

BROWN UNIVERSITY
Department of Computer Science
Master's Project
CS-96-M10

“Three-Dimensional User Interfaces
for Scientific Visualization”
by
Kenneth P. Herndon

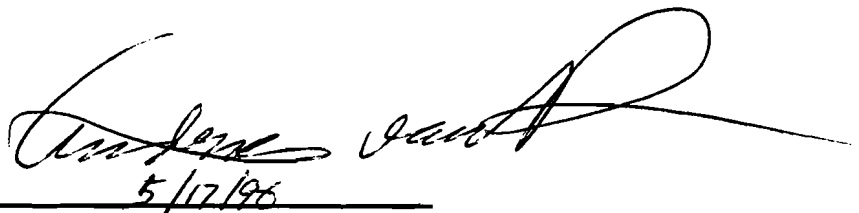
Three-Dimensional User Interfaces for Scientific Visualization

Kenneth P. Herndon

Department of Computer Science
Brown University

Submitted in partial fulfillment of the
requirements for the Degree of Master of
Science in the Department of Computer
Science at Brown University

May 17, 1996

A handwritten signature in black ink, appearing to read 'Andries van Dam', is written over a horizontal line. Below the signature, the date '5/17/96' is handwritten.

Professor Andries van Dam
Advisor

Table of Contents

1	Project Description	1
2	Accomplishments	1
3	Development Environment	2
3.1	Software	2
3.2	Hardware	3
4	3D Widgets and Interaction Techniques	3
4.1	Definitions	3
4.2	Taxonomy of Tasks	5
5	General Design Issues	5
5.1	Don't Mindlessly Transfer 2D User Interfaces into 3D Environments	5
5.2	Look to Real World Human-Object Interaction for Inspiration	6
5.3	Consider Characteristics of Input and Output Devices	7
5.4	Consider Geometric Properties of the Application Environment	8
5.5	Widgets should be as Non-Intrusive as Possible	8
5.6	Provide Adequate Feedback to the User	9
5.7	Map Widget's Geometry and Behavior to Spatial Quantities of Function it Controls	10
5.8	Consider Alternatives to Geometric Widgets when Appropriate	10
6	Implementations	10
6.1	Selection	11
6.1.1	Desktop	11
6.1.2	Virtual Reality	11
6.1.2.1	Virtual Input Devices and Physical Props	13
6.1.2.2	Touch	13
6.1.2.3	Touch Plus Intersection Ray	14
6.1.2.4	Laser Pointer	16
6.1.2.5	Target (Laser Pointer from Eye)	16
6.1.2.6	Aperture	17
6.1.2.7	Glove-Based Interface for Aperture	19
6.1.2.8	Orientation	19
6.2	Manipulation	21
6.2.1	Direct vs. Indirect Manipulation	21
6.2.2	Types of Manipulation in 3D Applications	22
6.2.3	Position and Orientation Techniques	22
6.2.3.1	Interactive Shadows	23
6.2.3.2	Object Handles	24
6.2.3.3	Grid-Aligned Handles	25
6.2.3.4	Data-Space Handles	26
6.2.4	Direct Manipulation Visualization Techniques	27
6.2.5	Manipulation after Selection in Virtual Reality	28
6.3	Navigation	29
6.4	Complex Widget Designs	29
6.4.1	Generalized Probe	30

6.4.1.1	Zero-Dimensional Probe	31
6.4.1.2	One-Dimensional Rake	31
6.4.1.3	Two-Dimensional Plane	32
6.4.1.4	Three-Dimensional Volume	33
6.4.1.5	Interface to the Generalized Probe	34
6.4.2	Menus and Buttons	34
6.4.2.1	Menus and Buttons in IVE's	34
6.4.2.2	Spherical Menus	35
6.4.2.3	Dial Menus	35
7	Other Visualization Techniques	37
7.1	Flux Ball	37
7.2	Smoke Rings	38
8	Annotation System	40
9	Techniques for Real-Time Interaction	40
9.1	Time-Critical Scheduling Algorithm	41
10	Evaluation	41
10.1	User study: Positioning tools	42
10.2	Pilot study: Selection techniques in IVE's	42
11	Future Work	43
11.1	Controlling parameters of visualization techniques	44
11.2	Annotation	44
11.3	Animation	45
11.4	User Studies and Pilot Studies	45
11.5	Navigation	45
12	Acknowledgments	46
13	References	46
	Appendix A: Tracker Calibration	50

1 Project Description

The main goal of this project was to develop novel and productive user interfaces for creating and managing visualizations of computational fluid dynamics (CFD) datasets. The existing commercial software applications for CFD visualization (including [2][16][24]) are full-featured systems for visualizing and analyzing datasets, but are all dominated by two-dimensional user interfaces that can complicate certain 3D operations. In contrast, our approach has been to develop user interface tools that exist within the same three-dimensional space as the data being visualized. We feel that this approach yields more intuitive user interfaces for tasks which are inherently 3D, thus reducing the cognitive distance between a user's intentions and actions. Building on our prior work in 3D user interfaces [12][19][38] and the Virtual Wind Tunnel project at NASA [6], we have implemented a series of interaction techniques and 3D widgets specifically tailored for scientific visualization tasks. The bulk of this work was done using conventional desktop hardware, but the most recent developments have focused on new interaction techniques which exploit unique characteristics of immersive virtual environments. The software for this research was written using our own comprehensive 3D graphics system.

2 Accomplishments

The research accomplishments of the past three years have centered on the designs and implementations of user interfaces for scientific visualization. To support this work, we first implemented a basic scientific visualization application development environment using the Graphics Group's own 3D graphics software system (UGA [44]) including the following visualization techniques: scalar and vector probes (with numerical, colored or tuft displays), streamlines and particle paths, isosurfaces and cutting planes (colored and tufts). Although commercial systems generally support many more visualization techniques than these, we feel that with our system we can experiment with a wide variety of user interface issues that directly relate to the general needs of the scientific visualization user community. Using this environment as a base, we have done the following:

- Designed and implemented a variety of interaction techniques and widgets for desktop and virtual reality systems (Section 4.2) for
 - selecting and placing probes in a dataset
 - controlling parameters of common visualization techniques
 - navigating through an environment
- Performed user and pilot studies to evaluate these user interface designs
 - positioning (Section 10.1)
 - selection tasks (Section 10.2)
- Formalized a methodology for the design of three dimensional user interfaces (Section 5)
- Implemented two visualization techniques (Section 7)
 - flux ball
 - smoke ring

- Implemented a system for graphically annotating fluid flows (Section 8)
- Developed a time-critical scheduling algorithm for managing the computational resources required by typical scientific visualization applications (Section 9)

The two new visualization techniques that we have implemented are described later in this document. The remainder of this report will describe each of the above items in more detail; some of these accomplishments have been presented at computer graphics and user interface conferences.

3 Development Environment

3.1 Software

Over the course of this research, we have implemented two applications with which to experiment with user interfaces for scientific visualization tasks. These applications were written using our in-house graphics system, called the Unified Graphics Architecture (UGA) [44]. The first application was implemented in the FLESH programming language, an interpreted, object-oriented language developed by the Graphics Group. This language served as an interface to the underlying functionality of the UGA system. This first application, which we used for the majority of our user interface experiments on the desktop, was ideal for this kind of experimentation because one could rapidly prototype new interaction techniques and widgets. The downside of FLESH, however, was its poor performance when using it for anything but simple designs.

The Graphics Group then implemented another system, called “trim-lite”. This newer system is written in C++ and consists of a relatively small set of libraries which include classes for many of the same components of 3D graphics applications that the FLESH programming language supported, including geometric objects, cameras, lights and input devices. We chose C++ for this system because it afforded much higher performance than the interpreted FLESH language did. As our interface designs became more complex, we needed this increased performance to properly evaluate new interface designs.

Applications written using the “trim-lite” libraries are compiled and thus run more efficiently than FLESH applications. As part of this grant, we added a library to “trim-lite” which performs all of the low-level scientific visualization functions of our prior application, and designed a framework in which we could continue our user interface experiments both on the desktop and in immersive virtual environments (IVE). The resulting application, which is still under rapid development, includes many of the user interface components of the earlier FLESH application. However, we have focused more strongly on implementing new techniques specifically tailored for virtual reality.

We have tested our system with relatively simple (less than 500,000 points) steady-flow datasets using both regular and curvilinear computation grids. The majority of our work has been done on a curvilinear dataset of airflow velocity (speed and direction) past the body of the Space Shuttle, though we have also used a number of other datasets including a rectilinear time-varying dataset of magma flows in the

Earth's mantle (courtesy of Brown's Geology department), and a rectilinear dataset of electrical potential in the human torso (courtesy of the University of Utah).

3.2 Hardware

On the desktop, we have used both conventional hardware (CRT and 2D mouse), and "fishtank virtual reality" hardware, including a Logitech 6D mouse and StereoGraphics LCD shutter glasses.

In our VR lab, we primarily use a single Ascension Bird tracker for one-handed input in conjunction with a Binocular Omni-Orientation Monitor (BOOM) for head tracking and stereoscopic display. The tracker is equipped with three buttons for additional input. In its default configuration, the tracker controls the position of a simple 3D crosshair cursor in the virtual world.

We also have a glove input device which can be used to input more complex data to the application such as postures and gestures of the hand. Though we have not yet used the glove directly within our scientific visualization application, we have developed the support software to recognize hand postures, and have considered a number of techniques which require this device. We discuss these techniques later in this document.

4 3D Widgets and Interaction Techniques

In the following sections, we will discuss the various user interface issues and designs that we have worked on over the course of this research. We begin with a few definitions, and present a taxonomy of 3D graphics application tasks, then discuss our user interface design methodology, and finish with descriptions of the specific interaction techniques and 3D widgets that we have implemented.

4.1 Definitions

A *widget* is an entity which possesses both geometry and behavior [12]. We define the geometry of a widget to be its visual appearance when rendered to an output device. The behavior of a widget represents its functional role in an application and defines both how it reacts to user interaction as well as how it affects aspects of the application environment when manipulated by a user. At this fundamental level, 2D and 3D widgets are identical. However, on a practical level, they differ in that 3D widgets exist in a 3D scene whereas 2D widgets exist within a 2D windowing environment. We do not consider 2D widgets that have a 3D "look" (typically achieved with drop shadows) to be true 3D widgets.

According to this definition, a wide range of entities can be called widgets. In practice, we divide this spectrum into specific categories (Figure 1). At one extreme are widgets with geometry but no behavior. These entities, which we call "primitive objects", are the building blocks of virtual environments, and are defined by their geometric attributes (position, size, orientation, color, etc.). When modified by a user, there are no side effects in the surrounding environment.

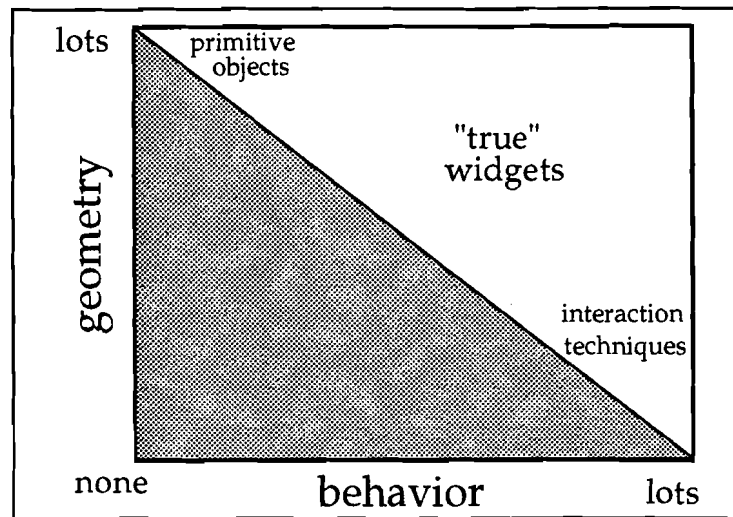


Figure 1: The spectrum of geometry and behavior in widgets. The ratio of geometry to behavior determines whether the object is a primitive, an interaction technique, or a "true" widget with a combination of both geometry and behavior.

At the other end of the spectrum are widgets with behavior but no geometry. The purpose of these entities, or "interaction techniques," is to translate raw user input (e.g., mouse deltas, button presses, etc.) into meaningful actions in a 3D scene. For example, an interaction technique for panning a camera in a desktop application converts 2D mouse deltas into transformations which are applied to the camera viewing a 3D scene. Likewise, an interaction technique for manipulating a 3D object converts mouse deltas into transformations which are applied to the object. Generally, the only feedback that an interaction technique supplies is its actual effect on the scene (e.g., the motion of the camera or object being manipulated).

In the middle of the spectrum lie an array of "true" widgets which contain a more balanced mix of geometry and behavior. Like interaction techniques, the behavioral components of these widgets serve to transform raw user input into values which are meaningful to primitive objects. The geometry of a widget acts as a virtual input device through which we can indirectly modify attributes of an object that may be unnatural or impossible to access directly through a simpler interaction technique (further discussion of manipulation is in Section 6.2).

Both interaction techniques and widgets can be used to modify spatial or non-spatial parameters of primitive objects. Widgets with geometry are often designed to provide feedback to the user, though this is not always necessary. For example, the rake widget, described later, utilizes a one-dimensional slider to set the number of streamlines along its bar. The position of the slider simultaneously determines the distance between the streamlines on the bar and provides feedback to the user about this spatial quantity.

4.2 Taxonomy of Tasks

In interactive 3D graphics applications, tasks can be classified according to the following three categories:

- selection (of objects)
- manipulation (of objects, parameters, etc.)
- navigation (of viewpoint)

Most of the interaction techniques and widgets that we have implemented fall neatly into one of these categories. Section 6 describes the techniques that we have implemented for our scientific visualization application.

This taxonomy expands on Robinett and Holloway's [36] presentation of the manually-controlled actions that may be implemented in an IVE which involve changing the location, orientation or scale of either an object in the IVE (manipulation) or the user herself (navigation). In Robinett's treatment, the selection task is implicitly included as part of manipulation. We have chosen to make selection a unique category because as with manipulation and navigation tasks, there are a wide variety of unique methods for performing this task.

5 General Design Issues

What follows is a description of some of the design issues that we have encountered in our exploration of 3D user interface development. In general, we try to adhere to a few rules of thumb when designing 3D widgets:

- Don't mindlessly transfer 2D user interfaces into 3D environments.
- Look to real world human-object interaction for inspiration.
- Consider the characteristics of input and output devices.
- Consider geometric properties of the application environment.
- Widgets should be as non-intrusive as possible.
- Provide adequate feedback to user.
- Map widget's geometry and behavior to spatial quantities of function it controls.
- Consider alternatives to geometric widgets when appropriate.

The following few sections describes each of these rules in more detail.

5.1 Don't Mindlessly Transfer 2D User Interfaces into 3D Environments

2D and 3D user interfaces are very different from each other, not only in the geometry which defines their visual appearance, but also in their relationship with the underlying application. Most 2D graphical user interfaces (GUI's) are built on top of a relatively small set of fundamental primitives objects (windows, buttons, menus, sliders, etc.) and are controlled by pointing, clicking and dragging with a mouse. 2D interfaces for even wildly different applications are thus very similar to one another in their "look and feel".

In contrast, 3D interfaces, especially those for IVE's, potentially have more task- and application-dependent components¹. This is because 3D interfaces exist within an environment which has many, if not all, of the degrees of freedom of the real world and thus affords more latitude for design than 2D environments. It is natural, therefore, to design specific tools for specific tasks in 3D applications, just as we build specialized tools for all manner of tasks in the real world. The components of 2D interfaces are more general in their applicability than many 3D widgets, and may find their place in 3D applications, even IVE's. However, we must be careful not to simply transfer 2D interfaces into 3D environments without considering the consequences². Instead, we should design 3D interfaces that exploit the unique characteristics of IVE's. Some of the following guidelines suggest ways to do this.

5.2 Look to Real World Human-Object Interaction for Inspiration

One of the most important guiding principles of user interface design, especially 3D interface design, is derived from the observation that human beings are very adept at using tools in the real world to interact with objects in their environment. This ability to design and use tools for constructive (and destructive) tasks sets us apart from other animals, and should be exploited as much as possible by user interfaces for computer applications. When designing user interface techniques, we therefore attempt to identify techniques in humans' real-world interactions that are similar to the tasks we wish to perform on a computer. In some rare cases, we can exactly transfer a real-world technique into a virtual world, but more commonly, we must implement a metaphorical interpretation of the real-world technique. We are forced to do this partly because of limitations of input and output devices, but also because particular tasks in a computer application are not exactly like any real-world tasks. Often times, however, a metaphor becomes a very powerful tool of its own and allows us to perform tasks on the computer which would be impossible in a real-world setting.

In general, we feel that the most successful user interface techniques are those that can be related in some way to human experience of the real world. Certainly, the computer is a relatively new tool in human experience, and thus presents new possibilities for interaction that were not conceivable with previous instruments. In some cases, therefore, people must learn new skills for interacting with computers that they would not otherwise have acquired (e.g., using a mouse). However, by recognizing peoples' familiarity with other tools in the real world, we can facilitate the learning of new tools on the computer.

Many of the interaction techniques and widgets that are described later in this document are derived from observations of real-world tools and human interactions. The other general design issues discussed below reflect some of the factors we must consider when transferring these real-world situations into user interfaces.

-
1. Despite their tendency toward application-dependence, there are still commonalities among 3D interfaces, as listed in the taxonomy in Section 4.2. Manipulation tasks can be further broken down into the affine transformations – (constrained) translation, (constrained) rotation and (constrained) scaling – which are used in application-specific widgets.
 2. Menus, for example, are a staple of 2D interfaces, but, as discussed later (Section 6.4.2.1), are problematic in IVE's.

5.3 Consider Characteristics of Input and Output Devices

Input and output devices play a crucial role in user interfaces because they are the primary channels through which human-computer interaction occurs. Without these devices, there is no communication between the human user and the computer application. Furthermore, each input and output device has its own particular characteristics which define how the device is used by a human and therefore what kind of interaction dialog can take place. For example, 2D mice have one or more buttons which produce discrete events, as well as a mechanism for producing data about relative location. This device was designed for so-called “point-and-click” interfaces, and is therefore difficult to adapt to other interaction paradigms. Certainly, mice have been used for gestural interfaces which push the boundaries of event-based point-and-click designs, but these are still two-dimensional in nature. Also, mice have been used in some research systems as continuous input devices, providing input to analytic functions within an application [15].

2D mice are often used to supply data for interaction techniques in 3D applications, but only when the techniques require just two input values. Since it is impossible to specify full three-dimensional motion with a 2D mouse, the interaction techniques in software must map the two-dimensional input onto some 2D subset of the 3D environment (e.g., translating an object in a plane). In many cases, it may be more appropriate to use a 3D device for this kind of task.

As a general rule, a given input device can only specify as many (or fewer) degrees of freedom as it supports. This fact may seem obvious, but it has important consequences on the design of user interfaces, especially those that support a variety of input and output devices. Specifically, any application which intends to support multiple types of devices must include interaction techniques and widgets which are appropriate for all of them. For example, one of the user studies we conducted (Section 10.1) showed that the interactive shadow widgets were difficult to use in desktop VR because subjects had a hard time using an input device with six degrees of freedom (only three positional) to manipulate an object with only two degrees of freedom. One solution to this problem is to only allow manipulation of the shadow widgets in the desktop environment with the 2D mouse. The shadows may still be displayed in an IVE, but they may not be interactive. However, if the shadows themselves serve another purpose, such as controlling the positions of light sources in the environment, then it would once again be advantageous to manipulate them in the IVE.

Output devices also impose constraints on the user interface. On the desktop, CRT's have a limited number of pixels with which to display interface components. Text must not be so small as to be illegible; widgets must neither be so large that they leave no room for the rest of the interface, nor so small that they are invisible to the user. VR output devices such as the BOOM and other head-mounted displays also have limited resolution and impose similar constraints on interfaces for IVE's³. However, since IVE interfaces typically enable users to navigate through the 3D environment, it is

3. In fact, today's head-mounted displays typically have very low resolution (640x480 pixels) compared with standard workstation screens (1280x1024 pixels), rendering users legally blind. Exceptions include the BOOM (which is technically a “head-coupled” display because it is not attached to the user's head), and some other high-end displays.

possible to get closer to 3D widgets to get a better look at fine details.

5.4 Consider Geometric Properties of the Application Environment

One of the factors that makes 3D interface design more complicated than 2D design is that since a 3D interface resides in an environment which can be viewed from arbitrary viewpoints, it must be designed so that it is still useful from any point of view. Also, unlike 2D user interfaces, which must fit within the physical dimensions of a CRT screen, a 3D interface can exist anywhere within an infinite 3D volume. As a result, the interface designer can not take it for granted that the 3D interface will necessarily be visible at all times. Certainly, a similar problem exists in a 2D desktop windowing environment when windows obscure one another, but the techniques one uses to navigate through the interface and find a particular component are different in 3D. In 2D, the number of pixels on a screen defines geometric boundaries within which the user interface components must fall, so a widget must be on the desktop. In 3D, the widget could be anywhere in a potentially very large volume.

In practice, IVE applications usually confine themselves to some finite region. For instance, in our scientific visualization application, we only interact within a relatively small region defined by the extent of the CFD dataset. Thus, the interface designer need not worry about users “leaving the interface behind” in another non-visible part of the IVE because the entire world is always visible (if not obscured by other objects in the scene, of course). In more complex environments composed of “rooms” ([9]), for example, this becomes more of an issue.

Even in relatively well-contained IVE's, however, user interface designers must consider the scale of the environment. In our application, for instance, we can read in many different kinds of datasets of all different sizes and scales, so the user interface must be able to adapt. It would be pointless to use a 3D widget that measured three inches long in a dataset the size of a one-inch cube. The obvious solutions are to either scale up the dataset, or reduce the scale of the widgets. In the case of a curvilinear dataset, however, there are data at many different scales within the same dataset, suggesting that the user interface must be able to dynamically change scale depending on its location in the scene. We have only begun to explore the problems that this kind of interface presents.

5.5 Widgets should be as Non-Intrusive as Possible

Since the primitive objects in a 3D scene are often the most important components, they must be visible at all times and not obscured by interface geometry. In a scientific visualization application, this is especially true of the visualization primitives (streamlines, cutting planes, etc.). As shown in the user interfaces described later on, we often choose to draw our 3D widgets in wireframe. This reflects a conscious attempt on our part to make the necessary geometry of the widgets less intrusive. In addition to wireframe rendering, we try to make widgets as small as possible. However, as the size of a widget decreases, it can become more difficult to see or pick it in order to manipulate it. A balance must therefore be met so that the interface is usable in a majority of the foreseeable cases.

In the desktop version of our application, we designed the widgets with this

balance in mind. Of course, in any perspective projection of a scene, it is always possible that a given object will be so far from the viewpoint that it is not pickable (though in an IVE, we use a selection technique, described later, which virtually eliminates this possibility).

Both pickability and visual appearance of the 3D user interface components are extremely important issues in IVE applications as well. As we have migrated the 3D widgets into a fully immersive environment, we have found that some of them are very difficult to use without significant redesign of either the widgets themselves, or, as will be discussed later, the interaction techniques used to select and manipulate them. For example, some of the components of the generalized probe widgets, such as the resolution and scale handles, were initially nearly impossible to select in our IVE because they were designed for a mouse-based system. Our evaluations of selection techniques through user studies for IVE's have shown that selection techniques designed specifically for virtual environments can make it easier to pick small objects.

5.6 Provide Adequate Feedback to the User

Feedback to the user is an essential component of a good user interface, and must be incorporated in the design of any interaction technique or widget. Deciding on the exact nature of this feedback is an important part of user interface design. For example, simple tricks like predictive highlighting so that the user knows what *will* happen when she clicks or releases the mouse button can only help usability. There are other more subtle feedback mechanisms, though, which are very important to implement correctly in 3D interfaces. On both the desktop and in VR, for example, we have found that it is very important to maintain correlation between the cursor and the object being manipulated. When moving objects in the real world with our hands, we maintain contact with the object until we let go of it. On the computer, we cannot touch objects without haptic devices, but we *can* maintain the illusion that we are directly manipulating objects by providing visual feedback which *suggests* the contact relationship between our input device and the object of interest.

Though feedback is sometimes discrete, it often takes the form of a continuous loop in our user interfaces. That is, for example, the user thinks to move an object to the left, so she drags the mouse to the left. She then sees the object move to the left, thus confirming the validity of her action. Based on this positive feedback, she continues to move the mouse to the left until she has placed the object at its destination. Ideally, the feedback loop has a very high frequency, thus reducing the cognitive load on the user. Maintaining physical correlation between a cursor and object being manipulated increases the frequency of the feedback loop because it provides *positive* feedback. An example of negative feedback would be an interface which moved an object up and down in response to left-right mouse motions⁴. The performance of the application

4. Unfortunately however, this kind of behavior is typical of 2D user interfaces for 3D applications. For example, moving a 3D object along the x axis is often accomplished by manipulating a 1D slider which is most likely not aligned with the x axis. As a result, users must devote significant cognitive effort to correlate their input with the visible effects on the 3D scene. Moreover, users may not be able to predict which direction on the screen an object will move in response to the slider manipulation unless additional interface components are available which display the current orientation of the scene, for example.

obviously plays a critical role in providing high-frequency feedback to the user (this fact in part motivated our switch from FLESH to “trim-lite”).

5.7 Map Widget’s Geometry and Behavior to Spatial Quantities of Function it Controls

Typically, widgets are used to visualize and modify specific properties of an primitive object or to control parameters of abstract functions. We generally attempt to design a widget so that the degrees of freedom of its manipulable components match the properties or parameters it controls. For instance, a widget for translating objects along a single axis in space should itself be constrained to move only along that axis. If it behaved otherwise, it would be misleading. Similarly, a widget to control a single scalar parameter of a function might also be constrained to translate along or rotate about a single axis (like a physical slider or dial).

Not only should a widget have the same degrees of freedom as the parameters it controls, but it should also, whenever possible, be designed so that its visual appearance matches the qualities of the parameter, and change its own appearance (position, size, color, etc.) along with the changing parameter. For example, the geometry of a widget which controls the size of a sphere of influence might be a bar whose length is the radius of the sphere. Changing the length of the bar modifies the radius of the sphere. In the rake widget described later, a one-dimensional slider controls the number of streamlines that the rake emits. We chose to represent this slider with the geometry of a cylinder in order to reflect the one-dimensional parameter it controls. In our design, the position of the slider on the rake relative to one end of the rake determines the spacing between the streamlines.

5.8 Consider Alternatives to Geometric Widgets when Appropriate

Still other research in the group has focused on developing gesture-based user interfaces for controlling 3D environments. The Graphics Group developed an application, called SKETCH [45], which uses these techniques to facilitate rapid construction of 3D models from simple sketched gestures. This application currently works with either a 2D mouse or tablet. To date, we have not applied this gestural interface research to the scientific visualization application, but we may consider doing so in the future. For example, using a sketch-style interface, scientists could quickly instantiate new widgets in a dataset by roughly sketching their salient geometric features into the 3D scene.

Eliminating the geometry of widgets from the scene is likely to be a good thing in some cases, but we must be sure not to lose their functionality in the process. Also, we must be wary not to misuse gesture-based interfaces when more traditional geometric widgets might be more effective.

6 Implementations

The following sections describe implementations of the user interfaces that we have experimented with over the course of this research. When appropriate, we discuss

how particular design decisions were made, and how these designs relate to the methodology described above. The first three sections address each of the tasks outlined in the taxonomy above. The final section on complex widgets describes some user interface designs for controlling abstract parameters of an application environment. These widgets require users to combine the selection and manipulation tasks to form a dialog with the interface.

6.1 Selection

Selection is one of the fundamental tasks in any interactive graphics application. Through selection, users indicate which objects they are interacting with and specify which parameters they wish to view or modify. The method used to select an object, however, differs greatly depending on the type of input and output devices at hand and the application itself. We have implemented a variety of techniques for both desktop and VR systems.

6.1.1 Desktop

There are a variety of configurations of input and output devices for desktop systems, ranging from the conventional CRT and 2D mouse combination, to more elaborate stereo displays and 3D input devices. Selection techniques for the latter will be discussed in the next section since many of the issues are the same.

When using conventional desktop hardware (2D mouse input and standard CRT output), the method we usually utilize for selection is *ray intersection*. We construct a ray based at the focal point (viewpoint) of the camera through the point on the film plane which corresponds to the position of the 2D mouse cursor. By testing for intersections between this ray and all of the geometry in the scene, we can determine which object the mouse cursor was “over” in the 2D projection of the scene, and select that object (usually in response to a button press). We have found that this technique is very effective on the desktop because from a perceptual point of view, it emulates the “point and click” behavior of 2D desktop windowing systems. Consequently, this is a general method of selection for desktop applications that have both 2D and 3D components. Also, since the mouse is a virtually noiseless and thus very precise input device, we are able to select very small objects which project to only a few pixels on the screen. As we will see, this is not the case for IVE input devices.

6.1.2 Virtual Reality

Just as on the desktop, selection techniques are used in IVE's to specify which object(s) in the environment the user wishes to interact with. In the real 3D world, of course, people “select” objects by touching them with their hands, or indicate a particular object in the distance by pointing in its general direction. When designing and implementing selection techniques for virtual reality applications, we can look to this real world human behavior for inspiration. However, in VR applications, the selection task is complicated by a number of factors, including limitations imposed by input and output devices as well as by software techniques.

In VR, user actions are mediated by input and output devices which are often imprecise and which either lack or distort perceptual cues that we take for granted in the real world such as haptic feedback, stereopsis, field of view, texture (both visual and tactile), and sound. Display devices like the BOOM provide stereoscopic views, but some people are not able to resolve depth from this type of display. Other head-mounted display devices typically provide fairly low-resolution images, making it difficult to resolve small objects. Magnetic 6D trackers introduce problems as well. First, they are not always accurate in many real environments because they are adversely affected by metallic objects and electronic devices in the physical environment. This can cause significant registration error between the actual and displayed position of the tracker. A number of techniques have been proposed and implemented to correct for static distortions (those that do not change significantly over time) of this type [5][13][17], including a method developed here at Brown [18] (see Appendix A for a discussion of these methods).

Secondly, the data reported by magnetic trackers is somewhat noisy, so even when they are held perfectly still, the 3D cursor in the scene appears to randomly jump around. The magnitude of this noise in our tracker is approximately 0.1". The combination of hardware and software used to convert magnetic fields into position and orientation values that can be used by our software introduces lag into the system. This results in a delay between the actual tracker movement and the display update in the IVE. As shown in [30], lag contributes significantly to error.

On the software side, the *selection tests* used by most virtual reality applications consist of precise, mathematical tests (e.g., ray intersection or point enclosure). While we have found that these seem to work well for desktop applications, they are not nearly as successful in IVE's. As our pilot studies have shown, this is in part due to the physical limitations of the tracking and display hardware we use. However, there may also be more subtle phenomena at work. For example, since IVE's strive to provide an experience which mimics many of the perceptual qualities of our real-world experience (including stereopsis, wide field of view, etc.), users of IVE applications may thus presume that they can interact with objects in an IVE in the same way that they interact with objects in the real world. Unfortunately, computers are not yet adept at inferring user intentions exclusively from the kind of vague indications which humans are accustomed to using for communicating with one another in the real world (pointing, gesturing and speaking, for example). As a result, designing effective selection techniques for VR applications is a tricky process. Our general design methodology has been to look to the real world for examples of how people select, indicate or manipulate objects, and transfer qualities of these interactions into software techniques in an IVE. Often, the resulting interaction technique in an IVE is very different from its real world source since the software technique must both cope with limitations of the hardware devices, as well as exploit the "magical" properties of an IVE (such as the ability to manipulate objects at great distance, which can not easily be done in the real world).

In the following sections, we will describe the most successful of the techniques that we have implemented. We are also conducting informal user studies of these techniques which are helping to guide our designs. These studies are discussed in a later section of this report.

6.1.2.1 Virtual Input Devices and Physical Props

A virtual input device in an IVE is analogous to the mouse cursor on the desktop. It is a piece of geometry which represents the state of the input device(s) currently being used. In the case of a single 3D tracker input device, the virtual input device (VID) might be a simple crosshair cursor (Figure 2a). Generally, the appearance of a VID depends on a number of factors, including the type of physical input device, the task it is being used for, and any physical modifications that have been made to the device itself (props). Some of the selection and manipulation techniques that we have implemented were motivated by observations of how people operate with tools in the real world and are implemented with these metaphors in mind. Therefore, when we have felt it appropriate, we have used props to emphasize the metaphor. In other research [21], props have been shown to aid users' understanding of user interfaces for virtual reality applications.

We make use of a number of different props in our lab, including a drumstick and a ski pole handle. We modify the geometry of the VID to suggest the shape of the prop (Figures 2b and 2c). Though this is not strictly necessary, it is often helpful because a user can more easily correlate what she sees in the IVE with the physical sensations she perceives of the actual object in her hand.

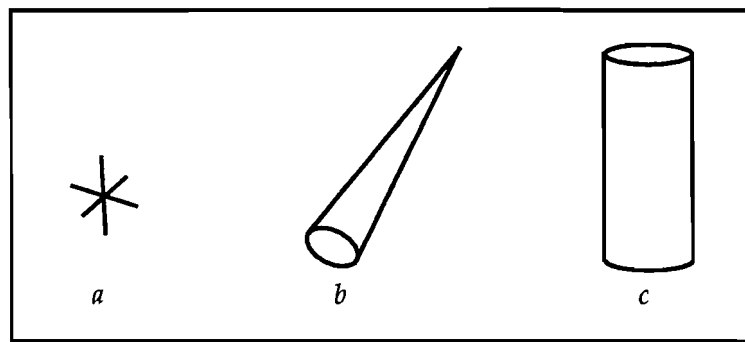


Figure 2: Three types of virtual input devices. *a*) a simple crosshair cursor; *b*) a cone (used with the drumstick); *c*) a cylinder (used with the ski pole handle)

6.1.2.2 Touch

In terms of borrowing ideas from the real world, this might be considered the most straightforward selection technique since humans are very familiar with touching objects first in order to manipulate them. From an implementation point of view, the input to this technique is also fairly simple, requiring only a 3D position (e.g., from a tracker) and a means of signaling to the application when to select an object (e.g., a button press, voice command, etc.). We have implemented two variations of the touch selection technique in our system. In the first, the position of the tracker determines the placement of the 3D cursor used for the selection test. When the 3D cursor is placed inside the geometric boundary of an object, that object can be selected by pressing a button on the tracker. We may use other input methods such as speech recognition to replace the button as a way for the user to signal selection. This technique is similar to touching objects in the real world, except that there is no haptic feedback (i.e., subjects

can not feel the object in the virtual world). One drawback of this technique is that it requires that the desired object be within reach. If the object is out of reach, then the subject must first navigate to the vicinity of that object in order to select it, and the selection task thus becomes a two-step process.

The second variation was inspired by observations of glass blowers who manipulate objects at a distance with long sticks so as not to be burnt by the hot glass. In our implementation, we use the drumstick prop, which emulates the glass blower's tool, and place the 3D cursor at the end of the corresponding VID in the virtual environment. Using the drumstick increases the distance from the user that objects can be selected, but may make it more difficult to select objects at close range due to the awkwardness of holding the stick in these positions. It is still not clear which of these techniques is better overall, but they clearly each have particular strengths and weaknesses. User studies will, of course, help to determine which approaches are better.

Note that due to noise in the position values reported by the tracker, coupled with normal jitter in the user's hand, the VID in this technique (and in others) appears to jump around even when attempting to hold it perfectly still. This phenomenon indicates that objects must be larger than some minimum size in order to be selectable with this technique (or that the technique itself must be modified). The pilot studies described later attempt to define this threshold. In the second variation, since the orientation of the tracker influences the position of the end point of the VID which is used for the selection test, this technique is susceptible to errors from noise in the orientation of the tracker as well as positional noise. In practice, these factors present severe usability problems which we have attempted to alleviate by modifying the technique (see the technique described in the next section).

Testing whether a given input point is within the geometric boundary of an object can be computationally very complex. To reduce this complexity, we can use a simplified geometric representation for objects in the selection test. In our first implementation, we used a sphere scaled to the 3D extent of the object. With this approximation, a simple analytic test could determine roughly when the cursor was in the vicinity of the target object. However, this simplified technique posed problems when the actual geometry of the visible object was very different from a sphere (e.g., if the object is convex). Currently, we are using a polygonal collision detection system called "I-Collide" developed at UNC, Chapel Hill [11], which can accurately detect exact collisions between a VID (represented by a geometric object) and the objects in the scene at interactive rates.

6.1.2.3 Touch Plus Intersection Ray

In our evaluations of the touch method described above, we found that even though it seems like a very natural technique, and seemed to work fine for selecting large objects, it is nearly impossible to use it to select small or narrow objects. The exact minimum size depends on a number of factors which we have attempted to quantify in our user studies described later. We have augmented the touch technique to remedy this shortcoming. In the basic technique, we simply test whether the cursor is inside the geometric boundary of each object (using either the simple spherical test or the true collision detection method). Due to the noise in the tracker and instability in the user's

hand, this technique is extremely susceptible to the effects of temporal aliasing. Since there is no haptic feedback in this system to alert the user that she has touched a given object, the VID is allowed to pass through any objects. If the frame rate is not sufficiently high, or if the noise in the tracker and user's hand is significant compared with the size of the target object, there may be cases when the user feels that she has placed the cursor in the correct position, but still can not select the object. This situation most often occurs when at time t , the cursor is either inside the object or just to one side of it (Figure 3). Then, at time $t+1$, the cursor has passed either outside the boundary of the object or to the opposite side. In the touch technique, when the button is pressed at time $t+1$, the selection test obviously fails⁵ (it can still succeed in some cases if the approximate spherical test is used). However, if we consider the line segment between the sample point at time t and the position of the tracker at time $t+1$, we can determine which object the cursor passed through by looking for intersections between this line segment and all of the objects in the scene. When the button is pressed at time $t+1$, we select the appropriate object.

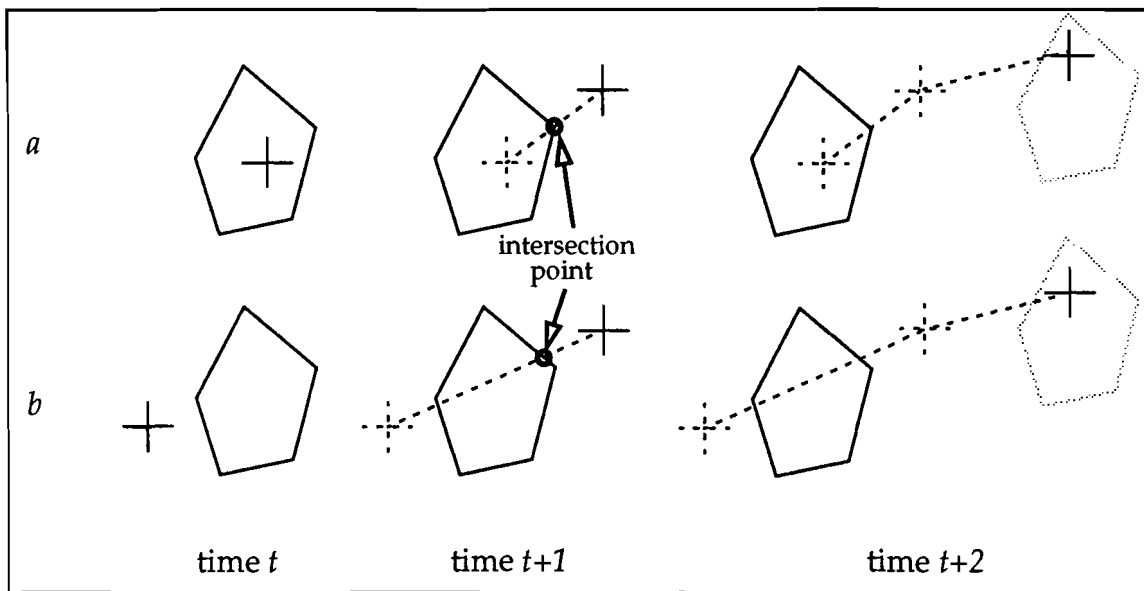


Figure 3: Augmenting the touch selection technique. At time t , the cursor is either inside the geometric boundary of an object (a), or just to one side (b). At time $t+1$, the cursor is outside the object (a), or on the opposite side (b). Finally, at time $t+2$, the cursor may have entered a second object (c). The dotted line segments in b and c are tested for intersection with the object.

Adding this heuristic does not by itself quite make a successful technique because if the user waits until time $t+2$ before pressing the button (which is likely considering each frame is displayed for less than 0.1 second), then the line segment between the two sample points may not intersect any objects. We have thought of two possible solutions to this problem. First is to “remember” the last object that was intersected for some time interval ($1/2$ to one second) during which any button press will select that

5. It is also possible that at time $t+1$, the cursor has entered the boundary of an adjacent object, in which case, that object would be selected instead of the intended one.

object. This method would still fail if the time between frame updates was longer than the memory interval. This will not occur in an IVE because the frame rate must be higher than 10 Hz.

Second, we might use a spatial measure to determine which object to “remember.” That is, until the cursor travels more than a certain distance away from the last intersection point, a button press will select the last object that the cursor passed through. Of course, both the length of the time interval and distance travelled by the cursor should be determined by user studies. While we have not yet determined the optimal parameters, our pilot studies do indicate that the addition of these heuristics greatly improves the usability of the touch technique.

6.1.2.4 Laser Pointer

This technique gets its name from real laser pointers which are used in darkened rooms to point out distant objects. It uses ray intersection to perform selection similar to the desktop selection technique described earlier. In this technique, however, the base point and direction of the selection ray are determined by the position and orientation of the tracker, respectively. The ray points down one of the principal axes of the tracker’s coordinate system (we have chosen the z axis). We generally use this technique with the drumstick prop, holding it as if it were a laser pointer. The physical prop reinforces this metaphor.

In our pilot studies, we found that this technique, though easily learned, presented severe problems when trying to select small objects even at close range. The noise from the device coupled with the inherent instability in one’s hand causes the direction of the intersection ray to fluctuate by as much as ± 5 degrees. At a distance of three feet, this error amounts to 6.25 inches, suggesting that objects any smaller than this are effectively unselectable at this distance. The techniques we describe below introduce methods to reduce the adverse effects of tracker and hand noise.

6.1.2.5 Target (Laser Pointer from Eye)

The target technique is also based on ray intersection, but borrows even more from the 2D desktop techniques than the laser pointer. In this technique, we cast a ray from the viewpoint, controlled by the position of the user’s head, through the 3D cursor in the IVE, determined by the position of the tracker. This technique was inspired by and is similar to looking at a target through the sight on the barrel of a gun, or to holding up one’s thumb to measure the size of a distant object. Although there is still noise in the position of the tracker, and jitter in the user’s head and hand, we have observed in our pilot studies that the distance between the head and hand, which determine the basepoint and direction of the intersection ray, respectively, do in fact reduce the overall error in this technique (compared with the laser pointer which is adversely affected by both positional *and* rotational noise in the tracker). Given the absence of haptic feedback in our system, users may also find it more “natural” to select objects from their point of view than from their hand (this may be similar to the decreased accuracy of shooting a firearm “from the hip” compared with using the sight on the barrel).

When using this technique, we must also consider the fact that a stereoscopic image of a scene is produced from two separate eye points (cameras) simultaneously that are positioned side by side to match the physical configuration of the human eyes. Since the base point of the intersection ray in this technique is controlled by an eye point, we must decide which eye point to use. We know that most people have a so-called “dominant” eye which they favor over the other when performing tasks in which a single point of view is required. A simple test can determine which eye is dominant, and we can adjust the technique appropriately.

6.1.2.6 Aperture

This technique augments the target technique with an additional feature to help alleviate the adverse effects of error from noise. The visual representation of this technique consists of a circular aperture centered on the cursor point which is marked with a crosshair (Figure 4a). We have experimented with two uses of the aperture. The first (Figure 4c) places the aperture at the location of the 3D cursor, and aligns it with the film plane of the camera viewing the scene. As with the target method described above, a user places the aperture “over” the object(s) she wishes to select, then presses the button on the tracker. This configuration can be used with or without the drumstick prop, and the VID is modified appropriately.

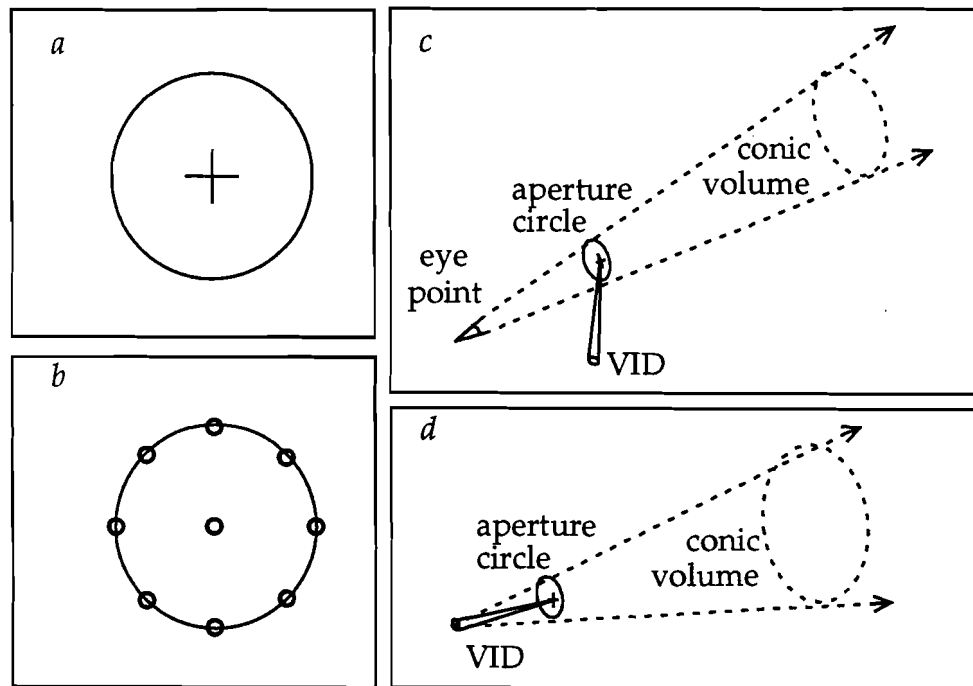


Figure 4: The aperture technique. a) The basic aperture geometry. b) Small circles represent the positions of intersection rays cast through the center of the aperture and representative points on the perimeter. c) The conic volume described by the viewpoint and aperture. d) The conic volume described in the “flashlight” configuration. Both c and d are shown with the drumstick VID. In our implementation, the conic volume is semi-infinite.

In the second configuration (Figure 4d) we use the drumstick prop and place the

aperture geometry at the end of the stick, but this time align it with the plane perpendicular to the axis defined by the stick (similar to the intersection ray in the laser pointer technique).

In either configuration, the combination of a direction vector and the aperture circle defines a conic volume in space. In the first configuration, the apex of this cone is coincident with the eyepoint of the viewer. Any object which appears from the user's point of view to be inside the circular aperture can potentially be selected. In the second configuration, the effect is more like a flashlight sweeping through a region of space; any object "illuminated" by the flashlight is potentially selectable. A similar technique using the flashlight metaphor was implemented by Liang [29] in the MR Toolkit system, but the from-eye version was not implemented.

The radius of the cone can be easily modified in the first configuration simply by drawing the VID closer to or further from one's eye. In the "flashlight" configuration, however, an external control is needed to change the radius of the aperture circle. Currently, we adjust this parameter from a command line interface, but we have envisioned a number of possibly more attractive possibilities including using voice recognition, a hardware dial on the physical input device, or twisting the stick around its long axis (the central axis of the conic volume). We did try this last technique, but quickly discarded it after we found it very difficult to continue pointing the stick in the same direction while twisting it.

Intuitively, the objects that the aperture-based techniques should select are those that fall within the conic volume. This can be expressed as a test for intersections between the conic volume and all of the objects in the scene. We have tried the following techniques:

- Ray intersection from the base point through center of aperture (by itself, this is a degenerate case equivalent to the laser pointer or target techniques).
- Ray intersection from the base point through representative points on the aperture circle (Figure 4b).
- Project vertices and edges of objects in the scene onto the plane defined by the aperture circle.
- Using I-Collide to detect interpenetration of the conic volume with the objects in the scene.

I-Collide obviously is the technique of choice because it returns an exact solution, though in some complex scenes (e.g., with isosurfaces), it may be too slow because of the large number of polygons. In most of the simple scenes we have used it in, however, I-Collide seems to work very well and at interactive rates.

The first two tests above use the same ray intersection tests that the laser pointer and target techniques use. Computationally, ray intersection is inexpensive compared with projecting vertices and edges of objects in the scene onto a plane, but can easily miss objects that do lie within the boundary of the aperture circle and thus should be selected. The vertex and edge projection tests reduce the 3D intersection test to a 2D problem. Once the vertices and edges are projected onto a plane, we need only determine whether they fall within the circle. However, this technique is computationally very expensive and currently can not be used in complex scenes at

interactive rates without hardware acceleration.

Since the aperture technique defines a volume of space, it is possible that we may have multiple candidate objects for selection. In the event that more than one object lies inside the conic volume, we may either select all of the objects, or identify a single object for selection based on some criteria. In practice, there are a variety of mathematical tests to determine which of a number of objects to select. One possibility is to choose the object whose center point is closest to the base point of the intersection ray (either the viewpoint or location of the tracker, depending on the particular configuration in use). This method presents a problem, however, when a user attempts to select an object near the center of the aperture, but a closer object is partially inside the edge of the circle. In this case, both objects are candidates, but the second, closer one will be selected because it is closer to the viewpoint.

Another possibility for choosing a single object among many potentially selectable objects, and one that appears to be the most intuitive, is to select the object closest to the ray passing through the center of the aperture circle. Note that this object may not be the closest object to the viewer. This method recognizes the user's intuition (backed up by anecdotal evidence in our user studies) that the more "centered" an object is in the aperture, the more likely that it will be selected.

A third option is to select an object based on its apparent size (e.g. select the largest object in the aperture). In practice, this is not such a good option, since it may be difficult to select a small object next to a larger one. Also, if two objects of roughly the same size are at different distances from the viewpoint, their apparent size will, in a perspective projection, be very different as well. Using the size test to determine which object to select may work in this case, but a test based on distance would probably work just as well. As mentioned, in practice, we have found that the distance test is more appropriate than an apparent size test.

Note that this technique is also subject to modification based on each user's dominant eye.

6.1.2.7 Glove-Based Interface for Aperture

Though we have not implemented it yet, we have designed a glove-based interface to the aperture technique which we feel is more natural than the two configurations described above. This technique utilizes the posture recognition software in our system to identify when the user has shaped her hand in a pinching posture. When this posture is recognized, the aperture geometry is drawn between the index finger and thumb. As with the first configuration of the basic aperture technique, the aperture is aligned with the film plane. Two advantages to this technique are that the user does not have to hold a prop, and also that the size of the aperture can be adjusted simply by moving one's fingers further apart or closer together.

6.1.2.8 Orientation

The orientation selection technique selects objects in an IVE by comparing the orientation of the tracker with the orientations of objects in the scene. Any objects which approximately match the orientation of the tracker are candidates for selection.

The inspiration for this technique was the observation that in the real world, when we attempt to grab an object with our hand and fingers, we first must configure our hand so that it conforms to the part of the object we reach for (e.g., the handle of a cup, the middle of a bar, etc.). At a very gross level, the task we perform is matching the orientation of our hand with that of the target object. We can approximate this with a simple mathematical test.

According to this technique, the shape of an object plays a direct role in determining how it can be selected. In the UGA system, primitive objects initially have a canonical uniform scale. Given this property, the scale components of an object's current transformation matrix (CTM) can reveal information about its shape. Long thin objects and flat objects can be easily identified by significant differences in their x , y and z scale components. Of course, objects which are long and thin but which are not aligned with a principal axis will not be so easily identified. In general, determining the shape of an object may be a harder, more subjective problem that involves at least an analysis of the object's geometry, and perhaps even some higher-level semantic knowledge about important features of the object (such as the handle of a cup or knob on a door). However, for some simple cases, we can get reasonable behavior under the current scheme.

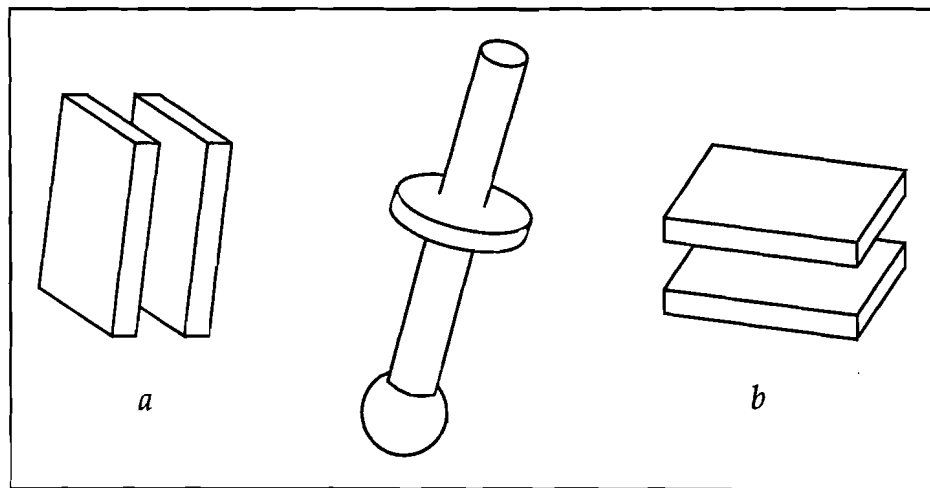


Figure 5: Orientation selection technique. For this technique the cursor geometry is a pair of parallel plates which indicates the current orientation of the tracker. This cursor may be used by itself or in conjunction with a VID such as the drumstick. a) shows the cursor orientation that would select long, skinny objects like the bar of the object in the middle of the figure. b) shows the cursor orientation that would select short, wide objects, like the disc on the bar. The ball at the end of the bar presents something of a problem for this selection technique. This can be remedied by adding a heuristic which identifies uniformly-scaled objects and compares distance to the cursor rather than orientation.

In our application, we tested this technique on the rake widget, which consists of a bar (a long thin cylinder), a slider on the bar (a flattened cylinder), and a ball at one end of the bar (a small sphere). We modified the geometry of the VID for this technique so that the orientation of the tracker was clearly represented (two parallel flat blocks placed side by side). As the user rotates the tracker, so does the cursor rotate and thus

indicate the kinds of objects that can be selected (Figures 5a and 5b).

Note that if this selection technique uses only the relative orientations of the tracker and objects in the scene to determine which objects it will select, and not their respective positions, it can select objects that are a significant distance from the tracker (or even outside the view). In our initial tests, we found this behavior unnatural. To cope with this drawback, we added the requirement that the tracker had to be within some reasonable distance from an object in order to select it, regardless of the similarity between their respective orientations. Checking distances in this way may also be required in order to choose among multiple candidates for selection when the scene is crowded.

The orientation of objects which are uniformly scaled, like the sphere at the end of the bar, is not readily apparent. In these cases, this selection technique uses only the distance measure to determine selectability. In this particular instance, a glove-based posture recognition interface might be more effective – the user would simply shape her hand to fit the desired object. Such an interface might compare the convex hulls of the user's hand and nearby objects and pick the one with the closest match.

6.2 Manipulation

Manipulation is a generic term which describes any of a number of ways to interactively modify the state of objects in a computer application. Manipulation in a 3D graphics context includes applying affine transformations to objects, discrete actions such as pressing buttons or complex actions like gestures or speech acts which are interpreted by a user interface as modifications of primitive objects. The key concept is that manipulation implies interactivity, and that therefore a user interface can be characterized by the types of manipulations it requires one to perform.

6.2.1 Direct vs. Indirect Manipulation

Most types of manipulation in user interfaces can be categorized as either direct or indirect. In his classic article on the subject [37], Shneiderman explains that a direct manipulation user interface is one in which the human user is presented with a visual model of a problem domain, and that the interaction dialog includes “continuous display of the object of interest” and “rapid, incremental, reversible operations whose impact on the object of interest is immediately visible.” This definition was proffered in 1983, when the art of graphical user interface design was still in its infancy. It stands in stark contrast to indirect manipulation found in batch, menu or command-line interfaces which generally require users to maintain an abstract mental model of a problem that conforms to a specific specification language (e.g., the command keywords and syntax). Since this first definition of direct-manipulation, many others have presented their own versions. For example, Laurel [28] stresses that direct-manipulation interfaces present the “continuous representation of the potential for action.”

Direct-manipulation interfaces, when implemented well, give users a sense of being in control of the application, and reduce the *cognitive distance* between a user's intentions and the resulting physical actions she must take. By relying more heavily on

visual perception and cognition (through the use of icons and other graphical elements) than on abstract thought processes required by text-based command interfaces, direct-manipulation interfaces can help users be more productive.

As we have experimented with different interaction techniques and widgets for 3D graphics applications, subtle variations of direct-manipulation interfaces have emerged. In its basic form, users directly control the object of interest and there are no side-effects. This type of direct-manipulation occurs in a 2D or 3D graphics application, for instance, when a user drags a shape or geometric object across the canvas or from one point in space to another. If manipulating this object has additional effects on other objects in the environment, then it is itself a component of the user interface. In this case, though the user has directly manipulated the widget, she has also *indirectly* manipulated some other part of the environment. Thus, direct-manipulation interfaces often incorporate and rely on indirect manipulation.

All of the widgets described below have significant direct and indirect manipulation elements. Some of the interaction techniques described earlier, however, do not provide a visual representation of themselves beside their effect on the scene. The use of predictive feedback, such as highlighting the object(s) that would be selected if the user pressed a button, for example, do provide a sense that the user is wielding a tool which can somehow modify the environment.

Designing good direct-manipulation interfaces is a tricky business, and requires deep insight into the exact nature of the tasks for which they are developed. A well-designed direct-manipulation interface can greatly help task execution, but a poorly-designed one can actually be more difficult to use than a non-graphical, indirect-manipulation interface. Thus, direct-manipulation does not necessarily equate with ease of use [23].

6.2.2 Types of Manipulation in 3D Applications

The most common types of manipulation tasks in 3D graphics applications are inherently geometric. That is, they involve changing the current transformation matrix (CTM) of 3D objects. Modeling, animation and scientific visualization applications all provide techniques for modifying the position, orientation and scale of objects, but the exact interaction techniques and widgets that one uses differ from application to application. Other attributes may also be manipulated, such as the color or transparency of an object, or higher-level attributes like the spacing of a gridded floor plane, or the number of streamlines on a rake in a scientific visualization application. However, whatever the parameter, a 3D user interface for modifying it almost always involves some kind of geometric manipulation. In the following sections, we discuss some of the widgets and techniques that we have developed for modifying parameters of 3D objects.

6.2.3 Position and Orientation Techniques

Positioning and orienting objects in 3D are two forms of manipulation that are widely used in 3D graphics applications. We designed the techniques described in the following sections for use with conventional desktop hardware: a 2D mouse and CRT.

Thus, these techniques aim to overcome many of the difficulties which result from using 2D devices for 3D interaction tasks. We have begun to use these same techniques in VR, and have found that while some are still useful, others must be (and have been) abandoned or at least significantly redesigned in order to be usable in VR. Finally, the selection techniques described above each suggest their own unique form of manipulation once an object has been selected. The final subsection below describes these manipulation techniques.

6.2.3.1 Interactive Shadows

When using desktop hardware, the default positioning technique in our system is direct-manipulation screen-aligned translation (objects move in the plane parallel to the screen plane). However, since this is a 2D mouse-based technique, the user must change the viewpoint to move objects in other planes. To move objects in three-space with the 2D mouse without changing the viewpoint, we have added “interactive shadow” widgets (Figure 6) to this environment. These shadow objects are generated for every 3D object, provide a valuable depth cue, and can be displayed on any axis-aligned plane. Further discussion of this tool is in [20]. Note that in the figure, the shadows on the floor plane do not contain all of the detail present in the widgets above them. This is done in part to decrease the number of polygons in the scene (since the shadow widgets are geometric copies of the widgets), but also because research has shown that human perception does not necessarily requires shadows to be exact [41].

Our initial attempts to use the shadow widgets as interactive tools in VR have been relatively unsuccessful. Since the input devices we use in VR provide the additional degrees of freedom that are lacking on the desktop, the shadows are no longer necessary as manipulation tools. They do, however, still provide a valuable depth cue. The major problems with the shadows as manipulation tools in VR is that they can be difficult to select, at least with the touch selection technique. Also, since the input device is not physically constrained to the same degrees of freedom as the shadow, users often feel disconcerted when the shadow does not exactly follow the position of their hand. We expect that the shadows may be somewhat more usable with the target and aperture techniques, and plan on testing this in the near future.

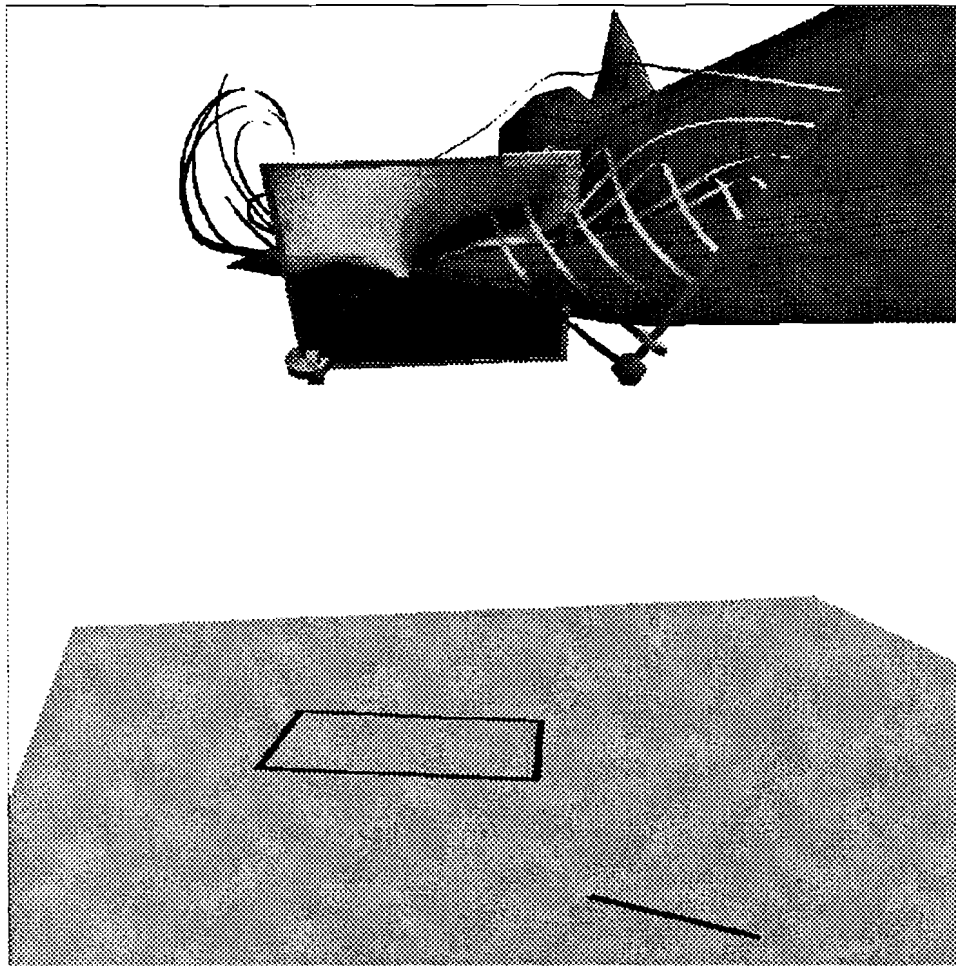


Figure 6: Interactive shadows. The shadow widgets for a cutting plane and a rake are projected onto the floor plane. The position of either probe can be modified by dragging the probe itself or its shadow. Manipulating a probe translates it in a plane parallel to the film plane of the camera viewing the scene; manipulating its shadow widget moves both the shadow and the probe in a plane parallel to the shadow plane. Note that the shadows also provide a useful depth cue.

6.2.3.2 Object Handles

Some geometry, such as the Shuttle's fuselage, can easily obscure the shadow widgets projected onto a floor or wall and thus render them unusable. To address this problem, we have implemented another technique for moving objects in 3D with a 2D mouse, called "object handles" (Figure 7) [12]. With this technique, we attach three new objects (in our case, simple line segments) to the 3D object and align them with the principal axes of world coordinate system. Dragging one of these handles translates the 3D object along the axis defined by that line. These widgets offer much of the same functionality as the "interactive shadows," but provide no depth cues. Their main advantage over shadows is that they are always attached to the 3D object. If the 3D object is visible, then so is the positioning widget.

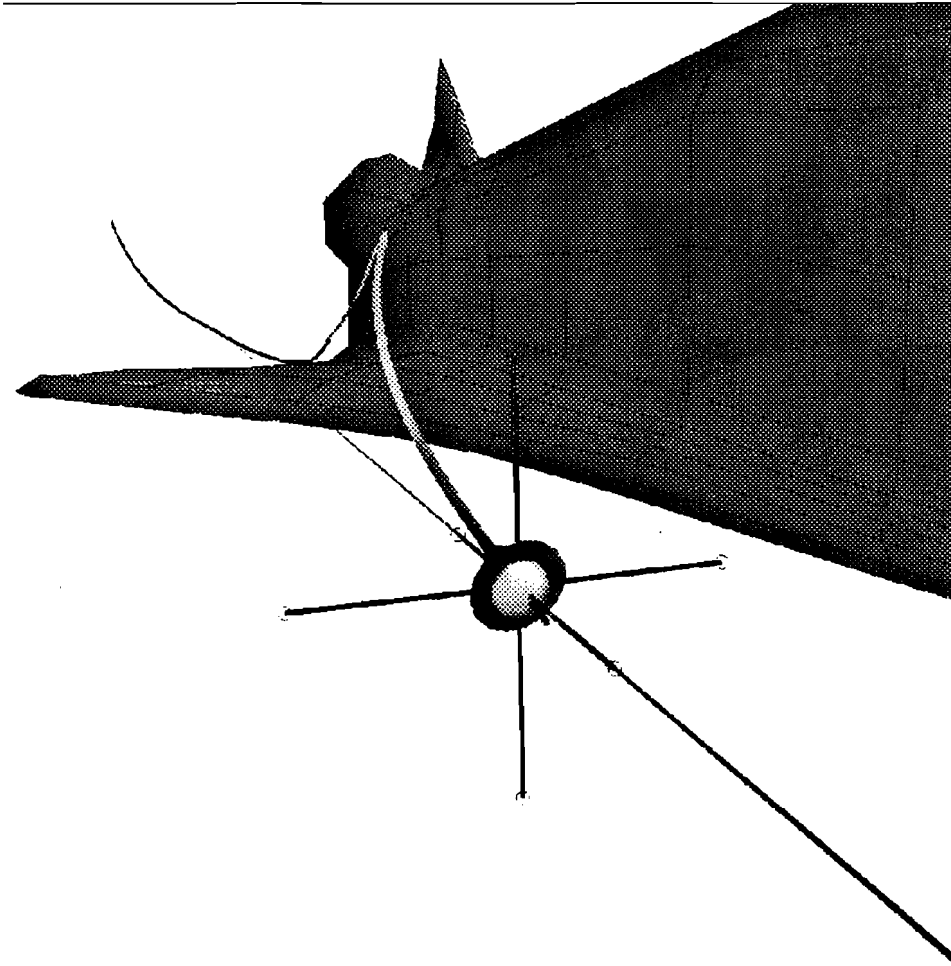


Figure 7: *Object handles*. The object handles widget is attached to a probe. The purple line represents the current translation axis.

These widgets also provide visual feedback to user actions in the form of projection lines and ghosting. A purple projection line⁶, drawn when a user drags one of the three handles, indicates the widget's restricted degrees of freedom. Also, a ghosted copy of the handles widget is drawn in the starting location to indicate the distance that the widget has been moved. When the user has finished dragging the widget (indicated by a mouse up event), the projection lines and ghosted copy are both undrawn.

6.2.3.3 Grid-Aligned Handles

Both the shadow and object handle widgets use axes in the Cartesian coordinate space to help position objects more easily in 3D. We have also designed similar techniques which constrain the movement of a probe to features in the underlying computation grid. We have implemented a version of our object handles, called "grid-

6. The projection line is light grey in our monochrome BOOM display.

aligned handles" (Figure 8) [19], which allow constrained translation along axes in the computational grid. With this technique, it is straightforward to move objects along the curved surfaces of a CFD object such as the leading edge of the Shuttle's wing.

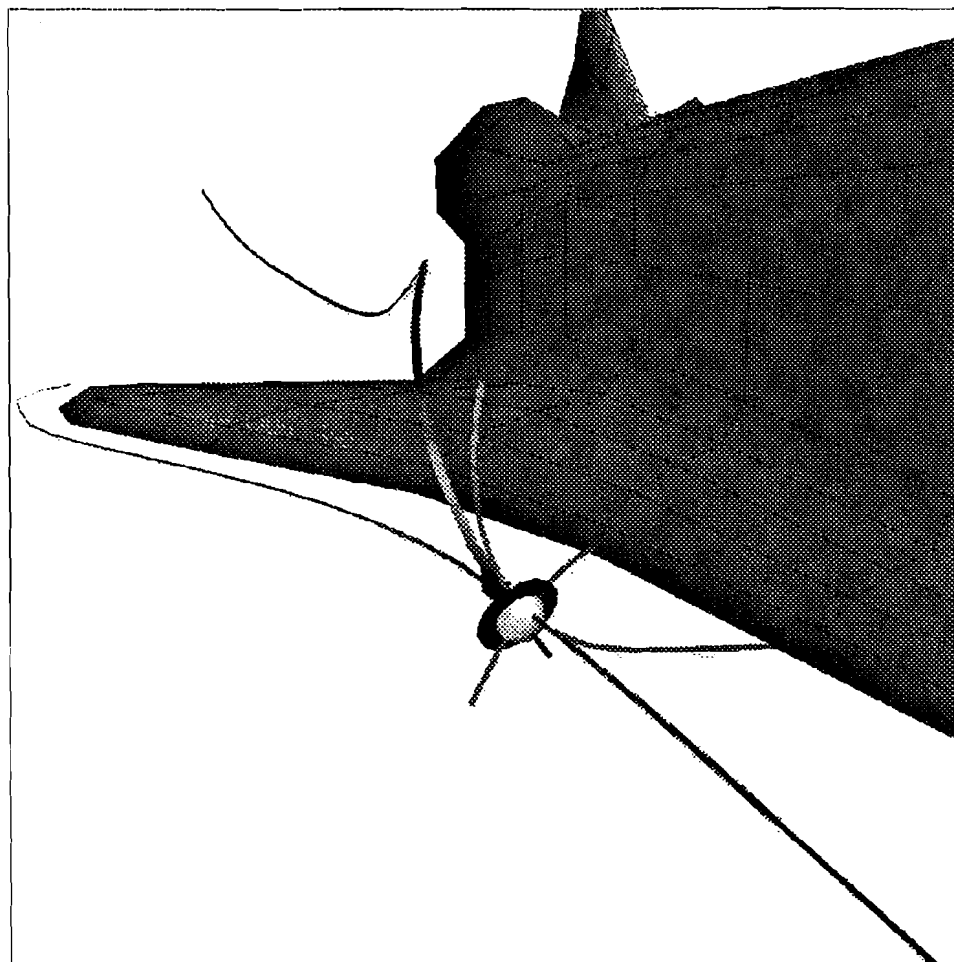


Figure 8: Grid-aligned object handles. The grid-aligned object handles widget is attached to a probe.

The grid-aligned handles work by tracing out lines in the computation grid from the point in the grid closest to a given sample point. An added benefit of these widgets is that they provide a visual representation of the local structure of the grid in the area surrounding the sample point. Users may exploit this information to gain a better understanding of the dataset and the behavior of visualization techniques.

Apart from its slightly different visual representation, this widget behaves identically to the object handles, complete with projection lines and ghosting.

6.2.3.4 Data-Space Handles

We have also developed some interaction techniques based on the actual data being visualized. For example, the vector probe widget (Figure 9) consists of a grey spherical sample point, an arrow representing the direction of the vector field at that point, and a disk representing the plane perpendicular to the vector. By dragging the

arrow component, the sample point can be moved along a streamline calculated at that point in the flow field. The disk is used to move the sample point perpendicular to the flow, allowing the user to explore nearby streamlines in the flow field.

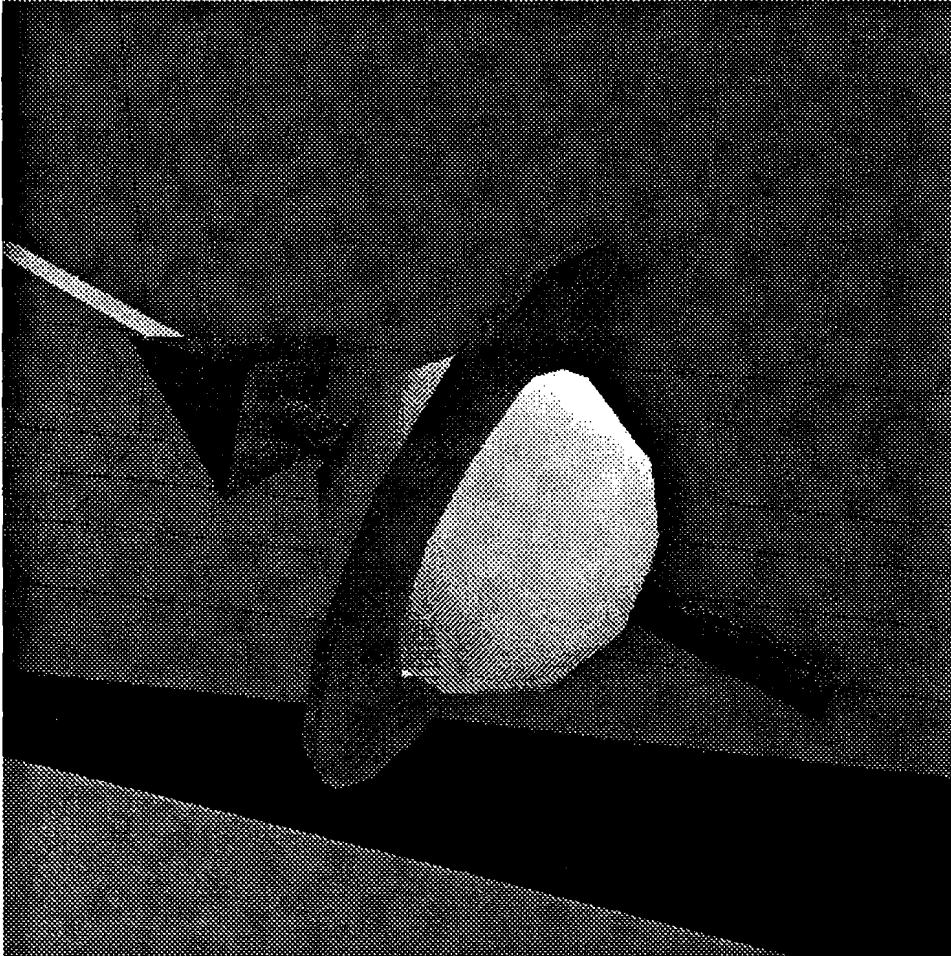


Figure 9: *Data-space handles.* The data-space handles on a point probe widget include a blue arrow and a reddish disk.

When we use this same general probe widget to visualize scalar data, the vector component displays the gradient of a scalar field. Pulling the vector changes the value of the isosurface that passes through the sample point, and translating the red disk moves the sample point along the isosurface itself. Note that moving the disk does not change the level of the isosurface, just the initial seed point from which the isosurface is computed. Since we are using an interactive isosurface algorithm [33], moving the probe in this way allows us to explore different regions of a single isosurface.

6.2.4 Direct Manipulation Visualization Techniques

We have also explored other direct manipulation techniques for modifying the position of sample points in a dataset. Normally, one describes a streamline by specifying a sample point from which a path is calculated and integrating forward through the dataset. Streamlines may also be constructed by integrating backward

from a sample point. We have implemented a streamline which serves not only as a visualization tool, but also as an interactive positioning widget. By clicking anywhere on this direct manipulation streamline, we specify a new position for the sample point, represented by a small sphere, which describes it. We can then drag this sphere using any of the manipulation techniques described above and the streamline recalculates in real time, integrating both forward and backward by the appropriate amount to maintain the original length of the streamline.

With this technique, users can manipulate a streamline very precisely in particular regions of interest that might otherwise be difficult to reach. For instance, if an interesting feature is observed at the very end of a given streamline, one can simply click on the streamline right near the feature and move it around with fine control. Without this ability, one can only modify the position of the sample point at the beginning of the streamline, where even small movements may cause very large changes in the streamline near the end.

We have also applied this same technique to translate rakes of streamlines. In addition, rakes can be rotated and scaled about the new sample points. In our system, we use a separate mouse button to “twist” the rake about the selected streamline. This has the effect of keeping the picked streamline constant but modifying the neighboring streamlines.

6.2.5 Manipulation after Selection in Virtual Reality

Each of the selection techniques described earlier that we have designed for use in VR represents a unique method for selecting objects in a scene. However, selection is only one task in a dialog that a user holds with an application. After selecting an object, a user will often modify that object in some way, either by transforming it, or changing some other parameter such as its color or transparency, or, in the case of a scientific visualization application, perhaps the visualization technique it generates. We have implemented a set of transformation techniques to accompany the selection techniques described above that allow a user to modify the position and orientation of objects once they have been selected.

The guiding principal behind these manipulation techniques is known as the principal of least surprise. This principal states very simply that what actually happens as a user interacts with a system is exactly what the user expects to happen. In the case of the selection techniques, an object has some relationship with the user’s hand (the cursor in the IVE) at the time of selection (e.g., the button press). During the subsequent manipulation, this relationship should be maintained as closely as possible. One way to look at this is from the point of view of the tracker device which, like any other 3D object in an IVE, has its own coordinate system. At the moment of selection, we can determine where in this coordinate system a selected object lies, and maintain this relative position and orientation throughout the following manipulation.

Unfortunately, exactly maintaining this relationship is not always possible. When the selected object is itself constrained to move only along a single axis or in a plane, we must project the position and orientation of the tracker onto this lower-dimensional space. The most obvious way to do this is to find the closest point in the constrained space to the reference position defined by the tracker. This approach seems to work

very well in our applications.

6.3 Navigation

To date, we have experimented with some navigation techniques both on the desktop and in VR, but have yet to fully explore the range of possible navigation interfaces. On the desktop, we mainly use a combination of techniques which are accessible via the right mouse button in conjunction with the shift and control keys on the keyboard. We use the right mouse button by itself for virtual sphere rotation about the center of the scene (the center point can change); with the shift key, we zoom in and out (by changing the size of the film plane); with the control key, we pan the camera in the plane parallel to the film plane; and by simultaneously depressing the shift and control keys and clicking on an object in the scene we focus the camera on that object (with a smooth animated sequence similar to that found in the Information Visualizer from Xerox [32]). In more recent work, we have developed an interface which affords these same controls, but without the use of the modifier keys (described in [45]). In this interface, the default behavior of the right mouse button is panning the camera. However, by pausing slightly after clicking the right mouse button, we get a combination of zoom and pan in which left-right motions map to zoom, up-down to pan⁷.

In our virtual reality application, we exploit the built-in degrees of freedom of the BOOM to provide most of the navigation. In most cases, this suffices because the majority of the objects that we interact with are at close range. In case we need to move beyond the somewhat limited range of the BOOM, however, we use the two buttons to “fly” forward and backward along the viewing axis. Generally, we “fly” at some constant velocity, but we have experimented with using an acceleration constant as well so that we can travel larger distances more quickly. We have yet to perform formal evaluations of any of these techniques, so we can not make conclusive statements about which techniques are better. We intend to explore this area of research further.

6.4 Complex Widget Designs

Each of the interaction techniques and widgets described so far was designed for a single purpose, such as modifying the position or orientation of an object, specifying which object in a scene is selected, or manipulating the viewpoint. We have also experimented with more complex, hybrid user interfaces which allow users to simultaneously visualize and modify multiple parameters of an object, including non-spatial quantities.

As with the interfaces described so far, the design of a complex widget can be very different for desktop and VR applications. In our experience, the complex widgets that we designed for desktop use consist of many small, grabbable parts that are usually fairly easy to pick with a 2D mouse. In our preliminary trials with these same widgets in our IVE, we have found that it is very difficult to grab these small parts using the available input devices. This has led us to reconsider the design of the user

7. Even though this means that we can only pan in one direction, this implementation seems to work well for the SKETCH application (it has yet to be supported by user studies).

interface for IVE's. Part of this redesign included implementing the new techniques for selecting and manipulating widgets described above. In other cases we have either redesigned existing widgets, or decided to do away with them entirely in favor of other approaches.

It is important to remember that though components of these widgets sometimes represent abstract quantities, they present these quantities in spatial terms. Thus, users may modify quantities by manipulating the position or orientation of the 3D widgets which represent them. In some cases, the positioning and orientation techniques described in the last section can be used, but more often, a widget is designed with built-in constraints so that components which represent scalar values, for instance, are constrained to translate along a single axis. If a quantity is bounded, the widget will likewise be constrained to move within appropriate physical limits.

6.4.1 Generalized Probe

The "generalized probe" that we have designed for our scientific visualization application provides a user interface to some of the parameters of the visualization techniques that we support.

Table 1:

	Number	Color	Tuft	Streamline	Isosurface
Zero-dimensional (point)	Numerical Probe	Colored Ball	Single Tuft	Single streamline	Single iso-surface
One-dimensional (rake)	Gradient rake	Colored bar	Rake of tufts	Rake of streamlines	Rake of iso-surfaces (onion)
Two-dimensional (plane)	Number plane	Colored cutting plane	Hedgehog	Multiple rakes	^a
Three-dimensional (volume)	Numbered points in volume ^b	Multiple cutting planes ^b	Multiple hedgehogs	Volume of streamlines ^c	^a

a. With partial transparency, this option may provide local volume rendering, though we have not implemented it in our system.

b. Works well only with a relatively low number of sample points.

c. There has been some research done on stream volumes, but we have not implemented this visualization technique in our system.

Most of the visualization techniques we have implemented in our system are generated by sampling single points in a dataset, calculating scalar or vector values, and displaying some visual representation of the data. The positioning techniques described earlier are designed to help scientists quickly place these sample points in a dataset, but we also need methods for controlling collections of sample points as a

group. To address this need, we developed the notion of a generalized probe which can manifest itself as a zero-, one-, two or three-dimensional widget. This generalized probe widget (Figure 10) is used to define the initial sample points for a variety of visualization techniques including streamlines, rakes, hedgehogs, cutting planes and isosurfaces. The design of the probe widget also includes components for modifying parameters of these techniques.

Table 1 shows the range of possible visualization tools that can be created using the generalized probe widget. Some of the entries correspond exactly to commonly used tools, such as the rake of streamlines, or the colored cutting plane. Other entries may require more esoteric visualization techniques to be useful.

6.4.1.1 Zero-Dimensional Probe

In its zero-dimensional form, this widget is a simple probe that samples a single point in the dataset. We use the probe with data-space handles (Section 6.2.3.4). From this point, we can choose to generate one of five visualization techniques: a number, color, tuft, streamline, or isosurface. Multiple visualization techniques can be generated simultaneously from a single sample point (though we have not yet devised a good user interface for controlling this functionality). Users may then use any of the positioning techniques described above to place this widget in the dataset. The direct-manipulation, “grab-anywhere” interaction technique described earlier (Section 6.2.4) applies only to the advected particle visualization technique.

6.4.1.2 One-Dimensional Rake

The one-dimensional widget is essentially the same as a rake tool commonly used in real wind tunnels (usually a steel pipe with holes drilled in it at intervals to generate smoke streams in an airflow). This widget produces a set of sample points at regular intervals in Cartesian space along a line; it can be translated and rotated freely and can be scaled along a single axis by translating the red ball at one end. Additionally, we supply a resolution handle, the orange disk, to change the distance between sample points. This resolution handle is free to move from one endpoint of the rake bar to the center of the bar. The distance between the handle and the endpoint determines the spacing between the sample points. One drawback to this method is that when the handle is moved close to the endpoint of the rake, very small movements up and down the bar can cause large changes in the number of streamlines. This non-linear behavior can be corrected by simply mapping the position of the resolution handle to a linear scale, but then one would lose the geometric correlation between the placement of the handle and the spacing between the sample points. In an IVE, this slider approach has a distinct drawback: Since the 6D tracker input device is not as stable as the 2D mouse on the desktop, noise from the tracker or user’s hand can make it difficult to precisely set a specific number of sample points. Alternative interfaces, such as the dial menus described later (Section 6.4.2), may be better approaches for tasks like this.

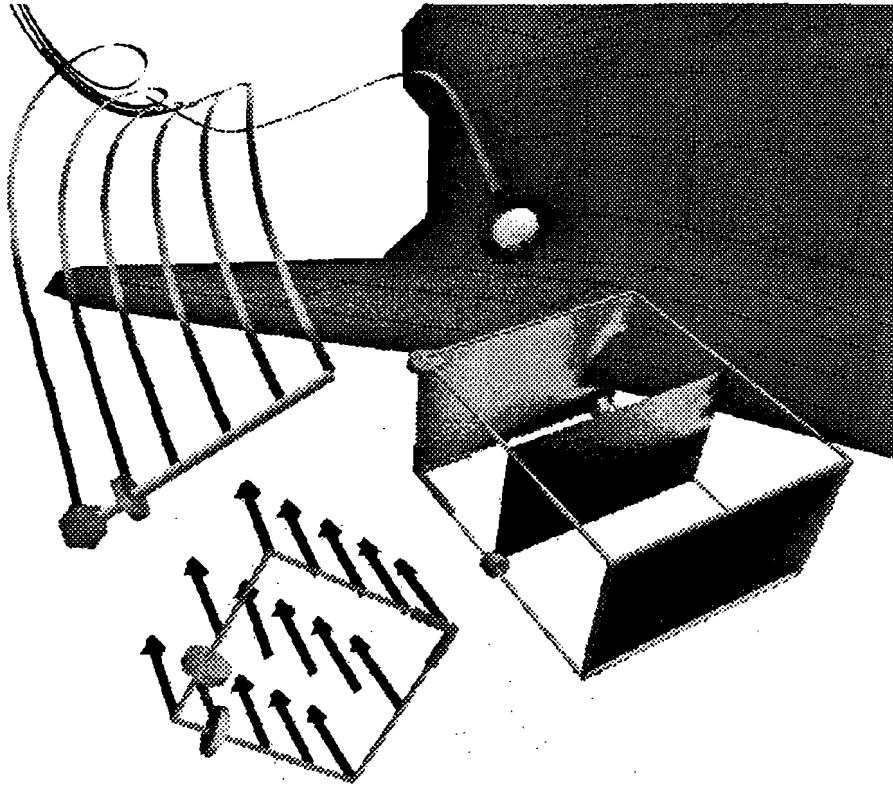


Figure 10: Probes. Counterclockwise from left, the point probe with a streamline, 1D probe with streamlines, 2D probe with tufts and 3D probe with color. Visualization techniques are colored by velocity of the vector field from blue (slow), through green, to red (fast).

Again, any of the visualization techniques listed earlier can be generated from this set of sample points; advected particles produce the familiar rake of streamlines, the color technique produces a colored bar, and the isosurface produces an “onion”–multiple isosurfaces at different levels of the dataset.

6.4.1.3 Two-Dimensional Plane

The two-dimensional widget samples a set of points arranged in a regular planar grid (similar to the one-dimensional widget). This widget can be translated and rotated freely, and can be scaled independently in two dimensions, much like a 2D window in a desktop-style graphical user interface (GUI). Also, the resolution of this widget can be changed independently in each dimension. Note that we maintain continuity between the different probes by using the same visual language for these handles. As with the one-dimensional widget, the resolution handles exhibit non-linear behavior.

Using the color technique with this widget produces a cutting plane; similarly, the

tufts produce a hedgehog. This widget can be confusing when the advected particle technique is chosen, especially if the sample points are very close together in both dimensions: the visual effect is something like a volume of streamlines, and is not very intelligible (this may be more useful with transparency or "stream volumes"). However, if we reduce the number of sample points in one dimension, say down to three, we effectively produce a widget which controls a set of three rakes as group. In this configuration (Figure 11), we have a useful tool once again.

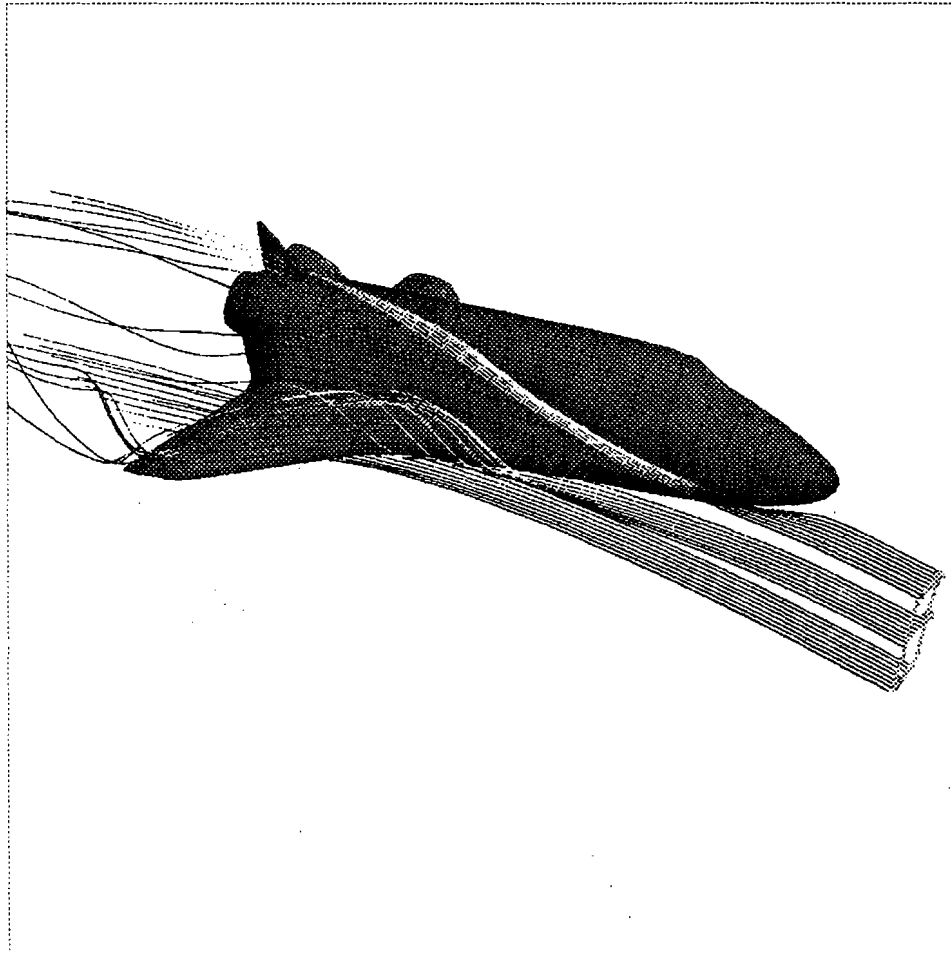


Figure 11: 2D probe with streamlines. Properly configured, the two-dimensional probe acts like a collection of one-dimensional probes which can be controlled in unison.

6.4.1.4 Three-Dimensional Volume

Finally, the three-dimensional widget generates a volume of sample points. It can be scaled in three dimensions and has resolution sliders for each dimension as well. Like the two-dimensional probe, this widget can produce very confusing visualizations if not parameterized correctly. However, by choosing the color technique and adjusting the resolution sliders so that there are lots of sample points in two dimensions and very few in the third, we can produce a set of cutting planes that can be moved around as a unit. One can imagine an alternate colored representation which utilizes volume textures, but we have not implemented this. This particular technique may be useful

for visualizing MRI data as well as fluid flows because it also has rich volumetric data (just as fluid flows have regions of higher or lower pressure or temperatures, for example, MRI data contains areas of differing density).

6.4.1.5 Interface to the Generalized Probe

In our desktop application, we provide a set of 2D buttons outside of the 3D view for changing the probes from one dimension to another. When a probe changes dimension, it fades from one representation to the next, thus maintaining visual continuity. Another set of 2D buttons changes the visualization technique generated at each sample point. When multiple widgets are being displayed simultaneously, the 2D buttons only affect the last widget used. In this way, we can have many probes on the screen, each in a different configuration and producing a different visualization technique.

6.4.2 Menus and Buttons

Menus and buttons are standard components of 2D user interfaces for desktop applications, and are often used as an interface to abstract parameters or commands within an application. However, they introduce many problems when added to a user interface in an immersive environment. 2D menus do not necessarily face the viewer, making visibility difficult. Also, selection of menu items is generally made with a six-dimensional device even though the items are arranged in a 2D array. The benefits of menus and buttons are many, including the access they provide to changing the state of objects or the environment, or to making abstract commands that do not have explicitly visual semantics. Menus and buttons are ubiquitous in desktop GUI's because they are relatively compact and easy to learn and use. However, care must be taken when transferring standard 2D interface components like these into IVE applications.

6.4.2.1 Menus and Buttons in IVE's

A number of different types of menus have been implemented in IVE systems, including a hierarchical menu in the "virtual wind-tunnel" [6], a panel with popup menus in the 3DM application [7], "hands-free" menus [13], the menus on the virtual tricorder developed here at Brown [43], as well as spherical (daisy) and dial (ring) menus in HoloSketch [14] and JDCAD [29]. They all have problems, however. The hierarchical menus in the VWT were modeled after 2D menus. When drawn in the 3D scene, they appear oriented with the film plane and are quite large, usually taking up most of the view, and thus obscure most of the virtual environment and the objects in it (thus violating one of our design rules)⁸. One chooses options from the menu by pointing at them with a laser-pointer style selection tool. Hierarchy navigation is also modeled after standard 2D menus and requires very good control over the pointer to do it effectively.

8. It is possible to render these menus transparently (though this is not done in the VWT). However, this approach is also problematic because the menu items can easily be confused with geometry in the background and can thus become difficult to read.

The 3DM application developed at UNC is a rectangular array of 3D icons and popup menus that can be “attached” to the user in the virtual environment. When attached, it follows the user wherever she goes, and remains located near the waist line. It can be moved around freely, and even detached from the user and left behind. The 3D icons represent tools, commands and toggles which are used to construct shapes or change the state of the environment. Apparently, it is quite easy to lose this panel in the IVE if the user detaches it.

Darken’s menus use speech-recognition techniques for selecting menu items in an IVE, and thus frees the user’s hands for other tasks. However, as with many other menu systems, the text of the “hands-free” menus must be visible in the 3D environment at all times, and thus must occupy some part of the valuable screen space.

6.4.2.2 Spherical Menus

Spherical menus are inherently 3D structures and therefore show promise for IVE applications. They operate as follows: the items in the menu are distributed on the surface of a sphere (which itself is not necessarily visible) which is placed at the location of the user-controlled cursor in the virtual environment. The menu items become visible usually in response to a button event. By rotating the tracker in place (a 6D tracker must be used for this kind of menu), users point at the desired item and release the button to activate it. However, it may be impossible to comfortably get at some of the items because it is very hard to rotate the tracker a full 360° in a single motion. In JDCAD, a clutch mechanism was implemented to alleviate this problem. By pressing a button on the tracker, the user can temporarily suspend input from the tracker, rotate her hand into a more comfortable position, release the button, and continue pointing. One drawback of the technique is that after clutching, there is no longer correlation between the orientations of the tracker and the cursor in the IVE. Also, the interaction dialog becomes complicated by multiple button events.

6.4.2.3 Dial Menus

We have implemented a kind of dial menu (Figure 12) in our application that may be more useful than these other types of menus. Our dial menus are generally associated with specific objects in the 3D scene, such as 3D widgets like the rake, which have many parameters. As with conventional 2D menus, only the root of a dial menu is initially visible. When the user clicks on the root, the associated menu items are drawn. Items are placed at intervals on a circle centered at the menu’s root, and can either be text or graphic icons. Both visually and behaviorally, these menus are very similar to the “marking menus” developed by Kurtenbach and Buxton [27] and implemented in the Alias/Wavefront Studio Version 7 modeling and animation software⁹. The main differences between the dial and marking menus are analogous to the differences between 2D and 3D widgets. Whereas marking menus are elements of a 2D interface, and behave similarly to pop-up menus, the dial menus coexist in a 3D environment with other geometry, and can be semantically attached to specific objects in the scene.

9. “Marking menus” are an extension of the original “directional” pie menus [9], and, as Buxton asserts, are superior to linear pull-down or pop-up menus because they exploit users’ spatial memory.

Dial menus can be used either on the desktop with a 2D mouse or in an IVE with a 3D tracker.

Unlike the marking menus, the user need not first select the object she wishes to modify before pulling up the menu – instead, these two operations are unified because some minimal geometry which represents the root of the dial menu is always visible as a component of the object that the menu is associated with.

To activate a dial menu, the user presses a button on the mouse or tracker, then drags to the appropriate item. As with the marking menus, a line trace is left behind the cursor as visual feedback of the gesture being performed. Expert users who have retained a mental model of the physical layout of the menu can quickly make menu selections by performing the appropriate gestural movement without first waiting for the menu items to appear. The dial menus are hierarchical, and users may navigate both down and up the hierarchy. When the user releases the mouse or tracker button, the menu items and trace are undrawn and the chosen action (if any) is performed.

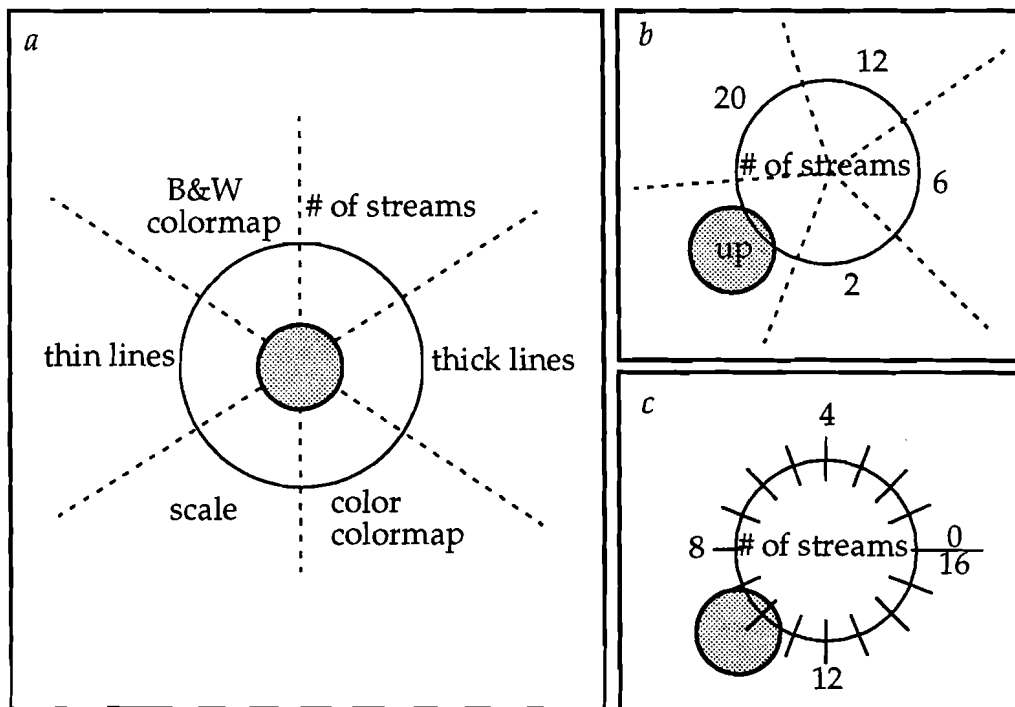


Figure 12: A sample dial menu. The root level menu on the left (a) has six items. Notice how related functions are placed on opposite sides of the dial (e.g., thick and thin lines).

The circle at the center represents the 3D geometry which is always visible. This geometry could be text or an icon. The dotted lines extending outward from the center indicate the six regions of this menu. By moving the cursor into the region labeled, "# of streams", one of the two submenus on the right (b and c) is displayed with just the root of the parent menu (the rest of the items on the root menu are undrawn). The menu in the upper right (b) is a hierarchical submenu with additional items. The "up" item on this submenu allows navigation back up the hierarchy to the root (a). The menu in the lower right (c) is a virtual dial. In an IVE, by rotating the tracker about a single axis, a continuous range of values can be selected. Navigation back up the hierarchy is accomplished by moving the cursor back toward the root.

collection. We lay out a grid of tape on the floor of the lab, at 1-foot intervals, and place a plastic tripod with plumb line above each grid crossing. The plumb line is marked at 1-foot intervals. This lets us place the sensor, carefully held level and grid-aligned, at each point of our $6 \times 7 \times 8$ foot grid and record the position and orientation reported.

Our own approach to interpolating data values is to first triangulate the domain of the function F , and then to estimate F by piecewise linear interpolation across this triangulation.

We divide each cubical “cell” of the grid into five tetrahedra. For each tetrahedron, there is exactly one linear function of position that agrees with the observed values at the four corners, so the interpolation scheme is unambiguous. For this piecewise-linear interpolation scheme to succeed, the division into tetrahedra has to be a *triangulation* of the entire cell array.

In practice, our approach has yielded excellent results. Correlation between actual tracker position and the observed location of the cursor in the IVE is significantly better than without the calibration.