Worlds within Worlds
Metaphors for Exploring *n*-Dimensional Virtual Worlds

Steven Feiner
Clifford Beshers

Department of Computer Science
Columbia University
New York, New York 10027

feiner@cs.columbia.edu
beshers@cs.columbia.edu

Abstract

n-Vision is a testbed for exploring *n*-dimensional worlds containing functions of an arbitrary number of variables. Although our interaction devices and display hardware are inherently 3D, we demonstrate how they can be used to support interaction with these higher-dimensional objects. We introduce a new interaction metaphor developed for the system, which we call "worlds within worlds": nested heterogeneous coordinate systems that allow the user to view and manipulate functions. Objects in our world may be explored with a set of tools. We describe an example *n*-Vision application in "financial visualization," where the functions are models of financial instruments.

n-Vision's software architecture supports a hierarchy of arbitrarily transformed, nested boxes that defines an interactive space within which information is displayed and input obtained. Our design, modeled in part after the hierarchical 2D windows of the X Window System, is intended to provide an environment that is well suited to the use of true 3D input and stereo display devices. Boxes are associated with event handlers that support 3D motion, enter, and leave events, and provide recognition of finger gestures.

CR Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*User interfaces*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Hierarchy and geometric transformations*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction techniques*

General Terms: Design, Human Factors

Additional Keywords and Phrases: virtual worlds, financial visualization

1 Introduction

One common problem in graphical user interface design has been the need to manipulate and view 3D environments using inherently 2D interaction devices and displays. Although graphics researchers have long been developing true 3D interaction and display devices [SUTH65; VICK70; KILP76], it is only over the past decade that high-performance 3D graphics workstations have been coupled with commercially available 3D devices such as polarized liquid crystal shutters for stereo viewing [TEKT87; STER89], head-mounted displays [VPL89], and the DataGlove [ZIMM87]. While the use of 3D devices for 3D data seems a natural match, there are many applications in science, mathematics, statistics, and business, in which it is important to explore and manipulate higher-dimensional data. In these applications, data can be defined by points in Euclidean *n*-space. A point's position is then specified with *n* coordinates, each of which determines its position relative to one of *n* mutually perpendicular axes. One goal of our research has been the development of interaction techniques and metaphors for the 4D and higher-dimensional worlds that this data represents.

2 The *n*-Vision Testbed

n-Vision is a testbed that we are developing for exploring *n*-dimensional virtual worlds. Input is provided largely through the use of an inherently 3D interaction device, the VPL DataGlove [ZIMM87]. The DataGlove uses a magnetic sensor to sense the 3D position and orientation of the user's hand. Fiber optic cables running along each finger monitor an additional ten degrees of freedom, determined by the positions of two joints on each of the five fingers. Output is displayed on a monitor viewed with liquid crystal stereo glasses [STER89] that make it possible for the user to have a strong sense of the three-dimensionality of the virtual space in which they interact. Interaction in *n*-Vision may also be controlled through mouse-operated control panels, and dial and button boxes. *n*-Vision is implemented using a hierarchy of nested boxes whose architecture is discussed in more detail in Section 4.

We have developed a set of interaction and display metaphors in the course of using *n*-Vision to investigate an example of what we call "financial visualization," in analogy to scientific visualization. Our users are interested in exploring the value of a portfolio of *options* to buy or sell foreign currency on a specified

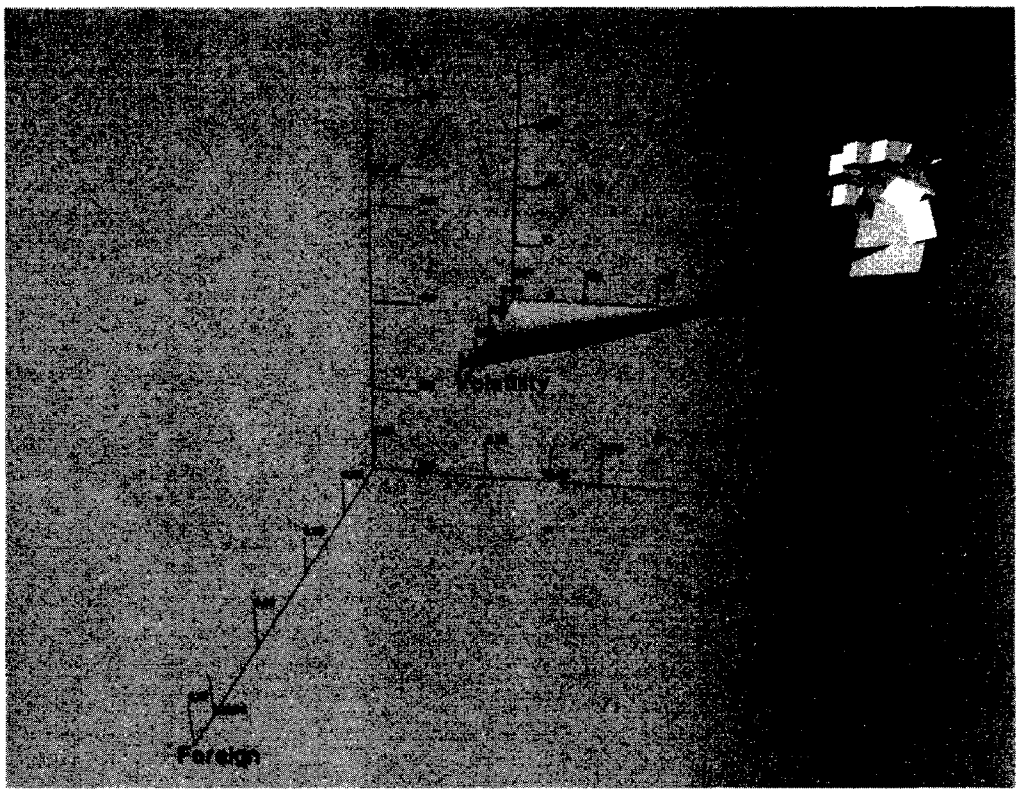

Figure 1 Value of a single call option.

date at a specified price. An option to buy is referred to as a *call*, while an option to sell is known as a *put*. Options that may only be exercised at a specified date are called *European options*. Each European option has a value that may be modeled as a function of six variables: the price at which the currency can be bought or sold at maturity ("strike price"), the price at which the currency is selling now ("spot price"), the time remaining to the date at which the option may be exercised, the interest rates for the domestic and foreign currencies, and the volatility of the market [HULL89]. These functions of six variables define surfaces in 7-space. Investors typically buy and sell combinations of different options that have been selected as part of an investment strategy that trades off risk against profit.

3 *n*-Dimensional Interaction Metaphors

Perhaps the simplest way to control the n values that define a point in n-space is to assign each value to its own valuator device. For example, a knob box may control as many independent variables as it has knobs. Similarly, the values of each of these n variables may be displayed by assigning each to its own numeric string or one-dimensional variable-length bar. In 2D and 3D applications, we commonly take advantage of our experience manipulating and viewing real objects, and instead use interaction and display devices that treat these objects as if they were part of a virtual 3D world. Our spatial positioning experience, however, is inherently limited to 3D. Therefore, from the standpoint of the user interface, the straightforward generalization of specifying a point in 2D by pointing in a plane, to specifying a point in 3D by pointing in space, does not extend to higher dimensions. What can we do instead?

In some cases, the multivariate data being presented has a familiar interpretation in our 3D world. For example, Ouh-young, Beard, and Brooks [OUHY89] allow users to explore a 6D space to find the energy minimum of positioning and orienting a rigid object in 3-space operated on by forces and torques. This task can be performed using visual and/or force-feedback presentations of the user's position in 6-space (3 dimensions each of position and orientation). The user sees a representation of the forces and torques as vectors of varying length or actually feels them by using force-feedback manipulators. Another approach is presented by Bly [BLY82], who demonstrated the ability of users to distinguish between multivariate data presented sonically by varying seven characteristics of a note: pitch, volume, duration, attack envelope, waveshape, and the addition of fifth and ninth harmonics. When presenting abstract multivariate data graphically, however, we must resort to visual displays that map abstract, nonvisual properties to visual properties, such as position, color, and texture, that can be represented in the display. Thus, real-world metaphors are still relevant, allowing us to map an abstract world into a concrete one. One possibility is to generalize 3D modeling transformations and viewing projections to higher dimensions [NOLL67]. Although systems based on these concepts are useful research tools [BANC78; FEIN82; BESH88], an intuitive understanding of these projections is often hard to acquire.

3.1 Worlds within Worlds

One common approach to reducing the complexity of a multivariate function is to hold one or more of its independent variables constant. Each constant corresponds to taking an infinitely thin slice of the world perpendicular to the constant variable's axis, reducing the world's dimension. For example, if we reduce the dimension to 3D, the resulting slice is a 3D height field. It represents a function of two variables that can be manipulated and displayed using conventional 3D graphics

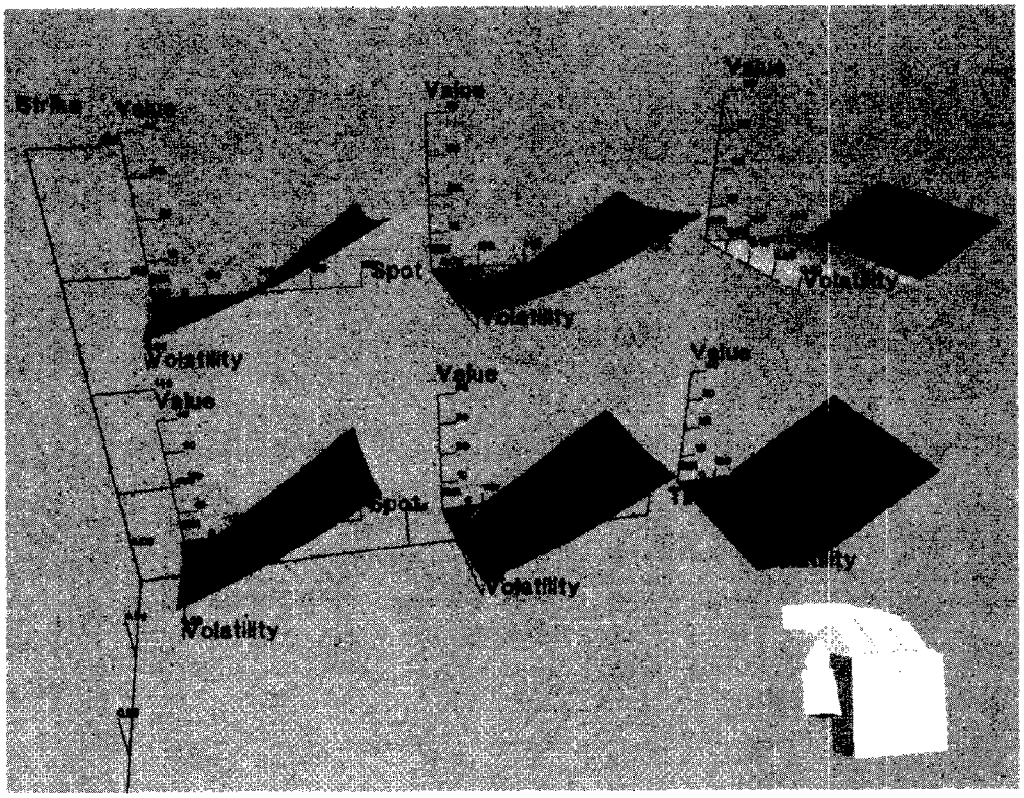

Figure 2 An array of call options.

hardware.

Although this simple approach effectively slices away the higher dimensions, it is possible to add them back in a controlled fashion. For example, if we *embed* a 3D world in another 3D world, we can represent three additional higher dimensions. The position of the embedded world's origin relative to the containing world's coordinate system specifies the values of three of the inner world's variables that were held constant in the process of slicing the world down to size. This process can then be repeated by further recursive nesting of heterogeneous worlds to represent the remaining dimensions.

Figure 1 is an example of a call option whose value is represented as a height field in a 3D inner world, plotted as a function of spot price and volatility. The outer world has axes of time to maturity, strike price, and foreign interest rate. The domestic interest rate has been held constant and is not assigned to an axis. Thus, the position of the inner world determines the time to maturity, strike price, and foreign interest rate used in evaluating the function.

In *n*-Vision, each world is contained within a rectangular box oriented with and containing its coordinate system. The lowest-level world in the hierarchy that contains the user's gloved hand is selected for manipulation, and its axes highlighted. Highlighted axes are drawn in black in the accompanying figures. (We also make it possible for a world to be selected and deselected for manipulation even when the hand is not inside it.) The action performed on a selected world depends on the "posture" that the user's hand assumes. One posture allows the user to translate the world. Translating a world that is embedded in an outer 3D world will change up to three otherwise constant variables, causing the object(s) within the world to change accordingly. Additional postures allow the user to rotate or scale a selected world about its origin, which makes it possible for the world to be viewed from another angle or at another size, without modifying its variables.

The kinesthetic feedback provided by an interactive system provides users with a feel for the functions being explored that they would not get from static displays.

The position of the origin of the selected world's coordinate system is shown with selectable tick marks on the axes of its containing world's coordinate system. A user can constrain the motion of an inner world along a single outer-world axis by selecting and translating the appropriate tick mark.

We can deposit multiple copies of the same world or copies of different worlds within a containing world, to allow the copies to be compared visually. Each copy, based on its position, has a different constant set of values of the containing world's variables. For example, Figure 2 shows an array of six inner worlds, each of which represents a call. A comparison of the calls, which vary only in strike price and time to maturity, indicates how market volatility has successively less effect as the time to maturity decreases, and how an increase in the price at which the currency can be bought ("strike price") makes for a lower profit, all things being equal.

Figure 3 is a stereo pair (with the left eye's image at the left) that shows a collection of worlds within a common containing world. The worlds include a put (in the foreground), two calls, and a "butterfly spread" (the surface at the left). The butterfly spread is a trading strategy in which call options for the same currency and maturity date are bought and sold. Two call options are sold with a strike price that lies within a range of strike prices established by buying one call with a lower strike price and one with a higher strike price. The strike price axis here controls only

Figure 3 Stereo pair of multiple worlds. (Left eye's image is at left.)

the price of the call options being sold. This strategy effectively limits the amount of money that can be lost (while also limiting the amount of money that can be made).

Note that the order in which variables are assigned to the nested coordinate system axes has a profound effect on the surface displayed. The two variables assigned to an innermost 3D world's axes determine the shapes produced, whereas the order in which variables are assigned to the ancestor worlds' axes determines the ease with which variables may be manipulated. For example, if multiple worlds are nested directly inside another world, then translating the common containing world modifies all of the nested worlds in the same way. Thus, the easiest way to restrict a set of worlds to share the same variables is to nest them inside a world whose ancestry defines the desired variables. In order to avoid the effects of translating one or more worlds, they can be nested directly inside of a world whose coordinate system has no variables assigned to its axes. These first-level inner worlds can then be positioned without changing their contents, for example to place them next to other worlds for comparison. Rather than being limited to 3D worlds, *n*-Vision provides support for worlds of from one to four dimensions, relying, in part, on techniques we have developed for transforming and displaying 4D objects in real-time using 3D graphics hardware [BESH88].

We have found that the combination of the DataGlove and stereo display is particularly powerful for picking and manipulating one of a number of worlds. The DataGlove's direct positional control allows the user to reach out and "grab" a world, rather than "steer" toward it. The stereo display provides visual feedback that makes 3D positioning significantly easier, while also resolving ambiguities in the projections of individual surfaces. For example, the inner world in Figure 1 is quite close to the nearest face of its containing world's box, as indicated by its tick marks. Disregarding the tick marks or any knowledge of the function's expected appearance, a user may be easily convinced that the Spot–Value plane is coplanar with the Time–Strike plane, based on the single projection shown here. When viewed in stereo, however, the discrepancy in their distances becomes obvious.

3.2 Tools

Users may explore worlds by using *tools* that are implemented by a kind of box called a *toolbox*. A toolbox is usually associated with a set of glove postures that specify when the user starts and stops using its tool. Each toolbox has access to all of the interaction device settings. We currently support a small collection of tools that include a *dipstick*, *waterline*, and *magnifying box*.

The *dipstick* is a small probe that the user may pick up and move within a surface. The 3D dipstick is sensitive to motion parallel to the plane above which the function is displayed, and displays the value of the function at the point it intersects. Figure 4 shows a dipstick being used to sample the value of a butterfly spread.

The *waterline* is a plane that is perpendicular to one of the axes in a world. It may be raised or lowered to slice a surface. Since it is processed by the same visible-surface algorithm as is the rest of the world, it can be used to locate local minima and maxima visually. Figure 5 shows a waterline being used to explore the value of a put option.

The *magnifying box* is a 3D version of the familiar 2D "detail" window that provides a higher-resolution display of part of another window [DONE78]. When a magnifying box is associated with another box, the actions performed in any one of the boxes are reflected in the other.

3.3 Metamorphosis

Using a stylized hand as a cursor provides a familiar mechanism for displaying the sixteen degrees of freedom that the DataGlove supports. Although graphics researchers using the DataGlove seem to be uniform in interpreting the glove data quite literally as a hand [STUR89; WEIM89], a graphical hand is not always the best way to interact with data. For example, feedback on all sixteen degrees of freedom is usually unnecessary; When the user has selected an object to rotate or translate it, the precise position of the fingers is of little consequence. Furthermore, there are some situations in which control of multiple degrees of freedom could be exploited better than by positioning and orienting a virtual hand and its wiggling fingers. In contrast, consider the commonplace use of multiple cursor definitions in 2D mouse-driven systems. The potential is even greater when a more powerful control mechanism with a larger number of degrees of freedom can be harnessed.

We exploit this capability in a limited way in *n*-Vision by allowing *metamorphosis*, turning the hand into one or more tools, instead of just attaching the tools to the hand. For example, turning the hand directly into a dipstick eliminates the visual interference of having the full hand rendered on the display, possibly obscuring part of a surface, when we need only to probe the surface at a specific point. Mapping the hand to other more exotic tools could eventually allow hand and finger motion to control nonanthropomorphic tool parts that move (or otherwise

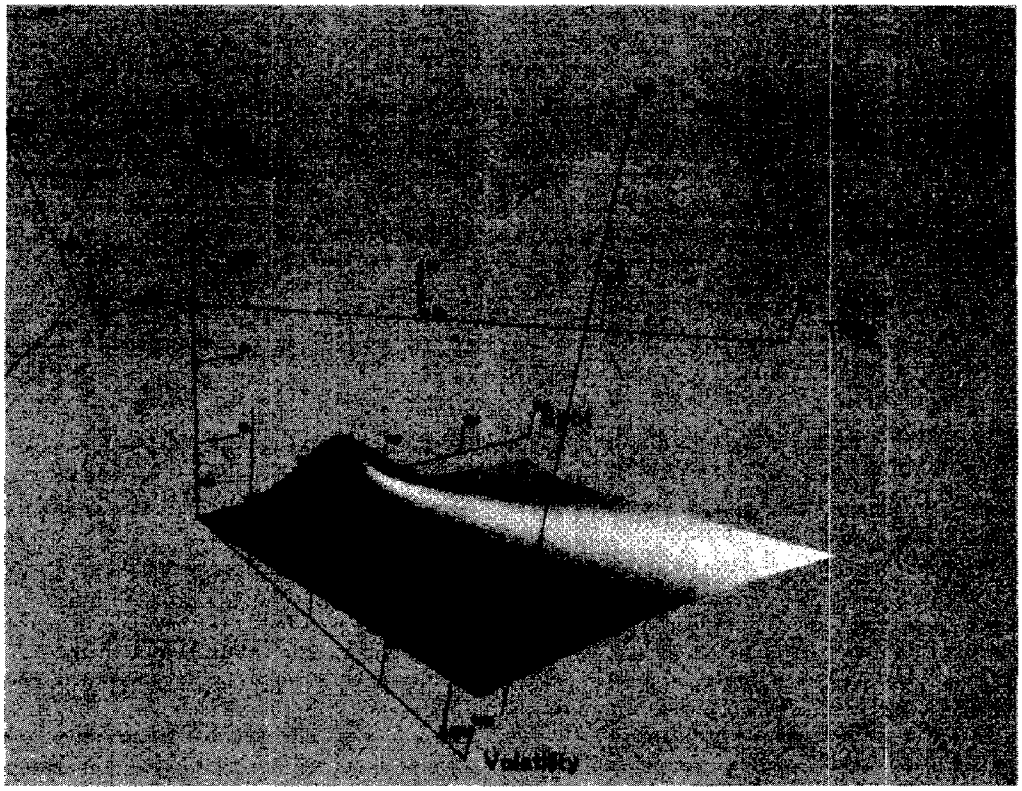

Figure 4 Sampling a butterfly spread with a dipstick.

change) differently than the user's hand and fingers.

4 Implementation and Architecture

n-Vision is implemented in C++ and runs under HPUX on an Hewlett-Packard 9000 375 TurboSRX workstation, which has hardware support for scan-conversion, shading, and *z*-buffer visible-surface determination.

The underlying structure that we have chosen for *n*-Vision is similar in spirit to that used in the X Window System [SCHL88]. Our world is a 3D hierarchy of nested boxes, each of which may be arbitrarily translated, scaled, and (unlike X) rotated relative to its parent. The hierarchy is an oriented tree: siblings are assigned relative priorities that determine the order in which they are rendered and picked in case of overlap. Rubin and Whitted [RUBI80] used a similar hierarchical structure to make possible fast 3D rendering. We have adopted this approach in an attempt to provide an understandable way to partition the space in which our users interact.

4.1 The Box Hierarchy

Boxes serve as containers for presenting graphical output and capturing graphical input. A box's coordinate system represents a transformation relative to that of its parent. Each box is an instance of a class that may be associated with event handlers that allow it to register for and react to a variety of different events. A box can map and unmap itself from the display. Mapped boxes are displayed and receive events for which they have registered; unmapped boxes are not displayed and do not receive events. By providing mapping and unmapping, we allow the creation of a controller box that owns a child box, and maps and unmaps it as it sees fit. This makes it possible to implement a low-resolution or schematic stand-in for a more complex object. For example, a box that is being moved may unmap its children during the motion and remap them only after the motion has ceased.

Each box has a list of associated event handlers, which are objects that request specific event types and are notified whenever a requested event occurs and is routed to that box. Event types are carefully designed to allow the application to track changes in the state of the system, including input arriving from a device, modifications to the box configuration, cursor movement across boxes, and box exposure. We use event handlers to implement data dependencies among boxes and to support graphics that are rendered within the box.

The body of the system's main loop performs the following series of tasks:

Sample the 3D pointer
Update the cursor
Find the regular box containing the cursor
Sample all other input devices
Dispatch all events
Redraw display

n-Vision has a 3D pointer that may be connected to any interaction devices, typically the DataGlove's position sensors. Whenever the 3D pointer moves, a special box, called the *cursor box*, is notified of the change. The cursor box differs from the other boxes in that it has a distinguished "hot point" that determines where input events are routed, and a function that sets the cursor box geometry according to the state of the pointer. The cursor box can also be set directly by the application.

Events are routed to the deepest box in the hierarchy on the

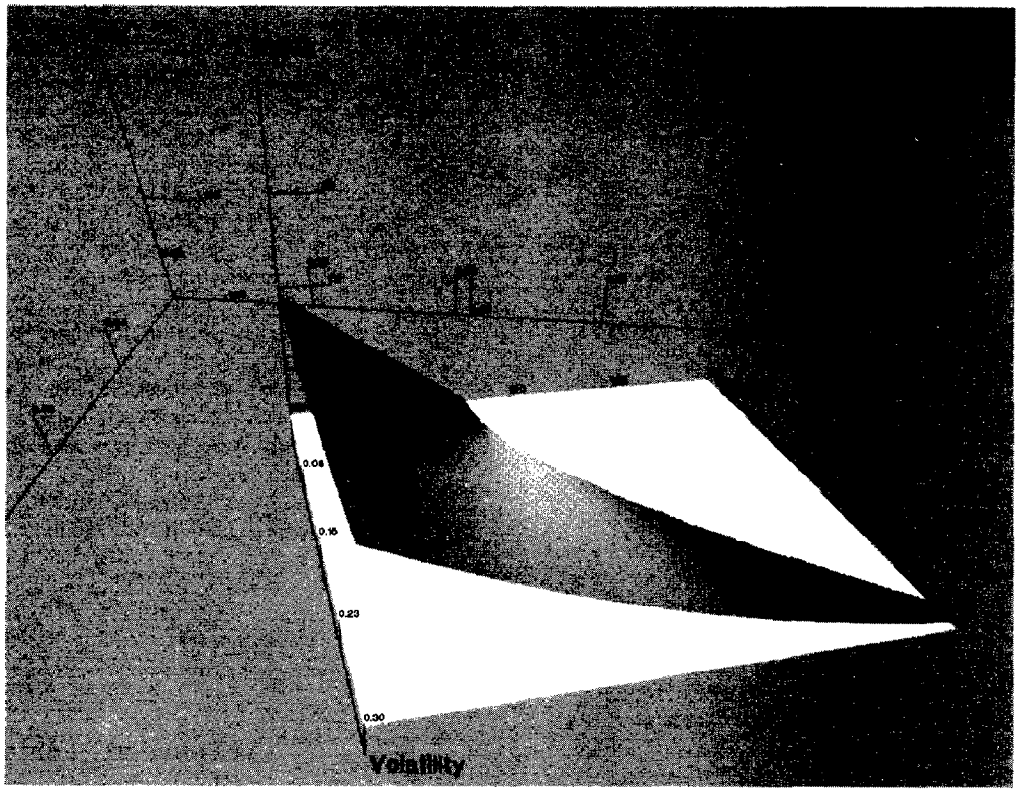

Figure 5 Examining a put option with a waterline.

highest priority branch that contains the cursor's hot point, excluding unmapped boxes and the cursor box. This is the *current box*. The box system begins at the current box and searches upward in the hierarchy until it finds a box with at least one event handler for the desired event type. It then dispatches the event to all interested event handlers attached to this box (except in the case of "grabs," which are described below.)

We also support X Toolkit events to make available standard input devices and techniques. These events are routed in the same way as other events, but only if no toolkit widget is interested in them. This gives priority to the toolkit for devices supported by X, and ensures that the box system does not interfere with the toolkit.

Events are generated by the server when a box changes state. *Mapping* events are generated whenever a box is mapped or unmapped, whereas *geometry* events are generated whenever a box is scaled, translated, or rotated.

The application can also track the path from the hierarchy's root to the current box by enabling *enter* and *leave* events. These are similar to the enter and leave window events of X, and are useful for providing graphical feedback. An event is generated for each box whose boundary is crossed by the pointer. The events indicate whether the pointer entered or left the box, and whether or not the box is or was the current box or one of its ancestors. All boxes between the old and new current box are notified of the change in status. An enter or leave event is routed first to the box for which it was generated, and then up the hierarchy to the lowest ancestor that has an event handler for the event.

The DataGlove can generate *posture* events that are triggered as it moves in and out of a set of user-definable postures that are maintained in a posture table. Each posture consists of a set of ranges for the finger bend sensors [STUR89]. One can enter or leave a posture, much like pressing or releasing a button. Unlike conventional button events, however, we do not currently support the ability to be in more than one posture at a time. Entry into or out of a posture event is used by box event handlers to allow interaction device settings (such as the DataGlove's position and orientation) to control the transformation of one or more boxes.

An event handler may "grab" a particular event type, such as pointer motion events. All grabbed events are routed exclusively to that event handler, rather than through the normal chain of event routing. This makes possible operations that control input events as long as they are active. We usually write posture-event handlers so that the box to which the posture event is routed grabs the DataGlove motion events until the posture is exited. This allows the user to assume rotation and translation postures that retain control of the box, even though the cursor may exit the box. This can occur if the box's movement is constrained (e.g., to rotate only about its origin and to maintain its origin within its parent).

The final step of the main loop is to redraw the display. This is accomplished with a depth-first traversal of the box hierarchy. *Expose* events get propagated down the box hierarchy for each new frame, with parents rendered before their children, and siblings rendered in order of increasing (higher) priority. Even though we rely on a hardware-supported z-buffer visible-surface algorithm, we must be able to specify the order in which objects will be rendered, because two objects whose projections overlap on the display may have identical z values at some shared pixel. Therefore, enforcing priority when rendering allows conflicts to be resolved consistently.

To implement the worlds-within-worlds paradigm, each world in the hierarchy is represented by a box that is assigned a geometry-event handler and an expose-event handler. The geometry-event handler constrains the box's position so that its origin always lies within the parent's box, and forwards information about the geometry changes down the hierarchy. The expose-event handler draws the box's axes. A "leaf" box that contains a surface is assigned an additional event handler that draws the surface in response to expose events. This event handler recomputes a sampled representation of the function whenever any of the function's variables have changed by more than a specified amount. The dynamic tick marks are contained in boxes that are siblings of the box whose position they control and report. Each tick-mark box has a geometry-event handler that supports constrained motion along the axis on which it is located and that propagates an event to the box it controls. Likewise, a regular box's geometry-event handler propagates events to the tick marks that represent its position.

4.2 Using Boxes to Partition the Environment

Although arbitrarily complex nested hierarchies of boxes can be constructed, each with its own interaction handler, it is clear that a complex environment could be quite confusing to the user. Even in 2D window systems, most system builders provide similar support for interaction inside of most windows. There are some places, however, in which providing spatial modes is important. For example, in using the buttons, dials, keyboard, or mouse while wearing the DataGlove the user may inadvertently enter or exit a posture. We could (and typically do) include in the posture table postures that turn posture recognition on and off. Rather than relying on the user remembering to issue these commands, we instead arrange a set of boxes that surround each of the real interaction devices, including the space needed for the hand using them. Each of these boxes has a handler for enter and leave events that turns posture recognition off and on, respectively.

While the stereo display system can provide a compelling sense of three-dimensionality, the effect is spoiled if the user reaches their hand out toward to the screen to interact with the environment. Unlike the rendered objects being displayed, the user's hand doesn't participate in the visible-surface determination algorithm; consequently, it can visually obscure parts of objects that are intended to be closer than it—an extremely disconcerting effect. Therefore, we box the view volume in front of the display, disabling both posture recognition and tracking to discourage the user from entering the volume. Unlike conventional interaction devices, the DataGlove continuously monitors a part of our bodies that we routinely use in communication with other people. This can result in the distracting effect of the user's glove cursor bouncing around on the display, while the user converses with others. Thus, disabling the glove in the view volume also ensures that gestures made when pointing at the screen are seen and interpreted only by other users, not by the system.

5 Conclusions and Future Work

We have described a new approach to visualizing and manipulating abstract n-dimensional worlds, and have discussed the software architecture underlying its implementation. The result is an environment, designed for true 3D input and output devices, that we feel provides familiar behavior when exploring quantitatively and qualitatively what are often unfamiliar objects. In contrast to generalized n-dimensional transformations and projections, the worlds-within-worlds metaphor encourages the user to think in terms of nested worlds, each of which is conceptualized and experienced as a physically realizable space.

There are a number of ways in which we would like to improve our n-Vision testbed. For example, there are many situations in which it would be more natural to demarcate arbitrary, possibly concave spaces, rather than rectilinear ones. Much as window systems such as NeWS and X11 provide arbitrary 2D window geometry, we are currently considering replacing our rectilinear box model with one based on an implementation of Thibault and Naylor's BSP tree representation for polyhedra [THIB87].

While the functions discussed in this paper have closed-form expressions that are relatively easy to compute, we are interested in exploring more complex equations. Since interactive performance is essential, we are developing a server that will evaluate the equations on a faster processor.

Currently, the user has sole responsibility for designing the world and deciding how to explore it. We are interfacing n-Vision to Scope [BESH89], a rule-based system that designs user interface control panels by choosing and laying out appropriate widgets. Scope's rule base will be expanded to include rules that we are developing for choosing presentation techniques, assigning variables to coordinate systems, and selecting tools.

Acknowledgments

This work is supported in part by a gift from Citicorp, by the Hewlett-Packard Company under its AI University Grants Program, and by the New York State Center for Advanced Technology under Contract NYSSTF-CAT(89)-5. We wish to thank Dan Schutzer for sharing his financial expertise with us, David Sturman for providing us with drivers for the VPL DataGlove, and Scott Novack for assistance with the implementation.

References

[BANC78] Banchoff, T. "Computer Animation and the Geometry of Surfaces in 3- and 4-Space." *Proc. Int. Cong. of Math*, 1978, 1005–1013.

[BESH88] Beshers, C. and S. Feiner. "Real-Time 4D Animation on a 3d Graphics Workstation." *Proc. Graphics Interface '88*, Edmonton, June 6–10, 1988, 1–7.

[BESH89] Beshers, C., and S. Feiner. "Scope: Automated Generation of Graphical Interfaces." *Proc. ACM SIGGRAPH Symposium on User Interface Software and Technology*, Williamsburg, VA, November 13–15, 1989, 76–85.

[BLY82] Bly, S. "Presenting Information in Sound." *Proc. CHI '82*, 1982, 371–375.

[BROO88] Brooks, F., Jr. "Grasping Reality through Illusion—Interactive Graphics Serving Science." *Proc. CHI '88*, Washington, DC, May 15–19, 1988, 1–12.

[DONE78] Donelson. W. "Spatial Management of Information," (Proc. ACM SIGGRAPH 78) ,*Computer Graphics*, 12(3), August 1978, 203–209.

[FEIN82] Feiner, S., D. Salesin, and T. Banchoff. "DIAL: A diagrammatic animation language." *IEEE Computer Graphics*

and Applications, 2:7, September 1982, 43–54.

[HULL89] Hull, J. *Options, Futures, and Other Derivative Securities*, Prentice-Hall, NJ, 1989.

[KILP76] Kilpatrick, P.J. *The Use of a Kinesthetic Supplement in an Interactive Graphics System*, Ph.D. Thesis, Univ. of North Carolina, Chapel Hill, 1976.

[NOLL67] Noll, M. "A Computer Technique for Displaying *n*-Dimensional Hyperobjects." *CACM*, 10:8, August 1967, 469–473.

[OUHY89] Ouh-young, M., D. Beard and F. Brooks, Jr. "Force Display Performs Better than Visual Display in a Simple 6-D Docking Task." *Proc. IEEE Robotics and Automation Conf.*, May 1989, 1462–6.

[RUBI80] Rubin, S.M. and T. Whitted. "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Proc. SIGGRAPH 80, Computer Graphics*, 14(3), July 1980, 110–116.

[SCHL88] Scheifler, R.W., J. Gettys, and R. Newman. *X Window System C Library and Protocol Reference*, Digital Press, Bedford, MA, 1988.

[STER89] StereoGraphics, CrystalEyes product literature, StereoGraphics Corp., San Rafael, CA, 1989.

[STUR89] Sturman, D., D. Zeltzer, and S. Pieper. "Hands-on Interaction with Virtual Environments," *Proc. ACM SIGGRAPH Symp. on User Interface Software and Technology*, Williamsburg, VA, November 13–15, 1989, 19–24.

[SUTH65] Sutherland, I. "The Ultimate Display." *Proc. IFIP 65*, 1965, 506–508

[TEKT87] Tektronix. "3D Stereoscopic Color Graphics Workstation (TEK 4126 product literature)." Beaverton, OR, 1987.

[THIB87] Thibault, W.C. and B.F. Naylor, "Set Operations on Polyhedra Using Binary Space Partitioning Trees," *Proc. SIGGRAPH 87, Computer Graphics*, 21(5), 153–162.

[VICK70] Vickers, D.L., "Head-Mounted Display Terminal." *Proc. 1970 IEEE International Computer Group Conference*, 1970. Reprinted in J.C. Beatty and K.S. Booth, *Tutorial: Computer Graphics, 2nd Ed.*, IEEE Computer Society Press, Silver Spring, MD, 1982.

[VPL89] VPL Research Inc. "EyePhone System Preliminary Specification v.4." Redwood City, CA, July 1989.

[WEIM89] Weimer, D., and S. Ganapathy. "A Synthetic Visual Environment with Hand Gesturing and Voice Input." *Proc. CHI '89*, Austin, TX, April 30–May 4, 1989, 235–240.

[ZIMM87] Zimmerman, T., J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill. "A Hand Gesture Interface Device." *Proc. CHI + GI 1987*, Toronto, Ontario, April 5–7, 1987, 189–192.