

Jack: A Toolkit for Manipulating Articulated Figures

Cary B. Phillips
Norman I. Badler

Computer Graphics Research Laboratory
Department of Computer and Information Sciences
University of Pennsylvania
Philadelphia, Pennsylvania 19104-0389

Abstract

The problem of positioning and manipulating three dimensional articulated figures is often handled by ad hoc techniques which are cumbersome to use. In this paper, we describe a system which provides a consistent and flexible user interface to a complex representation for articulated figures in a 3D environment. **Jack** is a toolkit of routines for displaying and manipulating complex geometric figures, and it provides a method of interactively manipulating arbitrary homogeneous transformations with a mouse. These transformations may specify the position and orientation of figures within a scene or the joint transformations within the figures themselves. **Jack** combines this method of 3D input with a flexible and informative screen management facility to provide a user-friendly interface for manipulating three dimensional objects.

1 Introduction

Many animation and simulation systems perform sophisticated operations on geometric figures in a particular configuration, but they leave the actual posi-

tioning of the figures to ad hoc techniques such as keyboard input of numerical joint angles or valuator input from a mouse. This requires the user to remember complex information about the coordinate frames with which objects are constructed and the axes around which the joints revolve. All too often, such systems provide minimal capabilities for describing basic 3D position and orientation, forcing the user to adapt to a primitive set of commands.

Much work[7, 5, 16] has shown the value of kineshetically appropriate feedback. Systems which do provide direct manipulation with multi-dimensional input devices often do not provide adequate feedback on how the motion of the input device produces world space transformations. Typically, several dozen degrees of freedom must be manipulated. For example, the human body model used by TEMPUS[2] has 18 joints and 48 degrees of freedom. Without adequate means of handling this complexity, systems can easily become cumbersome to use, and the problem of figure positioning becomes a tremendous burden for the user.

Jack is a system being developed at the University of Pennsylvania to provide a flexible user and programmer interface for manipulating complex articulated structures, particularly human figures in a 3D working environment. **Jack** is not a complete system in itself, but a toolkit of routines and operators for displaying and manipulating geometric figures. It is available to programmers as a basic framework which can easily be extended and tailored for specific applications. Currently, **Jack** provides input and control for applications involving lighting and image rendering, anthropometric modeling, dynamic analysis, and keyframe and constraint-based animation and simula-

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

tion.

Jack incorporates a simple but powerful mechanism for manipulating homogeneous transformations. These transformations describe the position and orientation of the figures and the displacement of joints, as well as other significant points in space or on geometric objects. **Jack** combines a flexible object representation with a visually informative screen management facility to yield a user-friendly working environment.

In this paper we describe the goals and philosophies behind the design of the **Jack** interface, with emphasis on its operations for moving geometric objects. In Section 2, we discuss the problem of figure positioning and describe some desirable features of positioning systems. In Section 3, we discuss previous work in the area of figure positioning and three dimensional input techniques. In Section 4, we discuss the **Jack** interface and object representation. In Section 5, we describe the techniques which **Jack** uses to manipulate articulated figures. Finally, we summarize the approaches which **Jack** has taken to the various aspects of the problem of figure positioning.

2 Figure Positioning

Much of the recent research effort in computer animation has been aimed at developing new techniques for describing and generating motion, with the hope that this will provide the user with better control over the figures in a scene. These new techniques show great promise, but there has been little effort towards improving the fundamental interaction between the user and the objects. Work which has been done in this area has dealt primarily with arranging rigid bodies in space and has not properly considered the problem of manipulating complex articulated structures.

The task of manipulating articulated figures is very common, particularly in animation. For keyframe animation systems, the importance of static positioning is obvious. However, tools for static positioning have usefulness outside of the domain of scene composition and keyframe animation. For example, systems which generate motion sequences from dynamic simulation[1, 10, 19] must still provide a mechanism for describing initial configurations and providing force and velocity information.

The importance of direct manipulation for figure positioning is clearly obvious. The ability to get immediate feedback on how the parameters of a figure affect its appearance has an enormous impact on the usability of a system. In the case of three dimensional design and animation systems, this translates to the ability to virtually reach into the environment and move objects

and figures around at will.

The failings of most interfaces for geometric manipulation can be classified as follows:

- Improper visual feedback of the current state of the objects;
- Improper visual feedback of the results of transformations;
- Inability to properly anticipate the effect of input;
- Inability to easily describe transformations with respect to arbitrary reference frames.

Based on this, we observe the following characteristics of a useful object manipulation system.

- Changing the view should be so effortless it is almost transparent to the user. This principal comes from the fact that when a person is presented with an object to observe, he will pick it up, turn it around and look at it from several directions. The user of an interactive design system should be able to do likewise.
- The display should give intuitive real-time feedback on the movement. The screen should be visually informative, and the user should be able to drag objects to see the effect of different positions. Recent advances in graphics workstations have made this goal easier to achieve. This makes the process of trial and error much more effective.
- The user should be able to easily predict what motion of the input device will yield the desired object motion. Many systems have failed to provide usable manipulation tools because world space transformations are encoded in the space of the input device, making the object space results of the input difficult to anticipate.
- Many types of movement (global and local translation and rotation) should be available at the user's fingertips, and the user should be able to experiment freely and quickly with position and orientation. The response time should not be hampered by having to repeatedly specify axes or reference items from a menu or keyboard.

3 Previous Work

Recent work has attempted to relieve the burden of figure positioning by automating the task completely. Many exciting and promising techniques have been developed, based on inverse

kinematics[11, 9], dynamics[19, 1, 9], and constraint based optimization[3, 4, 20]. While these are very valuable techniques for generating realistic motion and precise positions, it can be difficult to manipulate a figure exclusively in this way. Such systems should be developed in conjunction with techniques for direct manipulation.

In the case of constraint satisfaction this is particularly true, since the constraints themselves are usually specified relative to positions and orientations of points on the objects or in world space. This problem can easily turn a potentially powerful animation system into a frustration for the user. This underscores the importance of a simple and intuitive way of moving objects and points around in a three dimensional environment.

3.1 3D Input Devices

Since the task of positioning articulated figures is inherently three dimensional, some attempts have been made at using three dimensional input devices. In particular, [3] and [16] describe experiments using a 3SPACE Digitizer², a magnetic device which senses the position and orientation of a hand-held wand. The experiments described in [3] conclude that the multiple degrees of freedom of the input device were difficult to control simultaneously, and that it became easier to control the device with some of the degrees of freedom disabled. This, along with the lack of proper visual feedback, has limited the usefulness of this system.

3.2 3D Input From 2D Devices

Several techniques have been developed for describing three dimensional transformations with a two dimensional input device, such as a mouse. Nielson describes techniques for mapping the motion of a two dimensional mouse cursor to 3 dimensional translation and rotation, based on the orientation of the projection of a world space coordinate triad onto the screen [12]. Bier describes transformations formed by reference coordinate frames called *skitters* and *jacks*[5]. The user first positions the skitters and jacks in space and then uses their position and orientation to specify certain transformations. The skitters and jacks may be positioned directly on faces or along edges, or freely in space. Their position and orientation are manipulated by a set of dials.

²3SPACE is a trademark of Polhemus Navigational Sciences Division, McDonnell Douglas Electronics Company

4 The Jack System

The **Jack** system is being developed to serve the various needs for display and manipulation of geometric objects in an animation and simulation environment. Its goal is to provide a consistent and easy-to-use programmer utility for displaying and manipulating articulated structures, and at the same time provide a consistent and convenient user interface across various applications. **Jack** is implemented on Silicon Graphics IRIS Workstations, using *mex*, the Silicon Graphics window manager. Its intended user community consists primarily of people with education in computer science with a basic understanding of robotics and geometric modeling.

Jack is available to programmers as a "vanilla" system which provides many useful built-in features. The system is easy to modify for a particular application by developing new operations. The built-in facilities include commands for changing the view, creating and saving objects, moving objects, tailoring the display, and recording and executing sequences of commands. The programmer interfaces with the system primarily by defining new commands. These new commands generally use the same facilities for picking and moving objects as the built-in commands. Thus the user interface remains consistent across various applications.

One of the most significant features of the **Jack** system is its facility for moving objects. This feature is implemented internally as a "movement operator" which allows the same basic facility to be used in every instance where a geometric transformation must be specified by the user. This operator works interactively using the mouse, so that the user may "drag" objects around by moving the mouse around on its pad.

4.1 Articulated Figures

Jack is primarily a user interface which controls the interaction with articulated figures represented in a system called **peabody**. The name **peabody** refers to both the internal data structure representing the geometric objects and the external language for describing and storing them. **Jack** can be viewed as a graphical editor for constructing and manipulating **peabody** files.

Peabody represents figures composed of rigid *segments* connected by *joints*, also under the influence of *constraints*. Joints connect segments through attachment frames called *sites*. A site is a local coordinate frame specified with respect to the base coordinate frame of the segment to which it belongs. Joints connect sites belonging to different segments within the

same figure. Constraints are pseudo-joints between arbitrary sites in the environment. The segment is the basic geometric primitive. The state variables of each segment represent its mass and moment of inertia, as well as its geometry, stored as a boundary representation.

The user treats figures as arbitrary collections of segments connected by joints, without necessarily imposing a hierarchy upon them. The user is encouraged to think of the geometric objects as an arbitrary graph of segments connected by joints. **Peabody** computes the global position and orientation of each segment internally by first computing a spanning tree of the environment. This tree need only be recomputed when a new joint or segment is created or deleted, i.e. when the topology of the environment graph is altered.

Since this tree is computed internally, the user does not have to think of a figure as a strict hierarchy with a specific root. This simplifies the operation of "re-rooting" a figure, either to attach a figure to another figure, or to change the point of attachment of a figure to the world coordinate frame. This scheme makes it easy to specify transformations with respect to arbitrary frames.

4.2 The User Interface

Jack displays the screen in a visually informative way by drawing a ground plane grid, giving a perception of the orientation of the world coordinate system. **Jack** draws the orthogonal projections of the figures in the scene on each of the coordinate axis planes. The projections are drawn in a darker color than the figures themselves, so they do not heavily distract from the scene. This gives an easily interpreted visualization of the arrangement of the figures. Since all three projections are closely placed on the screen, the user can quickly reference the orientation and relative placement of neighboring objects in the scene.

All aspects of the display are optional and may be disabled. The user may choose to display the vertices, edges, faces, or sites associated with each segment on an individual basis. The sites are displayed as labeled coordinate axes. The orthogonal projections of the individual segments may be disabled as well. This allows the display to be tailored to suit a particular application, since all forms of display may not be appropriate for all tasks.

Jack is a menu driven system, but most of the commands in the menus have several options which can be invoked through keystrokes as the command is being executed. **Jack** controls almost all of the interaction through a three button mouse. In general, pressing

and releasing the mouse buttons are different events, and many functions require the user to hold down a mouse button while moving the mouse around on its pad.

The execution philosophy of **Jack** is to select a high level operation first, and then select the operands. The user executes commands from a menu, such as "move figure," then he pick the appropriate object by pointing at it with the mouse. Finally, he specifies the *value* of the operation, i.e. a transformation, which is usually manipulated interactively. Most operations such as moving are terminated by hitting a special key, such as the escape key.

4.3 Moving the View

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

5 Manipulating Objects

Internally, the **Jack** movement operator operates on two homogeneous transforms: the *reference* transform R , which is a global transformation, and a *relative* transform T , which is specified with respect to the reference R . The product of R and T is thus a global transformation G . The movement operator also operates on a global axis, defined in terms of a reference point and a direction vector, both specified with respect to the reference frame R . The purpose of the operator is to change the T transform. The operator can perform three types of movement: rotation around the axis; linear translation along the axis; or planar translation in the plane defined by the reference point and the axis vector. During the course of the movement, R remains fixed, and T changes so that the G transform rotates or translates appropriately. T is the *output* of the operation.

Some type of "action" is usually performed as this is done, such as updating the global position and orientation of figures and joints in the scene. This basic operator is used in many situations throughout **Jack** to move various things such as figures and joints, as well as individual vertices, edges, and faces of geometric primitives.

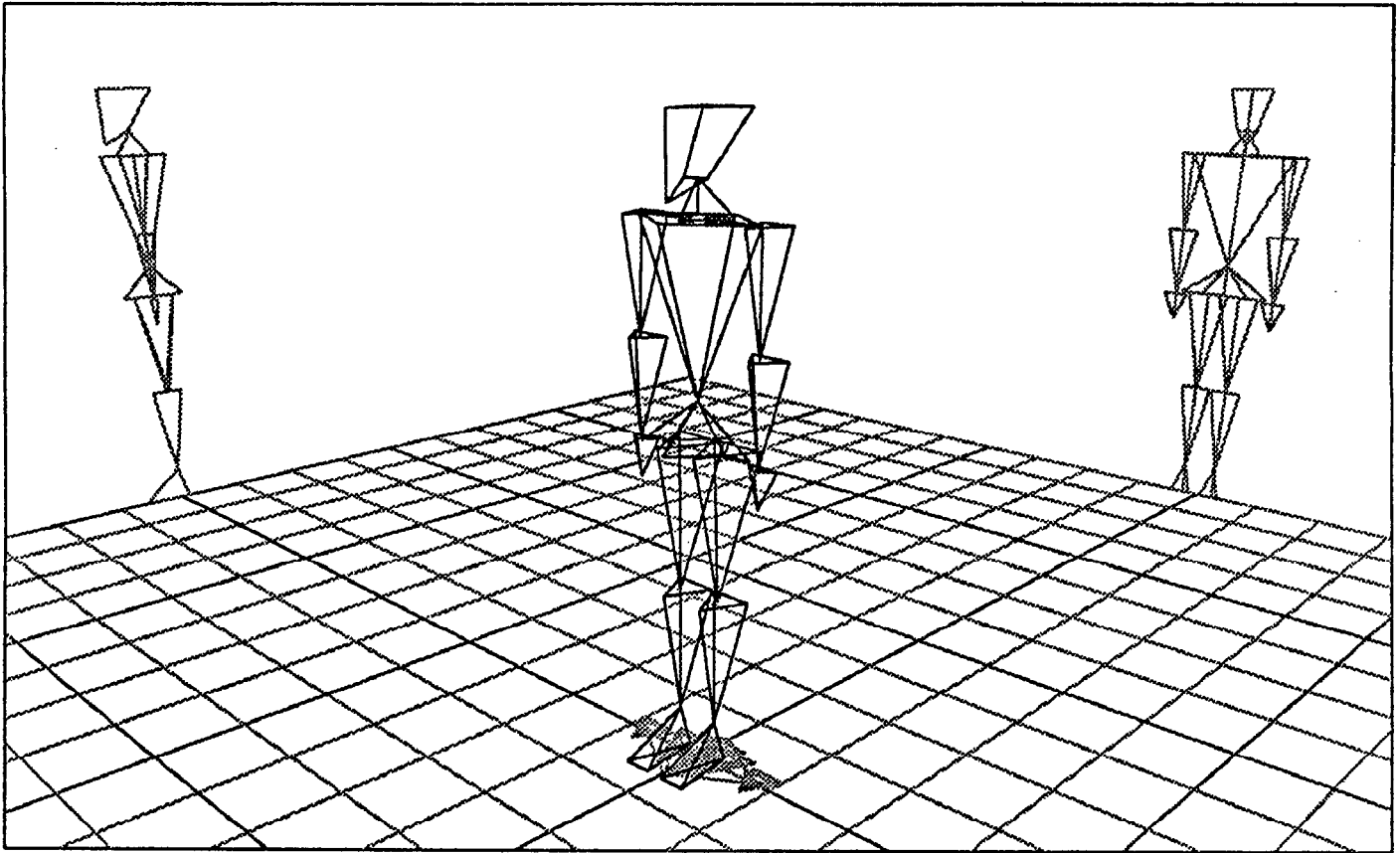


Figure 1: The Jack screen, with a human figure

The global and reference transforms and the rotation axes are selected internally by the movement operator depending upon the object being moved. The user sees them only as the frame of reference and resulting transforms, and the user may select from various rotational and translational axes by pressing various keys during the movement. The generality of this mechanism makes it possible to use this operator for many types of movement operation.

5.1 The Mouse Line

The rotation and translation of the global transform G is determined interactively by the ray in the world coordinates which is cast through the location on the screen where the mouse cursor lies. This line in space is referred to internally as the *mouse line*, and it can be easily computed by an inversion of the viewing transformation. *Jack* determines angles by intersecting the mouse line with the plane defined by the reference point and axis direction. It determines distances by intersecting the mouse line with the axis itself.

5.2 Translation

Under normal operation, translations along the x , y , and z axes of an object's local coordinate frame are encoded in the left, middle, and right mouse buttons, so that pressing down any mouse button enables translation along that axis. During the movement operation, when the user presses a button, translation is enabled along that axis. When the user presses two mouse buttons, translation is enabled along those two axes, i.e. in the plane spanned by those axes. Since the mouse is a two-dimensional device, it is not possible to translate along three axes simultaneously, so pressing three buttons at once has no effect.

When the buttons go down, a vector is drawn in world coordinates describing the translational axes. As the user moves the mouse, the object moves in world coordinates so that its location lies under the mouse cursor. As the figure moves, its global and local coordinates are displayed on the screen so that the exact position is available as well.

Translation along a single axis may be achieved by pressing only one button. The mouse may move anywhere on the screen, but the translation is restricted

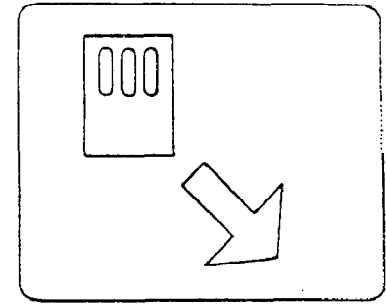
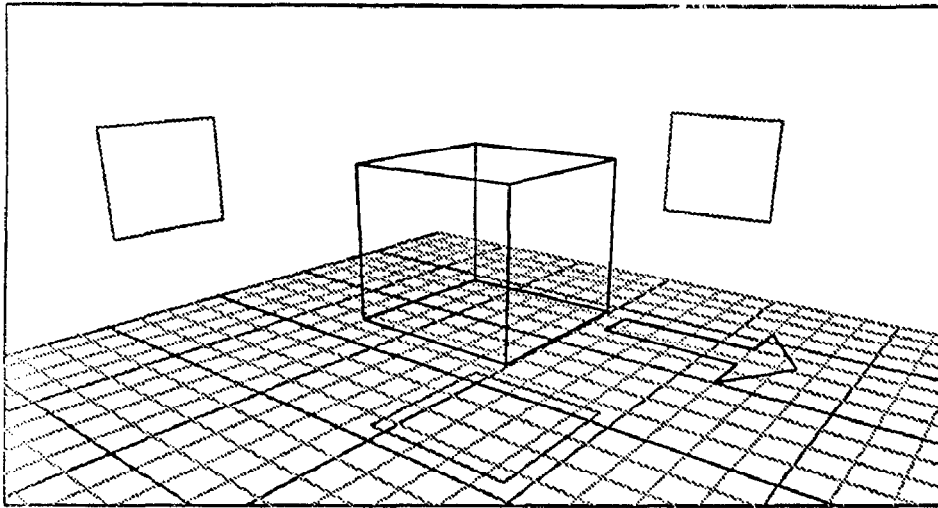


Figure 2: Translation

to a particular axis is space, which maps to a line on the screen. Therefore it is not possible to move the location of the figure along the axis such that it appears under the mouse cursor. In this case, **Jack** determines the position of the figure from the point along the translational axis which is nearest to the mouse line. **Jack** repeatedly repositions the mouse cursor so it lies along the line of translation.

5.3 Rotation

Rotation is accomplished in a similar way, by requiring the user to move the mouse around in circles on its pad. The three mouse buttons are encoded as rotation around the x , y , and z axes. When the user presses down on a button, a "wheel" is displayed at the origin of the figure describing the rotational axis. This wheel lies in the plane in which the rotation is to take place, with the origin of the wheel at the rotational axis. Then a vector is drawn from the current mouse line intersection with this plane, and as the user moves the mouse around in this plane, the object stays fixed with respect to this reference vector. Therefore, if the user moves the cursor around in circles on the screen centered at the origin of the object, the object rotates around the three dimensional axis.

5.4 Joints

The movement operations are helpful for positioning figures in world space, but usually an articulated figure is composed of a number of joints having specific degrees of freedom. **Peabody** provides a mechanism for associating arbitrary rotational and translational

degrees of freedom with a joint. In this case, the transformation at the joint is restricted to these axes.

Internally, when transforming a joint which has specific degrees of freedom, the rotational and translational axes are restricted to be the revolute or prismatic degrees of freedom of the joint. Externally, the same type of movement is employed, but it is simply encoded in the mouse buttons in a different way. A properly defined joint will never have more than six degrees of freedom. The first three axes are encoded in the left, middle, and right buttons, respectively. The second three are activated by holding down the shift key.

This mechanism has left unspecified how the joint displacement affects the position of the figure. When a joint is adjusted, one segment remains fixed with respect to the world frame and the other moves. In a strict hierarchy, this is a simple matter since there is a well defined "proximal" and "distal" segment. But **peabody** represents figures as a general graph, so it is ambiguous which segment remains fixed and which moves. In this case, **peabody** takes the proximal and distal segments from the underlying spanning tree of the environment, so that the segment which remains fixed is the one which is on the side of the figure which is rooted to the world. Since it is very easy to re-root a figure, it is a simple matter to arrange a transformation keeping any arbitrary segment fixed in space.

5.5 Reference Frame

The transformations described above are local to the coordinate frame being transformed, so that if an object is rotated and then translated, the translation will

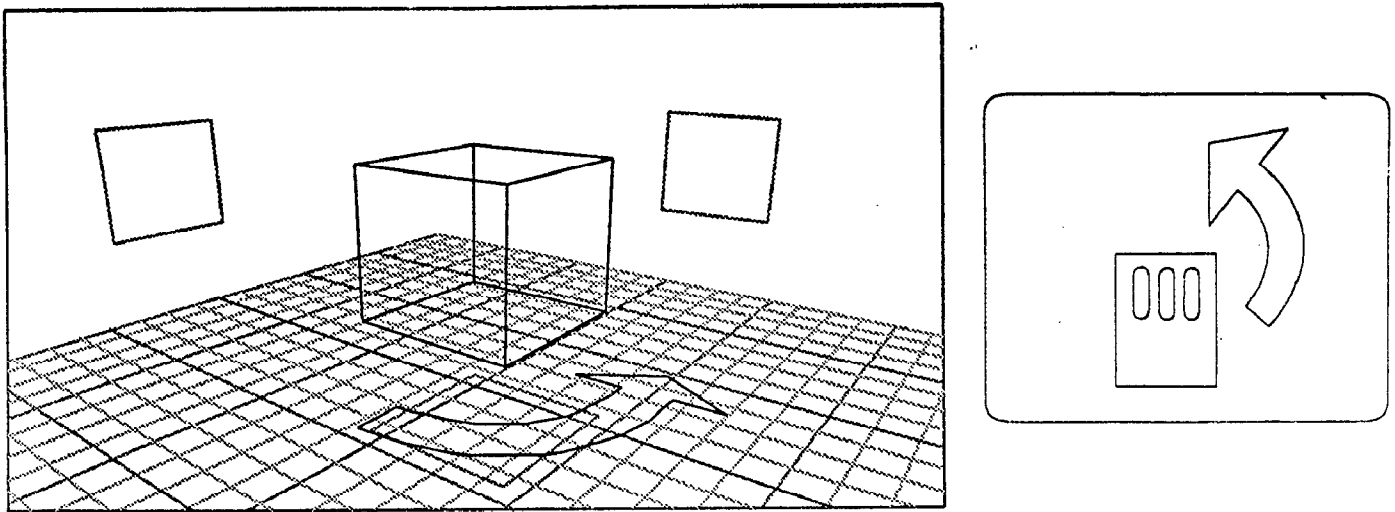


Figure 3: Rotation

be with respect to the rotated axes. The transformations may be specified with respect to the parent transformation by holding down the shift key. In the case of a joint, the parent is the segment on the proximal side of the joint. In the case of a figure, the transformation is taken with respect to the site to which the figure is connected. If this site is attached to the world coordinate frame, the transformation is relative to the world. If a figure is rooted to another figure, the motion is relative to that figure.

Since the control and shift keys are close together on the keyboard, they are easy to press in conjunction. Thus, the user may specify global or local rotation or translation by holding down some combination of the control and shift key and the three mouse buttons. The motion continues in this fashion until the user presses the control key, ending the transformation. The transformation may also be aborted by typing ^C.

5.6 Observations

A seeming drawback of this technique is that the user must remember the current orientation of the x , y , and z axes. However, the translation and rotation icons, along with the coordinate axis projections, give the user a simple means of determining this information quickly. Typically, the user has a particular direction in world coordinates in mind when he initiates a transformation. Since the icons are displayed as the keys are pressed and before the motion begins, the user can easily cycle through the available axes to select the appropriate set.

For instance, the user may want to “turn the fig-

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

Another seeming drawback of this technique is the inability to translate or rotate an object along an axis parallel to the line of sight, since in this configuration, small differences in the screen coordinates of the mouse may correspond to large distances in world coordinates. However, this is a transformation which the user should be discouraged from performing anyway. The first prerequisite for manipulating a figure, by computer or by hand, should be to position the figure in a convenient view.

5.7 End Conditions

The Jack movement operator provides a flexible way of manipulating transformations interactively, but it can be difficult to specify precise angles and distances in this way. This can be achieved in several ways. The most straightforward is to simply enter the coordinates from the keyboard, which can be done by typing ^K during the motion, after which the user will be prompted for the desired angles or distances. This need only be done when a specific known distance or angle is needed.

While a motion is taking place, the user may “snap”

key	effect
left mouse	x translation/rotation
middle mouse	y translation/rotation
right mouse	z translation/rotation
CNTL	rotation
SHIFT	transform w.r.t. parent

Figure 4: Key bindings during movement

the transformation to a particular face, edge, or vertex. To snap a transform, the user hits the snap key, ^Z, then specifies whether to snap to a vertex, edge, or face, and then picks the appropriate item. Then the transform is moved to tangency with the selected item. This tangency is accomplished by translating the transform to the desired vertex, edge, or face along the shortest possible distance, without rotation.

It is also possible to snap a transform to the direction of an edge or face. To do this, the user first orients the transform so that it is aligned approximately with the desired edge or face. He then hits the snap key as before, but specifies to snap to the *orientation* of an edge or face, rather than to tangency. Then the user is prompted to pick the appropriate item, and the snapping mechanism takes the local coordinate axis of the transform which is closest to the desired orientation and adjusts the transform so that that axis is aligned with the edge or face.

6 Future Work

One of the critical issues in the animation of human movement is the enforcement of joint limits. These limits should be enforced during the direct manipulation of joints, and the interaction should display rich visual feedback on the limits. Currently, Jack does not provide good visual feedback on joint limits, although the limits are enforced during manipulation.

The problem of joint limits is particularly difficult in the case of three degree of freedom ball joints such as the human shoulder, since the limits cannot be accurately represented by individual limits on the angles of each of the rotational axes. A good interactive system for manipulating such joints should give proper feedback on how the joint can be legally manipulated given its current configuration.

7 Conclusion

Jack provides a flexible and easy-to-use interface for

displaying and manipulating complex articulated figures. The windows, orthographic projections, and movement icons provide good visual feedback on the current state of the environment and the effect of the input. Since all three dimensional position and orientation information is provided by the same operator, there is a consistent interface for all aspects of the system. Since the movement operator generates homogeneous transformations based on the screen location of the mouse cursor, it is easy to anticipate the effect which movements of the input device will have on the objects. Since **peabody** is a very general mechanism for representing articulated structures, it is very easy to define attachment and reference points to specify transformations with respect to arbitrary frames.

References

- [1] Armstrong, W. W., and Mark Green, "The Dynamics of Articulated Rigid Bodies for Purposes of Animation," *The Visual Computer* 1, No. 4, 1985.
- [2] Badler, Norman I., Jonathan D. Korein, James U. Korein, Gerald Radack, Lynne S. Brotman, "Positioning and Animating Human Figures in a Task-Oriented Environment," *The Visual Computer* 1, No. 3, 1985.
- [3] Badler, Norman I., Kamran Manoochehri, David Baraff, "Multi-Dimensional Input Techniques and Articulated Figure Positioning by Multiple Constraints," In *Proceedings of 1986 Workshop on 3D Interactive Computer Graphics*, (Chapel Hill, NC, October 23-26, 1986), ACM, New York, 1987.
- [4] Badler, Norman I., Kamran Manoochehri, and Graham Walters, "Articulated Figure Positioning By Multiple Constraints", *Computer Graphics and Applications*, Vol. 7, No. 6, June, 1987.

- [5] Bier, Eric "Skitters and Jacks: Interactive Positioning Tools," In *Proceedings of 1986 Workshop on 3D Interactive Computer Graphics*, (Chapel Hill, NC, October 23-26, 1986), ACM, New York, 1987.
- [6] Bier, Eric, "Snap-Dragging," *Computer Graphics* 20, No. 3, 1986.
- [7] Britton, E.G. J.S. Lipscomb, M.E. Pique, "Making Nested Rotations Convenient for the User." *Computer Graphics* 12, No. 3, 1978.
- [8] Forrest, A.R., "User Interfaces for Three-Dimensional Geometric Modeling." In *Proceedings of 1986 Workshop on 3D Interactive Computer Graphics*, (Chapel Hill, NC, October 23-26, 1986), ACM, New York, 1987.
- [9] Girard, Michael and A.A. Maciejewski, "Computational Modeling for the Computer Animation of Legged Figures," *Computer Graphics* 19, No. 3, 1985.
- [10] Isaacs, Paul M. and and Michael F. Cohen, "Controlling dynamic simulation with kinematic constraints", *Computer Graphics* 21, No. 4, 1987.
- [11] Korein, James U., "A Geometric Investigation of Reach," MIT Press, Cambridge, MA, 1985.
- [12] Nielson, G.M. and Dan Olsen, Jr., "Direct Manipulation Techniques for Objects Using 2D Locator Devices," In *Proceedings of 1986 Workshop on 3D Interactive Computer Graphics*, (Chapel Hill, NC, October 23-26, 1986), ACM, New York, 1987.
- [13] Phillips, Cary, "Programming With Jack," unpublished programmer's manual.
- [14] Phillips, Cary, "Using Jack," unpublished user's manual.
- [15] Pique, M.E. "Semantics of Interactive Rotations." In *Proceedings of 1986 Workshop on 3D Interactive Computer Graphics*, (Chapel Hill, NC, October 23-26, 1986), ACM, New York, 1987.
- [16] Schmandt, Christopher, "Spatial Input/Display Correspondence In a Stereoscopic Computer Graphics Workstation," *Computer Graphics* 17, No. 3, 1983.
- [17] Shneiderman, Ben, *Designing the User Interface*, Addison Wesley, 1987.
- [18] Stallman, Richard, *Gnu Emacs Users Manual*.
- [19] Wilhelms, Jane "Using Dynamics for the Animation of Articulated Bodies Such as Humans and Robots," *Proceedings of Graphics Interface '85*, Montreal, 1985.
- [20] Witkin, Andrew, Kurt Fleischer, and Alan Barr, "Energy Constraints on Parametrized Models," *Computer Graphics* 21, No. 3, 1987.