

**Multi-Dimensional Input Devices and
Interaction Techniques for a Modeler-Animator**

Melissa Y. Gold

Department of Computer Science
Brown University
Providence, Rhode Island 02912

Technical Report No. CS-91-16
July 1990

Multi-Dimensional Input Devices and Interaction Techniques for a Modeler-Animator*

Melissa Y. Gold

Abstract

As powerful computers become less expensive, applications such as interactive three-dimensional modeling and animation systems, that are currently used by relatively few people, will become available to a much wider audience. It is therefore important to find ways to make these systems easier and more intuitive to use. One of the problems with current systems is that they typically provide users with one- or two-degree-of-freedom input devices with which to perform three-dimensional interaction tasks. This paper explores currently available multi-dimensional input devices as an alternative. It also describes experiments with using a six degree-of-freedom input device for the tasks of point location, translation, rotation, camera placement, and path drawing in a modeling and animation system. However, even with multi-dimensional input devices, certain tasks, such as precisely placing one object in relation to another, are difficult to accomplish using only basic transformations such as translation and rotation. Accordingly, this paper also investigates a snapping technique, independent of input device, which allows users to position one object at a specific point on the surface of another.

July, 1990

*This research was sponsored, in part, by grants from IBM, NCR, and Sun Microsystems

Table of Contents

1. Introduction	1
1.1 The Four-Dimensional Modeler: Its Use as a Testbed.....	3
1.2 Classifying Input Devices and Interaction Techniques.....	4
2. Multi-Dimensional Input Devices.....	7
2.1 Choosing a Multi-Dimensional Input Device.....	8
2.2 Notes About the Polhemus: Hype Versus Reality.....	15
3. Interaction Techniques in Moo.....	17
3.1 Extending Traditional Techniques Using New Devices.....	17
3.1.1 Review of Traditional Interaction Techniques.....	17
3.1.2 Previous Work Using Multi-Dimensional Devices.....	22
3.1.3 Using the Polhemus in Moo.....	25
3.2 Snapping: Extending a Technique.....	32
3.2.1 Snap-Dragging.....	33
3.2.2 Implementing Snapping in Moo.....	35
4. Extending this Work.....	43
5. Summary	44
6. Acknowledgements.....	47
7. Bibliography.....	48
8. Appendix	51

1. Introduction

"The screen is a window through which one sees a virtual world. The challenge is to make that world look real, act real, sound real, feel real."

- I. E. Sutherland, 1965

"Imagine a time far into the future, when all knowledge about our civilization has been lost. Imagine further, that in the course of planting a garden, a fully stocked computer store from the 1980's was unearthed, and that all of the equipment and software was in working order. Now, based on this find, consider what a physical anthropologist might conclude about the physiology of the humans of our era? My best guess is that we would be pictured as having a well-developed eye, a long right arm, a small left arm, uniform length fingers and a "low-fi" ear. But the dominating characteristics would be the prevalence of our visual system over our poorly developed manual dexterity."

- W. Buxton, 1986

Sutherland's vision of the future of computer graphics has only partially been realized, as Buxton's tale demonstrates. That is, today's technology is able to produce very realistic, even photo-realistic pictures on a computer; hence the anthropologist's conclusion about our highly developed visual system. However, as the anthropologist also noticed, computer graphics has yet to meet the challenge of allowing users to effectively interact with these pictures, as a means of interfacing with the objects the pictures portray.

There are many examples of computer graphics applications in which the user's ability to interact with the pictures is equally as important as the quality of the pictures themselves. Under the general terms *virtual worlds*, *scientific visualizations*, or *electronic books*, these applications typically present users with a view of an object or objects from the real world, (or the imagination) which they are able to move through, manipulate, change, or study. Such applications include architectural walkthrough programs [Brooks87], which allow users to proceed on tours of virtual buildings before they are built; molecular docking systems [Ouh-young89], which allow biochemists to manipulate pictures of molecules and proteins in order to study their behavior; driving simulators [Deyo89], which allow drivers to see

and feel what it is like to be behind the wheel without ever getting into a real car; "parts explosion" programs [Zelevnik89], which let users simulate taking apart their car engines, for instance, without ever getting their hands dirty; interactive modeling and animation programs [Conner89], which allow users to create their own simulated worlds; and CAD/CAM (computer-aided design/computer-aided manufacturing) systems for engineering, circuit design, etc. [Robinson90].

Most of these applications are designed for non-computer experts. As such, it is extremely important that they provide interfaces which are natural, intuitive, and easy to use. This means that a person walking through a virtual building should ideally be able to proceed by drawing upon her knowledge of how to walk through a real building, rather than her knowledge of how the computer system works. Similarly, any one of the applications mentioned above should force the user to look beyond the methods being used to provide simulation, and instead to look towards what is actually being simulated.

However, this problem is inherently difficult, given that even though what the user is looking at may appear "real" or "three-dimensional," it is actually being displayed on a two-dimensional surface. Moreover, most commonly used input devices, such as mice, joysticks, tablets and dials, provide at most two dimensions of information. Researchers are beginning to look into ways around these difficulties. Some are building three-dimensional displays [Wang90]. Others are designing alternative input devices which provide more than two degrees of freedom [Zimmerman87], and still others are experimenting with ways to provide better interaction techniques using the current display and input technology [Bier90].

The remainder of this paper describes research aimed at making the interaction techniques used in a specific application, a three-dimensional modeler and animator, more intuitive and easy to use. This research was performed on top of an existing system, so specialized display hardware was not explored. However, alternate physical input devices and interaction techniques using these and more common devices were examined.

1.1 The Four-Dimensional Modeler: Its Use as a Testbed

This research was written on top of the Brown University Graphics Group's four dimensional modeler (the fourth dimension is time), which is affectionately known as "Moo." Moo is an interactive program which was written as a tool for artists, scientists, students, and professors to use to produce computer animations. As such, it is important that any feature added to Moo be easy and intuitive enough for people without specific computer graphics or computer science knowledge to use.

Briefly, Moo allows users to piece together a few basic shapes (cube, cylinder, cone, sphere, torus) using constructive solid geometry to create more complex three-dimensional scenes. Much of the interaction is provided through a menu-driven interface, which for example, lets the user create, select, name, change the color of, change the visibility of, and delete objects. It also lets the user save the currently modeled scene, read in a previous scene, select from a number of views, and much more. Along with a mouse, interaction is performed using a tablet, a set of dials, and a keyboard. For a more detailed description of Moo and the larger software system it is part of, see [Conner89] and [Fisk89].

Moo is an ideal platform for experimenting with ways to provide natural and intuitive interaction. For one, as a modeling program, its usefulness is measured entirely in terms of how well it enables users to communicate their ideas to the computer. Without this ability to communicate, the program could not exist. In some sense the pictures are merely feedback for the interaction; they are not important by themselves. This is not true of all interactive programs. For example, consider building walkthrough applications like the one mentioned above. It is easy to see how even without the ability to interact with them, pictures of not-yet-built buildings could be very useful.

Another reason why a modeling program is a good place to study interface techniques is that the types of interactions generally necessary in a modeler are both basic and general. This means that interaction techniques implemented in a modeling program can be easily applied to other applications. As an example, consider the problem of letting users rotate objects in a modeling program. The methods developed for this solution can

immediately be applied to, say, a driving simulator, where the problem of what happens when the user turns the wheel involves rotation.

1.2 Classifying Input Devices and Interaction Techniques

Before going any further, it is important to define the terms *input device* and *interaction technique*, and their relationship, more clearly.

Input device is the term applied to any piece of hardware used to send information to a computer system. To sort through the many types of devices available, numerous classification schemes have been devised. For example, devices may be categorized by what mechanical or electrical properties they sense. Or, they may be grouped based on whether the data they return is discrete or continuous, relative or absolute, or direct or indirect. Yet another classification is concerned with how many dimensions of input a device provides. Figure 1-1 shows several common input devices grouped according to these properties. The remaining sections of this paper are mostly concerned with number of dimensions as a classification scheme, looking especially at devices which are greater than two-dimensional.

		Number of Dimensions							
		1		2		3+			
Property Sensed	Position	Rotary Pot	Sliding Pot	Tablet	Light Pen	Joystick	3-D Joystick	Sensing Types	
				Touch Tablet	Touch Screen		Polhemus		
	Motion	Continuous Rotary Pot	Treadmill	Mouse		Trackball			
		Thumbwheel							
	Pressure	Torque Sensing	Pressure Pad	Isometric Joystick		Spaceball			

Figure 1-1. Classifications of Input Devices [Buxton83]

At a more abstract level, input devices can be categorized based upon the type of functionality they provide from a software perspective. The categories of this grouping are called *logical devices*. There are five common types of logical devices [Foley90]: the *locator*, which specifies position or orientation, the *valuator*, which provides a single number, the *pick*, which selects a displayed entity, the *keyboard*, which inputs a character string, and the *choice*, which selects from a set of possibilities. Logical devices, which can be implemented in software as well as hardware, are important to designers of interactive systems, since they allow devices within the same category to be substituted for one another without changing the functionality of the system. This substitution is not complete, however, because even though two devices may be interchangeable technically, users may find that one facilitates their task while the other makes it nearly impossible.

The idea that two devices in the same logical class may not be equivalent in every sense provides the basis for much of the work described in this paper. That is, the goal was not necessarily to provide Moo with new

functionality (although that happened as well), but rather to make the existing functionality easier to use. Further, the intention was not to find one device which could be called the best at all tasks, but rather to experiment with the strengths and weaknesses of particular devices. A very simple example from Buxton [Buxton86] helps make these concepts clear. Consider, he says, two children's toys: the *Etch-a-Sketch*, and the *Skedoodle*. Both, in computer terms, are drawing applications that allow the user to control a cursor which traces out pictures on the screen. The only difference is in the control knobs. While the *Etch-a-Sketch* has two dials, one which controls left-right motion of the cursor and the other which controls its up-down motion, the *Skedoodle* has a single joystick which is used to control all motion. Obviously, these are equivalent logical devices. However, any user who is trying to draw orthogonal lines with the *Skedoodle*, or diagonal lines with the *Etch-a-Sketch*, will tell you that they are very different indeed.

This brings us to define the highest level concepts used in characterizing input, those of *interaction tasks* and *interaction techniques*. These classifications focus on the user's goals and how the user achieves those goals, respectively. Foley [Foley90] defines four common interaction tasks: *position*, *text*, *select*, and *quantify*. This paper will not be so specific, however, and will refer to more abstract terms, such as rotate, translate, scale, draw, position, etc., as interaction tasks as well. Interaction techniques encompass both what type of device is used, and the process by which the task is completed.

Of course there are many possible ways to perform a specific interaction task. For example, consider the task of translation in two-dimensions. One method of performing this task is to type in an object name and an amount and direction to translate by. This method is very good when the user requires that object placement be very accurate, such as when she is designing objects that are to be manufactured. However, in cases where such precision is unnecessary, typing becomes tedious. In these cases, it may be easier and faster to select an object and drag it to a new location using the mouse. This type of interaction is known as *direct manipulation*, and it typically provides intuitive ways for a user to perform a task. Furthermore, if the user is willing to trade off some speed for accuracy, a combination of direct manipulation and constraint-based techniques might be best, whereby an additional set of constraints to be satisfied by parts of the model is specified [Foley90]. This

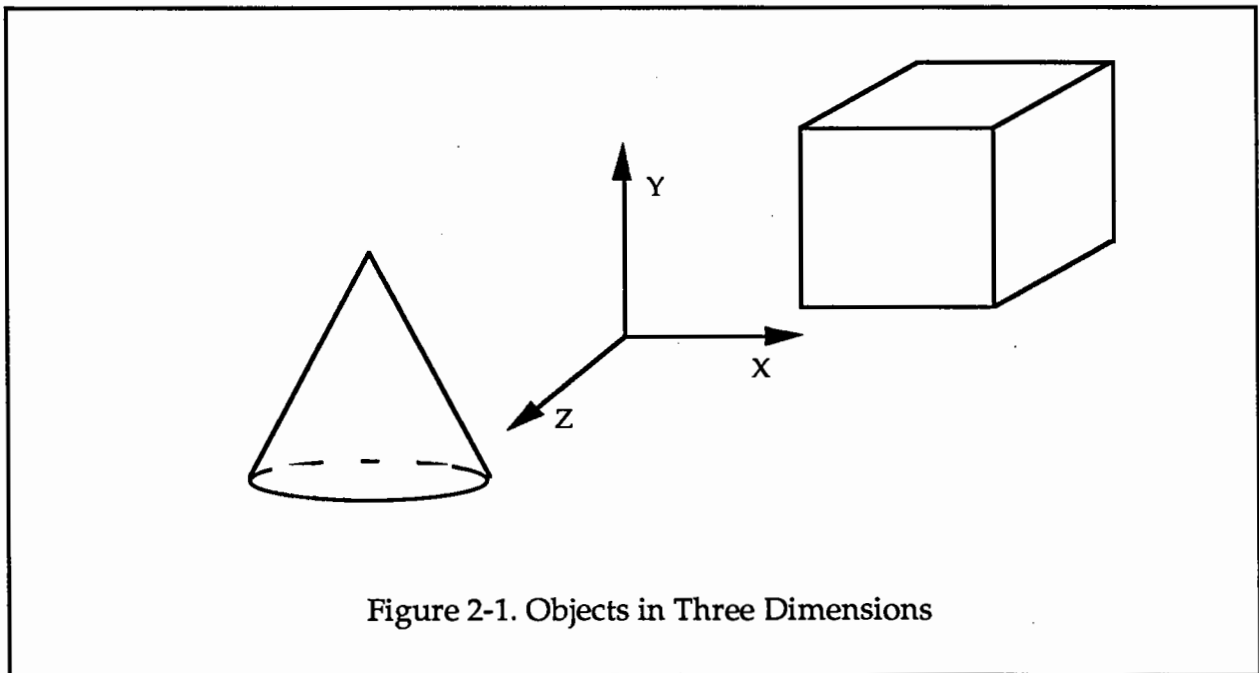
project is concerned with adding naturalness and ease-of-use to Moo, since the system already contains ways to specify tasks precisely. Thus, techniques which rely upon direct manipulation are of most interest for the purposes of this paper.

Moreover, another assumption made in this paper is that physical input devices which have three or more degrees of freedom are better than virtual devices with the same functionality. This is because with a physical multi-dimensional input device, the user can naturally manipulate all of the degrees of freedom of the device. On the other hand, even if a virtual device provides the user with many degrees of freedom, a physical device must ultimately be used to control the virtual one. As such, it is more natural to control a multi-dimensional device directly than to deal with the encoding from the physical device to the virtual one.

Using the terms of this section, the aim of this project has been to examine both new devices and new software methods in order to discover better interaction techniques for the interaction tasks found in a four-dimensional modeling system.

2. Multi-Dimensional Input Devices

Most interaction tasks in Moo involve thinking in three and four dimensions. However, devices such as mice, dials, tablets, and keyboards traditionally used to provide input to this and similar applications are at the most two-dimensional. A short example will serve to identify the specific problem inherent in this mismatch. A user is beginning to build a scene, and has created a cone and a cube, as in Figure 2-1. Now she wishes to translate the cone to be on top of the cube. How should this be accomplished, assuming that only the traditional input devices listed above are available? Obviously, none of the devices can be used to specify all of the directions of the translation at once. For example, the mouse can be used to move the cone in the two-dimensional plane of the screen; however, it is difficult to specify movement "into" or "out of" the screen. This is true of the tablet as well. Dials, on the other hand, can be used to express the complete motion, but one dimension of movement is specified at a time. Or, the keyboard can be used to input something to the effect of "Translate Cone by 3 in X, 2 in Y, and -4 in Z."



There have been many other schemes devised to provide three or more degrees of freedom using input devices of two dimensions or less, such as pressing the mouse button to indicate desire to move in a third dimension. Some of these techniques will be described later on in this paper. However, no matter what the procedure, specifying a three-dimensional interaction technique with a two-dimensional device is inherently unnatural, as these schemes make it impossible for the user to ignore that she is manipulating an input device and a computer, rather than a three-dimensional object. Thus, it seems that users would be able to work more effectively if they were provided with an input device which has at least the same number of degrees of freedom as the task at hand. Users could then concentrate on the task instead of the procedures necessary to map two dimensions of input to a three-dimensional scene.

2.1 Choosing a Multi-Dimensional Input Device

Deciding that Moo users could potentially benefit from the addition of a multi-dimensional (more than two-degrees of freedom) input device was the easy part; deciding which one to use was more difficult. Part of the problem is that whereas researchers have been experimenting with such devices for years, only recently have they become commercially available. Thus, much of what has been done with multi-dimensional input devices, including this work, is purely experimental; there are no well-developed methods for using them. (Note that in the remainder of this section, only devices which provide more than two dimensions of input will be discussed. Accordingly, the term *input device* will hereafter be used instead of multi-dimensional input device, unless otherwise noted.)

Several criteria were used to decide upon an input device for Moo. The first was that the device itself be commercially available, so that we could focus on building interaction techniques rather than hardware. Secondly, cost was an important consideration, not only because of our own modest budget, but also to ensure that any benefits of this work were not out of the reach of most users. Finally, the most important requirement of the device was that it might be used in as many parts of Moo as possible. A list of particularly important applications was compiled with the assistance of a group of Moo users. The device would be used as:

- a means to translate, rotate, and scale selected objects.
- a way to control the camera position.
- a control for a three-dimensional cursor, which could be used to select objects.
- a way to draw paths in three-dimensions.
- a device for manipulating points on spline paths and spline patches.
- an instrument for controlling tools in advanced modeling. For example, tools such as a chisel on wood or a "molder/finger" in clay.
- a way to digitize objects.

According to these criteria, there were three input devices that seemed potentially promising for Moo: the Spaceball, the DataGlove, and the Polhemus 3Space Isotrack (hereafter referred to as simply the Polhemus). Each of these is described below.

The Spaceball [Foley90] [SpatialSystems90] (see Figure 2-2) is a device consisting of a tennis-ball sized sphere mounted on a base. Sensors inside the sphere detect both pressure on and motion of the sphere's surface. (However, note that the ball itself does not move.) Thus, a Spaceball provides the user with six degrees of freedom at once, three translational and three rotational. It also has a programmable set of buttons, which can be used, for example, to restrict the input to one dimension at a time.

The Spaceball is available from several vendors, and at approximately \$2800, it falls within our budget. Furthermore, it seems to be an improvement over the dials and the mouse as a means to quickly translate, rotate, and scale objects, and a natural way to drive a camera through a scene.

For interaction tasks such as three-dimensional positioning and orienting, the advantages of a Spaceball over traditional two-dimensional input devices are clear. When using a Spaceball, users can manipulate an on-screen object as if it were the sphere in their hand. Pushing on the ball in one direction will cause the object to move similarly; the amount of pressure applied determines how far the object moves. Despite this advantage over traditional devices, the Spaceball has some shortcomings. Instead of pushing on an object to place it, as required when using a Spaceball, users find it is more natural to pick it up and place it in its new position. First time users report that it is not immediately intuitive, and that "it is considerably more difficult than learning to use a normal mouse" [Helliwell89].

Moreover, pushing and twisting a sphere does not seem to be an instinctive way to draw a path in three-space, nor to control abstract advanced modeling tools. Finally, it simply can not be used to digitize real objects. These shortcomings were the reason the Spaceball was not chosen.



Figure 2-2. A SpaceBall [SpatialSystems90]

The DataGlove [Foley87] [Foley90] [Zimmerman87] (see Figure 2-3) is an instrumented glove worn by the user. The fingers of the glove are laced with fiber-optic cables, which are anchored at an interface board at the wrist. At one end of each cable is a light-emitting diode (LED) and at the other end is a phototransistor. When one of these cables is bent, less light is able to pass through it. In this way the DataGlove senses the finger positions of the user. On the back of the glove is a Polhemus sensor, which provides information about the position and orientation of the user's hand. The Polhemus, an input device by itself, is discussed in more detail below.

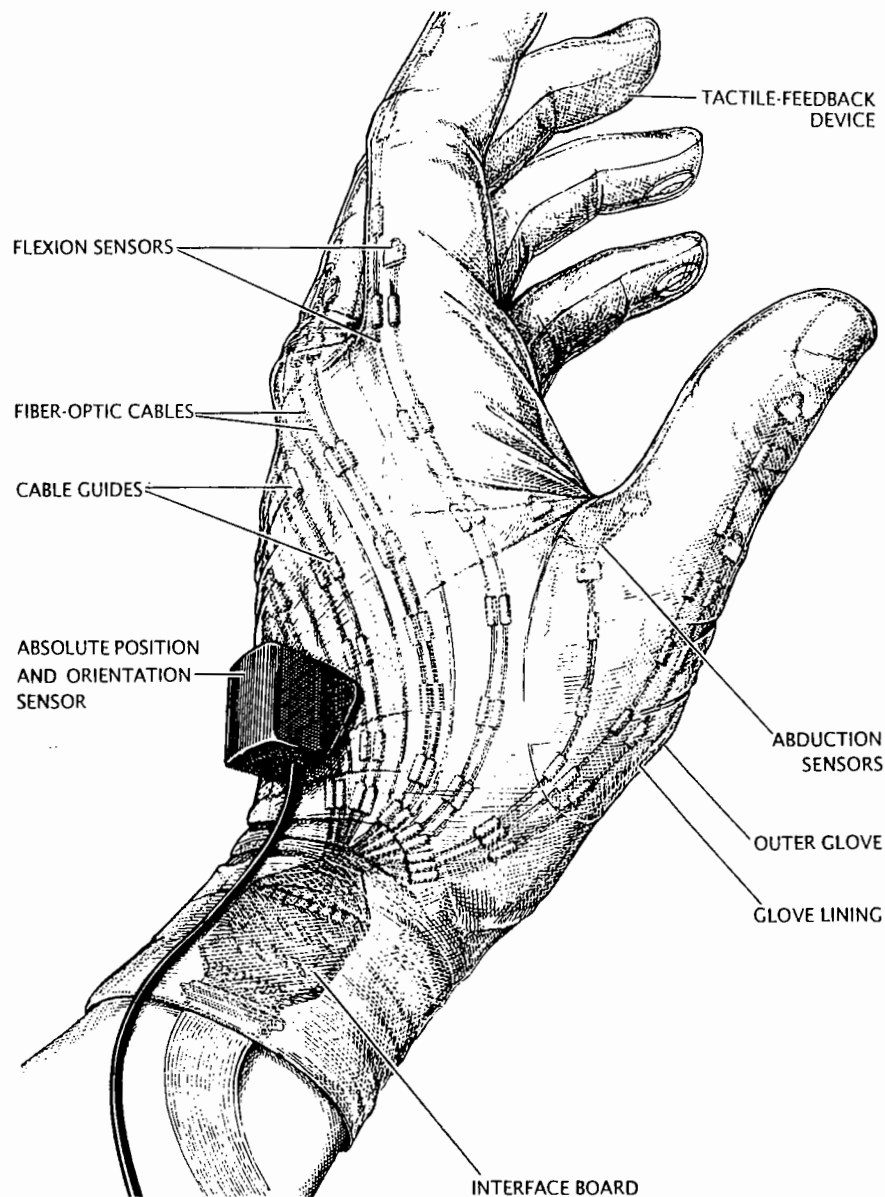


Figure 2-3. The DataGlove [Foley87]

In theory, for tasks that involve direct manipulation of objects, the DataGlove is a natural choice. Typical applications provide feedback by simulating the glove on the screen. Thus, in order to move an object on the screen, the user would simply “grab” the object as if it were in front of her, physically move her hand to another location, and let go. However, in practice there are several disadvantages to the DataGlove, at least as it is

currently implemented. For one, small changes in finger position are not detected. Thus, any system using the glove must rely on recognition of very distinct gestures in order to provide reliable input. Because of this, the user has to place her hand in very explicit, even unnatural positions, making the glove substantially less intuitive. Moreover, since the glove does not measure the actual position of the fingertips, it is difficult to precisely simulate grasping tasks. Also, unlike the SpaceBall, the user is not holding anything. This lack of tactile feedback not only lessens the effectiveness of the interaction, it also means a user's arm will get tired more quickly, since it must move around in the air with nothing to rest upon. Another problem which becomes important in a multi-user system such as Moo is that each user's hand is different, so recalibration must be performed often. Also, a single-sized right-handed glove may not be right for everyone. The final drawback is the price, approximately \$8000.

The multi-dimensional device which seemed most suited for Moo's purposes was the Polhemus [Foley87] [Foley90] [Polhemus90] (see Figure 2-4). Unlike the Spaceball, the Polhemus is an absolute device. The hardware consists of a small box to which a sensor and a source are attached. Within both the source and the sensor are three antenna coils, which are arranged at right angles to each other, forming a Cartesian coordinate system. Each transmitter antenna in the source emits a pulse which induces a current in the corresponding receiver coil. This technology enables the device, which has three translational and three rotational degrees of freedom, to calculate the three-dimensional position and orientation of the sensor with respect to the source. Typically, the source is mounted firmly near the display. The sensor can be mounted on any number of objects. For example, as mentioned above, a Polhemus placed on the back of the hand provides position and orientation information for a DataGlove. Others have mounted it on helmets for tracking head position in head-mounted displays [Wang90]. The Polhemus used in Moo is embedded inside a plastic stylus.

The Polhemus was the right choice for Moo for several reasons. First, it is in the correct price range (approximately \$3000). Second, from the users' point of view, it makes tasks such as translation and rotation seem intuitive. (However, unlike the DataGlove, the Polhemus does not give users the feeling they are directly manipulating objects; instead, they are controlling a tool which in turn affects the objects.) Also, because the tip of the device can

be aimed like a camera lens, the Polhemus is an intuitive device to control camera motion as well as to specify three-dimensional paths. Moreover, the Polhemus seems a natural device for advanced modeling. For example, the Polhemus could "become" the chisel in the wood carving example given above. In other words, its input could be connected to a representation of a chisel shown on the screen (in fact, since obtaining the Polhemus at Brown, a program of this sort, which uses the device to represent a clay sculpting tool, has been written [Galyean90]). Finally, the Polhemus, as it provides absolute data, can be used to digitize physical objects.

An interesting new variant of the Polhemus is the Flying Mouse. This device appears to be a normal 2D mouse, except for small "wings" on either side. The device in fact can be used, without modification, as a standard 2D mouse. However, encased within the mouse is a Polhemus; simply lifting the Flying Mouse into the air effects the transition from a 2D to a 3D device.

The strength of this device is its versatility- as a fully-functioning 2D mouse, it does not seek to eliminate existing devices, only augment their capabilities. Also important is the ease of transition between 2D and 3D modes; lifting the mouse into the air (made easier by the two wings on either side) is the only step necessary. Although this device was made available too late for testing for this project, it appears to be well-suited for Moo. As a fully-functioning Polhemus, it provides the same benefits as described above. In addition, the Flying Mouse promises to provide speed and ease-of-use by freeing the user from having to switch away from a 3D-device when Moo's normal mouse-driven features are used. Finally, the versatility of the Flying Mouse seems to justify its \$4000 price tag.

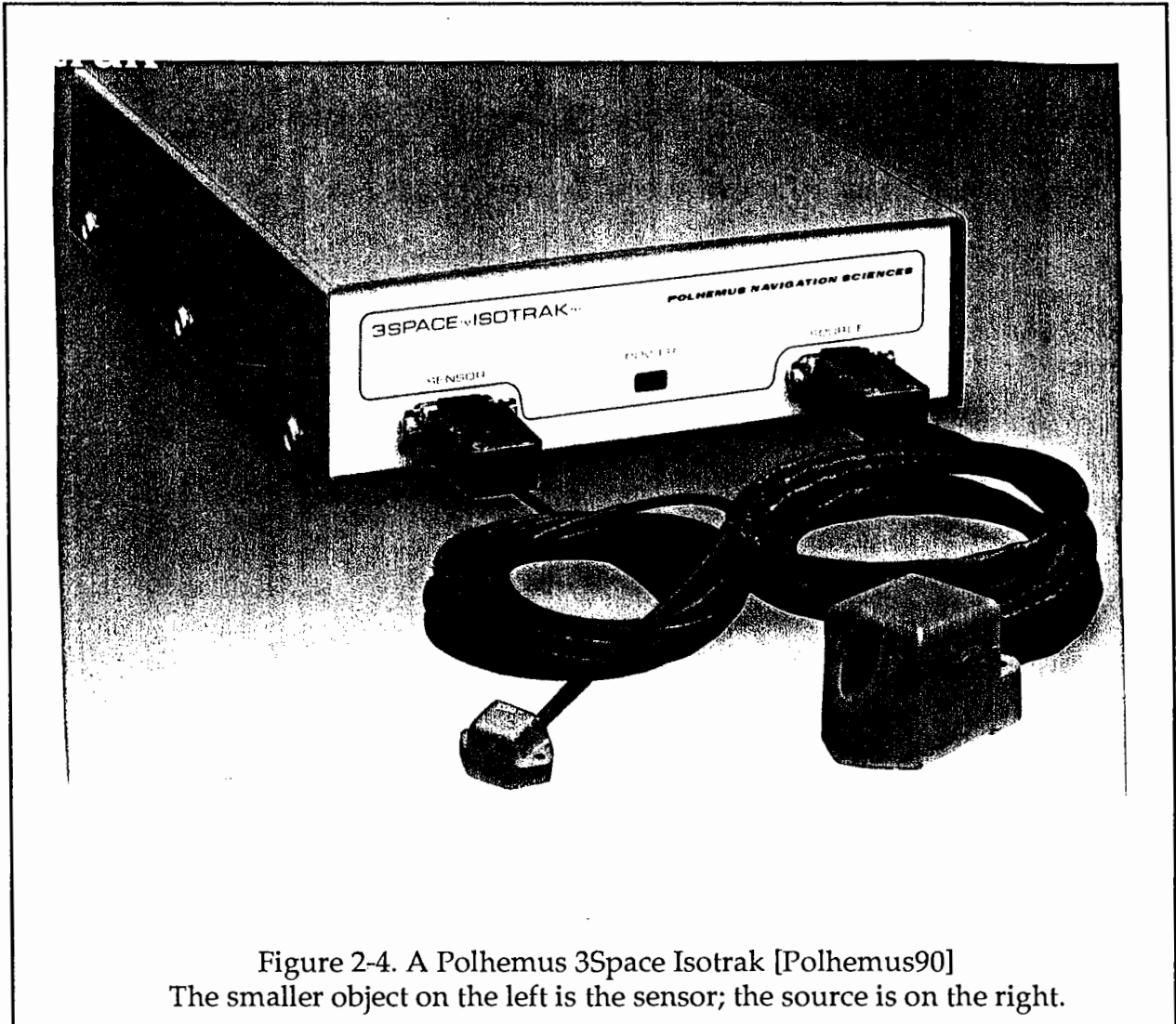


Figure 2-4. A Polhemus 3Space Isotrak [Polhemus90]
The smaller object on the left is the sensor; the source is on the right.

2.2 Notes About the Polhemus: Hype Versus Reality

The potential benefits of the Polhemus are clear. In theory, it provides an exciting new way to interact with a computer. It has been hyped as a "six-dimensional mouse," [Ware88] and the Polhemus company says it "is a powerful solution for linking the way we *naturally* respond to our environment to the most sophisticated 3D computer applications available today" [Polhemus90]. However, in practice, the Polhemus is not as easy to use

as this might suggest. This project, as well as the work of others, has shown that the Polhemus is still an experimental device.

Polhemus claims a positional accuracy of approximately one-tenth of an inch and a rotational accuracy of about two-fifths of a degree, under ideal conditions. However, there are several situations in which it provides less than accurate data. For one, it is easy to interfere with the magnetic field the device uses to calculate position and orientation. The manufacturers warn that "[l]arge metallic objects, such as desks or cabinets, located near the source or the sensor may adversely affect the performance of the system" [Polhemus90]. However, in practice, smaller objects, including computer displays, can cause problems with the Polhemus data. Of course this problem can be alleviated by isolating the system as much as possible. Nevertheless, this solution is not always practical and does nothing for the system's accessibility. Another limitation is that the device is most accurate when there is four to fifteen inches between the source and the sensor; the performance degrades noticeably beyond that. Also, because of the symmetric nature of the magnetic field, the Polhemus working area is a half-space rather than the full area surrounding the source. (The exact location of this half space can be arbitrarily defined, however.) Through informal testing in Moo, it seems that users often move outside of the bounds of the allowed operating space, only to become confused because their movements no longer correspond to what they are seeing on the screen. This could be alleviated by carefully controlling the working environment.

Using the Polhemus uncovered other difficulties with the device. For one, users found it difficult to control six degrees of freedom at once. Moreover, users wishing to restrict the motion to one or two dimensions at a time found this to be nearly impossible when all six variables were enabled. Thus, most of the interaction techniques implemented in Moo each make use of only a subset of the values sampled from the device. The difficulty in separating degrees of freedom partially stems from the fact that users holding a Polhemus have nothing against which to guide their movement. Not only is this drawing in the air an imprecise procedure, it also causes the user's arm to get very tired after long interaction sessions. A final problem which caused users some confusion was that of a lag between the time they moved the device and the time that movement was reflected on the screen.

3. Interaction Techniques in Moo

The remainder of this paper focuses on interaction tasks specific to four-dimensional modeling, and discusses various techniques used to perform these tasks. The first section describes a number of interaction tasks for which the Polhemus is useful. The second section describes the extension of a particular technique used in two-dimensional graphics, snapping, to three-dimensions.

3.1 Extending Traditional Techniques Using New Devices

Moo, like any complex modeling and animation system, employs a large number of different interaction tasks and techniques. Many of these can be readily extended to use multi-dimensional devices. We considered the following tasks for extension: rotation, translation and scaling of an object; positioning an object, both relative to another object and in relation to time; camera placement; and drawing a three-dimensional path, to be used either as a curve in space or a guide for an object to follow over time. We considered selection as well, although the extension is more complex. Following is a description of the Polhemus-specific techniques added to Moo. Before that, however, it is helpful to review other research in the area.

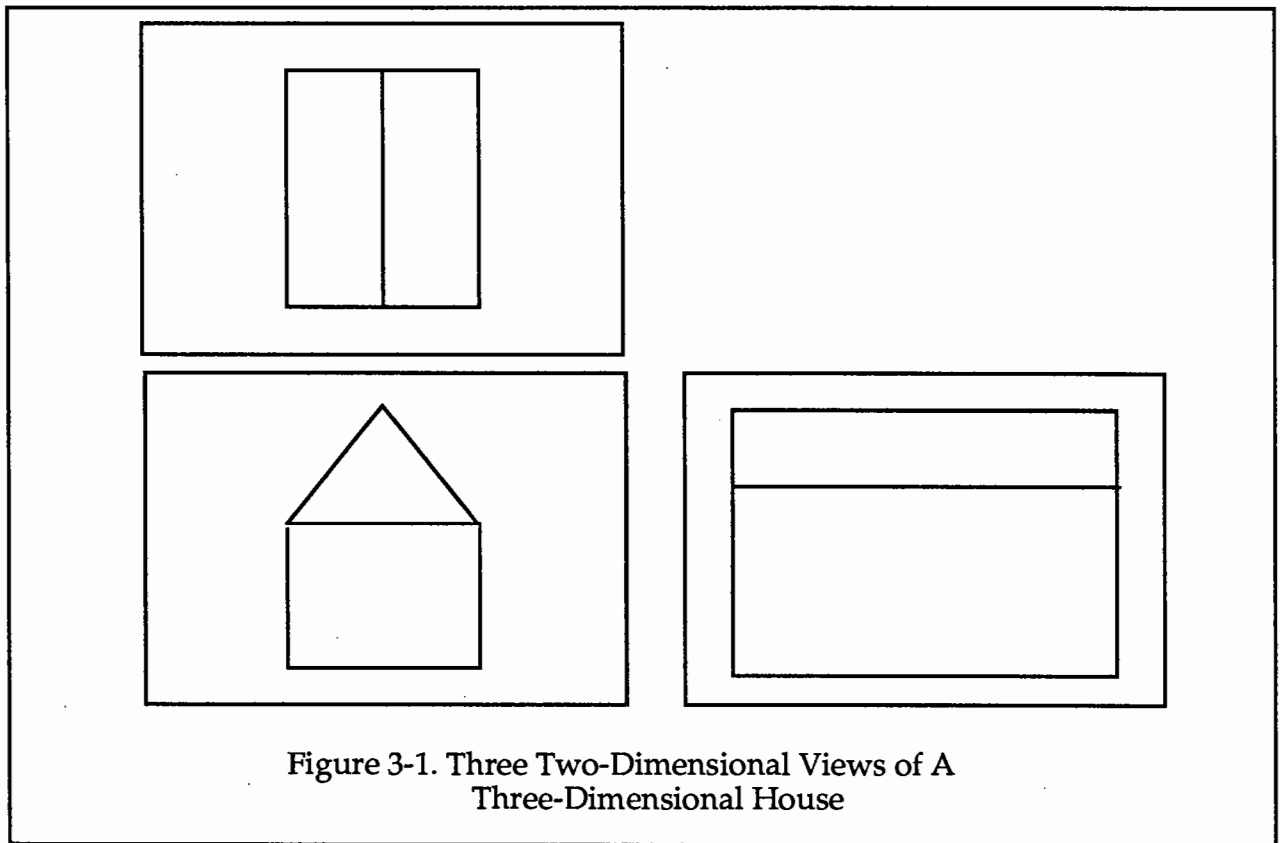
3.1.1 *Review of Traditional Interaction Techniques*

Most interactive modeling systems use a combination of one- and two-dimensional input devices to specify three-dimensional tasks. The most basic input device is the keyboard, used for typing commands. Moo makes use of this technique: an object can be selected by typing in its name, translated by entering a distance, and rotated by specifying an angle. Dials are another common device. For example, in Moo, translation, rotation and scaling are mapped to each of three dials. In addition, camera motion uses one dial for zoom, one for left-right motion, and one for up-down.

Common two-dimensional input devices include mice, tablets and the trackball. Modeling systems make use of these devices to provide input for

three-dimensional tasks. Interaction techniques using these devices can be differentiated by how they provide full interaction from only two dimensions of input.

One method, common to drafting, displays a three-dimensional scene by three two-dimensional views. (See Figure 3-1). Operations on objects are then restricted to the plane of a single view. However, many people find it difficult to piece the separate views together in order to visualize the full three-dimensional scene. Another technique maps the device's two degrees of freedom to two axes at time, chosen with the push of a button. For example, an object could be translated in X and Y with the mouse when the button is up, and in X and Z when it is pressed. The problem with this approach is that it is not effortless: the user must remain conscious of the state of the device. A more detailed discussion of these two techniques can be found in [Foley90].



Other techniques using two-dimensional input devices aim to provide the user with more of a feeling of direct manipulation. Nielson and Olsen [Nielson87] describe several such techniques. The authors use the two-dimensional projections of the three principle axes as a guide for specifying three-dimensional position. As the mouse moves, its direction is compared to the three projections and the closest match becomes the direction of motion. (See Figure 3-2). This provides users with a more direct feeling of control, even though they are restricted to moving in only one axis at a time. For translation, scaling, and rotation, Nielson and Olsen use a technique which relies upon the geometry of the objects being manipulated, so that the user's input is mapped to an edge or face of the object. Thus, even though the interaction is constrained to one or two degrees of freedom at a time, full three-dimensional transformations are allowed. Figure 3-3 depicts this technique for rotation.

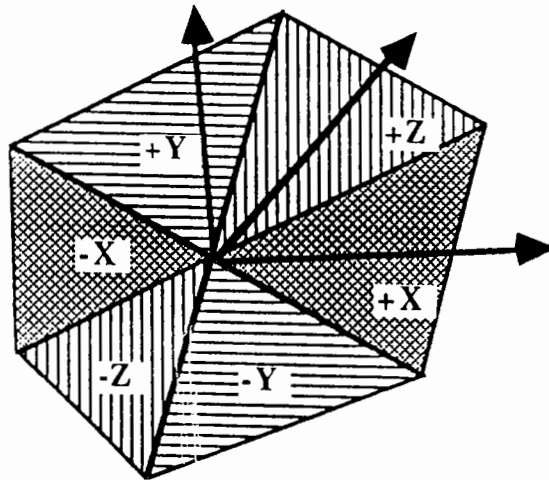


Figure 3-2. Cursor Movement Zones [Nielson87]

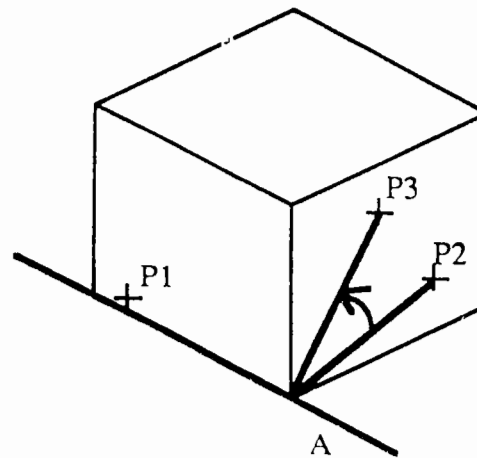


Figure 3-3. Edge Rotation [Neilson87]

The user selects point P1, which causes edge A to become the axis of rotation. Then the user selects P2 and P3, which are projected onto the face of the cube. The actual rotation is the angle between P2 and P3.

Chen, Mountford, and Sellen [Chen88] describe another method, specifically for rotation, known as the virtual-sphere. The virtual sphere simulates the effect of embedding an object inside of a trackball. On the display, the virtual sphere appears as a circle surrounding the object. When the user moves the mouse up and down or left and right inside of this circle, the object is rotated about the X-axis and Y-axis respectively. This is equivalent to rolling a trackball at its apex. Rotating an object about its Z-axis is achieved by moving the mouse along or outside of the edge of the circle. This is similar to twisting the trackball about its equator. The virtual sphere is shown in Figure 3-4; it is perhaps the most direct of the interaction techniques using two-dimensional devices, yet it still does not allow users to rotate objects displayed on the computer screen as they would if the object was in front of them.

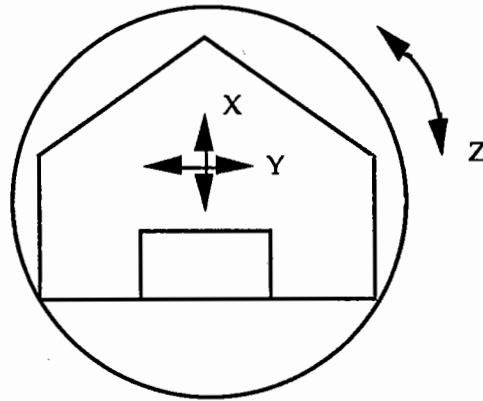


Figure 3-4. The Virtual Sphere [Chen88]

The arrows above indicate which axis is rotated around when the cursor moves in the indicated direction.

Finally, at Brown we have implemented several other techniques for direct interaction using two-dimensional input, in a system called TDUI (Three-Dimensional User Interface) [Galyean89]. For example, TDUI provides selection by tracing a ray directly back into the scene from the position of a two-dimensional cursor on the plane of the screen. The first object that this ray intersects is selected. This method cannot be used to select objects which are completely obscured; however, the camera position can always be changed in order to reveal hidden objects. To accomplish translation, TDUI uses Nielson and Olsen's technique of comparing mouse movement to a projection of the three principle axes onto a plane. This is more general than the technique they themselves use to translate along object faces and edges, because objects can be translated independent of their geometry. Also, objects such as spheres which are not composed of polygons may be translated using this scheme. Scaling in TDUI is performed in much the same manner as translation. The only difference is that unlike in translation, where the whole object follows the movement of the mouse, in a scaling operation, one

part of the object remains fixed, while another selected point moves. (Or, if the user desires, the entire surface of the object can be made to scale uniformly about the object's center.)

3.1.2 Previous Work Using Multi-Dimensional Devices

Despite the many attempts to provide intuitive interaction techniques using two-dimensional devices, the simple advantage of matching the dimensionality of an interaction task to an input device makes multi-dimensional input devices attractive. This section describes some previous systems which have made use of the Polhemus and the DataGlove.

One of the earliest systems to experiment with the Polhemus is an articulated figure program written by Norman Badler and others at the University of Pennsylvania [Badler87]. This system uses the Polhemus to perform two separate tasks: orienting joints of articulated figures, and positioning goal objects for these figures to "reach for." The authors of this work seem unsatisfied with the Polhemus as an input device for these tasks. One of their main concerns is with the lack of depth perception on the screen. This problem, they feel, "cause[s] the operator to consciously calculate how to move . . . an object, instead of intuitively performing the movement." They find that users have a much easier time if the Polhemus input is restricted to one or two degrees of freedom at once, although they admit that the tasks for which they have done this can be better performed with a different choice of (lower-dimensional) input devices. Badler has also experimented somewhat with the Polhemus as a device to control camera motion, and as a tool for digitizing real objects. However, neither of these tasks are included in the articulated figure program.

A second study of the Polhemus as an input device, specifically as a means to control a camera in a three-dimensional scene, was performed by Colin Ware and Steven Osborne of the University of New Brunswick [Ware90]. The authors experimented with three different metaphors for camera control: "eyeball-in-hand," "scene-in-hand," and "flying vehicle control." The eyeball in hand technique likens the Polhemus to a "virtual video camera," which is moving about a scene. The view from this video camera is what the user sees on the screen; as the device moves, the scene

changes. On the other hand, the scene-in-hand technique maps movement of the device directly to movement of the scene. For example, if the Polhemus moves left, so does the entire scene. Finally, the flying vehicle control method uses the Polhemus as a control device for a "virtual vehicle," from which the scene is perceived. The user's hand motion corresponds to changes in directional and rotational velocity of the viewpoint.

Ware and Osborne tested these metaphors by having the users perform several tasks with the Polhemus set up in each of the three ways. Their conclusion is that there is no best metaphor, but rather that each is useful for different tasks. For example, given a single object and the task of looking at specific features on the object, users find the eyeball-in-hand and the scene-in-hand metaphors to be appropriate, but have trouble performing the task with the Polhemus used as a flying vehicle. On the other hand, for the task of creating a camera path through the scene, users find the flying vehicle control metaphor best and the eyeball-in-hand worst.

The authors also have specific comments about the use of the Polhemus in each of these three metaphors. With the eyeball-in-hand method, their main comment is that the Polhemus "imposes awkward physical constraints." This is because the device must be operated within a specific area, so it is impossible to move the viewpoint very far from any part of the scene. Also, users had to make complex hand movements in order to view the scene from certain positions. This same problem is found when the device is used with the scene-in-hand metaphor. Furthermore, when the scene-in-hand method is used with the viewpoint far from the center of rotation, they find that a small change in Polhemus position maps to a large change on the screen; this becomes disorienting to the user. Finally, they find the flying vehicle control method the easiest to use with the Polhemus in terms of user comfort. This is because it does not require continuous movement by the user.

Finally, there is the modeling system implemented by David Weimer and S. K. Ganapathy at AT&T Bell Laboratories [Weimer89]. This system uses a data glove as an input device rather than a stand-alone Polhemus. However, it has limited relevance because the authors were also experimenting with another natural method of interaction, voice recognition.

Using this combination of input devices, the authors implemented techniques for performing some of the basic tasks of modeling: translation, rotation, and scale. These implementations do not make use of the position and orientation information received from the Polhemus on the back of the data glove. Instead, they work as follows. When the user says either "translate," "rotate," or "scale," she is then able to control of the size of the transformation (for example, how far the object would translate) with her thumb. More specifically, the angle between the thumb and the index finger is taken as a parameter to the operation. When the user is done, she simply says, "ok." Since this method uses the thumb as a one dimensional valuator device, the axis of the transformation has to be set explicitly beforehand. The authors do not mention any attempts to use more than one degree of freedom at a time.

A method of drawing curves in three-dimensions was also implemented in this system. In this technique, the positional information from the Polhemus is used to move a cursor around the scene. On the screen, this cursor is attached to the index finger of the hand representing the data glove, so the user seems to be moving the cursor with her finger. To draw a curve, the user says "ok" to leave a control point at the current position of the cursor, and then "close" or "quit" to complete the curve as either a closed or open path. The system also allows surfaces to be created by dragging these curves through space, or by rotating them about a specified axis. Moreover, control points on both curves and surfaces can be repositioned by a combination of saying "point-drag" and moving the thumb as described before.

In this system, the user changes modes and selects other features through the use of a "virtual control panel." This is a set of buttons displayed in the three-dimensional world of the screen, which the user can "press" with the index-finger controlled cursor.

In general, the authors of this system note that the hand tracking device they use is not as accurate and as noise free as they would like. They also comment that especially for the task of dragging around a curve, users' arms end up in uncomfortable, contorted positions.

3.1.3 *Using the Polhemus in Moo*

The Polhemus was not introduced in Moo as a replacement for all other currently used interaction techniques. Instead, the aim was to find ways to incorporate it as an enhancement to the already existing system. For example, because of this, unlike in the AT&T system, methods for manipulating menus and control panels using the Polhemus were not explored since the mouse has already proven to be an effective device for this task. Of course it might be said that the requirement to switch devices in order to perform different tasks might needlessly slow the user down. However, the assumption made in this work is that using a device for a task which it is not ideally suited would impair the user's progress even more.

Another benefit to having the Polhemus be one of many devices used is that it prevents the user from having to continually change modes each time she wishes to perform a different task. It is only possible for one input device to be connected to a limited number of interaction tasks at a time. For example, with nine dials, a different dial can be set to control X, Y and Z translation, rotation, and scale. However, if the user wishes to change the camera position, she will need to switch three of the dials to camera mode, thus losing at least some of the previous functionality. Similarly, the mouse can be assigned to manipulate a virtual sphere or to perform direct translation, but it can not do both at once. However, with dials, a mouse, a tablet, and a Polhemus as Moo now has, the user can have immediate access to many different functionalities simply by picking up the correct device. Moreover, in order to support a wide range of users, it is possible in Moo to specify what device controls what task.

This focus on the Polhemus as an addition to an existing system is a major difference between this work and the work described in the previous section. Also, unlike the other systems, Moo is a full-featured program used to produce high-quality animations. In contrast, the AT&T modeler is part of a highly experimental system which lacks basic features necessary to produce complex models. Similarly, the system produced at New Brunswick is intended solely for the purpose of testing the Polhemus, not for producing anything real. Another difference is that in Moo the Polhemus was used in the investigation of a wide range of interaction tasks, including three-dimensional point location, translation, rotation, camera motion, and path

drawing. Several of the other systems focus on one or two of these tasks; however, no single system aims to incorporate them all. The remainder of this section describes each of these techniques as implemented in Moo.

The task of point location is equivalent to that of positioning a cursor in three-dimensional space. In Moo, this cursor is drawn as a set of three mutually-perpendicular rectangular solids, as shown in Figure 3-5. (Three perpendicular lines were not used because Moo already uses this particular figure to indicate the current coordinate system.) In a first attempt to attach the cursor to the Polhemus, the device was continually sampled, and the absolute position data obtained was used to set the (X, Y, Z) coordinate of the center of the cursor. However, while this simple technique worked in the sense that the cursor was made to follow the motion of the Polhemus, several noticeable drawbacks were observed.

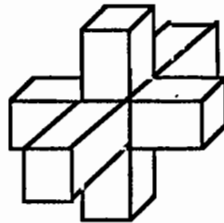


Figure 3-5. Moo's Three-Dimensional Cursor

The first problem had to do with the shape of the cursor itself. The three rectangular solids gave users helpful feedback about the coordinate space they were in. However, users found it difficult to make the connection between the stylus in their hand and the cursor on the screen. Possibly this was because the cursor only tracks the stylus tip, while users generally think in terms of moving the entire device. A solution to this problem, which has not yet been implemented in Moo, would be to simulate the entire stylus on the screen, showing its shape, orientation, and position.

The sense of user disorientation was exacerbated by the fact that the cursor did not remain steady as it moved with the Polhemus. Instead, its path was noticeably jittery. This could have been caused by interference by

other objects in the room, as discussed earlier, inadequate sampling software, or by problems with the accuracy of the device itself. Attempts to isolate the device as much as possible did not remove the problem. As well, there was not much that could be done about the accuracy of the device. Thus, software solutions were examined.

In order to smooth out the cursor's motion, an averaging algorithm was used. Every new sample value was averaged with the previous five samples, and the result became the new position of the cursor. This procedure removed some but not all of the jerkiness of the motion. A similar predictive tracking algorithm was used in Tinsley Gaylean's interactive volumetric modeler SCULPT. A new point is taken as a weighted average of a sampled point (weighed much more heavily) and a point calculated from the direction predicted by the previous two points; this method "works well except when there are fast, large movements" [Gaylean90]. More sophisticated averaging techniques could have been used, but as will be discussed later, improvements to the device driver made filtering less necessary.

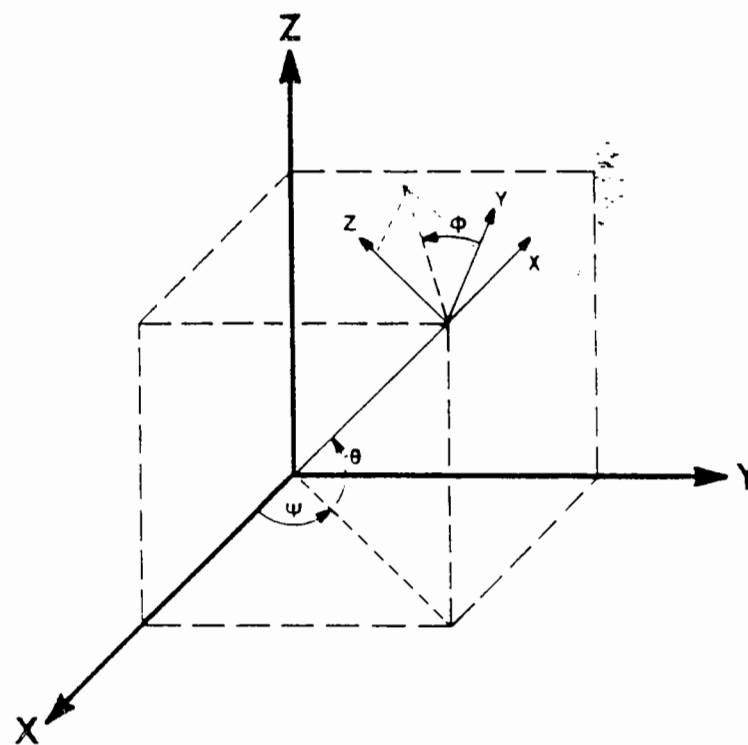
Unfortunately, averaging worsened another problem, that of lag. Lag refers to the delay which occurs between the time the user's hand moves to the time the cursor reflects that change. The greater the lag time, the less intuitive the device is to use. Lag had been a problem before; with the averaging algorithm the problem was greatly increased.

A relatively simple improvement was to modify the mapping from the unit of measure for the device to that for the modeler. This increased the density of the samples on the screen, thus minimizing the effect of the jitter. However, even with this improvement the problems did not completely disappear, so more work in this area should be done.

The task of translation is no more difficult than that of point location. In fact, in Moo, both tasks are implemented in the same way. The only difference is that instead of using the sampled Polhemus values to change the position of the cursor, they are used to move the currently selected objects. (Objects are selectable using the mouse or the keyboard, but not the Polhemus.) More specifically, when the Polhemus is activated, the current position of the stylus is set to be (0, 0, 0), and the current positions of all of the

selected objects are considered attached to the device. The objects then translate along with any movement of the stylus.

The task of rotation in Moo is also implemented by connecting the Polhemus stylus to the currently selected objects. These objects are then rotated about their own centers by the orientation amounts received from the device at each sample. Figure 3-6 explains the three orientation values, (Ψ , Θ , Φ), which are obtained from the Polhemus.



X, Y, Z = Alignment Reference Frame
x, y, z = Rotated Stylus or Sensor
Coordinate Frame
 ψ = Azimuth
 θ = Elevation
 ϕ = Roll
--- $\rightarrow$ = Polhemus Stylus Vector

Figure 3-6. Polhemus Orientation Values [Adapted from Polhemus87]

In order to change these values into an orientation for the selected objects, the following algorithm is used:

Rotate the X and Y coordinates of the object by Ψ about the object's Z-axis

Rotate the Z and rotated X coordinates by Θ about the object's rotated Y-axis

Rotate the rotated Y and Z coordinates by Φ about the object's rotated X-axis

While this method works in that it allows object orientation to be changed by the Polhemus, it is also fairly disorienting for the user. This is because it is difficult to tell what the three axes of rotation are at any given moment. Moreover, when people become confused and can no longer tell how their hand motion corresponds to screen output, attempts to correct things tend to make matters worse rather than better. With this in mind, a second implementation of rotation was attempted. This new method maps the three axes of rotation to the current camera up, at and right vectors (these correspond to the vectors directly vertical from the camera; directly into the screen from the camera; and the cross-product of the two, going to the right of the camera) instead of X, Y, and Z. The rotation is performed in screen space instead of object space; in other words, motion of the stylus in the real world is given a direct correspondence with the motion of the object. This is easier to use than the first technique, because it provides a static frame of reference against which users can coordinate movement of the stylus in their hand.

Using the Polhemus as a way to position the camera was the first technique tested in Moo using all six values from the device at once. The implementation begins with the default Moo camera, defined by three points (which may in turn be used to define the up, at, and right vectors described above): from-point (0, 0, 0), where the camera is located; at-point (0, 0, -1), where the camera is pointing; and up-vector (0, 1, 0), directly vertical from the camera. As samples are received from the Polhemus, the at-point and up-vector are rotated using the algorithm described for rotation above, and the from-point is placed at the location defined by the three position values. This is similar to the eyeball-in-hand metaphor of Ware and Osborne.

Users find it difficult to use the Polhemus in this manner. One of the reasons is again the jittery nature of the input, which is an even greater problem with camera motion than with translation. During translation,

typically one or two objects in the scene are moved at a time. When the camera moves, however, the entire scene shifts. When the entire scene is jittering instead of moving slowly, it is disorienting.

Another problem is that users have trouble dealing with all six degrees of freedom at once. Typically, a person operating a camera makes slow changes in one direction at a time. Using the Polhemus, this entails keeping most of the degrees of freedom steady while carefully controlling the others. However, the fact that the stylus is being moved around freely in the air makes this almost impossible to accomplish. As a solution to the problem, an alternate implementation was tried. This time, the up-vector remained constant at $(0, 1, 0)$, and the at-point was set to be the position of the currently selected object, independent of the Polhemus. The only value affected by the motion of the Polhemus was the from-point; it was set to be the same as the sampled position values of the Polhemus. Although this method restricts the motion of the camera (the focal point of the scene can not be changed), it has proven to be very effective. Users find it intuitive to be looking at the scene from the point of view of the Polhemus. When they want to view the scene from above, they simply move the stylus in their hand higher in the air.

The final test of the Polhemus in Moo was path drawing. To construct a path, the user activates the Polhemus in spline mode, and moves the stylus through the air. At each sample, a marker is drawn on the screen. Once the user is finished drawing, the list of markers is used to create a spline path.

Using the Polhemus as a tool to draw paths in this manner is not without problems. The most noticeable problem is lag, which is even worse than it is with the translation task. There are at least two reasons for this. First, the cost of drawing a marker at each sample (and the line connecting it to the previous marker) slows down the sampling of the device enough so that users no longer feel as if the Polhemus is tracking the motion of their hand. The second reason has to do with feedback. When a user translates an object, even if the object is lagging behind the motion of the stylus, the user knows exactly where the object is and can compensate for the lag accordingly. In path drawing, users do not know where they are until a point is actually drawn, and any attempt to compensate changes the path itself.

One way of reducing the lag is to have users move the stylus very slowly. This gives people much more of a feeling that their actions are being duplicated on the screen. However, this is not an adequate solution. In order to solve the problem, a method of reducing the number of control points per spline was added. In this scheme, as each sample value is returned from the Polhemus, the distance to the previous sample is calculated. If this value is less than a user-set minimum, the sample is thrown away.

Another problem with the original path drawing mechanism is that the resulting spline has far more control points than necessary to define the path. One method of reducing this number, implemented in Moo, is only to use points which indicate a significant change in direction. This technique makes the number of points more manageable, yet more work in this area is necessary.

3.2 Snapping: Extending a Technique

Although most of this paper has been about multi-dimensional input devices, the overall goal of the work it describes was to provide Moo users with easier ways to build four-dimensional models by providing better interaction techniques regardless of the type of input device used. With this aim in mind, we implemented a technique, known as snapping, which is not dependent on the type of input device used.

At the highest level, snapping is a technique for precisely positioning objects in relation to one another. For example, consider a user modeling an office. She has created both a lamp and a desk, and wishes to position the lamp so that its base is exactly resting on the top of the desk. This is not an easy task, no matter what input device is used, if the only tools provided are the basic transformations. The user must rotate and translate the lamp until the two objects appear to be correctly placed. However, while the scene might look fine from one angle, the base of the lamp could actually be intersecting the desktop. With snapping the user can easily place objects in the correct position.

The Moo implementation of snapping is based upon an existing technique known as "Snap-Dragging." In the next section Snap-Dragging is

described, and its applicability to Moo is evaluated. A discussion of the implementation of snapping in Moo follows.

3.2.1 *Snap-Dragging*

Snap-Dragging was first introduced by Eric Bier and Maureen Stone [Bier86], and was further extended by Eric Bier [Bier87] [Bier88] [Bier90]. The first implementation defined a technique for making precise line drawings in two dimensions: “a synthesis of the best properties of grid-based systems, constraint networks, and drafting” [Bier88]. Several features of Snap-Dragging include gravity, affine transformations, and alignment objects. Snap-Dragging uses a gravity function, causing the software cursor to snap to points, curves, and the intersection of curves when it is moved close enough to these objects. It also allows affine transformations to be specified by movement of the cursor. Gravity is still active during these transformations, which enables them to be precisely specified. Finally, the system creates alignment objects, which are gravity-active lines and circles that help the user construct objects to exact specifications. This feature is similar to the use of ruler and compass in traditional drafting. Interaction in the Snap-Dragging system is performed directly with the mouse and indirectly through an extensive set of mouse-based menus.

Figure 3-7 shows an example of Snap-Dragging. It depicts a triangle which the user wishes to make equilateral. The triangle is shown on the left. The two vertices with boxes around them have been selected by the user as centers for alignment circles. The radius of the alignment circle is specified by the user in a menu, which is not pictured. When the user moves the cursor near vertex C, it snaps precisely to that point. The user then drags the vertex towards the intersection of the two alignment circles. Since the circles also attract the cursor it snaps to their intersection as well. The figure on the right shows the completed triangle.

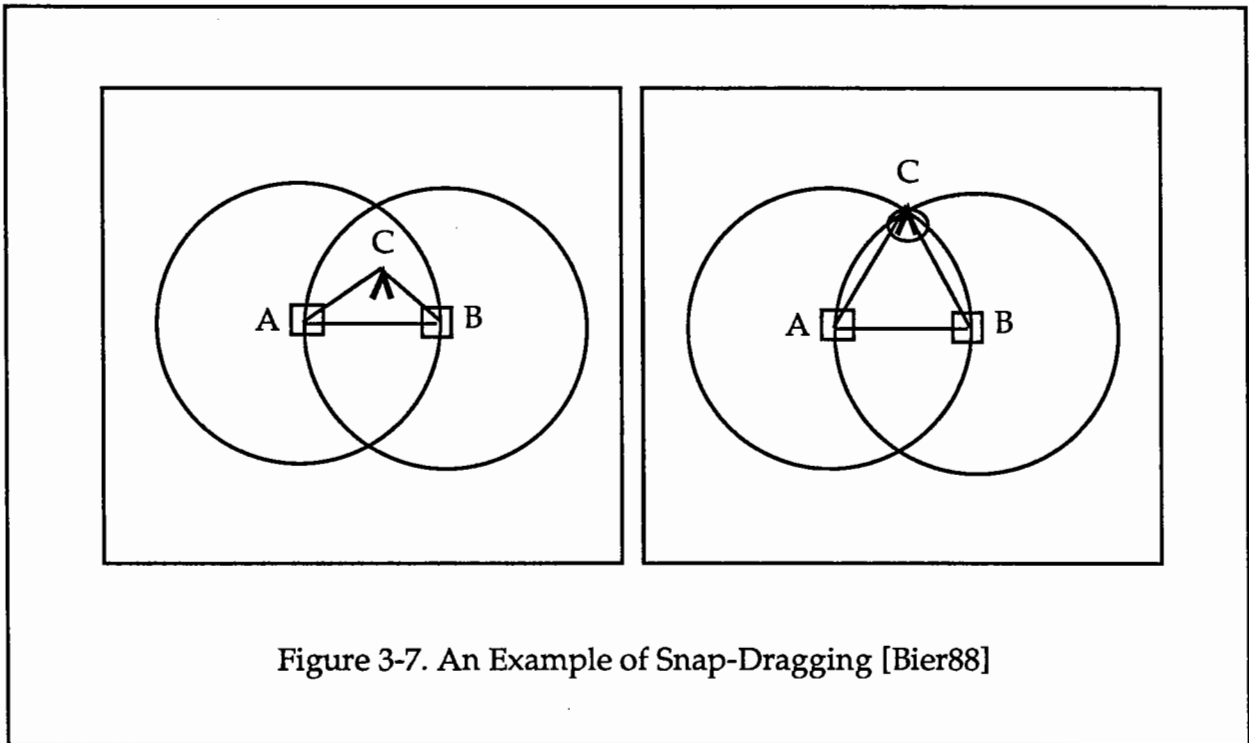


Figure 3-7. An Example of Snap-Dragging [Bier88]

In three-dimensional Snap-Dragging, surfaces and their intersections as well as points and curves can be made gravity-active. The technique is further extended by the introduction of the *skitter*. This is a mouse-controlled cursor, shaped like a three-dimensional coordinate frame, which moves along a plane. When the skitter is moved near a surface, edge, or point it snaps to that object. The orientation of the skitter is taken from the object it is snapped to. For example, if it snaps to a face, its Z axis becomes perpendicular to that face, and the X axis takes its position from the direction of a nearby edge. If the skitter is not near any object, it snaps to a default plane parallel to the screen.

The skitter can, of course, be used to specify transformations. For example, the user can invoke a "drop anchor" command, which places a permanent skitter at the position the cursor was at when the command was given. Like the skitter, the anchor defines a coordinate frame. It can be used for instance as a center of rotation, or as a general gravity-active point to which other items can be snapped.

Three-dimensional Snap-Dragging defines two additional alignment objects, planes and spheres. However, Bier also mentions that alignment

cylinders (the set of all lines at a known distance from a line), distance planes (two planes of a known distance from a given plane), and cones (the set of all lines making a given angle with a line at a given point) would be useful.

3.2.2 *Implementing Snapping in Moo*

Snap-Dragging provided the basis for the implementation of snapping in Moo. However, there are several differences between Bier's design and the needs of Moo. Most of these differences stem from the fact that Moo's purpose is to produce four-dimensional models, while the main application of the Snap-Dragging system is to create two and three-dimensional drawings.

For example, one goal of the Snap-Dragging system was to provide a way to switch easily between two and three-dimensions in the course of one user session. Since Moo is exclusively a three-dimensional modeling and animation system, this was not a concern. In addition, Bier's considerations that his system be entirely mouse-based is quite different from the philosophy behind Moo, which is that the use of multiple devices is an advantage.

Another difference is that while much of Bier's work is focused on the distinction between vertices, edges, and surfaces, in Moo, cubes, as well as cylinders, spheres, cones, and other items are the basic units of modeling. They are defined mathematically rather than by (polygonally approximated) vertices and edges. Thus, snapping in Moo concentrates more on the exact placement of objects in relation to one another than does Snap-Dragging, and takes advantage of specific properties of these objects in its implementation.

Snapping in Moo consists of two separate techniques. The first allows users to snap either a coordinate frame or an object to a specific point on the surface of another object. The second provides users with a way of moving along the surface of an object once they have snapped to it.

To the user, snapping to the surface of an object works as follows. First, a user positions the cursor over a specific point on an object, and pushes the mouse button. A set of coordinate axes then appears with its origin at the indicated position. (See Figure 3-8.) If the user then selects another point on another surface, the coordinate frame jumps to the new position.

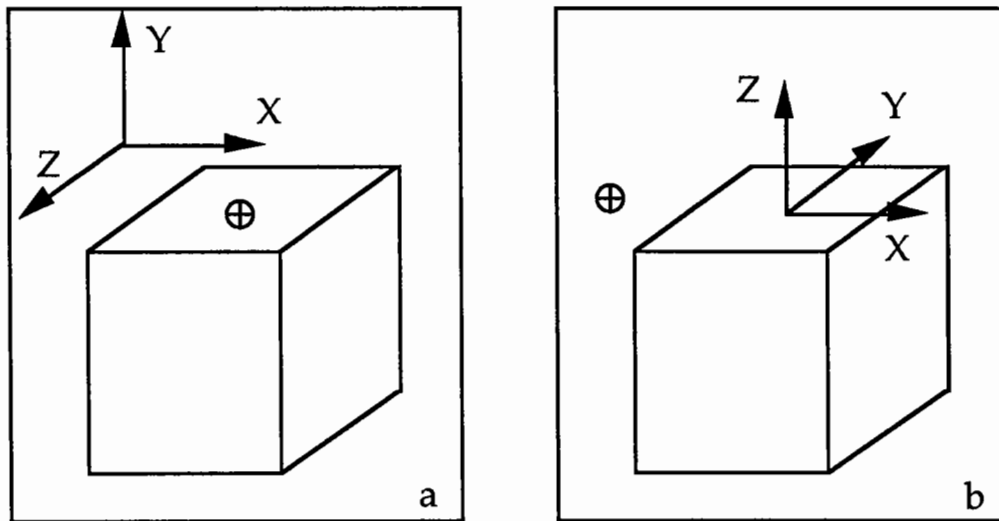


Figure 3-8. Snapping to A Cube

a) The user positions the cursor over an object and presses the mouse button. b) The coordinate frame snaps to the selected point. The cursor is free to move elsewhere.

Points are selected by tracing a ray from the cursor position on the plane of the screen directly back into the scene. The closest intersection of the ray with an object becomes the selected point. The origin of the coordinate axes is set to be this point, with its Z axis aligned with the surface normal. Once this has been done, the user can spin the coordinate frame about its Z axis. This coordinate frame can then be used as the basis for further transformations of the object, such as rotation.

Snapping is used to place one object in relation to another in the following manner. First, the user snaps the coordinate frame to an object, as above. This object is the one she wishes to position relative to the surface of another object. Then, she selects a menu item indicating that the coordinate axes should remain attached to the object. (Bier refers to this as "dropping an anchor.") Next, the user selects a point on a second object to which she wishes the first object to become attached. This time, instead of just the coordinate frame snapping to the second object, the first object snaps as well. The two objects end up tangent to each other, with their surface normals directly opposed. This is illustrated in Figure 3-9. Once this is done the first object can be spun about its own Z axis. Moreover, users can snap the two

objects to a pre-specified distance from each other. This places the snapped coordinate frame or object somewhere along the surface normal, but not necessarily touching the object.

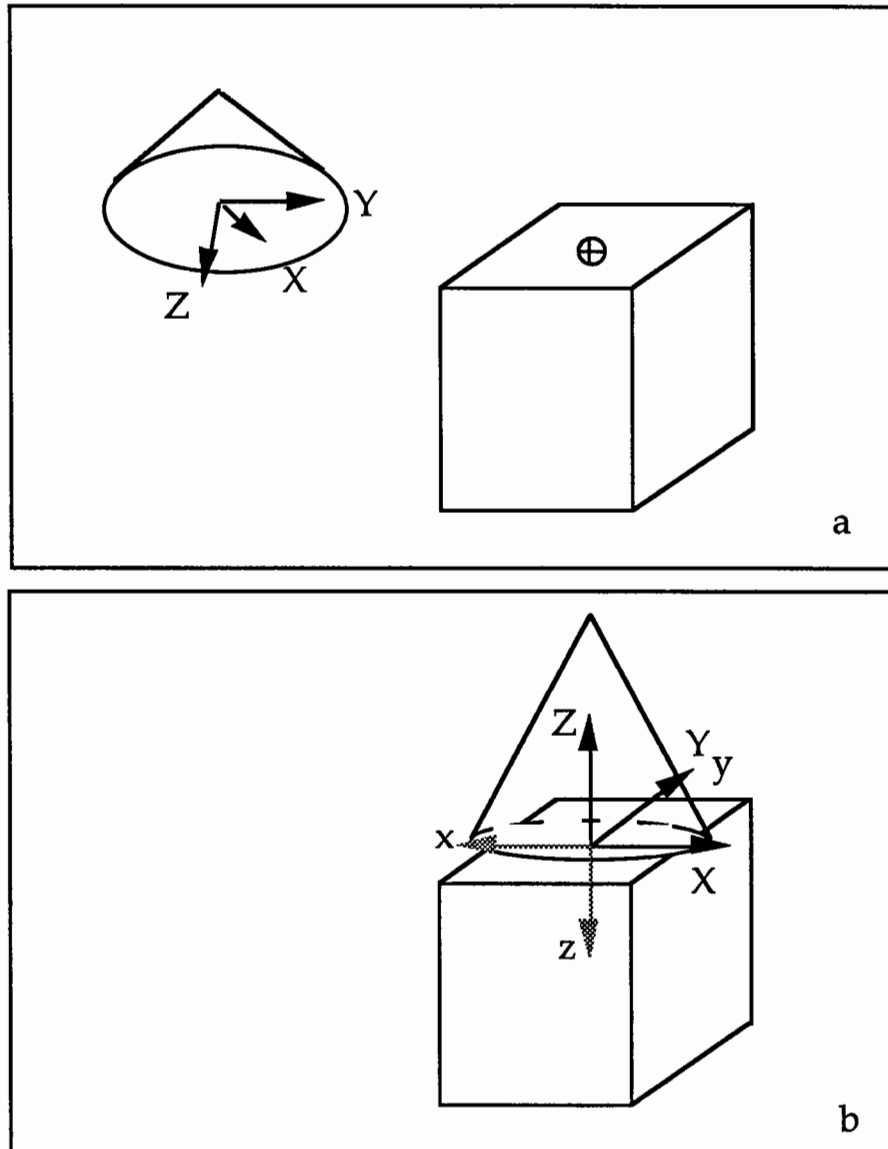
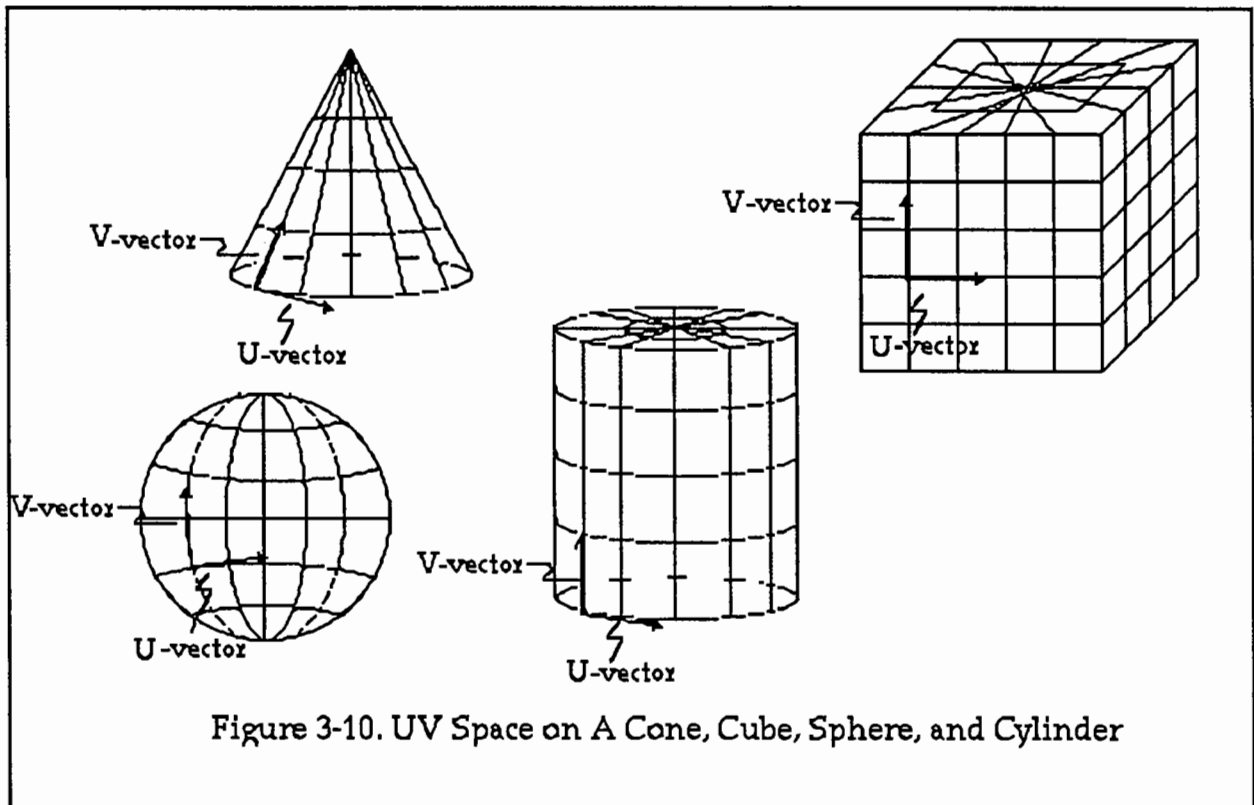


Figure 3-9. Snapping A Cone to A Cube

a) An anchor has been left on the bottom of the cone. The user selects a point on the cube. b) The cone snaps to the cube. The grey coordinate frame shows the anchor on the bottom of the cone.

The second snapping technique implemented in Moo allows users to move the snapped coordinate frame or object along the surface of an object. This is useful, for example, if a user wants to position one object relative to the surface of another, but is not sure exactly where. Instead of having to continually resnap the first object to the second, she can instead snap the object once, and then move it around until it is in the desired position.

Two different methods of moving one object constrained to the surface of another were implemented. Both of them make use of a UV coordinate space defined on the surface of every object. For example, the UV coordinate system on a sphere is equivalent to its latitude and longitude lines. Furthermore, at every point on the surface of an object, a U vector and a V vector are defined to be the unit tangent vectors in the direction of positive U and V, respectively. The UV space for several objects is shown in Figure 3-10.



Each of the two techniques uses the dials to perform the movement. One dial moves the object in the U direction, and the other moves it in the V direction. (If the object's axes have been rotated, movement is in the directions of the X' and Y' vectors instead.) The two methods differ by how they define a path along the surface given these vectors.

The first method is more simple than the second. Given the original position of the object and a direction of motion, a new position is simply calculated along this direction vector. The math is as follows. Let P denote the point which the user has snapped to. P has coordinates in UV space, (U_p, V_p) . Moreover, the X' and Y' axes of the coordinate frame can be stated in terms of the U and V vectors at P. Initially, $X' = U$ and $Y' = V$. However, since X' and Y' can be arbitrarily rotated about the Z' axis, the more general form is $X' = aU + bV$ and $Y' = cU + dV$, as shown in figure 3-11. Finally, consider that the "X'" dial has been turned a distance d. Then the coordinate frame is moved to a new UV point Q, (U_q, V_q) , which is calculated as $U_q = U_p + d * a$ and $V_q = V_p + d * b$. At Q, the Z' axis is set to be the normal vector, while the X' and Y'

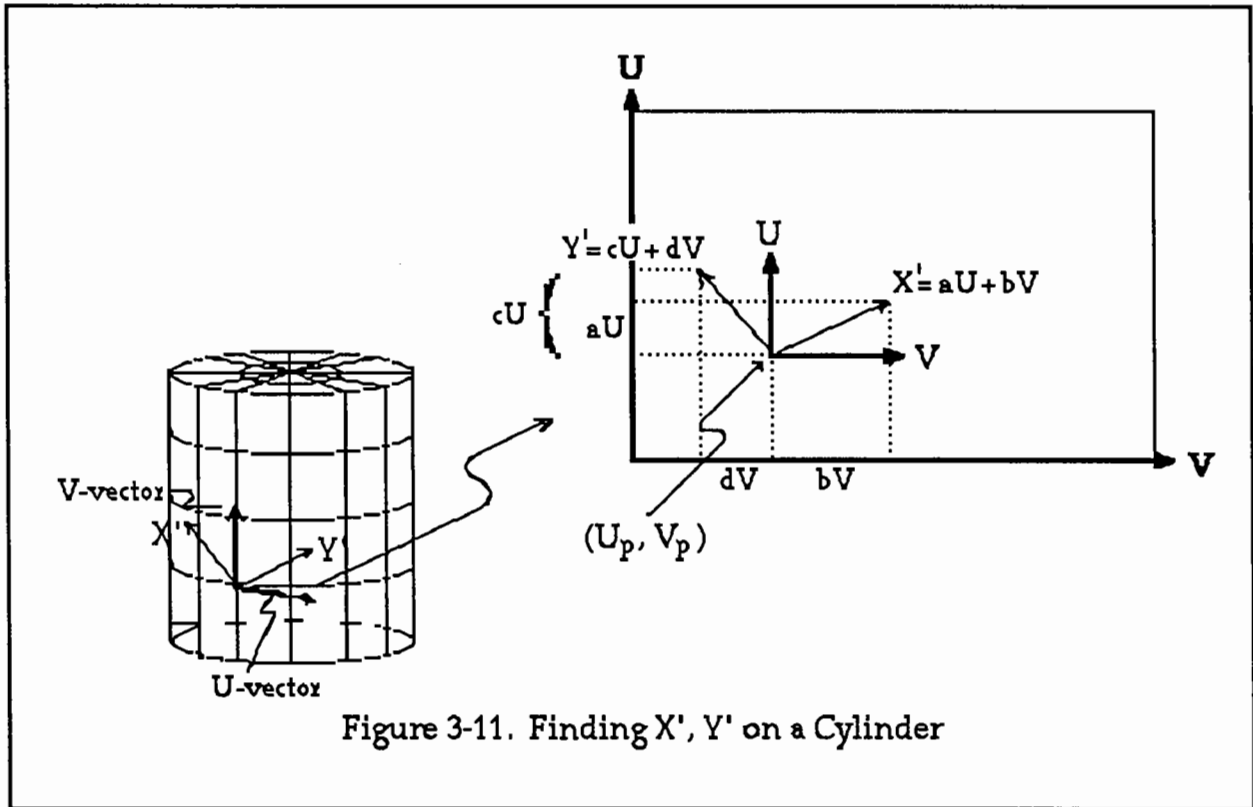


Figure 3-11. Finding X' , Y' on a Cylinder

axes remain aligned with the U and V vectors at P .

The second method allows users to move along geodesics on objects. A *geodesic* is defined as the shortest path between points on a surface. For example, a geodesic on a plane is simply a straight line. On a spherical surface it corresponds to a great circle, a circle whose center is also the center of the sphere. In general, geodesics are paths along a surface whose curvature relative to that surface is everywhere zero. Some geodesics along objects are shown in Figure 3-12. As with the first method, the motion along object geodesics uses a point P on the surface and the X' or Y' vector (expressed in terms of UV) in order to compute a new point Q . However, unlike the previous method, the curvature of the surface is also calculated and taken into account in the computation of the new point Q . The math is considerably more involved than that used in the first method. It is explained in detail in the Appendix.

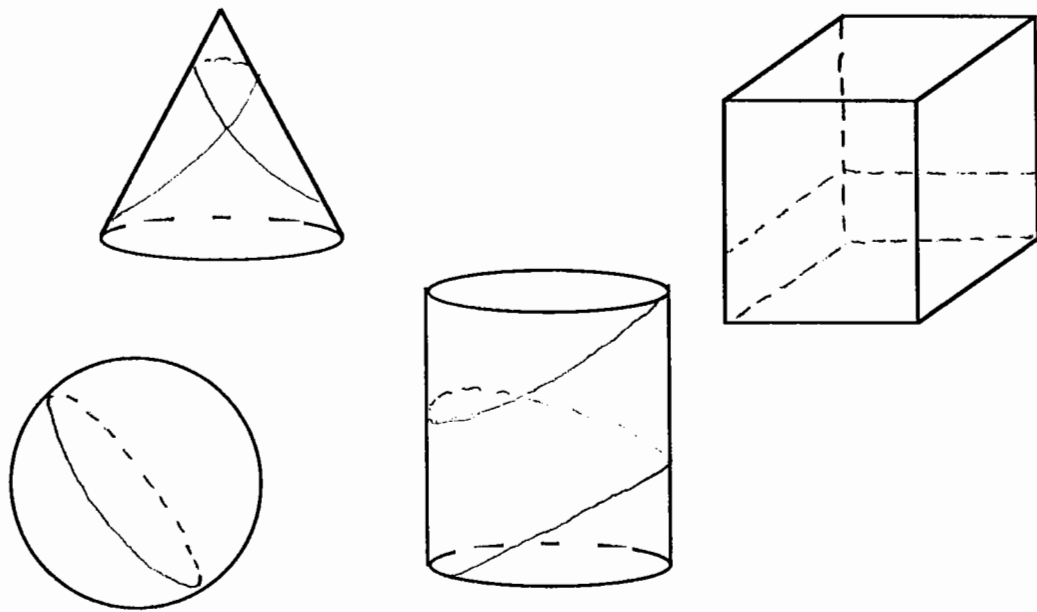


Figure 3-12 Sample Geodesics on A Cone, Cube, Sphere, and Cylinder

Both methods of moving along a surface have their advantages and drawbacks. The first allows users to move along certain interesting paths. For example, on a sphere, with the X' and Y' axes aligned directly with U and V , movement can be accomplished along lines of constant latitude or longitude. Latitude lines are not geodesics, so the second method does not permit paths of this type. Using the first method with the X' and Y' axes arbitrarily placed, however, may give users something other than what they expect. Consider a sphere with the X' axis forty-five degrees from the U vector. When the user turns the dial to move "in the direction" of X' , the coordinate frame will move towards and begin to spiral around one of the poles. Using the second method, on the other hand, the coordinate frame will move in a great circle, and return to the original point after one revolution. This is shown in Figure 3-13.

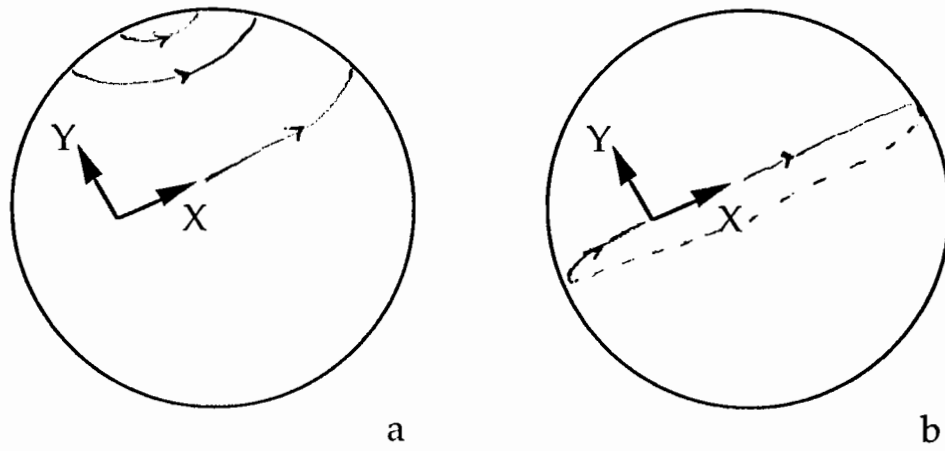


Figure 3-13. Paths Along A Sphere

a) The resulting path in the X direction using method one. b) The geodesic path.

4. Extending this Work

The broad and experimental nature of this work lends itself to many possible extensions. At the lowest level, work still needs to be done on discovering the best ways to utilize the raw data from the Polhemus. Perhaps a filtering technique more sophisticated than the simple averaging algorithm used here could be implemented to provide more accurate input. Moreover, other ways of mapping the sampled Polhemus data to each interaction task could be explored and compared to the techniques described above. Making stylus movement correspond to a smaller screen movement might make a difference in how intuitive a user finds a task, by increasing the device's apparent accuracy and control. Also, experiments could be performed testing the Polhemus using relative instead of absolute input values; an application could use the device in relative mode by only accepting sample values when the user is also pressing a button.

Another area in which more work could be done is in extending further traditional interaction techniques to use the multi-dimensional input device. For instance, although object selection is an important task in a modeling system, it was not implemented using the Polhemus in Moo. It would be nice to extend the point location technique such that when the three-dimensional cursor intersects an object, the user can push a button to select that object. Using this method instead of the mouse/ray-tracing selection technique, a user could select an object even if it were obscured. Moreover, if intersections with objects could be detected, the cursor could be prevented from passing through objects at all, thus providing users with feedback that they are working with "real" objects.

In fact, more needs to be done to improve user feedback when using a multi-dimensional input device. As mentioned, a cursor in the shape of a Polhemus might help make using the device more intuitive. For translating and rotating objects, this cursor could become embedded in the object being oriented, thus allowing users to have direct feedback as to how their hand position affects what is on the screen.

Another natural extension of this work is to implement snapping using the Polhemus. Consider the point location task, where the Polhemus is used to manipulate a three-dimensional cursor. One of the major problems with this technique as implemented in Moo is the imprecise nature of the

input. It is difficult for users to manipulate the cursor exactly where they desire. However, this imprecision would not be a problem if when the cursor approaches an object it is snapped to that object. Along the same lines, consider a path which the user has drawn with the Polhemus. This path could be edited by manipulating a three-dimensional cursor with the Polhemus. When the cursor nears a control point of the path, it snaps to this control point. The user then pushes a button to select the point and attach it to the cursor. The point then follows the cursor until the user releases the button.

Finally, although the work described in this paper was for a modeling and animation system, most of the techniques implemented were not applied to the time dimension. Thus, one of the main directions for future work is to extend these techniques to animation. For example, Moo users should not only be able to move the camera around a scene in order to look at an object from another side, they should be able to record that path for use in an animation. (A feature similar to this was implemented as part of the Ware and Osborne camera experiments.) Furthermore, a user should be able to attach an object to any path she draws with the Polhemus, and specify how the object moves along the path over time. On a similar note, the feature of snapping in Moo which allows motion along an object's surface should apply not only to objects being modeled, but also to objects being animated.

5. Summary

"The grand challenges in three-dimensional graphics are to make simple modeling easy and to make complex modeling accessible to far more people."

- R. F. Sproull, 1990

One major difficulty in the design of computer modeling systems is that while a user is conceptually building complex objects and scenes in a three-dimensional world, in reality she is interacting with a two-dimensional screen. Moreover, important tasks of interactive modeling, such object selection, translation, rotation, camera positioning, and path drawing are

most often performed using input devices of only one and two dimensions. These devices by nature require users to remain aware of how their actions are mapped to three dimensions.

One method of easing this interaction mismatch is to use a multi-dimensional input device, because their use more nearly matches three or four-dimensional modeling tasks. Currently available devices such as the Spaceball, the DataGlove and the Polhemus all improve interaction in this way, but each has its disadvantages. For the Moo modeling system, the Polhemus was the best choice based on cost and the fact that it potentially provided a natural way to perform a wider range of modeling tasks than did the others.

However, the Polhemus is not without its drawbacks. When it is operated outside of a limited range, or too close to a large metallic object, it provides inaccurate data. Efforts to compensate for these inaccuracies can introduce other difficulties. For example, the use of an averaging technique to smooth jittery data exaggerates the problem of lag.

Three dimensional point location, object translation and rotation, camera placement, and path drawing were all implemented using the Polhemus in Moo. This extension to a multi-dimensional device is an improvement because the new techniques enable the user to more naturally associate hand movement to action on the screen. However, the lack of accuracy from the device, as well as the lag problem, reduces the effectiveness of these techniques. In addition, while users found performing tasks such as camera placement more natural with three degrees of freedom than with one or two, they had difficulties managing the full six dimensions of information generated by the Polhemus.

Another way of making modeling easier is to isolate tasks which are especially difficult and to provide users with special methods for performing these tasks. One task for which this tactic is effective is precise object placement. Regardless of the type of input device used, it is difficult to translate and rotate objects so that they lie in specific relationships to one another. To address this, we implemented snapping. Using this technique, users can place one object tangent to another, and then rotate it and move it long the other's surface for very precise alignment. Because the implementation of snapping in Moo makes use of UV coordinate systems, there are two paths on surfaces along which objects can be moved. These

paths, one directly following the UV mapping and one following geodesics, can both be useful, depending upon the specific surface being traversed.

Multi-dimensional input devices are becoming more accessible to users as well as researchers. For example, a simple version of the DataGlove is currently available for under \$100. Moreover, a device similar to the Polhemus, which is not sensitive to metallic objects and costs half as much, is forthcoming. As their availability grows even more, the need for good interaction techniques using these devices will correspondingly increase. As well, since improved equipment will make three- and four-dimensional interactive systems more accessible, enhanced techniques for device-independent interaction will be needed. The work in Moo is a step in this direction.

6. Acknowledgments

I would like to thank my advisor, Andries van Dam, and the members of the Brown University Graphics Group for their help in completing this project. Thanks specifically to Bob Zeleznik and Tinsley Galyean for initially setting up the Polhemus, and to Henry Kaufman and Oren Tversky for their hard work on the device driver. Thanks to David Ascher, Brook Conner, Geoffrey Silverton, and Oren Tversky for their graphics project which facilitated defining UV coordinate spaces for objects in our system. Thanks to John Hughes for his help with the tricky math of UV coordinate systems and geodesics, for proofreading my paper, and for in general being there to answer questions and keep me on track. Thanks to Lisa Borden and Tinsley Galyean whose conversation helped keep me going on many days. Also, thanks to my Mom, my Dad, and my brothers, Josh and Dan, for all of their support and for putting up with me. Finally, special thanks to Jim Grandy, without whose help and support I would not have made it through this year.

7. Bibliography

- [Badler87] Norman I. Badler, Kamran H. Manoochehri, and David Baraff. "Multi-Dimensional Input Techniques and Articulated Figure Positioning by Multiple Constraints," In *Proceedings of the 1986 Workshop on Interactive 3D Graphics* (Chapel Hill, North Carolina, October 23-24, 1986). New York: ACM, 1987, pp. 151-69.
- [Bier86] Eric A. Bier and Maureen C. Stone. "Snap-Dragging," *Computer Graphics*, Vol. 20, No. 4, 1986, pp. 233-40.
- [Bier87] Eric A. Bier. "Skitters and Jacks: Interactive 3D Positioning Tools," In *Proceedings of the 1986 Workshop on Interactive 3D Graphics* (Chapel Hill, North Carolina, October 23-24, 1986). New York: ACM, 1987, pp. 183-96.
- [Bier88] Eric A. Bier. "Snap-Dragging: Interactive Geometric Design in Two and Three Dimensions," U. C. Berkeley Computer Science Division Report No. UCB/CSD 88/416, April 28, 1988.
- [Bier90] Eric A. Bier. "Snap-Dragging in Three-Dimensions," *Computer Graphics*, Vol. 24, No. 2, March 1990, pp. 193-204.
- [Brooks87] Frederick P. Brooks, Jr. "Walkthrough - A Dynamic Graphics System for Simulating Virtual Buildings," In *Proceedings of the 1986 Workshop on Interactive 3D Graphics* (Chapel Hill, North Carolina, October 23-24, 1986). New York: ACM, 1987, pp. 9-21.
- [Buxton83] William Buxton. "Lexical and Pragmatic Considerations of Input Structures," *Computer Graphics*, Vol. 17, No. 1, pp. 31-7.
- [Buxton86] William Buxton. "There's More to Interaction Than Meets the Eye: Some Issues in Manual Input," In *User Centered System Design: New Perspectives on Human-Computer Interaction*, Donald A. Norman and Stephen W. Draper, eds. Lawrence Erlbaum Associates, 1986, pp. 319-338.
- [Chen88] Michael Chen, S. Joy Mountford, and Abigail Sellen. "A Study in Interactive 3-D Rotation Using 2-D Control Devices," *Computer Graphics*, Vol. 22, No. 4, August, 1988, pp. 121-9.
- [Conner89] Brook Conner. "Using the BAGS Modeler," Brown University, November, 1989.

- [Deyo89] Roderic Deyo and David Ingebreetsen. "Notes on Real-Time Vehicle Simulation," In SIGGRAPH '89 Course #29 notes, *Implementing and Interacting with Real-Time Microworlds*. ACM, 1989.
- [Fisk89] Barton C. Fiske and Melissa Y. Gold. "Introduction to the Brown Animation Generation System," Brown University, August, 1989.
- [Foley87] James D. Foley. "Interfaces for Advanced Computing," *Scientific American*, Vol. 257, No. 4, October, 1987, pp. 126-35.
- [Foley90] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Reading, MA: Addison Wesley, 1990.
- [Galyean89] Tinsley Galyean, Melissa Gold, William Hsu, Henry Kaufman, and Mark Stern. "Manipulation of Virtual Three-Dimensional Objects Using Two-Dimensional Input Devices," Brown University, December 20, 1989.
- [Gaylean90] Tinsley Galyean. "SCULPT: Interactive Volumetric Modeling," Brown University, August, 1990, p. 12.
- [Helliwell89] John Helliwell. "3-D Display Helps Make Siggraph Fun - As Usual," *PC Week*, Vol. 6, No. 32, August 14, 1989, p. 22.
- [Millman77] Richard S. Millman and George D. Parker. *Elements of Differential Geometry*. New Jersey: Prentice-Hall, Inc., 1977.
- [Nielson87] Gregory M. Nielson and Dan R. Olsen. "Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices," In *Proceedings of the 1986 Workshop on Interactive 3D Graphics* (Chapel Hill, North Carolina, October 23-24, 1986). New York: ACM, 1987, pp. 175-82.
- [Ouh-young89] Ming Ouh-young, David V. Beard, and Frederick P. Brooks, Jr. "Force Display Performs Better than Visual Display in a Simple 6-D Docking Task," in *Proceedings of the IEEE Robotics and Automation Conference*, May, 1989.
- [Polhemus87] Polhemus, Inc. *3Space Isotrak User's Manual*, Colchester, Vermont, May, 1987.
- [Polhemus90] Promotional literature from Polhemus, Inc., Colchester, Vermont, 1990.
- [SpatialSystems90] Promotional literature from Spatial Systems, Inc., Billerica, Massachusetts, 1990.

- [Robinson90] Phillip Robinson. "MacCAD Takes Off," *MacUser*, Vol. 6, No. 8, August, 1990, pp. 114-128.
- [Sproull90] Robert F. Sproull. "Parts of the Frontier Are Hard to Move," Keynote Address in *Proceedings, 1990 Symposium on Interactive 3D Graphics*, (Snowbird, Utah, March 25-28, 1990). Published as *Computer Graphics*, Vol. 24, No. 2, March 1990, p 9.
- [Sutherland65] I.E. Sutherland. "The Ultimate Display," In *Proceedings of IFIP 65*, Vol. 2, pp. 506-8, 582-3.
- [Wang90] Jih-fang Wang, Vernon Chi, and Henry Fuchs. "A Real-time Optical 3D Tracker For Head-mounted Display Systems," *Computer Graphics*, Vol. 24, No. 2, March 1990, pp. 205-15.
- [Ware88] Colin Ware. "Using the Bat: A Six-Dimensional Mouse for Object Placement," *IEEE Computer Graphics and Applications*, Vol. 8, No. 6, 1988, pp. 155-60.
- [Ware90] Colin Ware and Steven Osborne. "Exploration and Virtual Camera Control in Virtual Three Dimensional Environments," *Computer Graphics*, Vol. 24, No. 2, March 1990, pp. 175-83.
- [Weimer89] David Weimer, S. K. Ganapathy. "A Synthetic Visual Environment With Hand Gesturing and Voice Input," In *Proceedings of SIGCHI*, May, 1989, pp. 235-40.
- [Zelevnik89] Robert C. Zelevnik. "Visualization of Disassemblies of Mechanical Objects," Brown University Department of Computer Science Master's Project CS-89-M5, May, 1989.
- [Zimmerman87] Thomas G. Zimmerman, Jaron Lanier, Chuck Blanchard, Steve Bryson, and Young Harvill. "A Hand Gesture Interface Device," *Proceedings Human Factors in Computing Systems and Graphic Interface*, 1987, pp. 189-92.

A. Appendix

As described in section 3.2.2, Moo provides a way for objects to move along surface geodesics. These calculations are described below. References and notation are taken from [Millman77], which also provides a more detailed explanation of the math.

We are given a two-dimensional coordinate system (u,v) -space with a starting point (u,v) , and a function $x: uv\text{-space} \rightarrow xyz\text{-space}$ which maps a (u, v) point onto a two-dimensional surface (in 3D space). Suppose we start at a point (u_0, v_0) ; we are also given a direction along the surface that we wish to travel, expressed as a direction (b^1, b^2) in (u,v) space. We wish to calculate the new u,v -coordinates of the point $(u_{\text{new}}, v_{\text{new}})$ after it has travelled for a (very small) time Δt . Furthermore, the point projected onto the surface – $x(u, v)$ – must necessarily travel along the geodesic uniquely defined by the initial position and the direction vector (theorem 5.7, p.111).

We must define geodesic; a geodesic is a unit speed curve $\gamma(s)$, defined as $\gamma(s) = x(\gamma^1(s), \gamma^2(s)) = x(u,v)$, which satisfies the equation:

$$(\gamma^k)'' = - \sum \Gamma_{ij}^k \gamma^{i'} \gamma^{j'} \quad \text{for } k = 1,2$$

Recall that in our case, we know $\gamma^1(0)$, which is u_0 , and $\gamma^2(0)$, which is v_0 ; we also know $\gamma^{1'}(0)$ and $\gamma^{2'}(0)$ – they are the components of the direction vector (b^1, b^2) for the motion. For our two-dimensional surface, i and $j = 1,2$. Hence to compute $\gamma^{1''}$ and $\gamma^{2''}$ we need to know the values of $\gamma^{1'}$, $\gamma^{2'}$, and Γ_{ij}^k for $i,j,k = 1,2$. We can then use our knowledge of $\gamma^{1'}(0)$, $\gamma^{2'}(0) = (u_0, v_0)$ and the first and second derivatives of γ^1 and γ^2 to make a quadratic estimate of $\gamma^1(\Delta t)$ and $\gamma^2(\Delta t)$ a short time later.*

* we are really using a second-order Euler method for solving the partial differential equation. Other methods (Runge-Kutta, etc.) are more accurate, but unnecessary in this application.

$$\begin{aligned}u_{\text{new}} &= u + u' = u + \gamma^1'(s) \Delta s \\v_{\text{new}} &= v + v' = v + \gamma^2'(s) \Delta s\end{aligned}$$

We must compute the Christoffel symbol, which is defined as follows (where x_1 denotes $\delta x / \delta u$, etc.):

$$\Gamma_{ij}^k = \sum_{\lambda=1}^2 \langle x_{ij}, x_{\lambda} \rangle G[\lambda][k]$$

(compute for $i, j, k = 1, 2$)

$G[\lambda][k]$ is defined as the (k, λ) entry of the inverse matrix of (g_{ij}) . The functions g_{ij} , called the metric coefficients, contains local information about the surface that depend critically on which coordinate patch has been picked. They are defined by the following equations:

$$\begin{aligned}g[i][j] &= \langle x_i, x_j \rangle, \text{ for } i, j = 1, 2 \\g &= \det(g_{ij})\end{aligned}$$

Hence, computing the inverse by Cramer's rule, we get:

$$\begin{aligned}G[1][1] &= g[2][2] / g \\G[1][2] &= -g[2][1] / g \\G[2][1] &= G[1][2] \\G[2][2] &= g[1][1] / g\end{aligned}$$

Computing Γ_{ij}^k , then, depends entirely upon the values of the partial derivatives (the x_i 's) and the partial second derivatives (the x_{ij} 's). Estimation is done using a symmetric difference around the given point; i.e.,

for a given point u , x is computed at $u + \Delta t$ and $u - \Delta t$, thereby reducing the bias in the computation.

Thus to estimate the partial derivatives, we write:

$$x_1 \cong \frac{[x(u + \Delta t, v) - x(u - \Delta t, v)]}{2\Delta t}$$

$$x_2 \cong \frac{[x(u, v + \Delta t) - x(u, v - \Delta t)]}{2\Delta t}$$

and for the partial second derivatives:

$$x_{11} = \frac{[x(u + \Delta t, v) + x(u - \Delta t, v) - 2x(u, v)]}{\Delta t^2}$$

$$x_{22} = \frac{[x(u, v + \Delta t) + x(u, v - \Delta t) - 2x(u, v)]}{\Delta t^2}$$

$$x_{12} = \frac{[x(u + \Delta t, v + \Delta t) - x(u + \Delta t, v - \Delta t) - x(u - \Delta t, v + \Delta t) + x(u - \Delta t, v - \Delta t)]}{2\Delta t^2}$$

$$x_{12} = x_{21}$$

Having computed all of the above, consider the original point on the surface $x(u_0, v_0)$. Call three vectors at this point X , Y , and N , with X and Y tangent to the surface and N normal to it. Write the direction, d , we wish to travel in as a linear combination of the X and Y vectors:

$$a^1 X + a^2 Y = d$$

The direction in UV space to travel, which we called (b^1, b^2) , can be computed by:

$$b^1 = \langle a^1 X, x_1 \rangle + \langle a^2 Y, x_1 \rangle = \langle d, x_1 \rangle$$

$$b^2 = \langle a^1 X, x_2 \rangle + \langle a^2 Y, x_2 \rangle = \langle d, x_2 \rangle$$

Finally, $(u_{\text{new}}, v_{\text{new}})$ is estimated as follows (using the two-term Taylor expansion):

$$u_{\text{new}} = u_0 + \Delta t b^1 - 1/2(\Delta t)^2 \sum_{i,j=1}^2 (\Gamma_{ij}^1 b^i b^j)$$

$$v_{\text{new}} = v_0 + \Delta t b^2 - 1/2(\Delta t)^2 \sum_{i,j=1}^2 (\Gamma_{ij}^2 b^i b^j)$$

However, once the new (u, v) point has been found, the vectors X, Y, and N must be updated as well. The first step in this process is to compute:

$$X^1 = \langle X, x_1 \rangle$$

$$X^2 = \langle X, x_2 \rangle$$

These values are then used to calculate:

$$X_{\text{new}}^1 = X^1 - \Delta t \sum_{i,j=1}^2 (\Gamma_{ij}^1 X^i b^j)$$

$$X_{\text{new}}^2 = X^2 - \Delta t \sum_{i,j=1}^2 (\Gamma_{ij}^2 X^i b^j)$$

Then, new partial derivatives must be computed at the point (u_{new}, v_{new}). The formula is the same as above. These new x₁ and x₂ values are used to finally calculate the new value of the X vector:

$$X_{\text{new}} = X_{\text{new}}^1 x_1 + X_{\text{new}}^2 x_2$$

Y_{new} is calculated in the same way as X_{new} , and $N_{\text{new}} = X_{\text{new}} \times Y_{\text{new}}$. Finally, to make these vectors an orthonormal basis, the following formulas are applied:

$$\begin{aligned} N &\leftarrow N / ||N|| \\ X &\leftarrow X - \langle X, N \rangle N \\ X &\leftarrow X / ||X|| \\ Y &\leftarrow N \times X \end{aligned}$$

After this, the incremental computation of u and v using the Taylor expansion can be repeated.