps elapsed since the start of the motion. The goal step is the duration of the motion in time steps. By expressing the state of every degree of freedom over time as a function we can deterministically position the puppet at every time-step.

Each degree of freedom is an Euler angle corresponding to a particular ball joint. The following equations for interpolation can also be applied to quaternions, which provide a compact form for expressing orientations. Interpolating quaternions follows the great arc between two orientations, while this is not guaranteed when interpolating Euler angles. However, the internal state of the puppet is expressed in Euler angles, and interpolating between quaternions would imply transforming the orientation representation back to Euler angles at every time step. The angles are interpolated as follows:

$$f = (q - y) \times f_I\left(\frac{step_C}{step_G}\right) + y\,., \qquad (4.1)$$

where f is the current angle, y is the start angle, q is the goal angle, $step_C$ is the current step, and $step_G$ is the goal step. f_I is the interpolation function associated with the particular movement.

At every time-step, the rotational angle for every axis of rotation in every ball joint is computed by equation 4.1. In order ensure the puppet will animate from its initial position to the final position, several constraints on the interpolation function f_I must be respected.

- $f_I(0) = 0.$

  This property ensures the motion will begin from the puppet's initial position. From Equation 4.1, if $step_C = 0$, then $f = y$.

- $f_I(1) = 1.$

  This property ensures the puppet will be in the goal posture at the end of the motion. From Equation 4.1, if $step_C = step_G$, then $f = q$.

- $0 \le f_I(x) \le 1\,, \quad \forall x \quad 0 \le x \le step_G\,.$

  This property ensures the puppet will not violate any joint limit constraints while interpolating since $MinLimit \le y \le f \le q \le MaxLimit$ or $MaxLimit \ge y \ge f \ge q \ge MinLimit$. We know $MinLimit \le y, q \le MaxLimit$ since the posture generator enforces joint limits for goal angles, and the start angle corresponds to the goal angle of the previous motion. This assertion is extended to the first task and interrupted tasks without justification.

The parameters in Equation 4.1 are taken from the motion model of the joint's currently executing task. The goal step and interpolation function, referring to the duration of the task's motion and its velocity over time respectively, are user-specified motion primitives. The possible values for these parameters are presented in Table 7.1. The set of goal angles for every degree of freedom in the motion's critical segments defines the posture to accomplish the task. If a general task is being performed, then the goal angles are hard-coded in the system and are resolved in the motion interface module presented in Chapter 6. If a reaching or sliding task is performed, then the posture generator will compute the goal angles required to position the end effector. The state of every degree of freedom is illustrated in Figure 4.18.

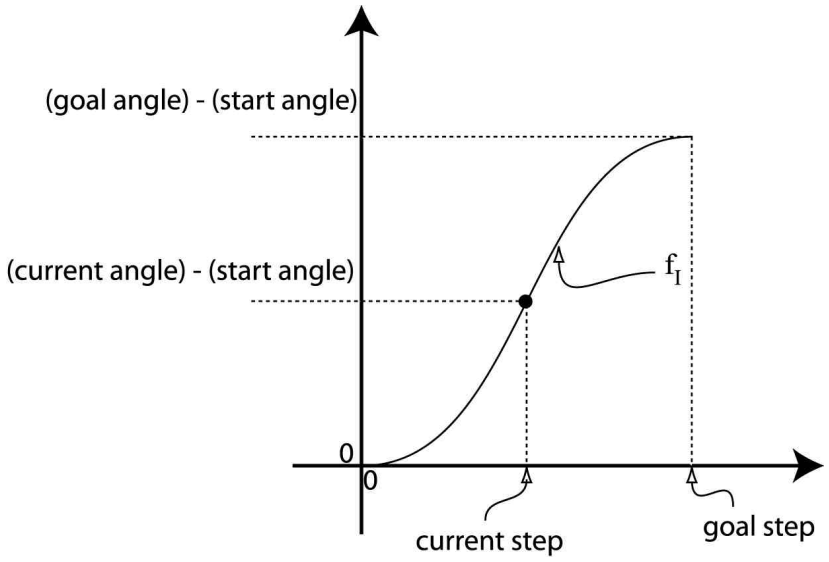

Figure 4.18 Computing the state of each degree of freedom.

We can determine the signal for a degree of freedom over the entire animation sequence by concatenating the interpolation function over time. For example, consider the rotation of the left wrist's y-axis while performing two tasks. Let us assume the puppet reaches for some object, followed by a scratching motion. The scratching motion is cyclic, composed of several wrist movements over time. The speed and interpolation functions for each task are user-specified parameters, so the signal will be dependent on user input. We present two examples of how the signal varies over time depending on different parameter values for each task. The first example performs a slow reaching motion followed by fast

scratching. Both tasks' velocity curves are characterized by a sinusoidal interpolation function consistent with research presented in Section 2.10. The second example shows a fast reaching motion followed by slower scratching movements. The reaching motion is executed with constant velocity, and the scratching movements are characterized by negative acceleration.

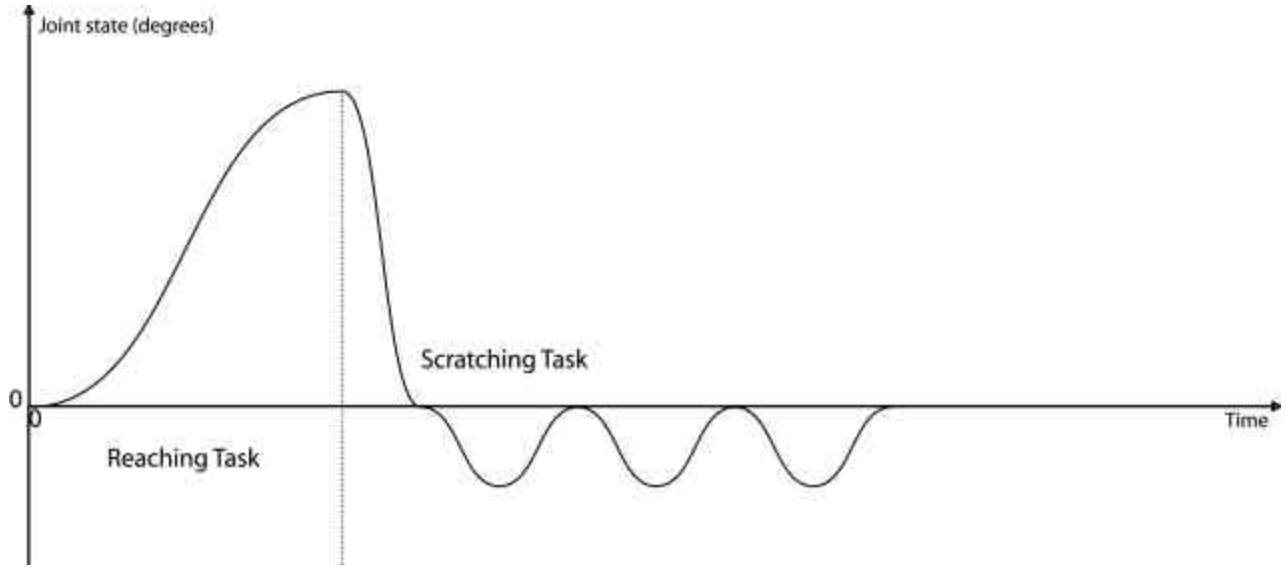

Figure 4.19 Velocity continuity between tasks.

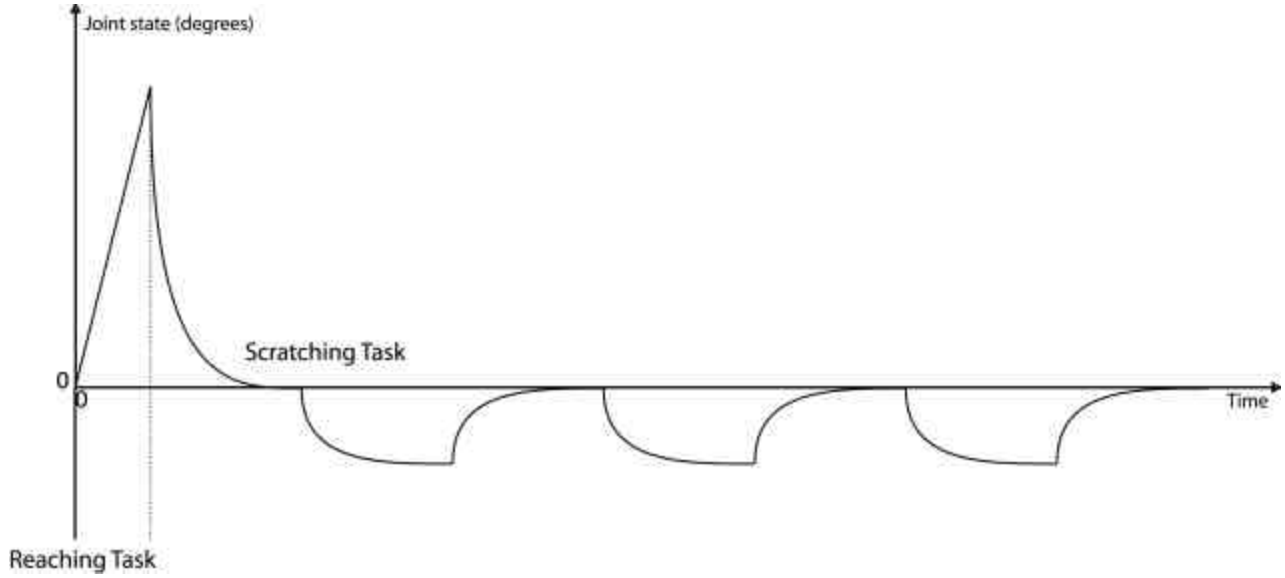

Fig. 4.20 Velocity discontinuity between tasks.

The first example shows C^1 continuity of motion. The second example has discontinuous velocity between movements. The velocity of each motion is a user-specified parameter, and hence the user is also responsible for enforcing continuity of velocity, if any.

Figure 4.19 mimics some properties of observed human behaviour, and simplifies the problem in several respects. The motion's velocity profile is a smooth, bell-shaped curve, which models fast, voluntary, single joint motions. The curve is scaled to accommodate variable amplitudes and time constraints, which models the experimental observations presented in Section 2.10. Many of the complexities of human motion are omitted from the animation of the puppet for several reasons. First, there does not exist a comprehensive model for arbitrary multi-joint voluntary discrete movements. Second, modeling all of the experimental observations in the biomechanics literature would imply much more autonomous control of the puppet's motion. For example, implementing Fitt's law as a hard constraint on the puppet's motion would restrict the user from controlling the duration of each task. Much of the applicable neuroscience research is focused on discovering preferred velocity parameters when executing various movements. However, the animator may not wish to be restricted to these parameters. Determining how more sophisticated biomechanical models can be effectively implemented in the system without limiting the user's control is a topic for future research. The most important deficiency in the current implementation for modeling realistic human motion is an overly simple approach to Bernstein's problem. In the current implementation, all joints rotate with respect to a single scaled interpolation function. In reality the acceleration of all joints in the human body is not uniform, and has been observed to be a function of the distance between the end effector and the joint's centre of rotation.

4.6 Summary

This chapter introduced algorithms and issues encountered when blending tasks together. Our method of combining several tasks into a single motion stems from the notion of critical segments discussed in Section 3.6.3. The user can dictate how a task will be combined with other tasks by specifying the value of the scheduling parameter. The motion scheduler's operation is summarized in Figure 4.3, and tasks are blending according to Figure 4.6 and 4.7. The motion resulting from scheduled tasks was discussed in Section 4.5.

Chapter 5

Posture Generator

The motion scheduler described in Chapter 4 is responsible for assigning tasks to groups of body segments. Some tasks may be performed independent of the environment, such as scratching or looking at one's watch. These tasks are referred to as *general tasks*, and are introduced in Section 3.6.1. General tasks are not ideally expressed as an inverse kinematics problem, so the position of the puppets' body segments accomplishing the task is hard-coded in the system's interface module in Section 6.1.3. Sliding and reaching motions are specific to the current state of the environment, and require positioning the puppet's hand according to the current position of the object or space. These tasks are formulated as inverse kinematics problems by the motion scheduler in Section 4.2. The problem is solved by the posture generator, which returns a forward kinematic specification of the joints assigned to accomplish the task. This chapter describes the IK solver used for computing the puppet's postures when performing tasks dependent on the state of the environment, such as reaching and sliding motions.

Realistic human inverse kinematics is remarkably complex, since the algorithm is modeling anatomical structures with abstract notions of comfort. Robotic manipulators have no esthetic preference for a single joint configuration over another. The solution space of the inverse kinematics problem applied to robotic manipulators is constrained by the mechanical

limitations of the robot. However, robotics engineers attempt to maximize the manipulability of the robot arm, which dictates the maneuverability of the manipulator in some configuration [DMSB95]. Humans on the other hand, have strong preferences for certain postures over others. This assertion is illustrated in Figure 4.4. These preferences constrain the solution space far more than the physiological limitations of the human body. Human motions appear to have common preferences as has been shown in numerous experiments discussed in Sections 2.9 and 2.10. Computer scientists have considered this problem as well, such as Lee et al. who proposed a model of human comfort for executing lifting tasks [LWZB85][BPW93]. Tevatia et al. investigated algorithms for effectively dealing with redundancies in humanoid robots [TS00]. A survey of research in positioning virtual humanoid figures with inverse kinematics is presented in Section 2.4.

To model realistic human motion, we must consider the decision-making process involved in performing some reaching motion which is not completely understood by neuroscientists. There are physiological limitations that would account for discarding most postures. For example, muscle and tendon elasticity and joint range of motion limitations both explain why humans do not reach for objects in certain ways. However, within some acceptable comfort level there is still a wide range of postures that humans consistently reject. Posture design applied to computer animation delves into neurology, psychology, and biomechanics to model how humans fundamentally perceive their surroundings and apply their bodies to accomplish certain tasks. Understanding human comfort and preferred postures is widely researched in ergonomic studies. The algorithm presented in this chapter considers results in both ergonomics and neurological research, as described in Sections 2.9 and 2.10.

The motion scheduler will analyze the contents of the motion queue, and schedule one or more tasks concurrently. This is accomplished by parsing the puppet's body into several kinematic chains, each with their own geometric constraints that specify a posture to accomplish a particular task. Given the algorithms presented in Chapter 4 and the groupings of critical segments presented in Section 3.6.3, there are only three possible kinematic chains associated with an IK problem from the motion scheduler. This assertion is presented without justification. A diagram of the posture generator's functional operation is presented in Figure 5.1.

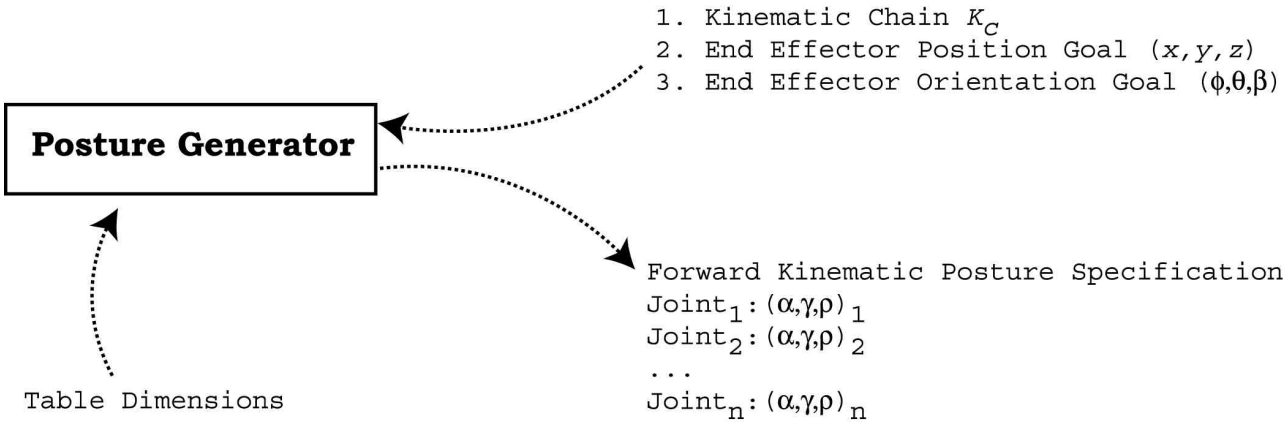

Figure 5.1 Functional diagram of the posture generator.

The kinematic chains associated with an IK problem can be one of three possibilities. Postures can be solved for a single arm, a single arm and the torso, or both arms and the torso. Kinematic chains that include a single arm has one end effector at the tip of the hand. When both arms and the torso are positioned by the IK solver, the torso is shared between two end effectors at the tip of either hand.

Section 5.1 presents an overview of techniques for solving inverse kinematics problems that will serve as a basis for presenting the algorithm implemented in Section 5.2. Section 5.3 describes how orientation goals are incorporated into the algorithm. Section 5.4 discusses the properties of our algorithm in terms of posture design. Section 5.5 describes how the algorithm proposed in Section 5.2 is applied to our 3D puppet model. Our method for dealing with table collisions is presented in Section 5.6. Finally, Section 5.7 describes the heuristics employed to find humanly natural inverse kinematics solutions.

5.1 Solving Inverse Kinematics

A kinematic chain is composed of a series of coordinate frames with a root joint and end effector. The root joint is at the base of the chain and has its own fixed coordinate frame relative to world coordinates. The end effector is the tip of the final joint segment in the chain. The end effector is not an active joint itself, but can be expressed as a translation from the base of the last joint in the chain. The end effector assumes the same orientation as its adjacent joint.

Formally, a kinematic chain K_C with n joints can be expressed as an ordered set of transformation matrices.

$$K_C = \{M_0, M_1, M_2, \ldots, M_{n-1}, M_e\}., \tag{5.1}$$

where M_0 and M_e are the root joint and end effector coordinate frames respectively.

$$Mo = \begin{bmatrix} R^0_{1,1} & R^0_{1,2} & R^0_{1,3} & T^0_x \\ R^0_{2,1} & R^0_{2,2} & R^0_{2,3} & T^0_y \\ R^0_{3,1} & R^0_{3,2} & R^0_{3,3} & T^0_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T^0_x \\ 0 & 1 & 0 & T^0_y \\ 0 & 0 & 1 & T^0_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R^0_{1,1} & R^0_{1,2} & R^0_{1,3} & 0 \\ R^0_{2,1} & R^0_{2,2} & R^0_{2,3} & 0 \\ R^0_{3,1} & R^0_{3,2} & R^0_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = T^0 R^0., \tag{5.2}$$

where T^0 and R^0 is a rotation and translation with respect to world coordinate space.

$$M_e = \begin{bmatrix} 1 & 0 & 0 & T^e_x \\ 0 & 1 & 0 & T^e_y \\ 0 & 0 & 1 & T^e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = T^e., \tag{5.3}$$

where T^e is a translation with respect to the adjacent coordinate space M_{n-1}. Intermediate joint position and orientation is expressed as a rotation and translation with respect to the adjacent coordinate frame.

$$M_i = M_{i-1} T^i R^i \ , \quad 0 < i < n \quad . \tag{5.4}$$

The configuration of the chain where all respective joint rotations equal zero is referred to as the *zero configuration* illustrated in Figure 3.10. When the puppet is in its zero configuration $R^i = I$ and $0 \le i < n$.

Given an arbitrary kinematic chain configuration, the end effector coordinates are calculated as a multiplication of transformation matrices.

$$e = \begin{bmatrix} e^o_{1,1} & e^o_{1,2} & e^o_{1,3} & e^p_x \\ e^o_{2,1} & e^o_{2,2} & e^o_{2,3} & e^p_y \\ e^o_{3,1} & e^o_{3,2} & e^o_{3,3} & e^p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = M_0 M_1 M_2 \ldots M_{n-1} M_e \quad . \tag{5.5}$$

The end effector position e^p and orientation e^o is isolated from the above matrix and expressed as

$$e^p = \begin{bmatrix} e^p_x \\ e^p_y \\ e^p_z \end{bmatrix} \quad , \quad e^o = \begin{bmatrix} e^o_{1,1} & e^o_{1,2} & e^o_{1,3} \\ e^o_{2,1} & e^o_{2,2} & e^o_{2,3} \\ e^o_{3,1} & e^o_{3,2} & e^o_{3,3} \end{bmatrix} \quad . \tag{5.6}$$

The end effector coordinates is a function of the kinematic chain configuration K_C.

$$e = f(K_C) . \tag{5.7}$$

Forward kinematics resolves the parameters of f to compute e. Inverse kinematics however, begins with the value of e and attempts to solve for the kinematic chain configuration K_C.

$$K_C = f^{-1}(e) . \tag{5.8}$$

There does not exist a general solution to this problem. The method employed to solve for a kinematic chain configuration requires intuition about the properties of the system and the constraints imposed by the problem.

There are a number of constraints one can impose on the solution, such as specifying end effector orientation e^o or restricting the range of motion of one or more links. One may

not require the end effector to rest at a specific point, but rather be satisfied with placing the end effector on a plane or along a line in space. The algorithm used to solve $f^{-1}(e)$ must also consider redundancies in the solution. That is, more than one solution may exist (often infinite solutions exist) and the solver may consider secondary constraints to deal with such under-constrained problems. A common secondary constraint is a prioritization of joints in the chain. Some joints are preferred over others and will more readily acquire a larger coordinate displacement. How redundancies are resolved depend on the properties sought in the final link configuration. Another variation of the problem involves $m > 1$ end effectors with one or more common links. The problem can be restated with more than one parameter for the function f.

$$K_C = f^{-1}(e_1, e_2, \ldots, e_m). \tag{5.9}$$

Constraints may be specified as "hard" or "soft" constraints [BB00]. Hard constraints are characterized as a necessary element in the final solution. An attempt is made to satisfy soft constraints, but not at the expense of any hard constraints.

Solving for the joint angles is difficult due to the non-linearity of the problem. Differential changes in the position of the end effector with respect to some joint i can be expressed as an m-dimensional vector, where m is the dimension of the end effector position. Let us assume the end effector position is expressed as a three-dimensional vector.

$$d_i = \begin{bmatrix} d_{i_x} \\ d_{i_y} \\ d_{i_z} \end{bmatrix}. \tag{5.10}$$

The vector d_i is referred to as the *joint velocity* of joint i, since it maps differential changes in joint coordinates to differential changes in end effector position. We can compute the differential end effector position by multiplying the incremental change in joint coordinates by the joint velocity vector.

$$de = \begin{bmatrix} de_x \\ de_y \\ de_z \end{bmatrix} = \begin{bmatrix} d_{i_x} \\ d_{i_y} \\ d_{i_z} \end{bmatrix} \cdot dj_i \ . \tag{5.11}$$

Unfortunately, the velocity vector d_i in a multiple link chain is dependent on the current state of the system. That is, d_i is a function of all other joints' coordinates in the chain. This introduces the non-linearity of the problem, since incremental coordinate changes in some joint i will affect the velocity of all other joints in the system.

A common approach is to linearize the problem about the current state of the system. We ignore the fact that joint velocities have to be recomputed after incremental changes in the state of the chain. The assumption is that joint velocities remain constant despite incremental changes. Considering the current state of the system, we compute velocity vectors for all joints and construct a Jacobian matrix.

$$J = \begin{bmatrix} d_{1_x} & \cdots & d_{n_x} \\ d_{1_y} & \cdots & d_{n_y} \\ d_{1_z} & \cdots & d_{n_z} \end{bmatrix}., \text{ where } n \text{ refers to the number of joints in the chain.} \tag{5.12}$$

The previous equation scales with the Jacobian matrix.

$$de = J \cdot dj \ ., \text{ where } dj = \begin{bmatrix} dj_1 \\ \vdots \\ dj_n \end{bmatrix}. \tag{5.13}$$

In this case, we are solving for differential changes in the end effector coordinates given differential changes in joint coordinates. However, we know the differential change in end effector coordinates and need to solve for the differential joint coordinates. For the inverse kinematics problem, we reformulate the equation.

$$dj = J^{-1} \cdot de \, . \tag{5.14}$$

Because of our linearization approximation, the resulting solution is only reasonable for small incremental changes in joint angle. As a result, we must iteratively solve the above equation and recompute J^{-1} for each iteration. Once dj is solved for a single iteration, we integrate to find the new joint coordinates.

$$j_i' = j_i + \int_0^{t_0} dj_i \, dt \,., \text{ where } t_0 \text{ is small.} \tag{5.15}$$

The Jacobian matrix is not always invertible. When the dimensionality of the end effector is less than the number of degrees of freedom in the kinematic chain there will be more columns than rows in the Jacobian matrix. In this case, the Jacobian matrix is redundant. If the rows in the Jacobian are not all independent, the matrix is singular and the chain is said to be in a *singular configuration*. Neither redundant nor singular matrices are invertible, which renders further iterations toward a solution impossible with the above formulation. Ill-conditioned matrices cause large spikes in joint velocity that intuitively lends to unusual contortions. Overcoming non-invertible and ill-conditioned matrices gracefully is difficult, and algorithms must take special consideration when dealing with such degenerate cases.

Another method for solving $K_C = f^{-1}(e)$ involves reformulating the problem as a non-linear optimization problem. An objective function is proposed, which is to be minimized or maximized, depending on the problem and the objective function. An obvious choice for an objective function is the distance formula, where we attempt to minimize the distance between the end effector and goal coordinates.

$$D = \left\| e^p - g^p \right\| = \sqrt{(e_x^p - g_x^p)^2 + (e_y^p - g_y^p)^2 + (e_z^p - g_z^p)} \,., \tag{5.16}$$

where e^p and g^p are the end effector and goal coordinates in world space respectively. The position of the end effector e^p is dependent on the configuration of the kinematic chain. Hence, D is also dependent on the chain configuration, and the objective function can be reformulated.

$$e^p = f_p(K_C). \tag{5.17}$$

$$D_{K_C} = \left\| f_p(K_C) - g^p \right\|. \tag{5.18}$$

The function f_p is an n-dimensional function, where n corresponds to the number of joints in the kinematic chain. Hence, D_{K_C} is also an n-dimensional function, and can be visualized as an $n+1$-dimensional surface D_{n+1}. The surface is $n+1$-dimensional because we have n joints corresponding to n independent variables, and one dependent variable D. We are looking for a chain configuration $K_{C_{\min}}$ that will correspond to a point $D_{\min}$ that minimizes the dependent variable D.

$$D_{\min} \le D, \quad \forall\, K_C\,. \tag{5.19}$$

Graphically, $D_{\min}$ corresponds to the shallowest point on the surface D_{n+1}.

Finding $D_{\min}$ is not always easy. There are a number of iterative search techniques and numerical methods designed to solve non-linear optimization problems. Often these methods are sensitive to the initial state of the system. The problem is further complicated if we do not know the value of $D_{\min}$. If a solution to the inverse kinematics problem exists, then $D_{\min} = 0$. However, this may not always be the case. Depending on the formulation of the problem, some $D'_{\min} > 0$ may be sufficient, where $D'_{\min}$ is the minimum distance found by our solver. If $D'_{\min}$ is not satisfactory for our purposes, there is no way of determining whether $D'_{\min} = D_{\min}$.

Assuming we know $D'_{\min} \neq D_{\min}$, it is difficult to determine where our solver failed. If the solver failed due to degenerate initial conditions, it is not always clear how a new initial configuration should be selected. If the solver itself is unable to find $D_{\min}$, it is not always obvious how one should modify its parameters. When $D'_{\min} \neq D_{\min}$ the configuration is said to be stuck in *local minima*.

5.2 Overview of IK Algorithm

The algorithm implemented in our system is an iterative optimization technique, where every joint is positioned independently to minimize the objective function. Our IK algorithm achieves a desired goal configuration by modifying joint angles one at a time, beginning with the most distal joints of the IK chain and terminating with the most proximal joints. This differs from inverse Jacobian methods, where each step of the algorithm involves a coordinated movement of all joints. The algorithm proceeds as presented below.

```
Step 1: Estimate a natural position to minimize the distance between the
end effector and the goal position.

Step 2: Refine the estimate with the IK solver to satisfy the position and
orientation constraints. Multiple solutions are computed.

Step 3: Select the best solution from among several possibilities that
maximizes the perceived naturalness of the posture, and minimizes the
error according to the objective functions and imposed constraints.
```

Step 1 uses studies introduced by Soechting et al. to generate a natural-looking estimate of the IK solution [SF89a][SF98b]. This estimate acts as the IK solver's initial conditions, and is presented in Section 5.5. Step 2 is accomplished with a 3D inverse kinematics algorithm. The algorithm applied in 2D is a derivative of the cyclic-coordinate descent method presented in Wellman's thesis [Wel93]. The algorithm is applied to the puppet's 3D ball joints by reducing the problem to a series of 2D cases. Solving the 2D problem is presented in Section 5.3, and its application to 3D ball joints is described in Section 5.5. Multiple postures are computed, and the best among them is selected in step 3 according to a score function presented in Section 5.7.5.

5.3 2D Inverse Kinematics

For all joints, we determine the rotation angle that will minimize the distance between the current end effector position E and the goal coordinates G. Assume the algorithm is currently considering some joint j_i. The algorithm proceeds as follows.

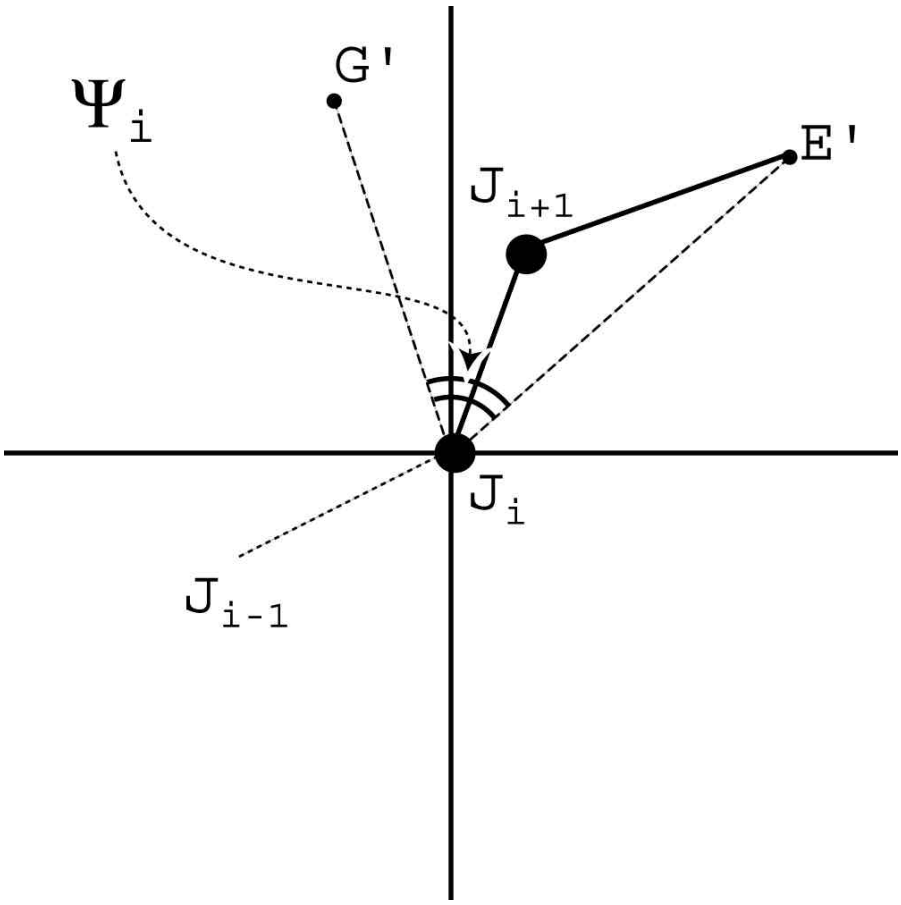

Figure 5.2 System state after step 1

1) Transform the current end effector and goal position to joint i local coordinates. Let us refer to the new positions as E' and G' respectively.

2) Determine the joint angle Ψ_i between the vectors E' and G'. Ψ_i is illustrated in Figure 5.2.

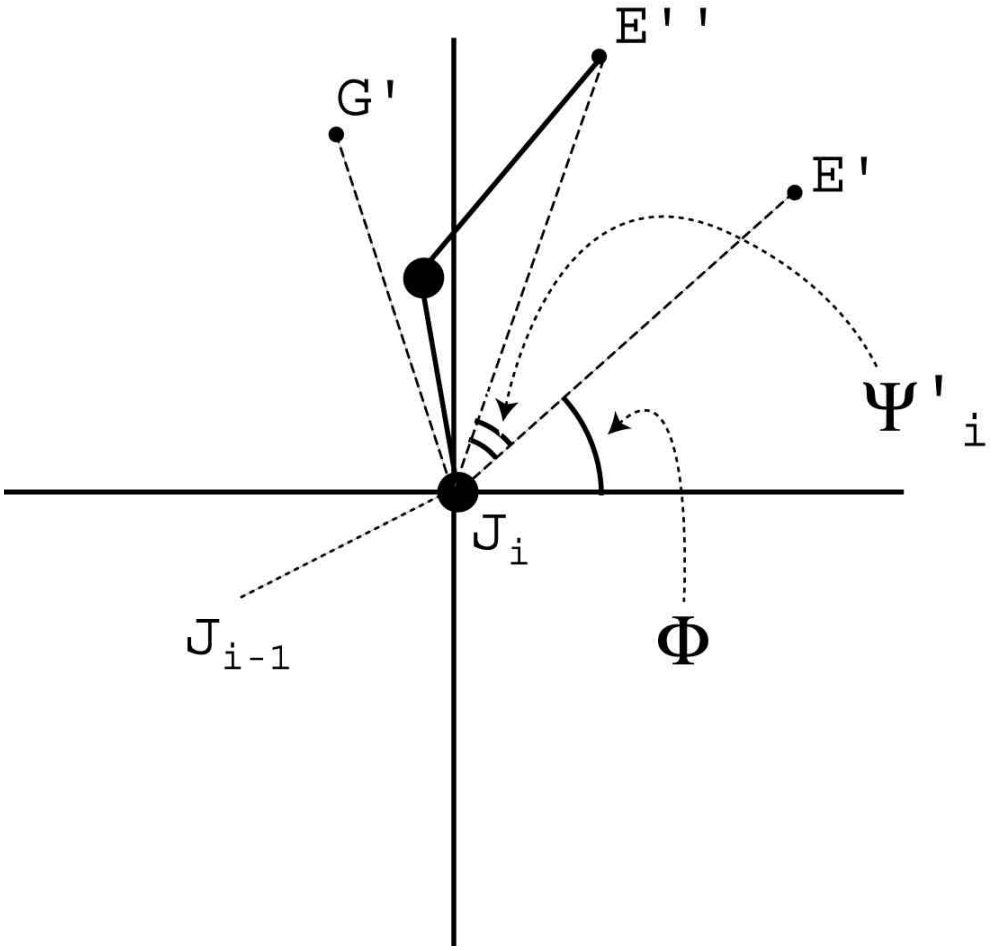

Figure 5.3 System state after step 3

2) We apply a weight factor to Ψ_i to control the amount of rotation applied to joint i. Let us refer to the damping value of some joint i as w_i, where $0 \leq w_i \leq 1$. w_i is an indication of the perceived "stiffness" of the joint. A damping factor of 0 implies a completely stiff joint, while 1 indicates a free moving joint that moves readily towards the goal. The new computed angle is calculated from the equation $\Psi_i^{'} = \Psi_i \cdot w_i$. The new computed end effector position E'' is shown in Figure 5.3.

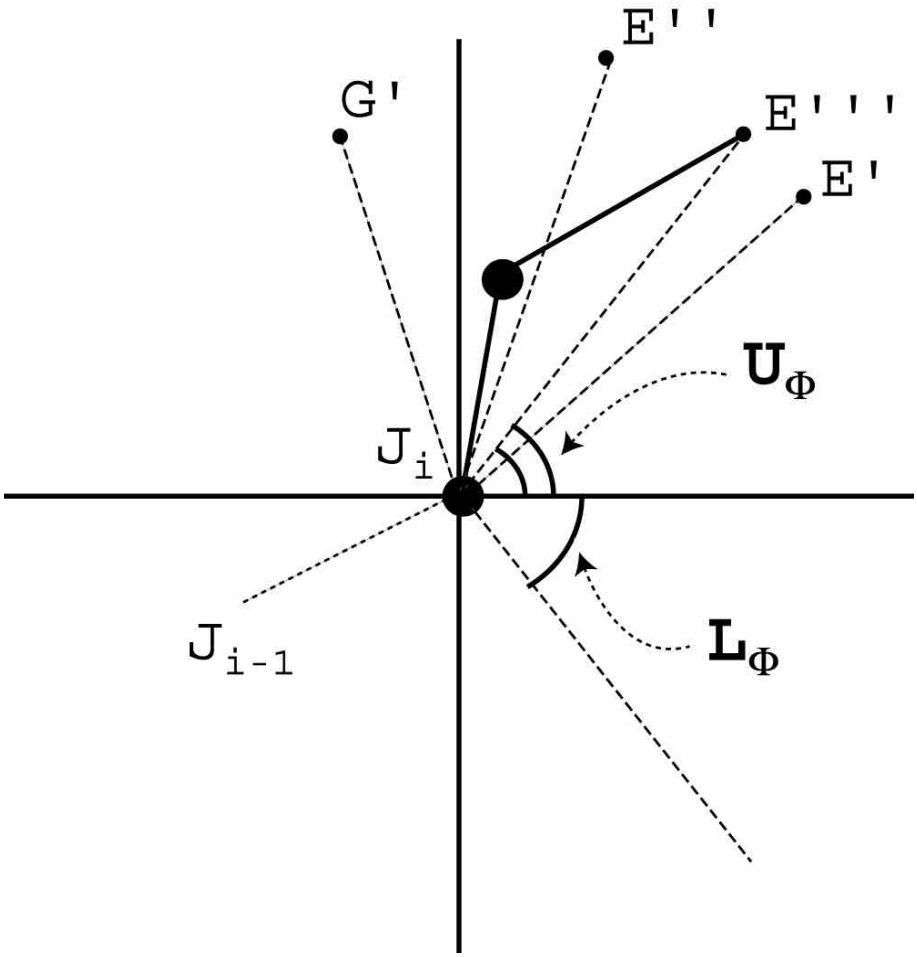

Figure 5.4 System state after step 4

4) If the original state of joint i is Φ, then the updated joint angle is denoted Φ', where $\Phi' = \Phi + \Psi'$. Joint rotation limits are applied to constrain the state of joint i. Let us denote the upper and lower bound of joint i as U_Φ and L_Φ respectively.

if $\Phi' > U_\Phi$ *then* $\Phi'' = U_\Phi$
if $\Phi' < L_\Phi$ *then* $\Phi'' = L_\Phi$
else $\Phi'' = \Phi'$.

The new state of joint i is Φ'' after constraining its range by the upper and lower bounds. Figure 5.4 shows the final end effector position E''' after being constrained by U_Φ.

The algorithm performs the above computation for every joint in the kinematic chain starting with the most distal link j_{n-1} and ending at the root joint j_0. In step 3 the algorithm updates the state of the system after every joint computation. For a kinematic chain with n joints, the state of the system will be updated n times per iteration. Prismatic joints do not exist in our puppet model and are not considered in the above algorithm.

As a non-linear iterative optimization algorithm there is an objective function maximized or minimized. For each joint, we calculate a rotation that positions the end effector as close to the goal as possible. This objective function can be expressed in several of ways:

1) Minimize the distance $D = \|E - G\|$.

2) Maximize the vector dot product $DP = E \circ G$.
   where E is the end effector and G is the goal expressed as vectors in world coordinates.

There are a variety of ways one can approach the two objective functions. There are numerical methods that iteratively converge on a minimum or maximum of non-linear functions, such as the above functions. The algorithm above proposes an analytical solution using simple geometric reasoning, which is executed in constant time.

5.4 Properties of the Algorithm

The benefits and disadvantages of non-linear optimization techniques versus Jacobian methods is a lengthy discussion introduced in Section 5.1. The most prominent disadvantage of non-linear optimization is that the algorithm may find a local minimum rather than the global minimum. Depending where one begins to decend towards a surface minimum and the algorithm used to dictate the direction and magnitude of the decent, we may discover a local minimum. Determining whether or not one has discovered a local minimum or the true global solution may require solving the original problem. There are several techniques for avoiding or escaping local minima, such as randomly selecting several initial configurations, or randomly shifting the state of the manipulator to escape a shallow cusp in the objective

function's height field. When stuck in local minima the algorithm will cease to converge on a potential solution, while Jacobian based methods are not subject to these anomalies. An in-depth discussion on local minima is beyond the scope of this thesis.

Our IK algorithm avoids local minima by setting the initial state of the system close to the global minimum. This is done by estimating a solution in Section 5.7. The estimate reduces the probability of encountering local minima while iterating towards the global minimum. The posture estimate also reduces the computation time of the algorithm, since the figure's initial position is close to a global solution. Several paths are considered when iterating towards the goal in Section 5.7.4. These paths are saved as several potential solutions to the IK problem, from which the score function in Section 5.7.5. chooses the best alternative.

In terms of computational cost, Welman observed better performance from the cyclic-coordinate descent method when compared with inverse Jacobian methods [Wel93]. As the accuracy demanded of the solution increases, the computation time of inverse Jacobian methods increases exponentially, while the above iterative technique increases approximately linearly. Under some circumstances the algorithm will begin to converge slowly, particularily when the kinematic chain is being "stretched" towards the goal. In these cases the joint angle perturbations become small and require many iterations to pull the end effector. An in-depth analysis of inverse kinematics algorithms is an open research area and is beyond the scope of this thesis.

One important property of this algorithm in terms of posture design is the preference of certain joints in determining a solution. Each iteration begins with the most distal link and progresses towards the base. Every joint in the chain maximizes its potential progress towards the goal, which can result in severe contortions of the most distal links. Beginning iterations at the root joint is not a feasible alternative for two reasons. First, the algorithm will be biased towards joints closer to the root, which is not any more desirable than distal links. Second, progression from the most distal link to the root is imperative or else the algorithm becomes susceptible to local solutions, rather than iterating towards the global minimum. This is illustrated in Figure 5.5, where the left-most diagram illustrates the problem, and the next two diagrams illustrate how the objective function is minimized if one considers distal

or proximal links, respectively. The figure in the third diagram from the left is stuck in local minima.

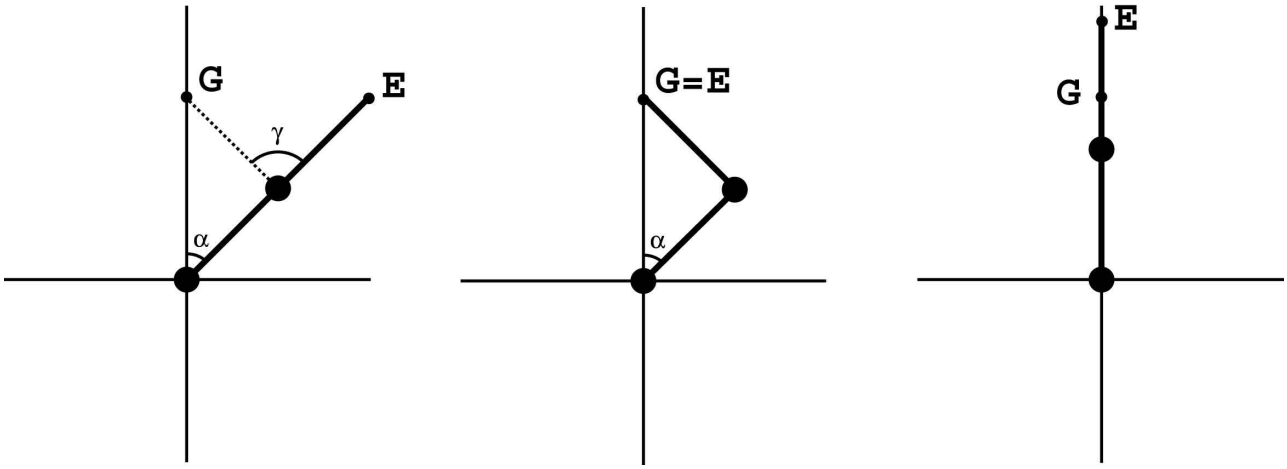

Figure 5.5 Discovering local minima when iterating from the base joint.

Preference for distal links is not conducive to posture design, where human beings tend to distribute joint rotations throughout the body to achieve some reaching task. This property is resolved by damping joint rotations at distal links, which implies applying a low weight factor for a particular joint. The damping factors applied to each joint is presented in Appendix C. Adopting any other inverse kinematics technique, such as inverse Jacobian methods, does not resolve our problems. The fundamental issue is to understand how humans delegate joint rotations for an arbitrary reaching motion, and this extends beyond the particular inverse kinematics algorithm selected.

5.5 3D Inverse Kinematics

In this section we describe how orientation and position objective functions are integrated into a single IK model. The theoretical description of end effector goals for our 3D puppet model is presented.

Orientation goals are specified by the value of the grip parameter for reaching tasks, and the orientation matrix for sliding tasks. The specification of reaching and sliding task

parameters is discussed in Section 3.6.1 and 3.6.2. The motion scheduler will assign orientation goals for each end effector in the kinematic chain as summarized in Figure 5.1.

Incorporating orientation goals into the IK solver requires some intuition about the properties of the figure being positioned and the IK algorithm itself. Difficulties may arise primarily when orientation goals and positional goals conflict with one another. At every iteration there is a dilemma of how to satisfy both positional and orientation constraints when the two objectives may suggest different solutions.

To compute a solution for satisfying orientation constraints, we consider the orientation of the current joint coordinate frame, the current orientation of the end effector, and the goal orientation of the end effector. Let us refer to these 3×3 matrices as O_j, O_e, and O_g respectively. The transformation applied to the current joint that will satisfy the end effector orientation constraints is denoted $O_g^{'}$. We compute the updated joint orientation $O_j^{'}$ by the equation below.

$$\begin{aligned} O_j^{'} &= O_j \cdot O_g^{'}. \\ O_g^{'} &= O_e^{-1} \cdot O_g. \end{aligned} \qquad (5.20)$$

Since each ball joint's orientation is defined in terms of Euler angles, we would like to express this transformation in terms of three rotations f, j, and y about the local x, y, and z-axis respectively. Without loss of generality we assume an arbitrary order of rotations.

$$O_g^{'} = Rot_X(f) \cdot Rot_Y(j) \cdot Rot_Z(y). \qquad (5.21)$$

Paul presents an analytical solution to solve for the local transformations [Pau81], and is presented in Appendix B. The local axis rotations that maximize the orientation objective function is computed in equation (5.21). The objective function we are attempting to maximize is the matrix dot product between the current and goal orientation, which are denoted by 3×3 matrices. The dot product DP is computed as the summation of the product of corresponding elements.

$$O_e = \begin{bmatrix} e_{0,0} & e_{0,1} & e_{0,2} \\ e_{1,0} & e_{1,1} & e_{1,2} \\ e_{2,0} & e_{2,1} & e_{2,2} \end{bmatrix}, \qquad O_g = \begin{bmatrix} g_{0,0} & g_{0,1} & g_{0,2} \\ g_{1,0} & g_{1,1} & g_{1,2} \\ g_{2,0} & g_{2,1} & g_{2,2} \end{bmatrix}$$

$$DP = O_e \circ O_g , \tag{5.22}$$

where O_e is the current end effector orientation, and O_g is the goal orientation.

$$DP = (e_{0,0} \cdot g_{0,0}) + (e_{0,1} \cdot g_{0,1}) + \ldots + (e_{2,2} \cdot g_{2,2}) . \tag{5.23}$$

Damping weight factors and joint limits are applied to the joint rotations the same as in step 3 and 4 of the algorithm in Section 5.3. These parameters are incorporated in the calculation presented in Appendix B.

To integrate both orientation and positional goals at every iteration, we consider a weighted sum of the optimal joint angles for both objective functions. The final joint state is a function of r and h that minimizes and maximizes the position and orientation objective functions respectively. The angles with respect to some figure configuration are illustrated in Figure 5.6. The final differential joint angle Δj_i for some degree of freedom j_i is calculated for every iteration in equation (5.24).

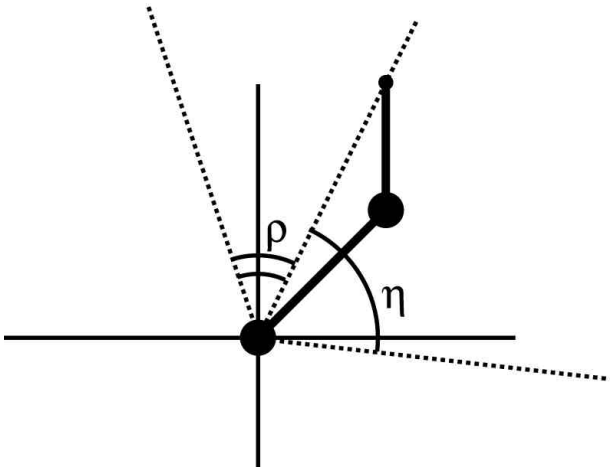

Figure 5.6 Joint displacement for optimizing orientation and position objective functions.

$$\Delta j_i = w_i \cdot (w_o \cdot r + w_p \cdot h), \quad (5.24)$$

where w_o and w_p are the orientation and position weight factors respectively, and w_i is the damping weight factor of the current joint introduced in Section 5.3 and presented in Appendix C.

$$w_o = (1 - w_p), \text{ and } 0 \le w_p \le 1. \quad (5.25)$$

Finding appropriate values for w_o and w_p is difficult, since the joint will neither optimize the orientation nor the position objective functions. It is not clear how one can find a solution if every iteration compromises between orientation and position, especially when $\left| (r - h) \right|$ is large. Our final implementation sets $w_p = 0$ for some degrees of freedom in some iterations, and $w_p = 1$ for the rest of the cases. The values of w_o and w_p parameters at various stages of the IK solver's iterative process is presented in Section 5.7.6.

Given the two-dimensionality of the algorithm in Section 5.3, additional calculations must be performed when applied to our three-dimensional puppet. Ball joints in our puppet can be modeled as three one-dimensional joints corresponding to each axis in the coordinate frame. All three one-dimensional joints have the same centre of rotation, and will rotate in a specific order about its respective axis. Given the pre-defined order of rotations, the first two joints have a link length of zero, while the final joint length is the length of the adjacent body segment. The same proximal to distal ordering is preserved for the puppet's three dimensional ball joints. Within each ball joint we consider each axis independently and in a sequence determined by the ball joint's specific order of rotations presented in Table 3.1. As the algorithm iterates each ball joint, the goal coordinates and end effector are mapped onto the two dimensional plane perpendicular to the current axis considered. This projection is illustrated in Figure 5.7. The distance objective function is minimized with respect to the rotating axis by the 2D algorithm presented in Section 5.3. The orientation objective function is maximized as in equation (5.22) and Appendix B. Let us assume the algorithm is considering some joint axis j_i. The computation proceeds as follows.

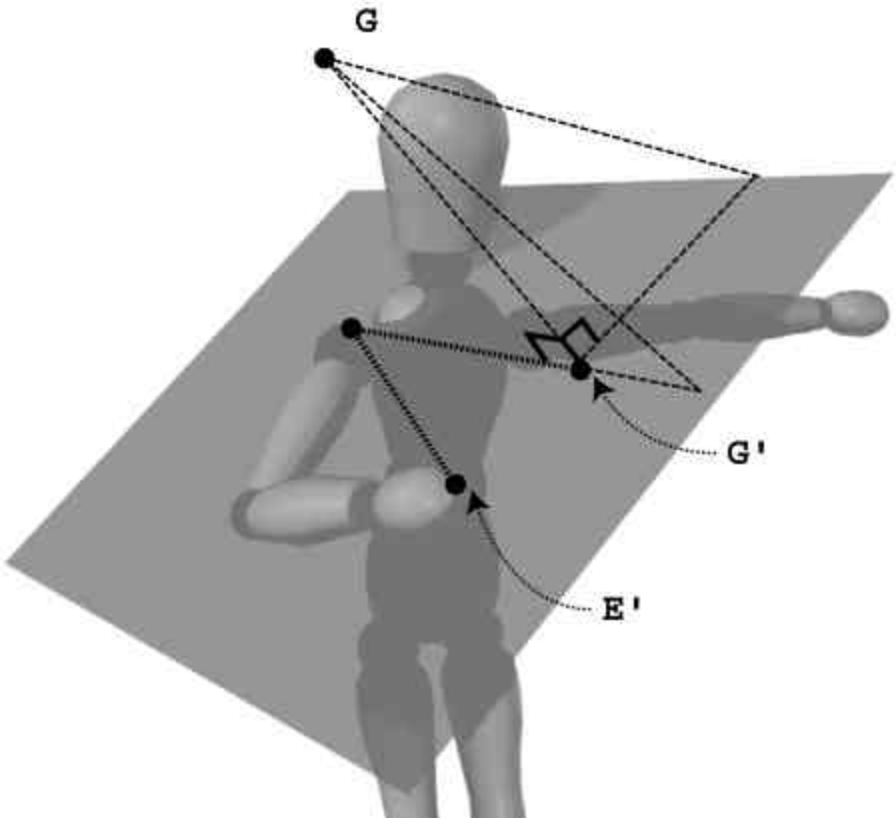

Figure 5.7 Projecting E and G for a shoulder rotation.

Step 1: Project the three-dimensional end effector and goal coordinates onto the plane perpendicular to j_i. Refer to Figure 5.7.

Step 2: Solve the 2D problem according to the algorithm presented in step 2 of Section 5.3.

Step 3: Determine the joint rotation to maximize the orientation objective function according to the calculations in Appendix B.

Step 4: Compute the differential joint rotation from equation (5.24).

Step 5: Apply joint limits according to the algorithm presented in step 4 of Section 5.3.

Step 6: The position of the hand, wrist, and elbow is updated. We check for collisions according to the algorithm in Section 5.6.

Step 7:

7.i. If j_i is the last axis in the ball joint's sequence of rotations, then the algorithm proceeds to the first rotation in the proceeding ball joint. If j_i belongs to the ball joint at the base of the kinematic chain, then the preceeding ball joint returns to the most distal joint in the kinematic chain.

7.ii. If j_i is not the last axis in the ball joint's sequence of rotations, then the algorithm proceeds to the next axis.

5.6 Coping with Collisions

An additional constraint imposed on the inverse kinematics algorithm is collision detection and avoidance. A comprehensive collision detection and path planning implementation is beyond the scope of the thesis. Relevant literature can be found in [BBGW94][GC95]. The algorithm proposed in this section is similar to [BB98]. The existing implementation only considers collisions between the arms and the table, which demonstrates how object avoidance can be incorporated into the existing posture design framework. There are two phases involved when considering collisions. One must first detect whether or not a collision has occurred. Second, one must remove the collision by modifying the state of the puppet.

In step 6 of the algorithm presented in Section 5.5, the state of the puppet is analyzed to see if a collision with the table has occurred. The joint positions are modeled as points that correspond to its centre of rotation. However, we must also consider that the puppet's geometry may have penetrated the table, even though a particular joint's centre of rotation has not. To accomplish this, we consider an offset value to compensate for the puppet's geometry when detecting collisions. This offset is denoted by T_{thresh} in equation (5.26). The offset is conservatively set to ensure the geometry does not penetrate the table, rather than invoking a collision correction only for deep penetrations of the puppet's geometry. The position of the puppet is corrected if the arm joints are within a certain distance to the table. Detecting collisions is simplified in our model since the table is parallel to the X-Z plane in world coordinates.

Let us refer to the position of the end effector, wrist, and elbow as $[e_x, e_y, e_z]$, $[w_x, w_y, w_z]$, and $[l_x, l_y, l_z]$ respectively. Let T_Y refer to the height of the table, and T_Z refer to the distance between the puppet and the edge of the table, as shown in Figure 3.7. A collision has occurred if at least one of the following statements is true:

$$\begin{aligned} &i) \quad e_y < T_Y + T_{thresh}, \text{ and } e_z > T_Z \\ &ii) \quad w_y < T_Y + T_{thresh}, \text{ and } w_z > T_Z \\ &iii) \quad l_y < T_Y + T_{thresh}, \text{ and } l_z > T_Z \end{aligned} \tag{5.26}$$

We also consider the geometry of the forearm as a possible source of collision. A primitive bounding cylinder with axis of symmetry L_1 approximates the geometry of the forearm. A line segment L_2 approximates the front edge of the table. A collision with the forearm has occurred if the cylinder approximation penetrates the front edge of the table. This is determined by calculating the distance between the most proximal points on line segments L_1 and L_2. If this distance is less than the cylinder radius R then a collision has occurred.

$$iv) \text{ Distance } (L_1, L_2) < R \tag{5.27}$$

There are several methods for coping with collisions. One possibility is to allow collisions to occur, then adjust the state of the puppet to remove table penetration. This method is termed *collision removal*, which removes collisions after they occur. The puppet's position must be adjusted to preserve the posture's character while removing the collision. Alternatively, *collision aviodance* does not update the state of the puppet if a collision is detected. The iteration that caused the collision is discarded and the algorithm considers the next joint. Collision avoidance did not work as well as collision removal since it severely hampered the IK algorithm's progress. Local minimum anomalies were common, since the technique does not reposition the puppet to iterate towards a global solution. Puppet configurations that are close to a collision state may conceivably be unable to adjust any

degree of freedom without penetrating the table. In these circumstances, the puppet will cease to move any of its joints. Our system implements collision removal, and the algorithm is presented below.

While detecting collisions is straightforward, removing the collision is more difficult. First, we must reposition the puppet so collision constraints are respected and the new posture remains natural. Second, we must reposition the puppet to ensure the progress of the IK algorithm. If one removes the collision yet moves the end effector drastically far away from the goal, it is conceivable that a solution may not be found if multiple collisions occur. Third, we must reposition the puppet so the IK will progress towards the global minimum. It is possible for thrashing to occur if the new puppet posture is close to the previous iteration that caused the collision. Future iterations will move the puppet into the same collision state repeatedly, while no progress towards the goal is made.

Our method first calculates a default posture that is guaranteed to remove the collision. A collision-free posture is computed by interpolating between the current colliding configuration and the default posture. An algorithm summary is presented below.

Step 1 : Compute a default state for the arm S_A that is guaranteed to be collision-free.

Step 2 : Linear interpolation in joint-space is used to determine the final puppet posture S_F according to the equation $S_F = f \cdot S_A + (1-f) \cdot S_B$, where S_B is the original colliding puppet configuration.

The value of f in Step 2 is determined by a simple sequential search procedure, $f = n \cdot \Delta f$. We choose the smallest value of f such that S_F is collision-free and thrashing is avoided.

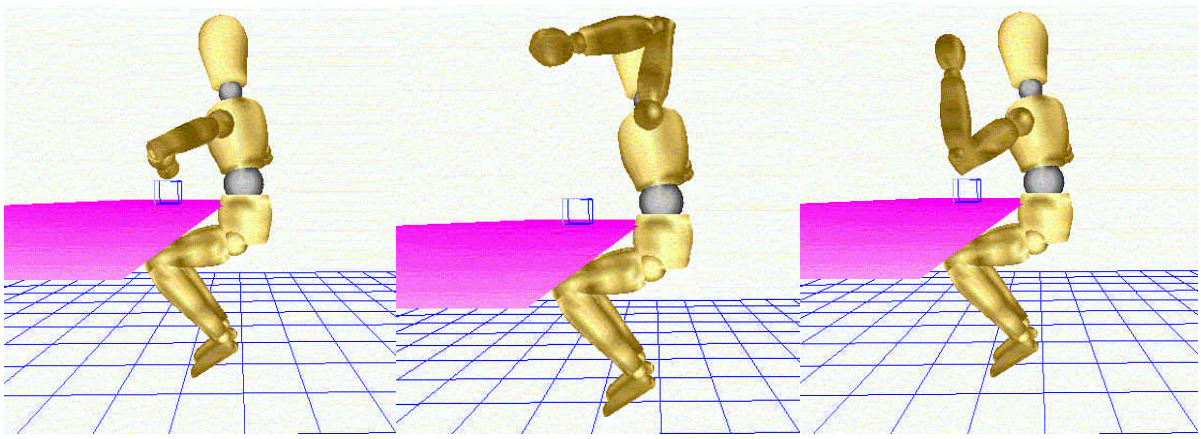

Figure 5.8 (From left to right) Original posture S_B, and corresponding S_A for elbow and hand collisions.

Determining the collision-free posture S_A requires intuition about the current state of the environment and the problem domain being considered. In our table scenario, a collision-free posture is computed by rotating the appropriate joints to position the geometry vertically in world space. Figure 5.8 shows examples of S_A and S_B. The first diagram is some arbitrary original posture S_B deemed to be colliding with the table according to (5.26) and (5.27). The second diagram illustrates S_A for elbow and forearm collisions, and the third diagram shows S_A for wrist and hand collisions. If an elbow or forearm collision occurs, the shoulder is rotated to position the upper arm vertically in world space. If the hand or wrist collides with the table, the elbow and shoulder is rotated to position the forearm vertically in world space. In both cases, S_A is computed by rotating the arm joints to modify the colliding configuration S_B.

When the shoulder is rotated to remove an elbow collision it is possible for hand or wrist collisions with the table to occur. It is also possible for multiple joints to collide with the table while the IK algorithm is iterating towards a solution. To overcome these anomalies, we first remove the elbow collision to generate a pose S_A. If wrist or hand collisions persist, we remove the collisions from S_A to generate a new posture $S_A^{'}$.

```
If ( elbow or forearm colliding with table )
      Remove elbow collision
If ( hand or wrist colliding with table )
      Remove hand collision
```

Figure 5.9 Illustrates the effectiveness of our collision removal algorithm for a simple two-handed grasping motion. The left diagram is the posture computed with collision removal enabled, and the right diagram shows the same task executed without collision removal.

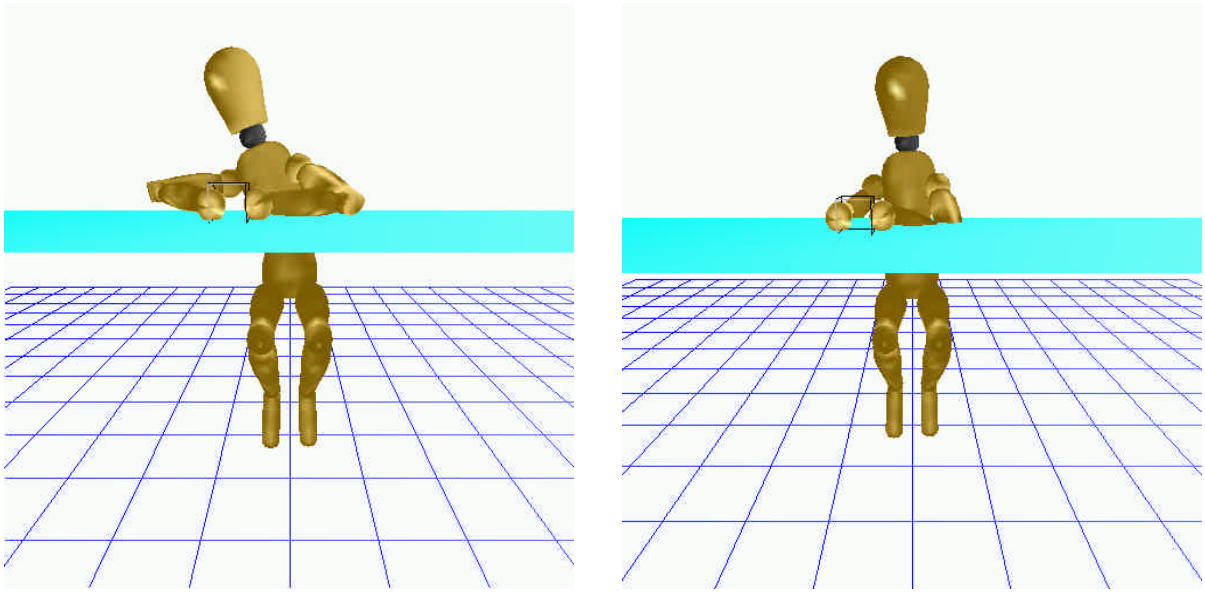

Figure 5.9 Grasping posture computed with and without collision removal.

5.7 Natural Postures

Before the inverse kinematics algorithm is invoked, a solution is estimated to position the end effector close to the goal with a natural-looking pose. Maximizing the naturalness of the postures and minimizing computational cost demands a constant-time function that maps end effector goals to a humanly natural IK solution. Unfortunately, such a function does not exist. Instead, we estimate a posture that is humanly natural but not an IK solution. We employ our inverse kinematics algorithm to position the end effector at the desired goal while preserving the naturalness of the original estimate.

The introduction of Chapter 5 categorizes inverse kinematics problems according to the body segments included in the kinematic chain. For each kinematic chain in Section 3.6.3, we estimate the position of each group of body segments in the following order.

```
Single Arm:
1) Estimate arm position

Single Arm and Torso:
1) Estimate torso position
2) Estimate arm position

Both Arms and Torso:
1) Estimate torso position
2) Estimate left arm position
3) Estimate right arm position
```

The estimate will serve as the initial conditions for the IK algorithm, which will be natural-looking and close to a global solution. The function used to position the arms is summarized in Section 5.7.1, and the torso estimate is presented in Section 5.7.2.

The estimate function is dependent on the relative distance of the goal coordinates to the puppet's body. Distant goal coordinates will generate different initial solution estimates than more proximal coordinates. We make the distinction since positions that require far stretching will require more pronounced bend at the torso. The arm posture will tend to be straight when stretching far for distant goal positions, as opposed to proximal goals where the arm position is the result of a more complex function. If the distance between the base of the head and the goal coordinates is greater than some pre-defined value D then the task requires far stretching by the puppet, and the distal estimate function is used. Proximal and distal posture estimates are shown in Figure 5.10. The left diagram shows the puppet's initial position after computing a proximal estimate. The right illustrates a distal approximation for the same task.

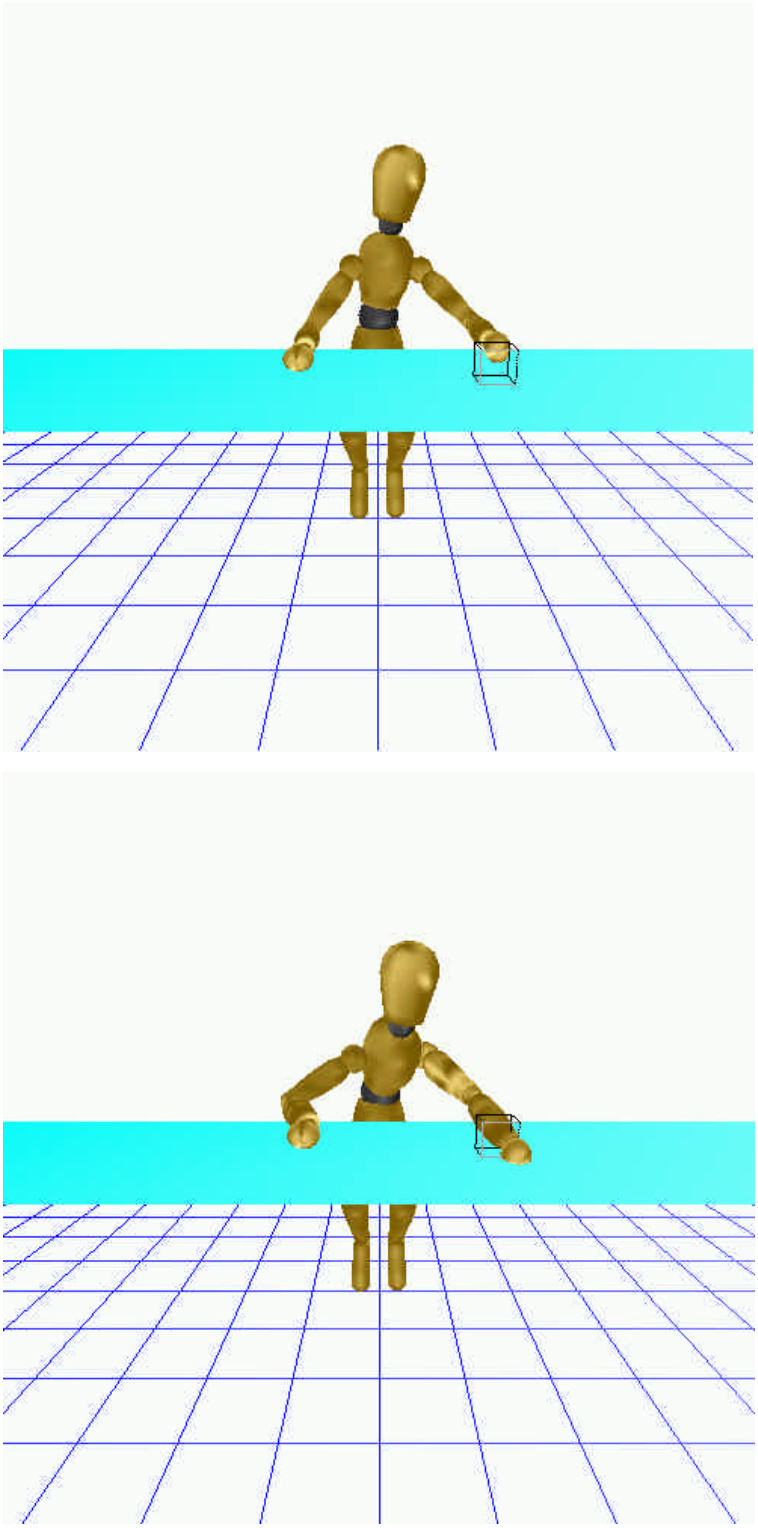

Figure 5.10 Proximal and distal posture estimates.

5.7.1. Estimating Arm Position

Estimating the position of the arm for proximal reaching tasks follows from results in neurophysiology, which found a near linear relationship between the position of a stylus held by a human and the orientation of the shoulder [SF89a][SF89b]. The goal position is expressed as spherical coordinates in terms of the shoulder local coordinate frame. Let R be the distance from the end effector goal to the origin. Let Φ and Ψ be the azimuth and elevation of the target respectively. The elevation and yaw of the upper arm, Θ and N respectively, is expressed in degrees as:

$$\Theta = -6.7 + 1.09 \cdot R + 1.10 \cdot \Psi. \quad (5.28)$$

$$N = 67.7 - 0.68 \cdot R + 1.00 \cdot \Phi. \quad (5.29)$$

After the shoulder positions the upper arm, the forearm is positioned to minimize the distance between the end effector and the goal position. Consider the two line segments (E,e) and (E,G), where (E,e) is the line originating at the elbow's centre of rotation and passing through the end effector coordinates, and (E,G) is the line originating at the elbow's centre of rotation and passing through the goal coordinates. The shoulder x-axis and elbow y-axis determine the forearm's orientation, and are adjusted to make (E,e) and (E,G) collinear. This is illustrated by the left diagram in Figure 5.11.

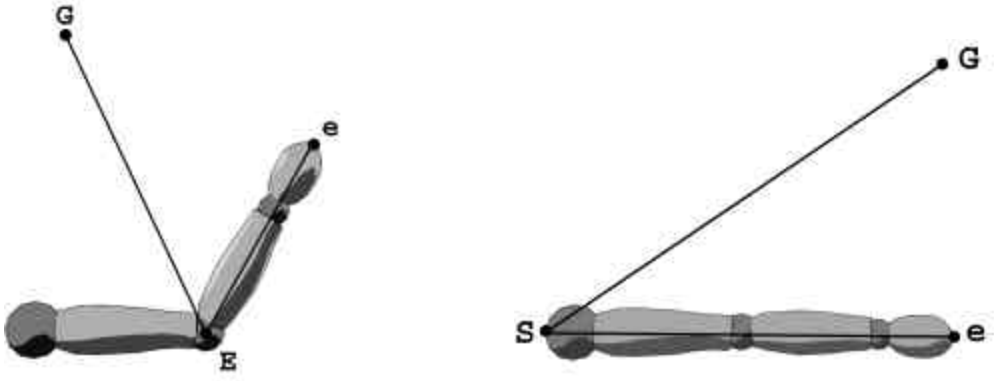

Figure 5.11 Positioning arm for proximal and distal estimates.

For distal reaching tasks, the shoulder simply points the arm straight at the goal position. The shoulder z-axis and y-axis is positioned to make the line segments (S,e) and (S,G) collinear, where (S,e) and (S,G) are line segments originating at the elbow centre of rotation and passing through the end effector and goal coordinates, respectively. This is illustrated by the right diagram in Figure 5.11.

5.7.2. Estimating Torso Position

If the chest and abdomen is included in the kinematic chain, we must estimate an appropriate orientation for the body as well as the arm. The estimate calculations were refined by observation and trial and error to determine which values ultimately generated the most natural postures. For proximal approximations the torso rotations are minimal, while distal goal coordinates induce more severe body contortions.

The initial orientation of the torso is determined by computing Euler angles in terms of the abdomen coordinate frame. Let us refer to the forward bend, sideways bend, and twist of the torso as g, l, and d respectively. The computed sequence of rotations first considers g, then l, and lastly d. d is computed to position the torso facing the goal. We compute g and l to minimize the distance between the base of the neck and the goal coordinates. The initial forward bend rotation g is computed with respect to the modified goal coordinates G' in Figure 5.12. G' is calculated by rotating the original goal coordinates G about the y-axis to align it along the z-axis.

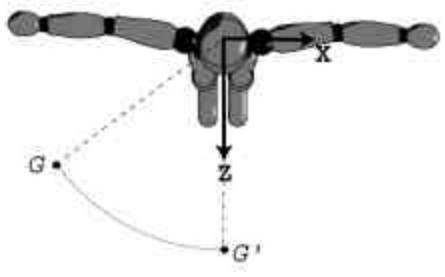

Figure 5.12 Transforming G for x-axis torso rotation.

Once the orientation of the abdomen coordinate frame that minimizes the distance between the base of the neck and the goal is computed, we determine the initial position of the chest and abdomen coordinate frames. Let us refer to the optimal computed Euler angles as g', l', and d', which correspond to rotations about the local x-axis, z-axis, and y-axis respectively. Each computed Euler angle is divided by some damping factor d that is unique to the particular ball joint and estimate being considered. Proximal estimates are characterized by high damping factors, while distal estimates have lower values. The puppet's initial abdomen and chest rotations are calculated from the Euler angles g', l', and d' in equation (5.30). Examples of estimated torso postures are shown in Figure 5.10.

$$\Gamma'_{abdomen} = \frac{\Gamma'}{d_{abdomen}}, \text{ and } \Gamma'_{chest} = \frac{\Gamma'}{d_{chest}}, \qquad (5.30)$$

Γ' corresponds to one of the three optimal Euler angles. $\Gamma'_{abdomen}$, Γ'_{chest} represent the estimated abdomen and chest rotation respectively. $d_{abdomen} \geq 2$ and $d_{chest} \geq 2$ are unique for the particular Euler angle and estimate being computed.

The joint limits for the chest and abdomen local z-axis rotation l' is a function of g' and the local y-axis joint limits. Modeling joint limits as a function of the current local orientation is justified by the equality (5.31). Figure 5.13 illustrates (5.31) with respect to the puppet model. The first diagram in Figure 5.13 corresponds to a transformation $RotY(-p)$. The second diagram illustrates a transformation $RotX(p)$. The third posture corresponds to the transformation on either side of the equation (5.31).

$$RotY(-p) \cdot RotX(p) = RotX(p) \cdot RotZ(p). \qquad (5.31)$$

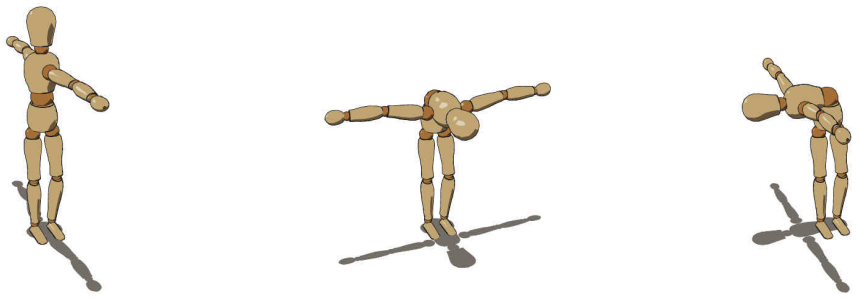

Figure 5.13 Posture transformations from (5.31).

By considering the rotation about the x-axis first, subsequent local z'-axis rotations are equivalent to an x-axis rotation with an initial y-axis component. To achieve the above position, we must consider how the z'-axis rotations are positioning the torso geometry in world space. That is, if the puppet is upright then z'-axis rotations will bend the torso sideways. Humans are not very flexible in this direction, so the upper and lower rotation limit is small. However, if the puppet is already bent forward, then rotations about the z'-axis is equivalent to the puppet first twisting then bending forward, as illustrated in Figure 5.13 and equation (5.32). To overcome this anomaly, we calculate the chest and abdomen z'-axis rotation limits as a function of the initial x-axis rotation. This concept is illustrated in Figure 5.14 and equations (5.32) and (5.33).

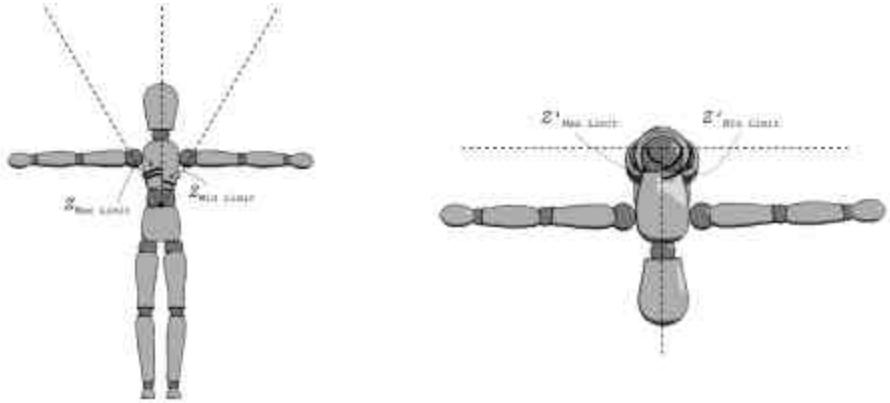

Figure 5.14 Abdomen z-axis rotation limit.

The abdomen and chest local z-axis joint limits are determined by the equation below, where $Y_{Min\ Limit}$, $Y_{Max\ Limit}$, $Z_{Max\ Limit}$ and $Z_{Min\ Limit}$ are the joint limits according to Table 3.1, and f_x is the absolute value of the x-axis local coordinate frame rotation. The below equations assume $f_x \leq \frac{p}{2}$.

$$Z'_{Max\ Limit} = Z_{Max\ Limit} + \frac{2 \cdot f_x}{p} \cdot (Y_{Max\ Limit} - Z_{Max\ Limit}). \quad (5.32)$$

$$Z'_{Min\ Limit} = Z_{Min\ Limit} + \frac{2 \cdot f_x}{p} \cdot (Y_{Min\ Limit} - Z_{Min\ Limit}). \quad (5.33)$$

5.7.3 Weight Schemes

A *weight scheme* is a term for an *n*-dimensional vector, where *n* refers to the number of degrees of freedom in the puppet. The elements correspond to the weight factor of a particular degree of freedom. Weight factors refer to the perceived degree of stiffness of the joint, and are incorporated by the w_i term in the IK solver presented in Section 5.5. w_i refers to the damping weight factor, and should not be confused with the weight factors w_o and w_p that determine the influence of the orientation and positional objective function on the optimal joint state in equation (5.25).

The algorithm presented in Section 5.2 computes each posture. Step 2 is performed three times with distinct weight schemes. The specific weight factors for each joint in each pass is presented in Appendix C. The first weight scheme makes the shoulder z-axis, abdomen z-axis, and chest z-axis degrees of freedom completely stiff. In the second pass of the IK solver, the abdomen and chest z-axis is permitted to deviate slightly from the initial estimate by assigning a very stiff weight factor, while the shoulder z-axis remains completely stiff. The third pass allows both the torso and shoulder z-axis degrees of freedom to deviate from their initial estimated state. These three degrees of freedom can lead to unnatural postures, since the torso will contort unnaturally in achieving an IK solution, while the shoulder z-axis will tend to raise the elbows.

With each pass we loosen the constraints on the degrees of freedom in an effort to find a solution at the expensive of perceived naturalness. In general, the first pass will generate the best posture, while each successive pass deteriorates the naturalness of the pose. The first scheme may not find a solution as readily as successive schemes since certain joints not permitted to rotate. However, under certain circumstances the second or third pass may generate better postures than the first pass. It is not obvious which pass generates the best posture for arbitrary initial and goal positions. For this reason, multiple passes will be computed even if an earlier, more constrained weight scheme successfully finds a solution. The resulting posture of each pass is evaluated by the score function in Section 5.7.5. Computing multiple passes is computationally expensive, but we benefit from a more robust IK solver.

In selecting a weight scheme, we must consider the properties of the IK algorithm itself and the desired postural characteristics derived from ergonomics and neurophysiology literature. The nature of the IK algorithm is conducive to slowly iterating towards the goal, since the algorithm is biased towards distal links. Certain anomalies and unusual contortions of the body can also arise since our algorithm considers each joint independently. These problems can be overcome by a weight scheme that slowly converges on a solution.

Incorporating postural preferences such as minimal torso rotations and limiting the height of the elbow is not trivial. Simply applying a narrow range of motion or a low weight factor to a particular degree of freedom may result in severe contortions in other areas to compensate for the joint's limited movement. Furthermore, our ultimate goal is to find a solution to the IK problem. Any solution is preferred over no solution, particularly for end effector goals that are "hard to reach". A weight scheme should be conducive to natural postures, but not at the expense of a solution.

Figure 5.15 illustrates reaching tasks with and without our implemented weight scheme. The postures in the first column were scored the best from the three weight schemes in Appendix C. The second column illustrates a naïve computation where the all degrees of freedom were assigned a weight factor $w_i = 1$ from equation (5.24). All other parameters in computing the four postures in Figure 5.15 remained constant. Raised elbows, severe torso contortions, and local minima are typical with the naïve weight scheme.

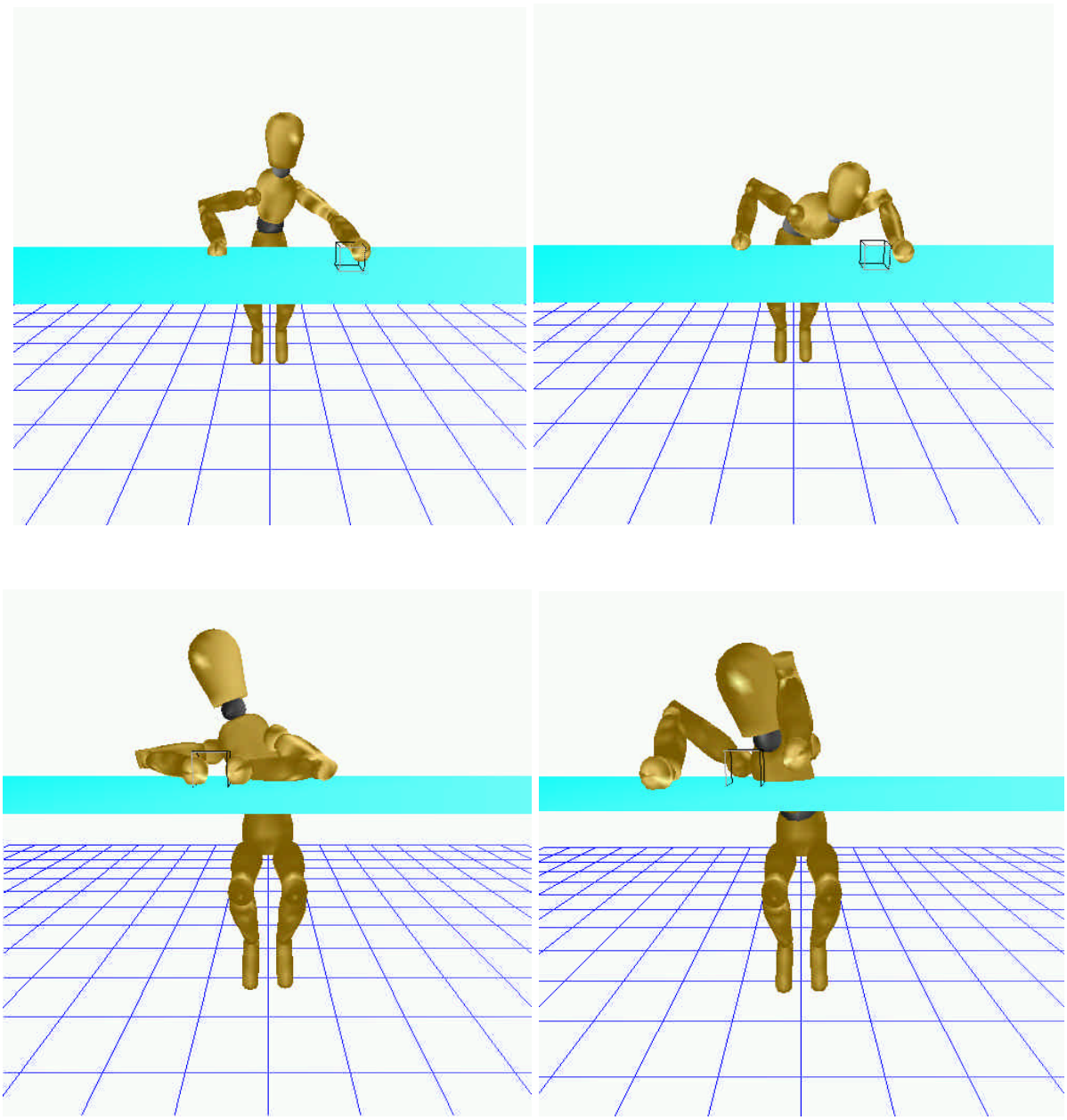

Figure 5.15 Comparison of our weight scheme (left) with a naïve calculation (right).

5.7.4 Score Functions

The quality of the puppet's posture is evaluated by a score function that considers the naturalness of the pose, and whether or not the pose solves the particular IK problem. After each pass, the posture generated is given a score based on certain criteria from ergonomics literature surveyed in Section 2.9. The score function is novel in the sense that we attempt to quantify the "naturalness" of a pose. The score function is derived from experimental observation, intuition, and trial and error.

The score function considers certain properties and degrees of freedom when assessing the quality of a posture. Each kinematic chain is evaluated with a distinct score function, since there is a unique set of body segments associated with every chain. We first summarize the criteria by which we evaluate each posture. A discussion of the issues and justification for evaluated postures based on these criteria is presented. The score function as it applies to each kinematic chain is then given.

Each posture is evaluated in terms of the distance between the end effector and the goal, the height of the elbow with respect to the wrist in world coordinates, and the state of specific degrees of freedom. The posture is assigned a score in each of the above criteria, and the final posture score is the sum of these scores.

The distance between the end effector and goal indicates whether or not a solution has been found. If the end effector is not within some error threshold of the goal, then a very bad score is assigned for the criteria. In our implementation, postures that do not solve the IK will certainly be rejected unless all other attempts to reach the goal have failed.

The height of the elbow is important in the perceived naturalness of the pose. Our score function is very biased against postures where the elbow is higher than the wrist. Such positions are highly unnatural and should be avoided unless absolutely necessary. We measure the vertical distance between the elbow and the wrist center of rotation in world coordinates. If the elbow is higher than the wrist, the vertical distance between the joints is multiplied by a large scalar to ensure the resulting posture is scored poorly. This measurement is illustrated in Figure 5.16.

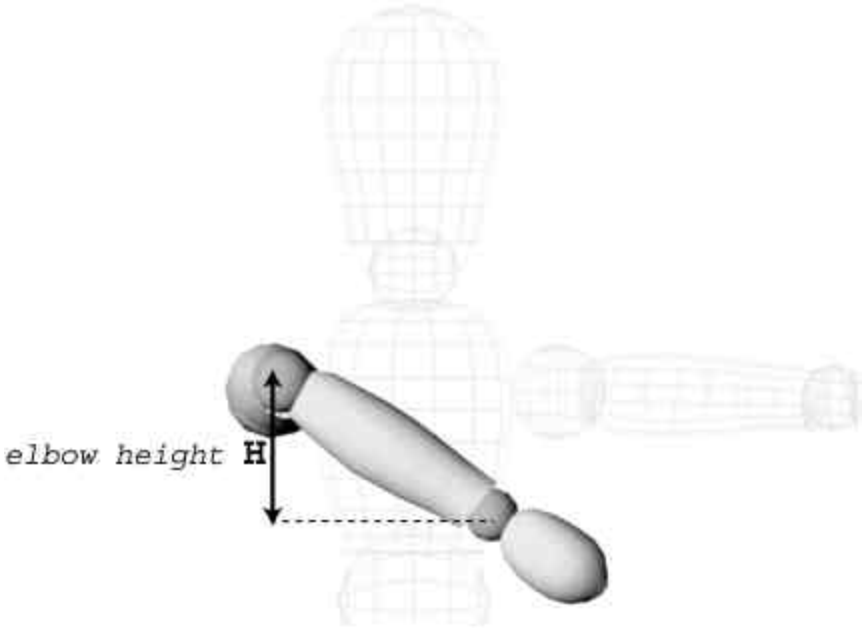

Figure 5.16 Calculating the height of the elbow.

The x-axis and z-axis of the shoulder, abdomen, and chest are critical to the naturalness of the posture. As these joints deviate from their centre of rotation, the perceived naturalness of the pose degenerates quickly. To measure the extent to which the joint degenerates the naturalness of the posture our score function measures the ratio between the current joint state and the respective joint limit, and the angular difference between the initial estimated joint state and the final computed rotation. The ratio is calculated as $r = \Phi / U_\Phi$ if $\Phi > 0$ and $r = \Phi / L_\Phi$ if $\Phi \leq 0$, where Φ is the current state of the joint and U_Φ, L_Φ is the maximum and minimum joint rotation respectively. The angular difference is calculated by $\Delta\Phi = \left|\Phi - \Phi_{init}\right|$, where Φ is the current state of the joint and Φ_{init} is the initial estimated position from the algorithm presented in Section 5.7.1 and 5.7.2. The naturalness of an individual joint's state is determined by the sum $r + \Delta\Phi$.

The ratio gives an indication of how close the joint rotation is to the joint limit. Joint rotations close to the respective limit are considered uncomfortable positions. The ratio term favors joint positions in the middle of its range of motion, assuming the joint limits extend equally in the positive and negative direction. Practically this is not necessarily true, particularly with the shoulder where humans comfortably function close to the minimum z-

rotation joint limit. The ratio term can also be perceived as favoring postures close to the zero. This implies that the zero posture embodies our notion of comfort, which may not necessarily be true. The puppet's zero posture presented in Figure 3.10 is unnatural only with respect to the shoulder Z-rotation. The arms extend outward, which is positioned in the middle of the range of motion although it is not a natural position in the sense that one would rarely perceive a human performing a reaching task in this way.

$\Delta\Phi$ measures the final joint state's deviation from the estimated position. Our assumption is that the estimated joint position is optimal in terms of comfort and perceived naturalness since it is derived from neurophysiological and ergonomic data. This assumption implies that the more the joint departs from the estimated posture, the more unnatural the posture will appear.

The naturalness of a joint's position is the sum of the ratio r and the deviated angle $\Delta\Phi$ expressed in radians. This implies that reaching the joint's rotational limit is equally as bad as deviating from the estimated rotation by 1 radian. The two values are not mutually exclusive, since deviating from the estimated posture may bring the joint close its limit.

Since the kinematic chains introduced in the introduction of Section 5.7 include variable number of end effectors and body segments, the above scoring criteria must be tailored to accommodate each case. The score associated with the distance between the end effector and goal is denoted D. The score associated with the vertical distance between the elbow and wrist is denoted H. The value of $r + \Delta\Phi$ for some joint i is denoted $(i)_{r+\Delta\Phi}$. The score function for the kinematic chain composed of a single arm, single arm and torso, and two arms and torso is denoted $f_{SingleArm}$, $f_{ArmTorso}$, $f_{2\,ArmTorso}$. The score functions are defined below.

$$f_{SingleArm} = D + H + (Shoulder\ \ x-axis)_{r+\Delta\Phi} + (Shoulder\ \ z-axis)_{r+\Delta\Phi} \tag{5.36}$$

$$\begin{aligned} f_{ArmTorso} = f_{SingleArm} &+ (Chest\ \ x-axis)_{r+\Delta\Phi} + (Chest\ \ z-axis)_{r+\Delta\Phi} \\ &+ (Abdomen\ \ x-axis)_{r+\Delta\Phi} + (Abdomen\ \ z-axis)_{r+\Delta\Phi} \end{aligned} \tag{5.37}$$

$$\begin{aligned} f_{2ArmTorso} = \frac{(f_{SingleArm,Left} + f_{SingleArm,Right})}{2} &+ (Chest\ x-axis)_{r+\Delta\Phi} + (Chest\ \ z-axis)_{r+\Delta\Phi} \\ &+ (Abdomen\ x-axis)_{r+\Delta\Phi} + (Abdomen\ z-axis)_{r+\Delta\Phi} \end{aligned} \tag{5.38}$$

5.7.5 Distributing Iterations

As mentioned in Section 2.9, we would like to restrict the torso rotations in favor of arm rotations when computing postures. Section 5.7.4 describes how the IK solver attempts to accomplish this by setting appropriate weight factors to favor certain degrees of freedom over others. However, we can also allow the arms to iterate more often than the torso. For example, assume we would like to reach for an object using the torso and one arm. We can iterate the arm joints to position the end effector close to the goal coordinates prior to considering the arm and torso. By iterating the arm more frequently than the torso, we can ensure the arm will account for the majority of the end effector's displacement.

In addition to ordering the iterations to favour certain body segments over others, we must also consider the value of w_o and w_p introduced in equation (5.24). As the IK solver iterates each joint towards a solution, we must consider whether the joint will optimize the orientation objective function, positional objective function, or both simultaneously.

Each of the three inverse kinematics problems is solved with the same general algorithm. Naturally, the algorithm is tailored to accommodate the distinct set of body segments included in the kinematic chain. The general algorithm proceeds by first positioning the end effector at the goal coordinates. We then attempt to incorporate the orientation goals by dedicating wrist rotations entirely to maximizing the orientation objective function. The elbow and shoulder will simultaneously minimize the positional objective function to correct the positional deviations incurred by the wrist. In practice, this method will generally find a solution that both maximizes and minimizes the orientation and position objective functions respectively. The general algorithm is presented below.

When the torso is shared between two reaching tasks, it is given an intermediate position that favors neither goal. This is accomplished in step 5 by moving the torso according to one task’s position goal for several iterations, then rotating with respect to the second task’s position goal for several iterations. The result is that the torso will converge on a position in between the two goals, and neither task is given priority. If no solution is found, then the IK solver will attempt to dedicate the torso entirely towards one reaching task, while a single arm will be responsible for accomplishing the second task. Postures sharing the torso between two tasks is illustrated in Figures 5.9, 5.15, 5.19, 7.8, and 7.9.

For 3 passes

Step 1) Load the joints' corresponding weight scheme (per Section 5.7.3)

Step 2) Estimate initial posture (per Section 5.7)

Step 3) Set $w_o = 0$, $w_p = 1$.

Step 4) Iterate the puppet's arm(s) for n_1 iterations (per Section 5.3)

Step 5) Iterate the whole kinematic chain for n_2 iterations (per Section 5.3)

Step 6) Set $w_o = 1, w_p = 0$ for the wrist joint. Set $w_o = 0$, $w_p = 1$ for the elbow and shoulder.

Step 7) Iterate the puppet's arm(s) for n_1 iterations (per Section 5.3)

Step 8) Score the resulting posture (per Section 5.7.4)

Output best posture

Figure 5.17 Posture generator algorithm.

In practice, we found that $n_1 = 500$ and $n_2 = 40000$ solve the problem consistently with realistic postures. This process is quite robust in practice, and converges quickly on a configuration that satisfies both positional and orientation constraints in several hundred iterations. The disadvantage of dedicating the wrist entirely to satisfying orientation constraints is that the joint may be bent severely in some circumstances. This situation does not arise often, but is more common with awkward reaching tasks that require the puppet to stretch far to the side or close to the body. In these circumstances, the orientation of the forearm may differ significantly from the goal orientation. One can notice this anomaly in Figure 7.8. It should be noted that it is not always unnatural for the wrist to come close its range of motion limits when trying to satisfy some orientation goal, as observed in Figure 5.19.

The problem is considered solved when the end effector is within some acceptable positional error. Step 4, 5, and 7 are considered complete once the end effector position is within some acceptable distance from the goal coordinates, or when the number of iterations exceeds the maximum permitted. For our purposes an acceptable error threshold is 0.1% of the height of the puppet model. For standard IK problems, fewer than a thousand iterations is

typically necessary. Over twenty thousand iterations may be required for more difficult problems that require the puppet to stretch for the goal or that generates frequent collisions. The number of iterations required to solve a problem also depends on how close the estimated posture positions the end effector to the goal.

Figure 5.18 illustrates three postures computed by each of the three passes in Figure 5.17. Clockwise from the top left corresponds to the first, second, and third pass of the algorithm. The top left posture was scored the best among the three possibilities. Figure 5.19 shows a human performing simple reaching tasks. The posture generator computed postures for similar tasks, which are presented beside the photos in Figure 5.19.

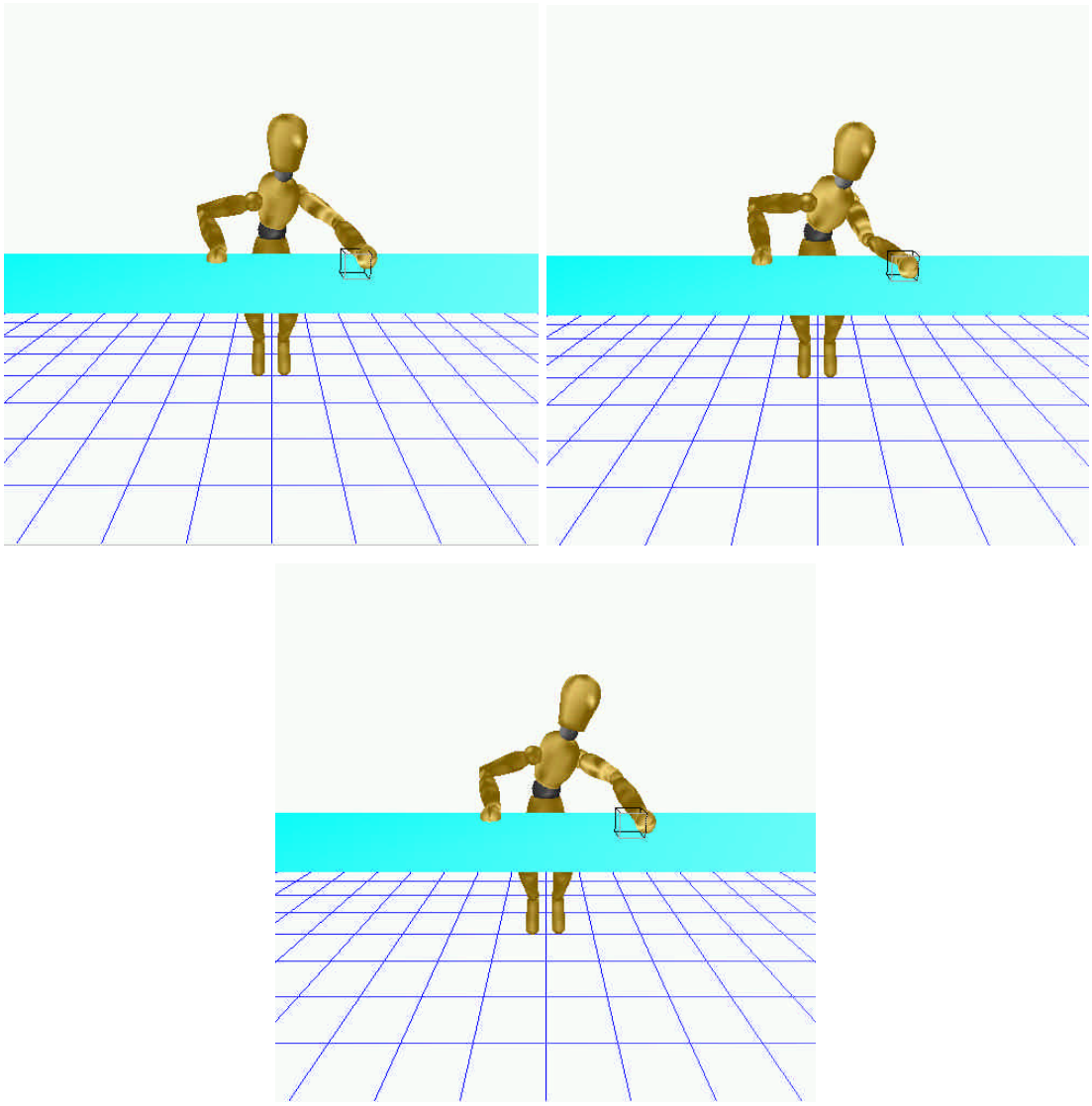

Figure 5.18 Postures generated by each of the three passes in Figure 5.17.

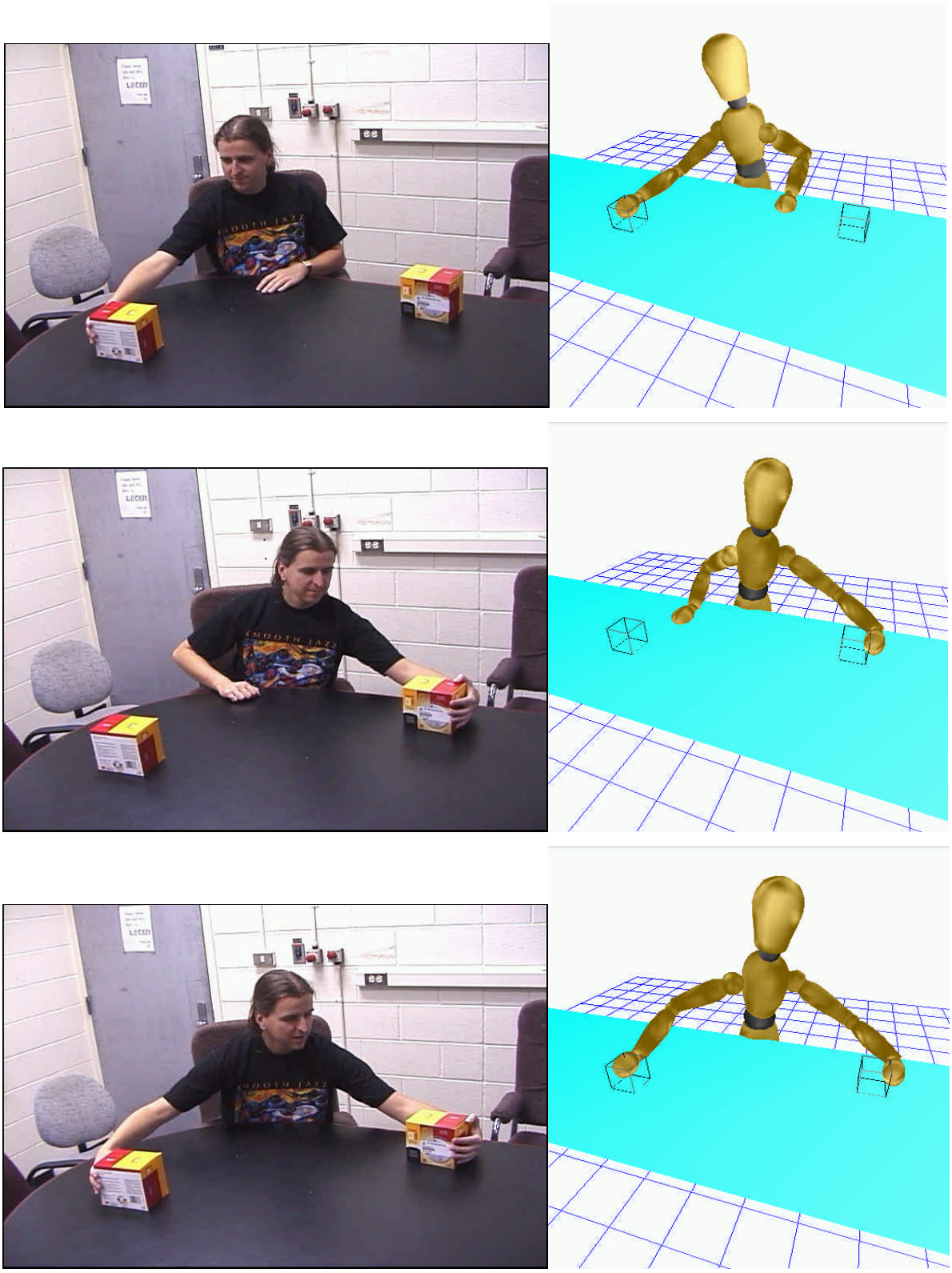

Figure 5.19 Human and puppet performing similar tasks.

5.8 Summary

This chapter described our system's posture generator module, which is responsible for positioning the puppet with human realism given certain end effector constraints from the motion scheduler. A robust posture generator is important to guided control to relieve the animator of specifying reasonable joint rotations for every task. The algorithms and heuristics described in this chapter attempt to model how humans position their bodies for arbitrary reaching motions. Sections 5.1, 5.2, and 5.3 give the background for computing three-dimensional inverse kinematics. Section 5.6 describes how external geometric constraints can be incorporated in the existing posture design model. Section 5.5 describes some methods and issues in generating realistic postures for humanoid articulated figures.

Chapter 6

Motion Interface

The motion interface is responsible for converting the user's input into geometric constraints. The user inputs a higher level motion command, which the motion interface will interpret according to the state of the puppet and the environment. For every task, the position and orientation of both hands must be resolved. When the user specifies a task for a particular hand, the hand is referred to as the *primary* hand since its position and orientation goals accomplish the *primary task* as defined in Section 3.6.3. The other hand, referred to as the *secondary* hand, will be assigned a secondary task autonomously by the motion interface. The objective of the motion interface is to interpret the user input and determine an appropriate position and orientation of the primary and secondary hand. Similar work on interpreting and resolving higher-level tasks to lower level goals was done at the University of Pennsylvania [BPW93][LB94].

Once both hands are assigned geometric goals, a motion model is created to encapsulate the user's intentions. The motion model includes posture constraints as well as the timing, velocity, and scheduling parameters of the task introduced in Section 3.3. The contents of the motion model are dependent on the type of motion building block. Reaching, sliding, and general tasks include data specific to their execution. The motion model is then

placed in the motion queue and processed by the motion scheduler described in Chapter 4. If the motion performed by the puppet modifies the environment's state, then the environment is updated.

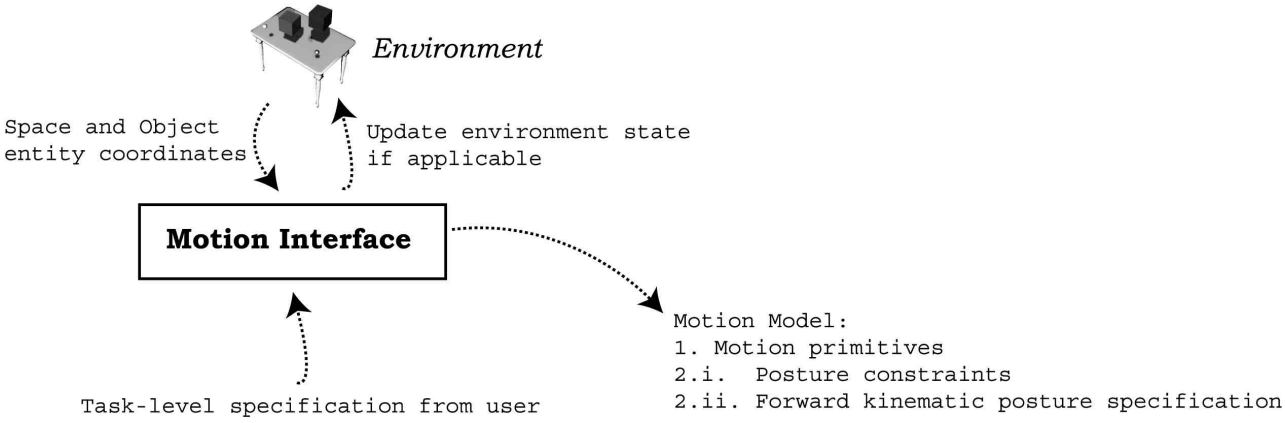

Figure 6.1 Functional diagram of the motion interface.

The orientation and position constraints are resolved depending on whether the task is a reaching, sliding, or general task. Determining primary hand constraints for the three motion building blocks is described in Section 6.1. Secondary goal resolution is introduced in Section 6.2. Section 6.3 describes how and when the motion interface updates the state of the environment. There are advantages and limitations with our proposed way of specifying motion with respect to objects and space. These issues are discussed in Section 6.4. Finally, some anomalies arise in the motion interface's interpretation from Section 6.3. Section 6.5 describes the solutions implemented to handle such cases.

6.1 Primary Hand Constraints

The task specified by user input dictates the motion of the primary hand. The motion interface will determine the geometric constraints that generate a posture to accomplish the input task. The input parameters required to infer these geometric constraints are dependent on the specific motion building block; these are introduced in Section 3.6.3. The geometric constraints specified by the motion interface are simply position and orientation goals for the

particular hand performing the task. The following subsections describe each motion building block in terms of its distinct input parameters and the geometric constraints imposed by the motion interface. Specifying these input parameters is presented in Chapter 7.

6.1.1 Reaching Motions

Reaching motions allow a user to position the hand relative to some object or space entity in the environment. Properties of space and object entities are introduced in Section 3.4. The position and orientation goals of the end effector are determined by the user specified *grip* parameter. Our current implementation considers grasping the top, side, or bottom surface of the object's cubic approximation. A grip's corresponding orientation and position depends on whether the right or left hand is performing the motion. The following table maps the input parameters to end effector constraints. Satisfying these constraints will accomplish the reaching task specified by the user. In the following table, O_X, O_Y, O_Z is the centre of the object volume and O_W is the volume width. Orientation is specified about the (Z, Y, X) axis respectively.

	Side	Top	Bottom
Left Hand	Position: $(O_X + O_W,\ O_Y,\ O_Z)$ Orientation: (0, -90, -90)	Position: $(O_X,\ O_Y + O_W,\ O_Z)$ Orientation: (0, -90, 0)	Position: $(O_X,\ O_Y - O_W,\ O_Z)$ Orientation: (0, -90, -180)
Right Hand	Position: $(O_X - O_W,\ O_Y,\ O_Z)$ Orientation: (0, 90, 90)	Position: $(O_X,\ O_Y + O_W,\ O_Z)$ Orientation: (0, 90, 0)	Position: $(O_X,\ O_Y - O_W,\ O_Z)$ Orientation: (0, 90, 180)

Table 6.1 Mapping orientation and position goal for object reaching tasks.

Space reaching motions position the hand at some space entity. Similar to grasping objects, the hand can be placed at the space entity with the palm facing downward, on the side, or

facing upward. Independent of whether the left or right hand is performing the motion, the tip of the hand is positioned at the corresponding point in space. In the following table, S_X, S_Y, S_Z is the position of the space entity in world coordinates. Orientation is specified about the (Z, Y, X) axis respectively.

	Side	Top	Bottom
Left Hand	Position: (S_X, S_Y, S_Z) Orientation: (0, -90, -90)	Position: (S_X, S_Y, S_Z) Orientation: (0, -90, 0)	Position: (S_X, S_Y, S_Z) Orientation: (0, -90, -180)
Right Hand	Position: (S_X, S_Y, S_Z) Orientation: (0, 90, 90)	Position: (S_X, S_Y, S_Z) Orientation: (0, 90, 0)	Position: (S_X, S_Y, S_Z) Orientation: (0, 90, 180)

Table 6.2 Mapping orientation and position goals for space reaching tasks.

Figure 6.2 Interpreting reaching tasks.

6.1.2 Sliding Motions

Sliding motions position the hand relative to its current position in space. The specification of position and orientation is not relative to any particular object or space entity. Instead, the user explicitly specifies the goal orientation and displacement of the hand. In addition to providing lower-level control over the puppet's motion, slides allow the user to specify end effector goals with respect to its current position, rather than fixed points in space. The end effector positional constraint computed by the motion interface is summarized by $(x, y, z) \leftarrow (x_0 + \Delta x\ ,\ y_0 + \Delta y\ ,\ z_0 + \Delta z)$, where (x, y, z) is the goal end effector position,

(x_0, y_0, z_0) is the current end effector coordinates, and $(\Delta x, \Delta y, \Delta z)$ is the user-specified displacement vector. The orientation is specified by the user-specified local transformations (a, b, g).

Figure 6.3 Interpreting sliding tasks

6.1.3 General Motions

General tasks are executed using canned postures embedded in the motion interface. A forward kinematic specification of a posture that completes the task is output by the motion interface. The IK solver does not compute the posture, so no geometric constraints are specified. The end effector position and orientation is determined by the joint coordinates output by the motion interface.

Figure 6.4 Interpreting general tasks.

6.2 Secondary Hand Constraints

The primary hand positions specified in Section 6.1 are derived from user input. The position and orientation of the hand not involved in the task is specified by the system, rather than user input. Secondary motion is applied to all body segments that are not critical segments in the user-specified task. This default motion is specified by a posture that placed the secondary hand at some pre-defined position and orientation. Figure 6.5 illustrates how geometric constraints are applied to particular body segments. User input will determine the goals of its critical segments, while the system determines the secondary motion assigned to the non-critical body segments.

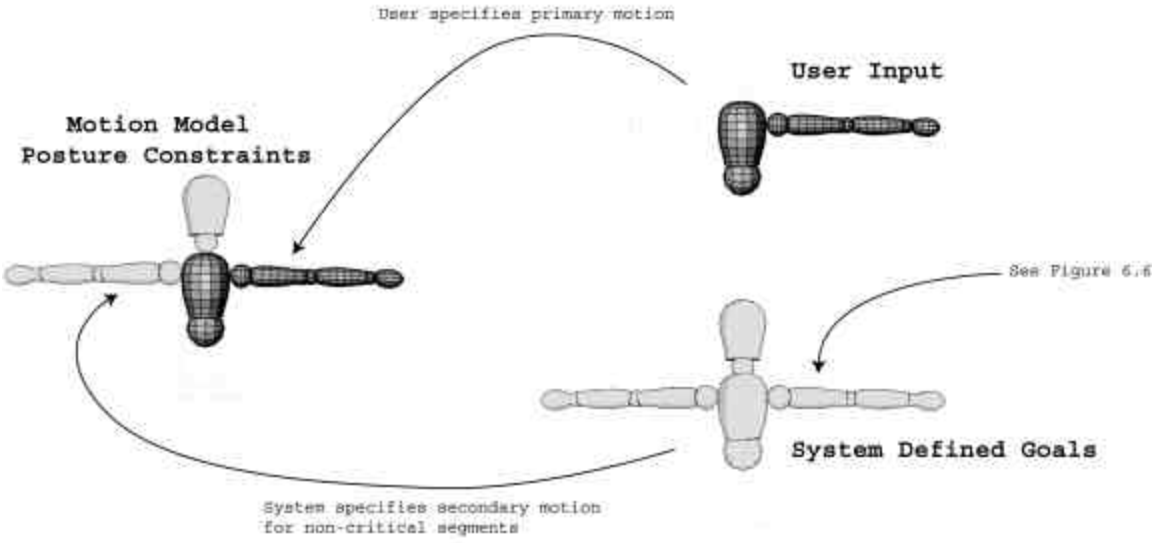

Figure 6.5 Applying primary and secondary motion to a task.

In our prototype the head default position is to observe the primary hand performing its task. The secondary hand is placed flat close to the table's edge. The user can also override the default end effector constraints for the secondary hand. This is accomplished with *posture locks* introduced below.

When a puppet completes its task, the primary hand can be locked in its final position. Locks are interactively set and released by the user. Both the left and right hand maintain their own instance of a lock. We refer to these instances as the *left lock* and *right lock* respectively. Posture locks influence the puppet's motion by overriding default secondary constraints. If a lock is set, the secondary hand will assume positional and orientation goals from the lock, rather than the system default values. Once a lock is released, the corresponding hand returns to satisfying default constraints.

The locks specify the secondary hand task in the same level of abstraction as when the lock was set. For example, if the user reaches for some object *A* and sets the lock, the lock will specify the secondary task as *"Reach for object A"* rather than *"Reach for coordinates x,y,z"*. This implies that even if the object moves, the locked hand will maintain its position relative to the object. Locking a sliding motion will fix the end effector at a specific position and orientation. A locked general task will store the joint coordinates of the

canned executed posture. An example of specifying secondary goals while the right lock is set with a reaching motion is presented in Figure 6.6.

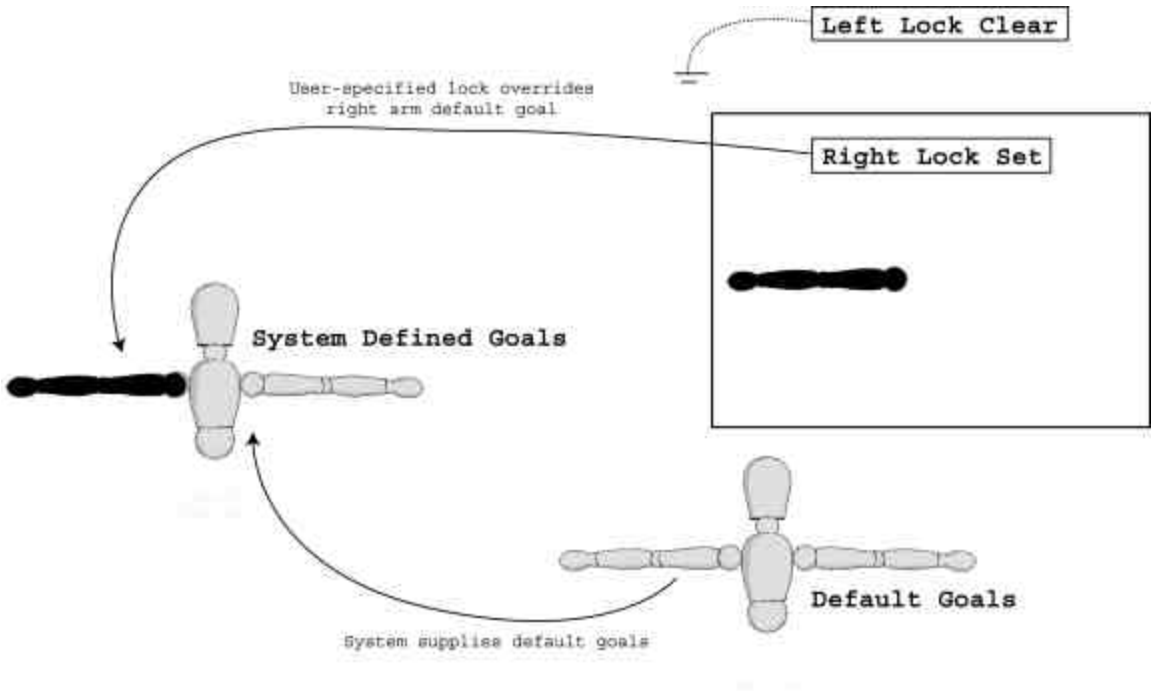

Figure 6.6 Overriding default secondary goals with locks.

Assume the puppet is executing a left-handed task as in Figure 6.5, and the right-hand is among the group of non-critical segments. Let the right lock be set by some previously executed task as in Figure 6.6. In this case, the left hand is positioned according to the user's input in Figure 6.5 and the right hand is positioned according to the right lock in Figure 6.6. If the right lock were not set, then the right hand would be scheduled default secondary goals. These two scenarios are illustrated below. The left column summarizes the user and system-defined parameters input to the motion interface, and the right column shows the resulting output.

System State	Posture Goals
User Input: Reach object B with left hand Left Lock Frame: Empty Right Lock Frame: Reach space A Default Secondary Motion: Place hand on table	Left hand task: Reach object B Right hand task: Reach space A

Table 6.3 Overriding default secondary goals.

System State	Posture Goals
User Input: Reach object B with left hand Left Lock Frame: Empty Right Lock Frame: Empty Default Secondary Motion: Place hand on table	Left hand task: Reach object B Right hand task: Place hand on table

Table 6.4 Assigning default secondary goals.

If the user specifies a task for a hand while its corresponding lock is set, then the lock is ignored and does not override the primary motion. This concept is illustrated in Table 6.5. Under some circumstances, specifying motion for a locked hand can update the state of the environment, and is discussed in Section 6.3.

System State	Posture Goals
User Input: Reach object B with right hand Left Lock Frame: Empty Right Lock Frame: Reach space A Default Secondary Motion: Place hand on table	Left hand task: Place hand on table Right hand task: Reach object B

Table 6.5 Ignoring lock because of user input.

6.3 Modifying the Environment

The user can specify motion that modifies the position of object and space entities in the environment. Objects and space are introduced in Section 3.4. Certain space entities, referred to as *dynamic space*, can be repositioned by performing sliding motions while *static space* has fixed coordinates. There are two dynamic space entities corresponding to the left and right hand. Sliding motions performed by the respective hand can control the position of these entities. For example, if the puppet performs a sliding motion with the left hand, then the left hand's dynamic space entity is placed at the new end effector position. Updating the position of dynamic space is illustrated in Figure 6.7.

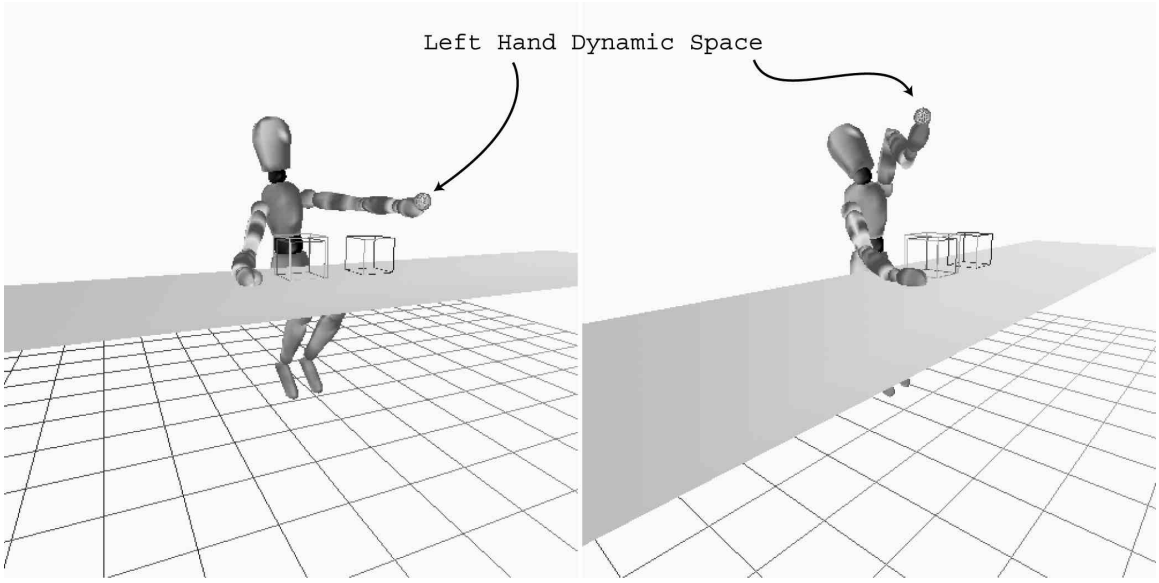

Figure 6.7 Sliding task moving the left hand dynamic space entity.

The dynamic space entities can be referenced the same as static space. This implies objects or the opposite hand can be positioned at dynamic space, which results in rich variety of multi-limb cooperative tasks. While static space entities resolve end effector goals to a pre-defined point in space, dynamic entities determine end effector goals relative to the puppet's current hand position. For example, the user can interactively specify a motion that positions the palm upward, then place an object in the hand. An example of this motion is illustrated by the second image in the third row of Figure 7.9.

To move an object the user must first command the puppet to grasp it. Performing an object reaching motion, then locking the task to the respective hand will be interpreted by the system as a *grasp*. Once the object is grasped, subsequent reaching motions will be interpreted as a command to move the object. If the puppet reaches for a space entity while grasping an object with the same hand, then the object will be placed at the specified space entity. An example of moving some object *A* to space entity *B* is presented in Table 6.6.

System State	Posture Goals
User Input: Reach space B with right hand Left Lock Frame: Empty Right Lock Frame: Reach object A Default Secondary Motion: Place hand on table	Left hand task: Place hand on table Right hand task: Move object A to space B

Table 6.6 Moving objects.

If the user commands a puppet to reach for an object after performing a grasp with the same hand, then the motion interface will interpret the input as a stacking motion. The following series of commands will stack object *A* on object *B*.

System State	Posture Goals
User Input: Reach object B with right hand Left Lock Frame: Empty Right Lock Frame: Reach object A Default Secondary Motion: Place hand on table	Left hand task: Place hand on table Right hand task: Stack object A on object B

Table 6.7 Stacking objects.

If an object is moved to a space occupied by one or more objects, then the object is placed at the topmost position in the stack. If an object A is to be stacked on top of an object B, object A is placed at the topmost position in the stack that includes object B as one of its members. Table 6.8 is a non-exhaustive series of commands identical in their interpretation. The resulting configuration of objects is presented in Figure 6.8.

Case 1	Case 2	Case 3
Move Object A to Space A	Move Object A to Space A	Move Object A to Space A
Move Object B to Space A	Stack Object B on Object A	Move Object B to Space A
Move Object C to Space A	Stack Object C on Object A	Stack Object C on Object B

Table 6.8 Series of commands with identical interpretation.

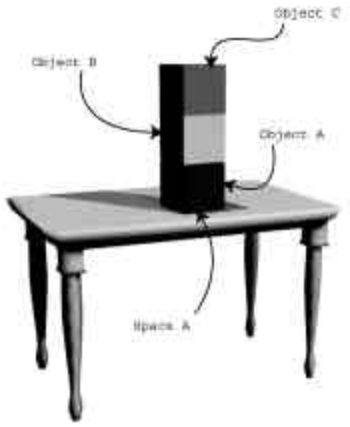

Figure 6.8 Stack of objects.

If the hand's lock frame is set with an object reaching motion, then slide motions are interpreted as sliding the object. For example, if the left hand lock frame is set as *"Reach for object A"*, then sliding the left hand will be interpreted as *"Slide object A"*. The system state for sliding objects is shown in Table 6.9.

System State	Posture Goals
User Input: Slide with right hand Left Lock Frame: Empty Right Lock Frame: Reach object A Default Secondary Motion: Place hand on table	Left hand task: Place hand on table Right hand task: Slide object A

Table 6.9 Sliding objects.

Motions requiring the coordination of both hands are possible given the current model of dynamic space and locks presented in this chapter. Two examples of coordinated motion are moving an object with both hands and placing an object in the hand. The first example is performed by locking both hands on an object in a single stack. Moving the hand locked on the bottom-most object will cause the entire stack of objects to move, and both hands will subsequently move with the stack. This is a result of object locks that position the hand at a particular object independent of its position. An example of moving a stack of objects with both hands is given in Section 7.3.2. A single object is simply a stack with one object, and moving it with both hands is illustrated in Figure 6.9. The state of the system and task specification is presented in Table 6.10. Moving an object to dynamic space, which is positioned at the hand following a slide motion, places an object in the hand. Figure 7.9 illustrates a cup being placed in the palm of the puppet's hand.

The combination of static and dynamic space, pre-defined grips, interactive orientation specification, specifying tasks with respect to objects or space, and posture locks provides a powerful interface for interactive control. The number of motions and tasks that can be expressed in terms of the above control primitives is large. Examples of possible scenarios are mentioned in Section 3.6.1.

System State	Posture Goals
User Input: Reach space B with right hand	
Left Lock Frame: Reach object A	Left hand task: Move object A to space B
Right Lock Frame: Reach object A	Right hand task: Move object A to space B
Default Secondary Motion: Place hand on table	

Table 6.10 Coordinated task execution with both hands.

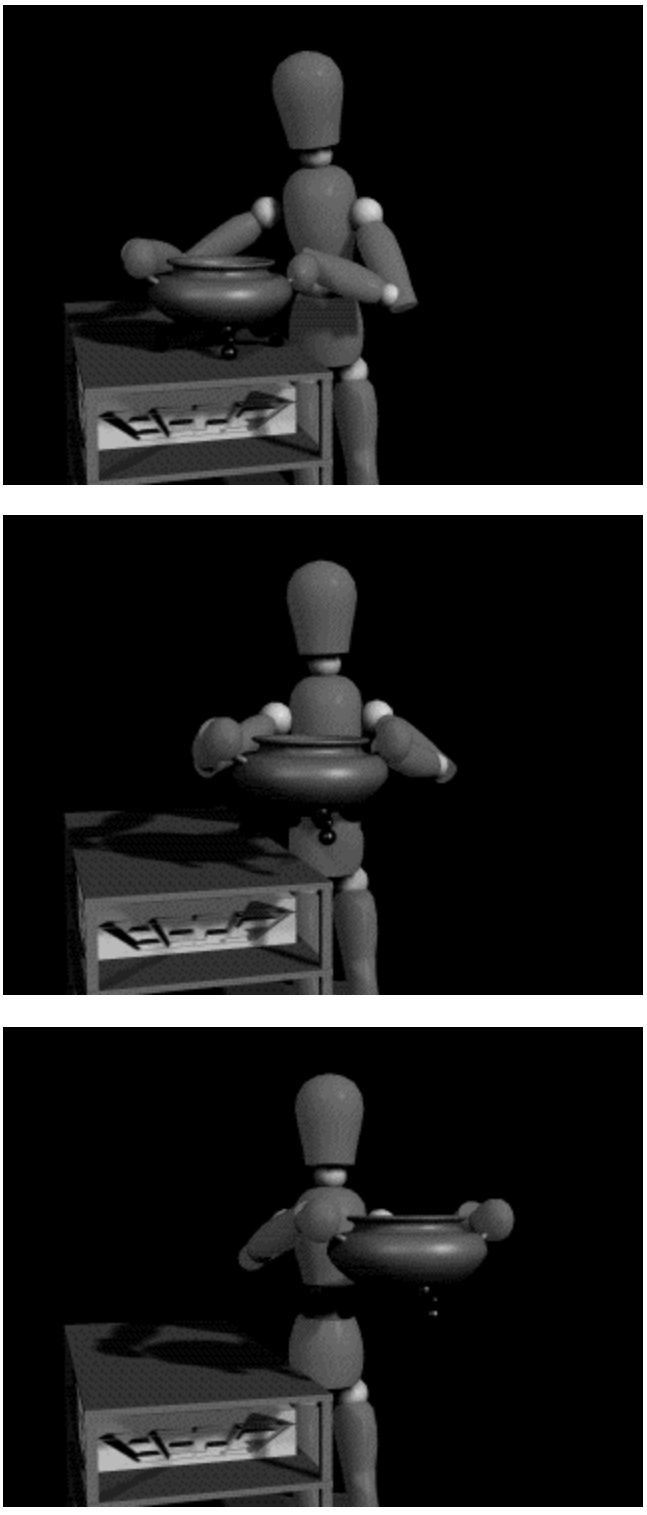

Figure 6.9 Moving a cauldron with both hands.

6.4 Properties of the Interface

Thus far in the discussion, the orientation of the puppet's hands is restricted to one of three pre-defined grips. The puppet is also restricted to positioning its hands and moving objects to predefined points in space. These restrictions are useful for several reasons. It is cumbersome for the user to specify the movement of hands and objects in terms of coordinates in world space. Generating the motion of a puppet stacking three blocks on top of each other is tedious with respect to determining and specifying appropriate end effector coordinates for every task in the animation.

The high-level references for objects and space allow the animator to specify an otherwise complicated sequence of tasks in compact form. Our system allows the user to reference objects and space independent of their position in world coordinates. This is useful when the animator does not know the arrangement of the environment prior to determining the tasks to be executed by the puppet. Furthermore, the position of space and object entities can be changed without modifying the sequence of tasks. This is particularly useful if the animator wishes to rearrange the environment while preserving the sequence of tasks executed. This feature is illustrated in Figure 7.6.

Specifying tasks with respect to objects is a particularly useful component of the interface. If one could only specify tasks with respect to space, then difficulties can arise when modifying a long sequence of tasks. Assume one would like to place a cup on a saucer. To specify the correct task, the animator would have to know which space entity occupies the saucer and input the command *"Move cup to space A"*. If the saucer is moved several times prior to this task, the animator must keep track of the current position of the saucer. Consider a long script of tasks involving a particular object that is moved frequently throughout the animation. Assume the animator would like to modify the initial position of the saucer, or a task that moves the saucer to another point in space. The animator would have to change all tasks in the script affected by the change to accommodate the new position of the saucer.

High-level references for objects and space also allow interactive specification of tasks. Labeling objects and points in space with unique values is conducive to keyboard control over the puppet. Presumably, each object and space entity maps to a particular key on the keyboard. The user can interactively specify an entity associated with a task by pressing the appropriate key in Figure 7.4.

The disadvantage of uniquely referencing entities is that tasks are limited to positioning a single object at one of several points in space. The number of object and space entities that can be effectively referenced by the animator is constrained by the properties of the input device. These issues are discussed briefly in Chapter 2, and cover issues in human-computer interaction. We partially resolve this problem by incorporating sliding motions that can position the end effector anywhere in space. However, specifying these motions are more tedious and cannot be expressed as conveniently as static space. Chapter 7 discusses the modes of input implemented by our system.

Aside from problems in specifying entities related to a task, there are certain types of tasks that cannot be specified using this notation. Tasks such as *"Grab any object"* or *"Push object forward"* ambiguously references objects and points in space. The level of control presented in this thesis requires an unambiguous specification of the entities involved in a particular task. Resolving ambiguous tasks is possible, and would be appropriate future work. Presumably, to incorporate such tasks the interface would reformulate the command in terms of specific entities in the environment. When the user inputs a task such as *"Grab closest object"*, the interface would compute the closest object, and reformulate the task as *"Grab object C"*, where C is the closest object. Likewise, a task such as *"Move any object forward"* would be resolved to *"Move object B to space E"*.

6.5 Special Interpretations

Given the model of interpreting grasping, moving, and stacking objects presented in Section 6.3, there are certain sequences of commands which require special consideration. Each case is presented in the following paragraphs with our proposed interpretation.

If one would like to move a hand to some space entity that is occupied by one or more objects, it is not clear how one should execute the task. One may place the hand on top of the stack, beside the stack, or reject the motion all together. Our implementation will position the hand with a side grip at the bottom-most object in the stack.

Figure 6.10 Placing hand at an occupied space.

There is an issue of how to deal with hand positions that penetrate objects. When objects are organized in a stack, it is not possible to reach for an object with a top or bottom grip without penetrating adjacent objects in the stack. If one hand is locked on object *A* with a top grip, and we try to place another object *B* on top of *A*, then *B* will penetrate the hand geometry. Likewise, if a hand is grasping object *A* with a bottom grip, and we attempt to place it on object *B*, then penetration between the hand and object *B* will occur. This is also the case if a hand is locked at a particular space entity, and an object is moved to the space. To overcome colliding configurations, the system will modify the input or lock to adjust the hand position and avoid penetrating the object. In all circumstances where a hand may penetrate some object *A*, the hand will assume a side grip on object *A*. This concept is shown in Figure 7.8.

Another scenario requiring special attention is moving objects within the same stack. If the user requests to stack an object on top of itself, the input is concerned nonsense and ignored. Likewise, if the user requests to stack object *A* on top of object *B*, and object *B* is positioned lower in the same stack, then the task requested has already been achieved and the input is ignored. Finally, assume the user commands object *A* to be stacked on top of object *B*, and *A* is lower than *B* in the same stack. The request can be interpreted as a swapping task, where objects *A* and *B* trade positions within the stack. This particular motion is not implemented in our system so the input is ignored.

6.6 Summary

This chapter described how higher level tasks can be formulated as posture constraints. The motion scheduler analyses the body segments assigned to a particular constraint to identify motion conflicts. The constraints are later satisfied by the posture generator to position the puppet. Constraints for some body segments were derived from user input, while others were assigned system default values. The default system posture constraints can be overridden by user specified locks. Tasks input for locked hands are interpreted differently than unlocked hands, and results in motion that modifies the environment, such as moving objects and dynamic space entities. The range of potential interpretations may also result in non-intuitive, redundant, or complex motion. These cases were handled in our system by choosing one interpretation over another when no significant advantage existed among the set of possible interpretations, or by simply ignoring the input.

Chapter 7

Results

This chapter describes animations resulting from our prototype application. The principal features of our system are guided-level control of the puppet, which provides a rich variety of motions at the user's disposal, and autonomous interpretation and execution of user-specified tasks. Tasks and primitive motion specifications mapped to standard input devices is presented. Three animations demonstrate the principle features of our work, and give examples of our system’s capability. The first scenario demonstrates the puppet reaching and stacking blocks in variable environments. The second scenario demonstrates our system’s autonomous interpretation of the user’s commands. The third scenario incorporates a variety of objects with specific manipulation motions associated with them.

Section 7.1 describes the modes of input and output of our prototype system. Section 7.2 presents a sample mapping of tasks and motion primitives to keystrokes. This mapping was used to generate the three animations discussed in Section 7.3.

7.1 Input and Output

The user interacts with our system at several levels. This section describes the modes of user interaction in terms of how and when input is specified, and the output produced. The advantages and disadvantages of various modes are briefly discussed, while subsequent sections describe the semantics of the input.

7.1.1 User Input

The user specifies the state of the environment and inputs the motion specifications to the puppet. The state of the environment is characterized by the position of the table, the dimensions and position of the object entities, and the position of the space entities in space. Motion specification includes both the primitives that modify the characteristics of a motion, such as speed and interpolation functions, and the task to be accomplished. The environment and motion models are introduced in Section 3.4 and 3.6 respectively. The motion signal to satisfy these specifications is generated by the system and displayed to the user.

The user specifies the environment by a script. The script includes the position of all space entities in world coordinates. Objects' dimensions are determined by a single value corresponding to the width of the approximating bounding cube as illustrated in Figure 3.3 and 3.6. Each object entity is associated with a space entity to determine its initial space relationship. A sample environment specification is as follows:

```
0.6 0.2 0.0              Table

1- 0.1 0.0 0.2           Space Entities
2- -0.2 0.1 0.15
3- -0.1 0.0 0.1
4- 0.2 0.0 0.05
5- 0.0 0.0 0.15
6- -0.7 0.2 0.9

A- 0.1  1                Object Entities
B- 0.04 4
C- 0.02 2
D- 0.09 6
```

Figure 7.1 Sample environment script.

The first line determines the position of the table in world coordinates. The values correspond to the height, front, and skew of the table respectively. The following six lines specify the position of the space entities with respect to the centre of the front edge of the table. All space entities are specified as an offset from table coordinates so moving the table does not modify the space and objects' position with respect to the table. The approximating cube's width and the space the object initially occupies, which correspond to the first and second value in the environment script, define the state of the objects.

To effectively understand the input mechanisms of the system, it is useful to be familiar with the visual aspects of the system. The application includes three windows on the graphical display. One window displays the puppet and the current state of the environment. The user can observe the motion as it is processed. The user is able to zoom and rotate the position of the camera with respect to the origin of world space.

A Tcl/Tk window provides the user with a command-line interface to modify the value of certain variables. Recall that a sliding motion is defined by the value of a translation vector and an orientation matrix. Three scroll bars correspond to the x, y, and z components of the translation vector. Three other scroll bars correspond to x, y, and z rotations, which defines an orientation matrix. Subsequent slide motions will move according to the values specified by the scrollbars. The user can also select a joint from its corresponding radio button, and directly modify the joint's state. Three other scrollbars can be manipulated to adjust the x, y, and z local rotations of the selected joint. A message is printed at the top of the Tcl/Tk window describing the current task being executed. Examples of messages are, *Reaching for object A* or *Looking at watch*. A third window is a system console that prints computation statistics and error messages. The three windows are presented in clockwise order from the top left in Figure 7.2.

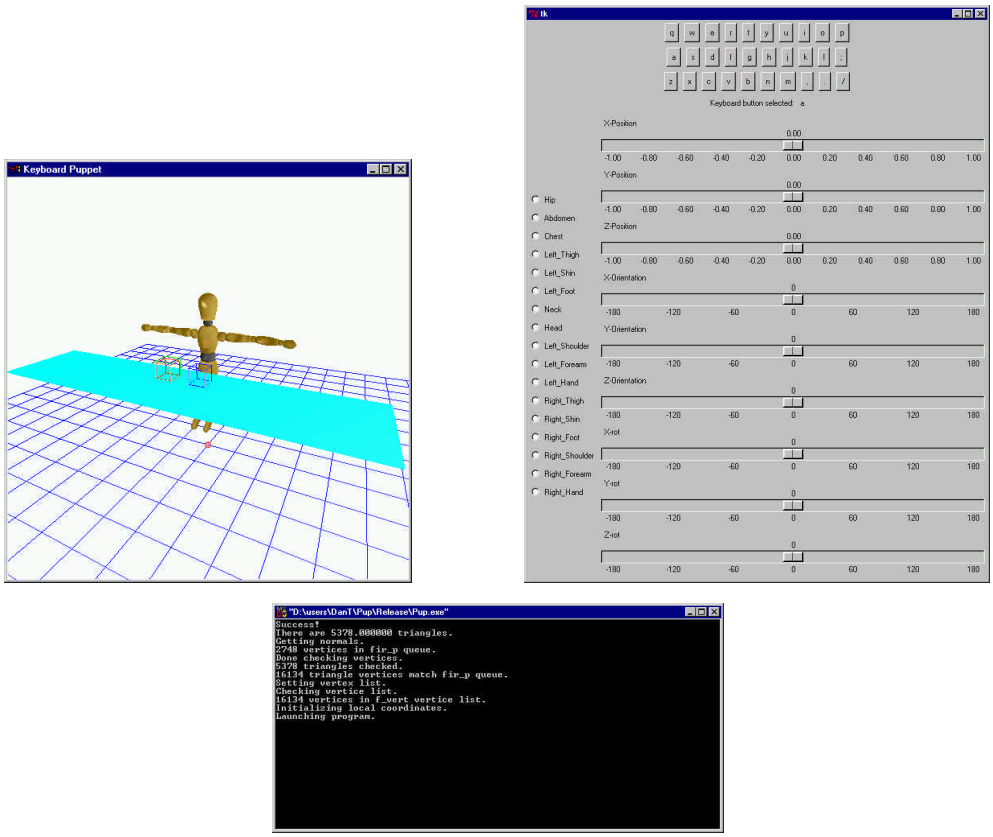

Figure 7.2 Appearance of the prototype application.

The user has several options when specifying tasks and motion primitives. Our system permits the user to input directives by script, keyboard, and mouse. The mouse is limited to specifying sliding motion primitives by adjusting the translation vector and orientation matrix in the Tcl/Tk window.

Keyboard input can specify tasks or modify motion primitive variables. The key presses are processed in real-time. A key press specifying a task will immediately invoke the motion interface and create a motion model to be placed in the motion queue. Motion primitives are updated immediately from keystrokes.

Since keystrokes are processed in real-time, keyboard input allows the user to interactively control the motion of the puppet. The posture computation time is too long to consider the keyboard interaction as real-time. Complicated postures such as grasping two objects at the same time may take several minutes to compute. However, for canned motions

the computation required is minimal, and the user can take advantage of the interactivity to carefully correlate the timing of certain motions.

The user can write a script specifying all tasks and motion primitives over time. The script is input and processed when the program is launched. The user can observe the motion as it transpires on the graphical display. Our script notation simply specifies keystrokes at specific points in time. A sample script is presented below:

```
:nq
92
6))==__G3DL
110
?r
```

Figure 7.3 Sample motion script.

The first line is executed at the start of the animation. Every second line is an integer. This number refers to the time that the next line should be read and executed. In the example presented above, the line "`6))==__G3DL`" will be read and processed at frame 92 of the animation. The sample script above specifies six tasks to be executed over approximately 150 frames. Pauses can be incorporated into the script by scheduling no motion for some period of time. For example, if the motion queue is exhausted at frame 100, and the next task will not be queued until frame 120, then the final animation will have still motion for 20 frames.

The notion of time is expressed as frames. For higher-level task specification, the user will want to specify timing parameters in terms of seconds. This is can be incorporated into any future scripting notation developed for the system. For the purposes of developing movies rather than specifying motion, knowing the duration of certain events in frames rather than seconds is useful. The motion output by our system can be run on a number of commercially available rendering programs at whatever frame rate the animator desires. A time specification in frames is consistent among any group of motion reproductions.

The interactivity of the system can be categorized as a hybrid between real-time specification of tasks and script-based systems. The keyboard interface provides some feedback to the user as the animation progresses. However, the computation time is too long to be categorized as performance animation. One adjustment that provided near real-time interactivity was to eliminate the inverse kinematics from the posture generator all together. Postures were computed simply from the estimate in Section 5.7. As the user entered input

from the keyboard, visual feedback was provided in real-time. A script with identical notation as illustrated above can be generated from the user's input. Hence, the user's input is used to generate a script rather than the final motion. Although the estimated postures do not place the end effector precisely at the goal, the estimate is good enough to give the user a general idea of the final motion. The script generated is then processed in a second pass to generate the final animation. This approach is the most promising to implement real-time interactivity. Optimizing the inverse kinematics to achieve even near real-time posture generation does not seem feasible with our current IK solver.

Sliding motions provide another problem to keyboard interaction. Specifying an orientation and translation component is tedious. The current implementation allows the user to modify these parameters with the mouse or the keyboard. As previously mentioned, the mouse can adjust scrollbars to resolve the parameters. Alternatively, the user can increment or decrement the vector components and axis angles of rotation from keystrokes. This is not very practical for real-time interaction. Our current implementation has several frequently used orientation and translation vectors hard-coded to specific keys. For the majority of sliding motions, this subset of translation and orientation specifications is sufficient. A mapping of keystrokes to motion primitives and tasks is shown in Table 7.1.

7.1.2 System Output

Animation is output in two forms. First, the animation is shown on the graphical display as motion is processed by the system. In addition to the visual feedback from the system, a script file is output with complete forward kinematic specification of the puppet at every time step. The file output is in BVH format, which is a standard file format for recording motion capture data. The script can be input to a commercial rendering package to generate a polished movie.

7.2 Keyboard Mapping

The mapping of task and motion primitive specifications to keystrokes is presented in this section. The mapping of tasks and motion primitives to keys is not presented with the intention of proposing an optimal interface to control the puppet, but rather to illustrate the level of control at the user's disposal. Effectively specifying motion parameters involves issues in human computer interaction, and is considered appropriate future work. The set of parameters that can be specified by the user is the focus of the work, rather than the mode of specification. Figure 7.4 illustrates the keyboard layout of the motion directives in Table 7.1.

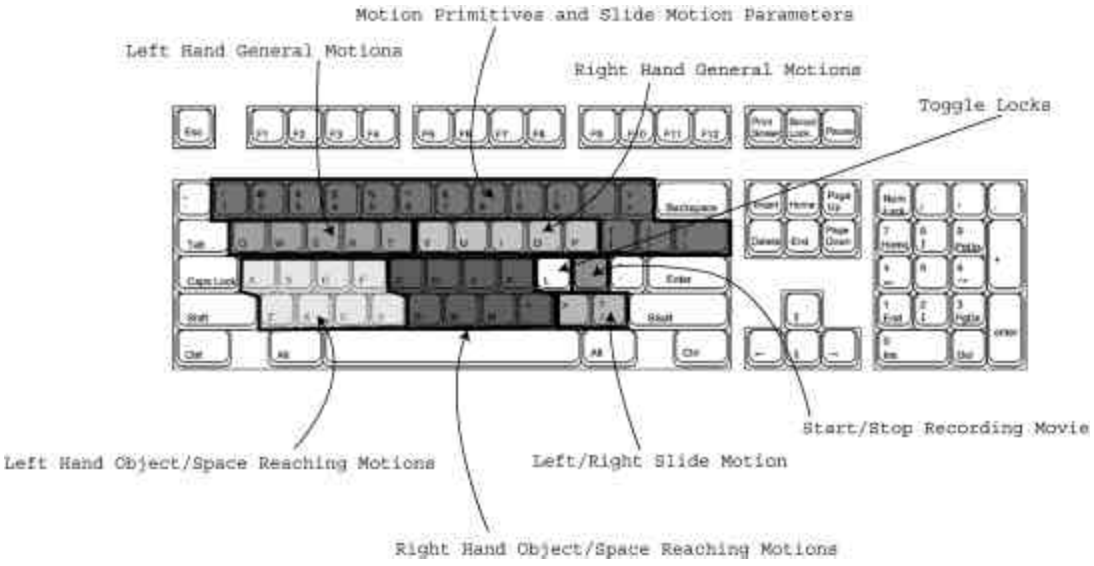

Figure 7.4 Example keyboard layout of user-specified parameters.

The three interpolation functions implemented allow the user to specify stylistic parameters of the executed motion. $func_1$ corresponds to a sinusoidal interpolation function illustrated in Figure 2.9 and 4.18. $func_2$ is a quadratic function, where the velocity of the motion increases linearly over time. The drinking motion in the animation presented in Section 7.3.3 was performed with this interpolation function. Finally, $func_3$ is a square root function that is characterized by high velocity at the beginning, and deceleration for the remainder of the motion.

Keystrokes	Corresponding Task
A,S,D,F,Z,X,C,V	Reach for Object Entities 1,2,3,4,5,6,7,8 with left hand
a,s,d,f,z,x,c,v	Reach for Space Entities 1,2,3,4,5,6,7,8 with left hand
G,H,J,K,B,N,M,<	Reach for Object Entities 1,2,3,4,5,6,7,8 with right hand
g,h,j,k,b,n,m, ,	Reach for Space Entities 1,2,3,4,5,6,7,8 with right hand
> , ?	Slide left/right hand
. , /	Reach for left/right dynamic space entity
: , ;	Start/Stop recording motion
q , y	Scratch head with left/right hand
w , u	Check watch with left/right hand
e , i	Drink from coffee cup with left/right hand
r , o	Pour coffee with left/right hand
t , p	Turn on lamp with left/right hand
1,2,3	Set scheduling parameter to *No overlap*, *Partial overlap*, *Full overlap*
4	Set scheduling scheme to *interrupt mode*
5	Freeze motion
6,7,8	Set grip parameter to *Side*, *Top*, *Bottom*. The slide motion orientation matrix is set to the left hand values in Table 6.1.
0 ,)	Increment/decrement slide motion translation vector x-component by 0.05
- , _	Increment/decrement slide motion translation vector y-component by 0.05
= , +	Increment/decrement slide motion translation vector z-component by

	0.05
[, {	Increment/decrement slide motion orientation matrix x-axis rotation 45 degrees
} ,]	Increment/decrement slide motion orientation matrix y-axis rotation 45 degrees
\ , |	Increment/decrement slide motion orientation matrix z-axis rotation 45 degrees
9	Set slide motion translation vector to (0 , 0 , 0)
(	Set slide motion orientation matrix to identity matrix
l , L	Toggle left/right hand lock
! , @ , # , $, %	Set speed to 20 , 30 , 40 , 50 , 80
^ , & , *	Set interpolation function to $func_1$, $func_2$, $func_3$ (see below)

Table 7.1 Sample key mapping for animations in Section 7.3.

7.3 Animations

Several animations were developed to illustrate the practicality of the system. The animations were generated with the intention of showing how the concepts described in this work satisfy the proposed thesis contributions in Section 1.2. The first animation manipulates blocks to show how the puppet executes tasks in context with the current state of the environment. The second animation shows some of the interpretation and autonomous reasoning about the user's intentions. The third animation shows the rich set of motions that a guided-level control animation system is capable of.

7.3.1 Manipulating Blocks

Our system successfully produced an animation of stacking blocks. The puppet was directed with a simple script presented below. The script is presented below as a natural language translation of the original keystroke specification script, along with the system's hard-coded default secondary motion.

Body Segment	System Default Goals (see Figure 6.6)
Left arm	Place hand on table with top grip
Right arm	Place hand on table with top grip
Torso	Upright
Head	Look at hand executing primary motion

Table 7.2 System default settings for animation #1.

Script for Animation #1
1. Reach yellow block with left hand. Motion Primitives: Grip = top; Speed = 30; Function = $func_1$
2. Lock left hand.
3. Reach blue block with left hand. Motion Primitives: Grip = side; Speed = 60; Function = $func_1$

Table 7.3 Script for animation #1.

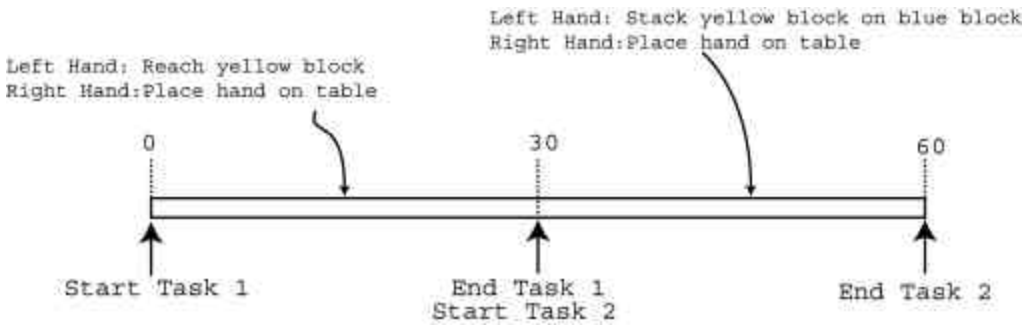

Figure 7.5 Task execution timeline for animation #1.

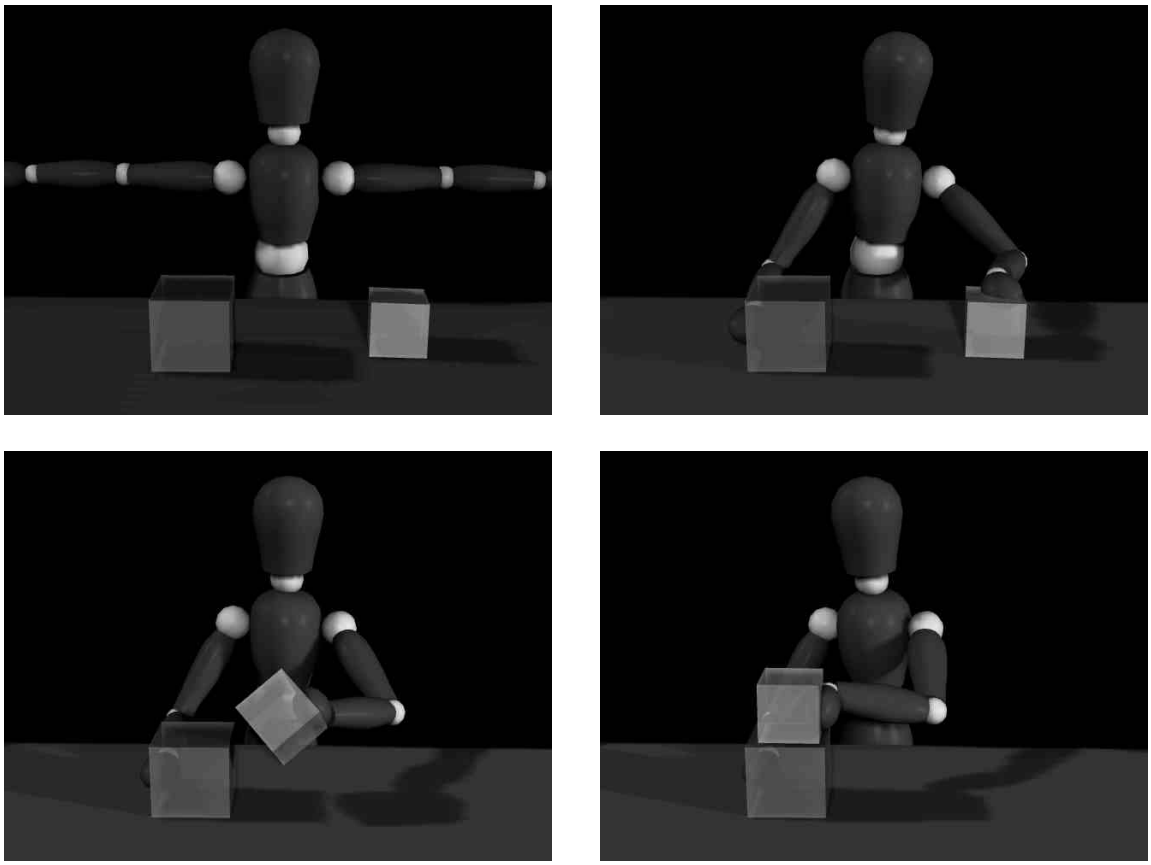

Figure 7.6 Selected frames from animation #1.

This script is used to illustrate the perceived intelligence of the puppet by appropriately executing the tasks according to the current state of the environment. The above script is executed four times to generate four animations. Each animation is generated with a unique environment script. The position and size of the objects is modified, while the motion script remains unaltered.

The resulting four animations illustrates the strengths and deficiencies in our current implementation. The red puppet executes the script with objects of simple dimensions and positions, and is illustrated in Figure 7.6. Images from the other three animations are not presented in this text. The top row of images in Figure 7.6 corresponds to frame 0 and 30, and the bottom row corresponds to frames 45 and 60. The yellow and green puppets are given awkward environment specifics. The red, yellow, and green puppets execute the tasks without any serious physically implausible results. The purple puppet is forced to stretch far

for the blue block. Although the final posture of the stacking motion is realistic, the interpolating movement passes the hand and the cube through the table. This anomaly can be overcome by integrating geometric constraints throughout the motion signal, rather than the starting and ending points of task execution [LS99][BBGW94]. The final posture does not position the cube flat on the blue cube. This is a result most likely due to the joint limits of the hand, and the biased towards the positional objective function over the orientation when computing postures.

The four animations were computed on a Pentium II 400MHz. Computing the final motion for each script took approximately 1 minute.

7.3.2 Blending and Interpreting Tasks

The motion for a script requiring blending tasks and autonomous reasoning is successfully produced. Two reaching motions are combined into a single task. Two cubes are stacked on top of each other, and the puppet's hand position is adjusted appropriately to avoid object penetration. The script includes an implausible motion that is interpreted and rejected by the system. The final task requires reasoning about the current state of the system to generate appropriate motion. The script is presented below.

Body Segment	System Default Goals (see Fig. 6.6)
Left arm	Place hand on table with top grip
Right arm	Place hand on table with top grip
Torso	Upright
Head	Look at hand executing primary motion

Table 7.4 System default settings for animation #2.

Script for Animation #2
1. Set scheduling parameter to ‘full overlap’
2. Reach yellow block with left hand. Motion Primitives: Grip = top; Speed = 30; Function = $func_1$
3. Reach blue block with right hand. Motion Primitives: Grip = top; Speed = 30; Function = $func_1$
4. Lock left and right hand.
5. Set scheduling parameter to ‘no overlap’.
6. Reach yellow block with right hand. Motion Primitives: Grip = top; Speed = 30; Function = $func_1$
7. Reach blue block with left hand. Motion Primitives: Grip = top; Speed = 30; Function = $func_1$

8. Reach for space #1 with left hand. Motion Primitives: Grip = top; Speed = 30; Function = $func_1$

Table 7.5 Script for animation #2

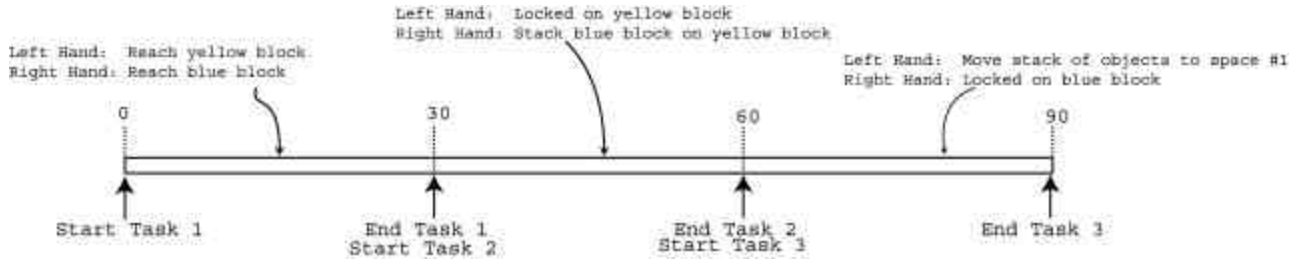

Figure 7.7 Task execution timeline for animation #2.

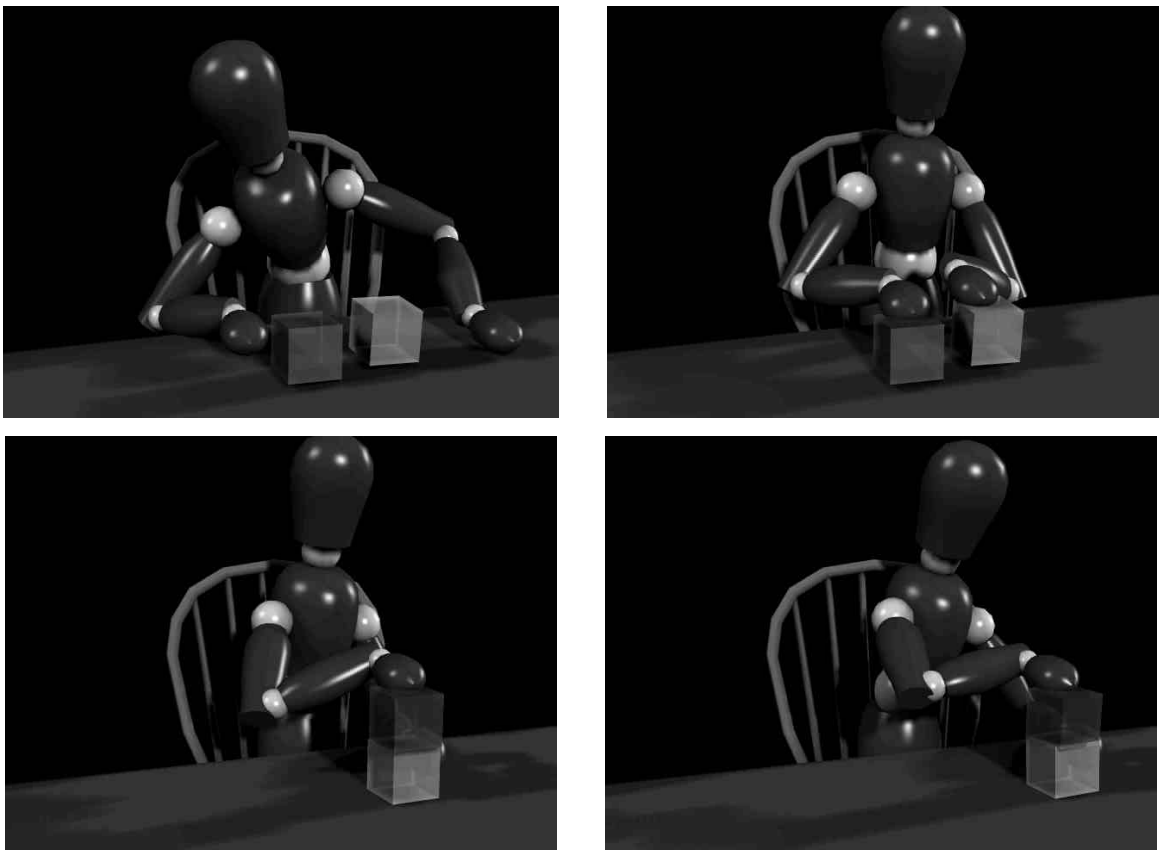

Figure 7.8 Selected frames from animation #2.

Frames from the animation are presented in Figure 7.8. The top row of images corresponds to frames 0 and 30, and the bottom row corresponds to frames 60 and 90. This script illustrates some of the autonomous interpretation of user commands implemented in our system. Tasks 1 and 2 are blended together in a single motion. The blue block is then stacked on the yellow block. However, the left hand is grasping the yellow block with a top grip. If the blue block is placed on top of the yellow block without making any adjustment, the block will collide with the hand. The system autonomously adjusts the left-hand grip to a side grip to avoid the blue block.

Task 7 is interpreted as swapping the position of the blue and yellow cubes. Swapping objects' positions in a stack is not implemented in our current system. The system does not know how to handle swapping motions, so the task is rejected by the system and no motion results from the command.

Task 8 requests reaching for space entity #1 with the left hand. The left hand is locked on the yellow object, so the task is interpreted as a request to move the yellow block to space #1. The blue block is stacked on top of the yellow block, and the resulting motion will move the stack of objects to space #1, instead of the yellow block alone. The right hand is positioned on the blue block in its new position. This movement of the right hand is performed since the right hand is locked on the blue block and both the yellow and blue block are being move to space #1.

The final motion of both hands is a result of the system interpretation of the user's input described in Chapter 6. The resulting motion was generated in 6 minutes on a Pentium II 400 MHz.

7.3.3 Drinking Coffee

We built a scenario of a puppet pouring and drinking coffee from several cups. Cyclic and non-cyclic general tasks such as scratching the head and looking at a watch are presented. Using the motion building blocks described in Section 3.6.1, we successfully executed tasks such as pouring coffee, turning on a lamp, and unscrewing the lid of a jar. The tasks are

overlapped and executed with variable speeds and interpolation functions. The entire animation is the result of a single script, which included some pauses in between motions.

This example is an attempt to show how multi-limb manipulation of objects can be effectively controlled by our system. An analysis of the script is not justified, as opposed to the previous two examples. The animation script is written with the intention of animating a rich variety of motions, rather than demonstrating particular features of the implemented system. Images from the animation are presented in Figure 7.9.

7.4 Summary

This chapter demonstrated that the algorithms and models presented in Chapters 3, 4, 5, and 6 can be integrated into a practical prototype animation system that generates movies. A sample mapping of parameters to a common input device is presented in Section 7.2. Section 7.3 demonstrates simple animation sequences resulting from guided control over the puppet's motion. The three animations illustrate the success of the system in meeting the requirements of guided control presented in Section 1.2.

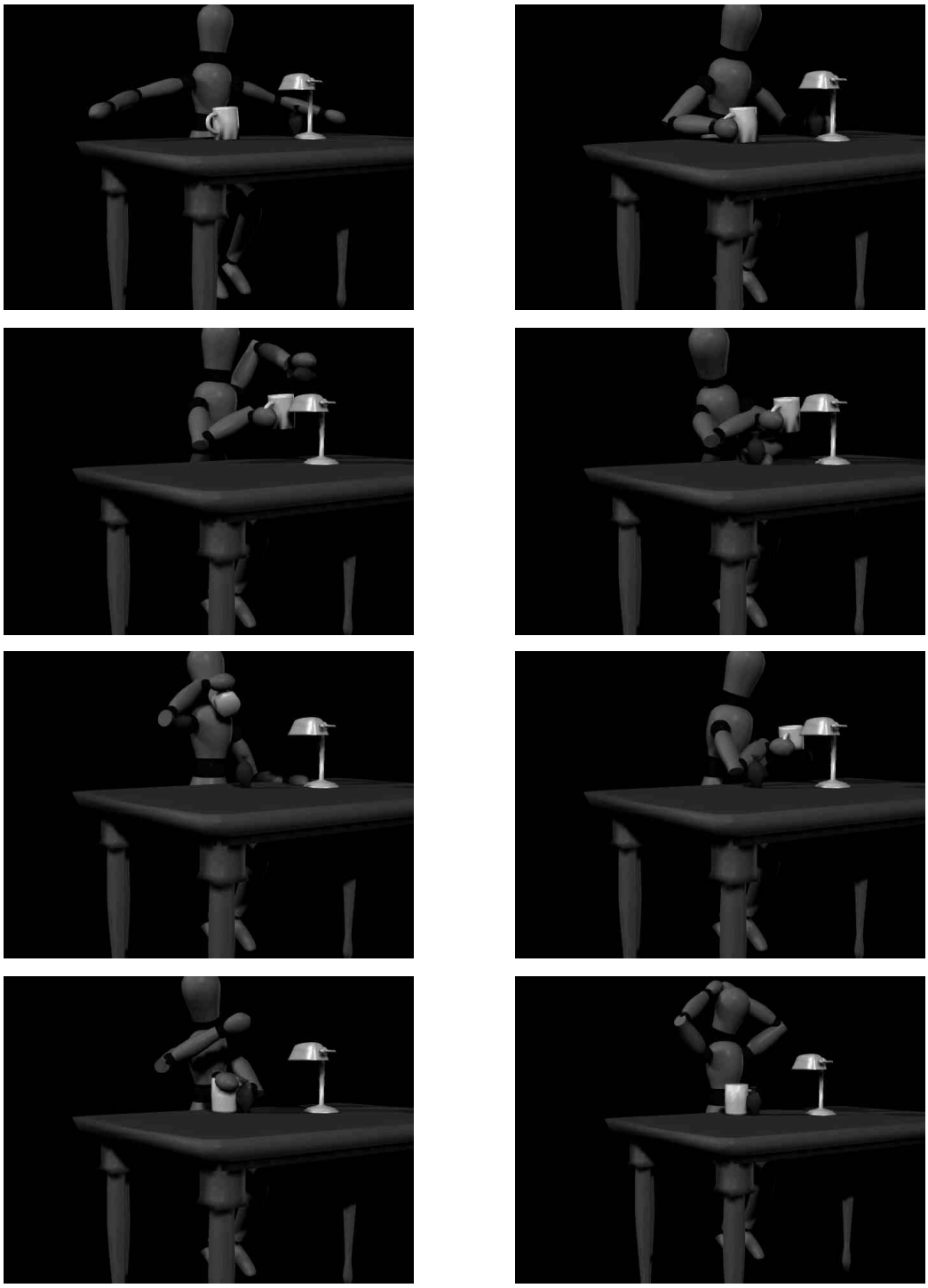

Figure 7.9 Selected frames from animation #3.

Chapter 8

Conclusion

In this thesis we explored issues in developing an intelligent guided-level control animation system. Solutions to some of the issues are proposed as part of this work, while others are left for future research. Section 8.1 presents appropriate future work for issues not addressed in this work, or for implemented solutions deficient in some respect. Section 8.2 summarizes the contributions of the thesis.

8.1 Outstanding Problems

Guided-level control can be extended in a number of directions. Our vision of guided-level control of articulated figures presented in Section 1.2 extends beyond a single humanoid figure seated at a table. Guided control is a level of abstraction that can be applied to a number of characters and scenarios for improved convenience and control over the resulting motion. The topic is broad and can benefit from results in areas of research not directly related to computer graphics. The following paragraphs introduce some of the unresolved issues encountered in our research.

Results from ergonomic and neurophysiology research were instrumental in developing posture generator algorithm in Chapter 5. However, the computation time is too long for real-time interactivity. The implemented IK algorithm may benefit from

optimizations that will yield adequate postures in real-time. The current prototype only allows approximate posture estimates when used in real-time. Considering a more sophisticated knowledge-based approach, such as incorporating a database of human postures, may improve run-times and the perceived naturalness of the postures. Implementing more advanced constant-time neurological models may also result in improved postures and run-times for certain tasks. Our current puppet model is simplified in several respects as discussed in Section 3.5. It would be interesting to see how the lower limbs add to the complexities of posture generation and executing multiple tasks simultaneously [BB00]. At the heart of this level of control is an understanding of how the character effectively uses his body to perform arbitrary tasks, which may delve into areas of neurology and biomechanics. Topics in realistic human kinematic motion are presented in Section 2.10.

Awkward grasping motions, although biomechanically feasible, are not well suited to our current implementation. The problem of grasping and manipulating objects of variable size, weight, and geometric configurations with a realistic model of the human hand is challenging. Positioning the fingers to model a realistic handhold on an object would be relevant future research [RG91]. This problem reduces to modeling how humans psychologically perceive objects and apply learned behaviours to manipulate them.

The motion of the puppet has been simplified by interpolating between the puppet's initial and final state. Although our interpolating functions correlate with biomechanical studies of single joint motions, the problem of accurately modeling multi-joint human movement is much more complex. Incorporating more sophisticated biomechanical models can contribute to the realism of the puppet's motion. A survey of applicable research is presented in Section 2.10.

Considering emotional characteristics of motion as a user-specified parameter would be another interesting improvement upon our current implementation. Having a user specify “happy” or “angry” rather than making a reference to a particular interpolation function would be a powerful feature for generating complex movement. The current state of the art does not make this immediately feasible, and would have to incorporate motion capture data into the system. Unfortunately, this would narrow the range of possible motions considerably. Facial expressions are also important to conveying the mood of the character performing a task [Per93]. A reasonable short-term plan for incorporating emotion into our

system is to texture map a face onto the existing puppet model, and allow the user to interpolate between various possible expressions.

Our initial intention was to implement full real-time interaction with the puppet. Assuming the computational cost of the inverse kinematics is overcome, the optimal mode of specifying a large number of parameters for controlling a virtual puppet in real-time is not clear. Guided-level control implies that there may be several motion primitives to specify for every task command. Executing several tasks quickly would require an interface conducive to such a level of interactivity and control. These issues benefit from future research in human computer interaction. The current implementation provides multi-modal interaction with the puppet. The effectiveness and learning curve associated with the interface was not considered in this thesis.

Applying collision detection and removal throughout the motion signal is imperative for a robust animation system [BBGW94][WP95]. Path planning to avoid objects and self-collisions are important for the perception of realism in arbitrary environments. Currently, our system only ensures collisions are avoided at specific keyframes, and does not check for possible collisions occurring between keyframes. The model proposed in Section 5.6 illustrates how collision avoidance of geometry in the environment can be integrated in the system.

Guided-level control of multiple characters in a shared environment would be interesting. The user's intentions would be interpreted in terms of the other character's state, previously performed tasks, and the state of the environment. Allowing a user to specify cooperative, multi-limb tasks of several characters is applicable to many scenarios. Applying guided-level control to articulated figures of arbitrary dimensions would be an enhancement of the posture generator discussed in Chapter 5. Attempts to implement guided-level control of other species, such as quadrupeds or inanimate objects, would be another prototype that would not benefit from the work presented in Chapter 5, but would extend the issues discussed in Chapters 4 and 6.

8.2 Contributions

The work presented in this thesis provides a novel set of motion directives to effectively control a 3D articulated figure in a specific scenario. The animations presented in Chapter 7 were generated with a level of control, effort, and interactivity consistent with Figure 2.3. Chapter 4 introduces how multiple tasks can be performed concurrently at variable velocities over time. The foundation for more sophisticated interpretation of user intentions is set in Chapter 6. A new posture design algorithm for a 3D, 27-degree of freedom virtual puppet is introduced in Chapter 5.

A prototype animation tool is implemented with multi-modal puppet interaction. The interaction with the system is described in Section 7.1, and the architecture of the system is illustrated in Figure 3.2. Our system generates human-like motion to accomplish multi-limb object manipulation tasks. The user can effectively control the puppet's hands to perform multiple independent tasks simultaneously, or a single task requiring cooperative use of both hands. Finally, our system exhibits intelligent behaviours by adapting postures and interpreting the user's intentions in context with the current state of the environment.

The results and proposed solutions suggest that guided control systems can be an effective tool for animating characters in variable environments with a limited set of objects for manipulation. Task specification in our prototype suggests that it is feasible to express 3D character motion in a level of abstraction similar to other forms of artistic expression.

8.2 Summary

The challenge of human animation invokes an introspective examination of human motor control. How and why we move a certain way and are able to distinguish between living and synthetic motion is a mystery modern science cannot explain. The ultimate goal of human guided control is to effortlessly specify abstract tasks resulting in motion that cannot be distinguished from a real human being. This was not achieved in this thesis, but it is hoped that my humble contribution has advanced us closer to the goal.

Appendix A

The extent to which the scheduler will attempt to concurrently execute motions depends on the value of the scheduling parameter. The scheduling parameter can be set to one of three values: *no overlap*, *partial overlap*, *full overlap*. The following notation is used to describe the three scheduling schemes.

- m_0, m_1 : the first and second frame in the motion queue respectively.
- $\{j_0, j_1, \ldots j_8\}$ is the set of all joints in the puppet.
- $\{m_k.critical\}$ is the set of joints corresponding to critical body segments in motion frame m_k .
- $m_k.sched\_param$ is the value of the scheduling parameter in motion frame m_k .
- $S\_P_{m_k}$ is the set of ball joints scheduled positional goals from motion frame m_k .
- $S\_T_{m_k}$ is the set of ball joints scheduled trajectory data from motion frame m_k .

Before we can schedule posture and trajectory goals, we group the joints twice into two mutually exclusive sets ($S\_P_{m_0}, S\_P_{m_1}$, *and* $S\_T_{m_0}, S\_T_{m_1}$). Every degree of freedom in the puppet is assigned data from m_0 or m_1, depending on which set the respective ball joint belongs to. For example, the x-axis data structure of ball joint j_i (denoted $j_{i,x}$) will reference positional data in motion frame m_k if $j_i \in S\_P_{m_k}$, and trajectory data in $m_{k'}$ if $j_i \in S\_T_{m_{k'}}$.

The Group_Positions() procedure will include all the puppet's eight ball joints in one of two mutually exclusive sets, $S\_P_{m_0}$ or $S\_P_{m_1}$. The joints referenced in $S\_P_{m_0}$ will copy positional data from motion frame m_0. Likewise, $S\_P_{m_1}$ is the set of joints to copy positional data from m_1.

Group_Positions():
If (($m_1.sched\_param == no\_overlap$)||(One Motion Frame in Motion Queue)) {
$S\_P_{m_0} \leftarrow \{j_0, j_1, \ldots j_8\}$
$S\_P_{m_1} \leftarrow \{\ \}$
}
Else if ($m_1.sched\_param == partial\_overlap$) {
if $\begin{pmatrix} (\{j_2, j_3, j_4\} \in (\{m_0.critical\} \cap \{m_1.critical\})) \| \\ (\{j_5, j_6, j_7\} \in (\{m_0.critical\} \cap \{m_1.critical\})) \end{pmatrix}$ {
$S\_P_{m_0} \leftarrow \{j_0, j_1, \ldots, j_8\}$
$S\_P_{m_1} \leftarrow \{\ \}$
}
else{
$S\_P_{m_0} \leftarrow \{m_0.critical\} \cup (\{j_0, j_1, \ldots j_8\} - (\{m_0.critical\} \cup \{m_1.critical\}))$
$S\_P_{m_1} \leftarrow \{m_1.critical\} - \{m_0.critical\}$
}
}
Else if ($m_1.sched\_param == full\_overlap$) {
if $((\{m_0.critical\} \cap \{m_1.critical\}) == \{j_0, j_1\})$ {
$S\_P_{m_0} \leftarrow \{m_0.critical\} \cup (\{j_0, j_1, \ldots j_8\} - (\{m_0.critical\} \cup \{m_1.critical\}))$
$S\_P_{m_1} \leftarrow \{m_1.critical\}$
}
else if $\begin{pmatrix} (\{j_2, j_3, j_4\} \in (\{m_0.critical\} \cap \{m_1.critical\})) \| \\ (\{j_5, j_6, j_7\} \in (\{m_0.critical\} \cap \{m_1.critical\})) \end{pmatrix}$ {
$S\_P_{m_0} \leftarrow \{j_0, j_1, \ldots, j_8\}$
$S\_P_{m_1} \leftarrow \{\ \}$
}
else{
$S\_P_{m_0} \leftarrow \{m_0.critical\} \cup (\{j_0, j_1, \ldots j_8\} - (\{m_0.critical\} \cup \{m_1.critical\}))$
$S\_P_{m_1} \leftarrow \{m_1.critical\} - \{m_0.critical\}$
}
}

1) If $m_1.sched\_sys == no\_overlap$, then each frame is executed independently. All joints are scheduled posture data from the head of the queue, m_0.
2) If $m_1.sched\_sys == partial\_overlap$, we first consider any conflicting critical segments between m_0 and m_1. If m_0 and m_1 have a common arm in the set of critical segments, then no motion blending can occur and we schedule all positional data from m_0. Otherwise, then all the critical segments in m_0 are scheduled positional goals from m_0. Critical segments in m_1 that are not critical in m_0 are scheduled positional goals from m_1. The rest of the body segments are scheduled from m_0.
3) If $m_1.sched\_sys == full\_overlap$, then we check if j_0 and j_1 are critical segments in both m_0 and m_1. If j_0 and j_1 are common critical segments in m_0 and m_1, then the two joints are shared between the two tasks (i.e. the motion frames are 'blended'). Body segments that are not critical in either m_0 or m_1 are scheduled positional data from m_0. If $\{j_0, j_1\}$ are not common critical segments, then 'full_overlap' schedules positional data identical to the 'partial_overlap' scheme

Similar to Group_Position(), the Group_Trajectory() procedure will divide the puppet's ball joints between two mutually exclusive sets, $S\_T_{m_0}$ and $S\_T_{m_1}$. Trajectory data is copied from motion frame m_0 or m_1 depending on which set the joint belongs.

Group_Trajectories():
If (($m_1.sched\_param == no\_overlap$)||(One Motion Frame in Motion Queue)) {
$S\_T_{m_0} \leftarrow \{j_0, j_1, \ldots j_8\}$
$S\_T_{m_1} \leftarrow \{\ \}$
}
else if(($m_1.sched\_param == partial\_overlap$)||($m_1.sched\_param == full\_overlap$)){
if $(\{m_0.critical\} \cap \{m_1.critical\} == \{\ \})$ {
$S\_T_{m_0} \leftarrow \{m_0.critical\} \cup (\{j_0, j_1, \ldots j_8\} - (\{m_0.critical\} \cup \{m_1.critical\}))$
$S\_T_{m_1} \leftarrow \{m_1.critical\}$
}
else{
$S\_T_{m_0} \leftarrow \{j_0, j_1, \ldots j_8\}$
$S\_T_{m_1} \leftarrow \{\ \}$
}
}

1) If the scheduling parameter of m_1 is *no overlap*, all joints are scheduled trajectory data from m_0.
2) If the scheduling parameter of m_1 is *partial overlap* or *full overlap*, and there are no conflicting critical segments between m_0 and m_1, then the critical segments from m_0 and m_1 are scheduled trajectory data from m_0 and m_1 respectively. Body segments that are not critical in m_0 or m_1 are scheduled trajectories from m_0. If there is conflicting critical segments, then all the joints are scheduled trajectory data from m_0.

Appendix B

There are a number of analytical and numerical methods which can be employed to satisfy the orientation objective function. Our implementation uses an algebraic approach introduced by Paul [Pau81]. The orientation of some joint i with respect to world space is calculated as follows:

$$O_i = M_i M_{i-1} M_{i-2} \ldots M_1 M_0,$$

where M_k is the orientation of joint k with respect to joint *k-1* local coordinate frame.

Let G_O be the goal orientation with respect to world coordinates. Let joint k be the link adjacent to the end effector. Let us assume joint k orientation is specified by rotations about the z, y, and x-axis respectively. Hence, we would like to find a , b , k such that

$$G_O = RotZ(a) \cdot RotY(b) \cdot RotX(k) \cdot O_{k-1}.$$

G_O and O_{k-1} are known values. We solve for a :

$$RotZ^{-1}(a) \cdot G_O \cdot O_{k-1}^{-1} = RotY(b) \cdot RotX(k).$$
$$RotZ^{-1}(a) \cdot M' = RotY(b) \cdot RotX(k).$$

Since matrix equality implies equality of corresponding elements, the above equation gives us:

$$-\sin(a) \cdot M'[0][0] + \cos(a) \cdot M'[1][0] = 0.$$
$$\tan(a) = \frac{M'[1][0]}{M'[0][0]}.$$
$$a = \arctan 2(M'[1][0], M'[0][0]).$$

Now a is a known value so we have $M'' = RotZ^{-1}(a) \cdot M'$ and $M'' = RotY(b) \cdot RotX(k)$.

$$M''[2][0] = -\sin(b).$$
$$M''[0][0] = \cos(b).$$
$$\tan(b) = \frac{-M''[2][0]}{M''[0][0]}.$$
$$b = \arctan 2(-M''[2][0], M''[0][0]).$$

$$M''[1][2] = -\sin(k).$$
$$M''[1][1] = \cos(k).$$
$$\tan(k) = \frac{-M''[1][2]}{M''[1][1]}.$$
$$k = \arctan 2(-M''[1][2], M''[1][1]).$$

a, b, k satisfy the orientation constraints G_O. b and k are dependent on a. One should consider joint limits and weight factors before calculating b and k. Let $q' = f(q)$, where f applies weight factors and ensures joint limits are respected. We calculate b and k with respect to constrained a', since b' and k will maximize the objective function given the orientation after the Z-rotation. If we calculate b and k before constraining a, then we will be maximizing the objective function assuming a is unconstrained. The algorithm for maximizing the orientation objective function is:

$$M' = G_O \cdot O_{k-1}^{-1}.$$
$$a = \arctan 2(M'[1][0], M'[0][0]).$$
$$a' = f(a).$$
$$M'' = RotZ^{-1}(a') \cdot M'.$$
$$b = \arctan 2(-M''[2][0], M''[0][0]).$$
$$k = \arctan 2(-M''[1][2], M''[1][1]).$$
$$b' = f(b).$$
$$k' = f(k).$$

Appendix C

The following is a table of weight factors for all joints in every pass of the algorithm in Figure 5.17. The values in the table correspond to the variable w_i in equation (5.24).

Joint	Pass 1	Pass 2	Pass 3
Abdomen x-axis	0.001	0.001	0.001
Abdomen y-axis	0.001	0.001	0.001
Abdomen z-axis	0	0.001	0.001
Chest x-axis	0.001	0.001	0.001
Chest y-axis	0.001	0.001	0.001
Chest z-axis	0	0.001	0.001
Left shoulder x-axis	0.05	0.05	0.05
Left shoulder y-axis	0.4	0.4	0.4
Left shoulder z-axis	0	0	0.005
Right shoulder x-axis	0.05	0.05	0.05
Right shoulder y-axis	0.4	0.4	0.4
Right shoulder z-axis	0	0	0.005
Left forearm x-axis	0.1	0.1	0.1
Left forearm y-axis	0.4	0.4	0.4
Left forearm z-axis	0	0	0
Right forearm x-axis	0.1	0.1	0.1
Right forearm y-axis	0.4	0.4	0.4
Right forearm z-axis	0	0	0
Left hand x-axis	0.4	0.4	0.4
Left hand y-axis	0.4	0.4	0.4
Left hand z-axis	0.4	0.4	0.4
Right hand x-axis	0.4	0.4	0.4
Right hand y-axis	0.4	0.4	0.4
Right hand z-axis	0.4	0.4	0.4

References

[AF84] S. Adamovich, A. Feldman. Model of the Central Regulation of the Parameters of Motor Trajectories. *Biophysics*, 29(2):338-342, 1984.

[AGL86] W.W. Armstrong, M. Green, R. Lake. Near-Real-Time Control of Human Figure Models. *Proceedings of Graphics Interface*, pages 147-151, 1986.

[AM91] M. Ayoub, M. Miller. Industrial Workplace Design. *Workspace, Equipment and Tool Design*, Mital and Karwowski (ed.), pages 67-92, 1991.

[Bad86] Norman Badler. Animating Human Figures: Perspectives and Directions. *Proceedings of Graphics Interface*, pages 115-120, 1986.

[BPW93] Norman Badler, Cary Phillips, Bonnie Webber. *Simulating Humans*. Oxford University Press, Oxford, NY, 1993.

[BBGW94] Norman Badler, Ramamani Bindiganavale, John Granieri, Susanna Wei, Xinmin Zhao. Posture Interpolation with Collision Avoidance. *Proceedings of Computer Animation*, pages 13-20, 1994.

[BB98] Paolo Baerlocher, Ronin Boulic. Task-Priority Formulations for the Kinematic Control of Highly Redundant Articulated Structures. *Proceedings of IEEE IROS '98*, pages 323-329, Oct. 1998.

[BB00] Paolo Baerlocher, Ronan Boulic.. From Soft to Hard Priorities for IK Control of the Full Body. *Poster Proceedings of Graphics Interface*, pages 2-3, 2000.

[BGF86] M. Berkinblit, I. GelFand, A. Feldman. Model of the Control of the Movements of a Multi-Joint Limb. *Biophysics*, 31(1):142-153, 1986.

[BG95] Bruce Blumberg, Tinsley Galyean. Multi-Level Direction of Autonomous Creatures for Real-time Virtual Environments. *Computer Graphics Proceedings*, Annual Conference Series, pages 47-54, 1995.

[BT97] Ronan Boulic, Daniel Thalmann. Complex Character Positioning Based on a Compatible Flow Model of Multiple Supports, *IEEE Transactions on Visualization and Computer Graphics*, 3(3), July-Sept. 1997.

[BN88] Lynne Brotman, Arun Netravali. Motion Interpolation by Optimal Control. *Computer Graphics*, 22(4):309-315, 1988.

[BC89] Armin Bruderlin, Thomas Calvert. Goal-Directed, Dynamic Animation of Human Walking. *Computer Graphics*, 23(3):233-242, 1989.

[BC93] Armin Bruderlin, Tom Calvert. Interactive Animation of Personalized Human Locomotion. *Proceedings of Graphics Interface*, pages 17-23, 1993.

[BC96] Armin Bruderlin, Tom Calvert. Knowledge-Driven, Interactive Animation of Human Running. *Proceedings of Graphics Interface*, pages 213-221, 1996.

[BW95] Armin Bruderlin, Lance Williams. Motion Signal Processing. *Computer Graphics Proceedings, Annual Conference Series*, pages 97-104, 1995.

[CS86] Danny Cachola, Gunther Schrack. Modeling and Animating Three-Dimensional Articulated Figures. *Proceedings of Graphics Interface*, pages 152-157, 1986.

[Cha87] D. Chaffin. Biomechanical Aspects of Workplace Design. *Handbook of Human Factors*, John Wiley and Sons (ed.), New York, pages 602-619, 1987.

[Cor90] Paul Cordo. Kinesthetic Control of a Multi-joint Movement Sequence. *Journal of Neurophysiology*, 63(1):161-172, Jan. 1990.

[CB76] E. Corlett, R. Bishop. A technique for assessing postural discomfort. *Ergonomics*, 19(2):175-182 , 1976.

[DMSB95] Keith Doty, Claudio Melchiorri, Eric Schwartz, Claudio Bonivento. Robot Manipulability. *IEEE Transactions on Robotics and Automation*, 11(3), June 1995.

[DT86] Karin Drewery, John Tsotsos. Goal-Directed Animation using English Motion Commands. *Proceedings of Graphics Interface*, pages 131-135, 1986.

[FN71] R. Fikes, N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, Vol. 2, pages 189-208, 1971.

[FvDFH90] J. Foley, A. van Dam, S. Feiner, J. Hughes. *Computer Graphics: Principles and Practice*. Second edition, Addison-Wesley, Reading, MA, 1990.

[FTT99] John Funge, Xiaoyuan Tu, Demetri Terzopoulos. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. *Computer Graphics Proceedings, Annual Conference Series*, pages 29-38, 1999.

[GLM94] Christopher Geib, Libby Levison, Michael Moore. *SodaJack: an architecture for agents that search for and manipulate objects.* Technical Report MS-CIS-94-16/LINC LAB 265, Department of Computer and Information Science, University of Pennsylvania, 1994.

[GC95] K. Glass, R. Colbaugh. Real-Time Collision Avoidance for Redundant Manipulators. *IEEE Transactions on Robotics and Automation*, 11(3), June 1995.

[Gle98] Michael Gleicher. Retargetting Motion to New Characters. Computer Graphics Proceedings, Annual Conference Series, pages 33-42, 1998.

[GCJA88] G. Gottlieb, D. Corcos, S. Jarie, G. Agarwal. Practice improves even the simplest movements. *Experimental Brain Research*, Vol. 73, pages 436-440, 1988.

[GTH98] Radek Grzeszczuk, Demetri Terzopoulos, Geoffrey Hinton. NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models. *Computer Graphics Proceedings, Annual Conference Series*, pages 9-20, 1998.

[HWBO95] Jessica Hodgins, Wayne Wooten, David Brogan, James O'Brien. Animating Human Athletics. *Computer Graphics Proceedings, Annual Conference Series*, pages 71-78, 1995.

[HP97] Jessica Hodgins, Nancy Pollard. Adapting Simulated Behaviors for New Characters. *Computer Graphics Proceedings, Annual Conference Series*, pages 153-162, 1997.

[HS86] Donna Hoffman, Peter Strick. Step-Tracking Movements of the Wrist in Humans: Kinematic Analysis. *The Journal of Neuroscience*, 6(11):3309-3318, Nov. 1986.

[HvP96] Pedro Huang, Michiel van de Panne. A Planning Algorithm for Dynamic Motions. *Computer Animation and Simulation '96*, pages 169-182. Eurographics Workshop, September 1996.

[KvP00] Maciej Kalisiak, Michiel van de Panne. A Grasp-based Motion Planning Algorithm for Character Animation. *Computer Animation and Simulation 2000*, pages 43-58. Eurographics Workshop, August 2000.

[KMMS98] Prem Kalra, Nadia Magnenat-Thalmann, Laurent Moccozet, Gael Sannier, Amaury Aubel, Daniel Thalmann. Real-Time Animation of Realistic Virtual Humans. *IEEE Computer Graphics and Applications*, pages 42-55, Sept./Oct. 1998.

[KKKL94] Yoshihito Koga, Koichi Kondo, James Kuffner, Jean-Claude Latombe. Planning Motions with Intentions. *Computer Graphics Proceedings, Annual Conference Series*, pages 395-408, 1994.

[Kon94] Koichi Kondo. *Inverse Kinematics of a Human Arm*. Technical Report CS-TR-94-1508, Deparment of Computer Science, Stanford University, 1994.

[KB82] James Korein, Norman Badler. Techniques for Generating the Goal-Directed Motion of Articulated Structures. *IEEE Computer Graphics and Applications*, pages 71-81, Nov. 1982.

[LvP96] Alexis Lamouret, Michiel van de Panne. Motion Synthesis by Example. *Computer Animation and Simulation '96*, pages 199-212. Eurographics Workshop, September 1996.

[Las87] John Lasseter. Principles of Traditional Animation Applied to 3D Computer Animation. *Computer Graphics*, 21(4):35-44, 1987.

[LvP00] Joseph Laszlo, Michiel van de Panne. Interactive Control and Composition of Physically-Based Animations. *Computer Graphics Proceedings, Annual Conference Series*, pages 201-208, 2000.

[LvPF96] Joseph Laszlo, Michiel van de Panne, Eugene Fiume. Limit Cycle Control and its Application to the Animation of Balancing and Walking. *Computer Graphics Proceedings, Annual Conference Series*, pages 155-162, 1996.

[Lat93] Mark Latash. *Control of Human Movement*. Human Kinetics Publishers, Champaign, IL, 1993.

[LS99] Jehee Lee, Sung Yong Shin. A Hierarchical Approach to Interactive Motion Editing for Human-like Figures. *Computer Graphics Proceedings, Annual Conference Series*, pages 39-48, 1999.

[LWZB90] Philip Lee, Susanna Wei, Jianmin Zhao, Norman Badler. Strength Guided Motion. *Computer Graphics*, 24(4):253-262, 1990.

[LB94] Libby Levison, Norman Badler. How Animated Agents Perform Tasks: Connecting Planning and Manipulation Through Object-Specific Reasoning. Toward Physical Interaction and Manipulation, *AAAI Spring Symposium Series*, 1994.

[LMY97] R. Loftin, J. Maida, J. Yang. *Inverse Kinematics of the Human Arm*. Annual Report 1996-1997, Institute for Space Systems Operations, University of Houston, TX, 1997.

[Mai96] R. Maiocchi. *3-D character animation using motion capture, in Interactive Computer Animation*. N. M. Thalmann and D. Thalmann (ed.), Prentice-Hall, Englewood Cliffs, NJ, pages 10-39, 1996.

[MFV84] Ann Marion, Kurt Fleischer, Mark Vickers. Towards Expressive Animation for Interactive Characters. *Proceedings of Graphics Interface*, pages 17-20, 1984.

[Mat99] Michael Mateas. An Oz-Centric Review of Interactive Drama and Believable Agents. *Lecture Notes in Artificial Intelligence*, pages 297-321, Springer-Verlag, Berlin, 1999.

[vMGvGG90] J. van der Meulen, R. Gooskens, J. Denier van der Gon, C. Gielen, K. Wilhelm. Mechanisms Underlying Accuracy in Fast Goal-Directed Arm Movements in Man. *Journal of Motor Behavior*, 22(1):67-84, 1990.

[MC90] Claudia Morawetz, Thomas Calvert. Goal-Directed Human Animation of Multiple Movements. *Proceedings of Graphics Interface*, pages 60-67, 1990.

[ML87] B. Mustard, R. Lee. Relationship between EMG patterns and kinematic properties for flexion movements at the human wrist. *Experimental Brain Research*, Vol. 66, pages 247-256, 1987.

[NA89] B. Naderi, M. Ayoub. *Cumulative Hand Trauma Disorders*. Technical Report, Institute for Ergonomics Research, Texas Technical University, 1989.

[Nag89] H. Nagasaki. Asymmetric velocity and acceleration profiles of human arm movements. *Experimental Brain Research*, Vol.74, pages 319-326, 1989.

[OGH92] Lee Ostrom, Gay Gilbert, Susan Hill. Development of an ergonomic assessment checklist and its use for evaluating and EG&G Idaho print shop: A Case Study. *Advances in Industrial Ergonomics and Safety IV*, pages 469-474, 1992.

[OCM87] D. Ostry, J. Cooke, K. Munhall. Velocity curves of human arm and speech movements. *Experimental Brain Research*, Vol. 68, pages 37-46, 1987.

[vPF93] Michiel van de Panne, Eugene Fiume. Sensor-Actuator Networks. *Computer Graphics Proceedings, Annual Conference Series*, pages 335-342, 1993.

[vPKF94] Michiel van de Panne, Ryan Kim, Eugene Fiume. Virtual Wind-up Toys for Animation. *Proceedings of Graphics Interface*, pages 208-215, 1994.

[Pau81] Richard Paul. *Robot Manipulators: mathematics, programming, and control*. MIT Press, Cambridge, MA, 1981.

[Per93] Ken Perlin. Layered Composition of Facial Expression. *Visual Proceedings of Computer Graphics (SIGGRAPH '97)*, pages 226-227, 1993.

[Per97] Ken Perlin. Real-time Responsive Animation with Personality. *Siggraph '97 : Course Notes*, Vol. 17, 1997.

[PG96] Ken Perlin, Athomas Goldberg. Improv: A System for Scripting Interactive Actors in Virtual Worlds. *Computer Graphics Proceedings, Annual Conference Series*, pages 205-216, 1996.

[PG99] Ken Perlin, Athomas Goldberg. Improvisational Animation. *Proceedings of The Society for Computer Simulation International Conference on Virtual Worlds and Simulation (VWSIM'99)*, San Francisco, California. January 17-20. 1999.

[PB91] Cary Phillips, Norman Badler. Interactive Behaviors for Bipedal Articulated Figures. *Computer Graphics*, 25(4):359-363, 1991.

[PZB90] Cary Phillips, Jianmin Zhao, Norman Badler. Interactive Real-time Articulated Figure Manipulation Using Multiple Kinematic Constraints. *Computer Graphics*, 24(2):245-250, 1990.

[PW99] Zoran Popovic, Andrew Witkin. Physically-Based Motion Transformation. *Computer Graphics Proceedings, Annual Conference Series*, pages 11-20, 1999.

[RH91] Marc Raibert, Jessica Hodgins. Animation of Dynamic Legged Locomotion. *Computer Graphics*, 25(4):349-358, 1991.

[Ree81] William Reeves. Inbetweening for Computer Animation Utilizing Moving Point Constraints. *Computer Graphics*, 15(3):263-269, 1981.

[Rey87] Craig Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics*, 21(4):25-34, 1987.

[RHC86] G. Ridsdale, S. Hewitt, T.W. Calvert. The Interactive Specification of Human Animation. *Proceedings of Graphics Interface*, pages 121-130, 1986.

[RG91] Hans Rijpkema, Michael Girard. Computer Animation of Knowledge-Based Human Grasping. *Computer Graphics*, 25(4):339-348, 1991.

[RGBC96] Charles Rose, Brian Guenter, Bobby Bodenheimer, Michael Cohen. Efficient Generation of Motion Transitions using Spacetime Constraints. *Computer Graphics Proceedings, Annual Conference Series*, pages 147-154, 1996.

[SS91] Bruno Siciliano, Jean-Jacques Slotine. A General Framework for Managing Multiple Tasks in Highly Redundant Robotic Systems. *Proceedings of ICAR '91*, Vol. 2, pages 1211-1215, 1991.

[SF89a] J. Soechting, M. Flanders. Sensorimotor Representations for Pointing to Targets in Three-Dimensional Space. *Journal of Neurophysiology*, 62(2):582-594, Aug. 1989.

[SF89b] J. Soechting, M. Flanders. Errors in Pointing are Due to Approximations in Sensorimotor Transformations. *Journal of Neurophysiology*, 62(2):595-608, Aug. 1989.

[Sor89] Peter Sorenson. Felix the Cat – Real-time Computer Animation. *Animation Magazine*, pages 13-14, Winter 1989.

[SB85] Scott Steketee, Norman Badler. *Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control*. 19(3):255-262, 1985.

[Stu84] David Sturman. Interactive Keyframe Animation of 3-D Articulated Models. *Proceedings of Graphics Interface*, pages 35-40, 1984.

[Stu94] David Sturman. Character Motion Systems. *Siggraph '94 : Course Notes*, Vol. 9, 1994.

[Stu98] David Sturman. Computer Puppetry. *IEEE Computer Graphics and Applications*, pages 38-45, Jan./Feb. 1998.

[TS00] Gaurav Tevatia, Stefan Schaal. Inverse Kinematics for Humanoid Robots. *Proceedings of IEEE International Conference on Robotics and Automation*, pages 294-299, Apr. 2000.

[Tha99] Daniel Thalman. Towards Autonomous, Perceptive, and Intelligent Virtual Actors. *Lecture Notes in Artificial Intelligence*, pages 297-321, Springer-Verlag, Berlin, 1999.

[Tis78] E. Tischauer. *The Biomechanical Basis of Ergonomics*. John Wiley and Sons (ed.), New York, 1978.

[TvP98] Nick Torkos, Michiel van de Panne. Footprint-based Quadruped Motion Synthesis. *Proceedings of Graphics Interface*, pages 151-160, 1998.

[TT94] Xiaoyuan Tu, Demetri Terzopoulos. Artificial Fishes: Physics, Locomotion, Perception, Behaviour. *Computer Graphics Proceedings, Annual Conference Series*, pages 43-50, 1994.

[UAT95] Munetoshi Unuma, Ken Anjyo, Ryozo Takeuchi. Fourier Principles for Emotion-based Human Figure Animation. *Computer Graphics Proceedings, Annual Conference Series*, pages 91-96, 1995.

[Wel93] Chris Welman. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation*. MSc. Thesis, 1993.

[WP95] Andrew Witkin, Zoran Popovic. Motion Warping. *Computer Graphics Proceedings, Annual Conference Series*, pages 105-108, 1995.

[Zel82] David Zeltzer. Motor Control Techniques for Figure Animation. *IEEE Computer Graphics and Applications*, Vol.2, pages 53-59, Nov. 1982.

[Zel85] David Zeltzer. Towards an Integrated View of 3-D Computer Character Animation. *Proceedings of Graphics Interface*, pages 105-115, 1985.

[Zha96] Xinmin Zhao. *Kinematic Control of Human Postures for Task Simulation*. PhD. Thesis, 1996.