WiMP (Widgets, Menus and Pointing) Design Tools for Virtual Environments

by

Doug Bowman and Larry F. Hodges

GIT-GVU-94-37
September 1994

Graphics, Visualization & Usability Center

Georgia Institute of Technology
Atlanta GA 30332-0280

WiMP (Widgets, Menus and Pointing) Design Tools for Virtual Environments

Doug Bowman and Larry F. Hodges

Graphics, Visualization & Usability Center
College of Computing
Georgia Tech

ABSTRACT

Many early virtual environment (VE) applications were simple walkthroughs. The effect was intended to be visual, with little or no direct manipulation of the space by the user. A logical next step is the creation of interfaces and tools that allow a user to design within an immersive space. In this paper we identify issues and desired attributes for immersive design tools. We also describe a preliminary design metaphor, WiMP, that combines menus, pointing, direct manipulation and widgets to provide a useful set of design tools.

Beginning in the summer of 1994, a group of computer scientists, architects and engineers began regular meetings at Georgia Tech to discuss issues related to both teaching design concepts and doing design within virtual environments. This paper is intended as both a report on those discussions and a description of some of the ideas that have been implemented and tested.

Motivation and Challenges

Virtual environments (VEs) offer an immersive geometry in which one can physically and psychologically be in the space that one is designing. Many of the initial virtual reality applications were walkthroughs and visual inspections of architectural spaces (Brooks, 1986; Teller and Sequin, 1991). The ability to walk through a virtual model of a design has been generally accepted as an effective way to increase understanding of the model. Some of the advantages of an immersive geometry for understanding spatial designs include:

- The ability to inspect lines of sight, and scale and form of objects within a geometrically correct full-scale visual model of the space.
- The option of exploring a space from different points of view, such as shrinking the user to see the space from a child's point of view or examining the space with respect to the viewpoint and accessibility of a person in a wheelchair.
- The ability to dynamically interact with lighting conditions affecting a model. The perception of an architectural space is much affected by considerations of volume, material and light.
- The illusion of "presence" in the space. In a way that we still do not completely understand, a person's experience of a situation in a virtual environment may evoke the same reactions and emotions as the experience of a similar real-world situation. This may be true even when the virtual environment does not accurately or

completely represent the real-world situation. (Rothbaum, et al., 1994).

It is inevitable that the process of inspecting a space via a virtual reality walkthrough will result in the desire to make changes in the model of the space. A logical next step for VR and design is the creation of a virtual reality CAD (Computer-Aided Design) system that allows a user to design from within an immersive space. Attempts to create such a system, however, have not yet lived up to the expectations of professional designers. Prototypes have been "toy systems" lacking both the rich set of tools and display modalities available for workstation-based CAD.

There are a number of challenges associated with creating immersive, interactive environments for computer-aided design. Some of these challenges, such as the limited resolution of head-mounted displays, inaccurate data from tracking devices, and real-time computation of complex scenes, are being addressed by the rapid improvement in hardware devices associated with VR and computer graphics. Conceptual issues, however, concerning *how design should be done* and *if it should be done* within immersive environments are almost all open research questions. In particular, the more general problems of how to do 3-D interaction and what type of tool interfaces are appropriate to support that interaction are actively being discussed (Gomez, et al., 1994; Snibbe, et. al., 1992).

Some general issues include:

- The familiar physical input and interaction devices such as keyboard, mouse, buttons, and dials are not present inside a virtual environment.

- VR input and interaction devices such as 3-D mice and whole-hand input devices (gloves) are still in the development stage. It is not yet clear which types of devices will prove to be useful or even how they should be used.

- Familiar 2-D metaphors such as the desktop, pop-down menus, cursors, widgets and icons must be modified or completely redesigned in order to be usable in an immersive 3-D space.

- Although an immersive environment is very helpful for a local understanding of geometry and visual impact of design changes, it is not helpful for global understanding of where the user is within the entire design and how changes in one part of a structure will affect other parts. Simultaneous multiple presentations of the structure are still needed to make design decisions.

- Multiple windows in a head-mounted display obscure the user's view, and also detract heavily from the sense of immersion.

- The space within which the user is working is visually more confusing. Spatial locations extend in all three dimensions, and dimensions may not be bounded.

DESIGN TOOL ATTRIBUTES

During meetings with architects and engineers from the School of Architecture at Georgia Tech and the Architectural firm of Lord, Aeck and Sargent, Inc. in Atlanta, our discussions focused on defining what types of design tasks should be done within immersive virtual environments, what types of tools were useful, and what tools users would actually use and accept.

Two paradigms for tools emerged from these discussions. The industry architects preferred tools that were extensions of those used in traditional PC- or workstation-based CAD. The academic architects and engineers, who ranged from expert CAD users to non-CAD

users, also indicated an interest in virtual environments in which tools were available which were metaphors for those tools they were accustomed to using in the physical world.[1]

Whichever of these two paradigms is used, it became clear that all tools for immersive interaction should share some minimal set of attributes. This set of attributes includes the following:

- Input tools should have both a physical and virtual manifestation, i.e., they should be felt as well as seen inside the immersive environment.
- The user should be able to interact with anything he can see, regardless of the scale of the space or the apparent distance of an object from his current location. This extension of the user's powers (being able to perform what would be superhuman tasks in the physical world) is one of the most compelling reasons for the use of virtual reality.
- Tools should provide visual, tactile and/or auditory feedback to indicate what is currently being picked, chosen or located.
- Tools should be simple and intuitive. As far as is feasible they should be an extension of 2-D tools and utilities with which users are already familiar. Alternately, they may be metaphors for real-world tools.
- Tools should operate the same way for every user, and the tools should be reusable in many different types of applications.
- Users should retain a feeling of immersion while using tools.

[1]This second idea is similar to the self-disclosing widget discussed by Snibbe, et al. (1992).

WIMP: WIDGETS, MENUS AND POINTING

Most CAD systems, along with almost all other contemporary workstation applications, use WIMP (Windows, Icons, Menus and Pointing) graphical user interfaces. (GUIs). This approach has a twenty year history and has been very successful. We have pursued a strategy of combining some of the aspects of WIMP, particularly menus and pointing, with direct manipulation in 3-D and 3-D widgets (Gomez, et al., 1994). Because it was too good an acronym to pass up, we call this approach the WiMP (Widgets, Menus and Pointing) interface for immersive environments. Our prototype system, known as Conceptual Design Space (CDS) is built using the Simple Virtual Environment (SVE) toolkit. SVE supports both the creation of virtual environments (VEs) and the creation of VE interface tools and objects (Kessler, et al., 1994).

MENUS

Menus, in general, offer a number of advantages for human-computer interaction. The user is not required to memorize long lists of commands (keyboard input, hand signals, or multiple button combinations) in order to carry out desired actions. Instead, one can choose from a list of possible actions. We have developed a menu system for virtual environments that is effective, intuitive, and does not detract from the immersive sense of the user.

Issues in the creation of immersive, 3-D menu systems include the appearance of the menus in the virtual environment, the location of menus, visual cues when choosing menu items, and toolkit support.

For our first implementation, it was decided that menus should be three-dimensional, solid objects just like other objects in the virtual world. Each entry is represented by a

trapezoidal box of a fixed size. Each menu has a "header entry" or "title bar" which names the menu and defines its top. Menu entries are placed sequentially below the title bar. Each header and entry is associated with a text string which is printed on the front of the box.

At first, menus were always visible, but remained fixed in a single position. If the user moved too far in any direction, or went "behind" a menu, the commands were no longer readable or reachable. The next idea was to have menus "follow" the user around the virtual environment. With this enhancement, the commands would always be available, but the user might have to turn away from his/her work to select an action. Finally, we agreed that menus should be attached to the user's head position, so that they always appeared in the same position in the view area (Figure 1).

We also had to consider the fact that menus could obscure much of the actual working environment. In the initial implementation, menu entries were always visible to the user. In a complex application, though, this would mean that most of the limited screen real estate would be taken up by menus! The solution was taken from the traditional 2-D menu metaphor, in the form of "pull-down" menus. In this implementation, only menu title bars are visible initially. When the user selects a title bar, the entries associated with it appear (Figure 2). After the selection process has been completed, the entries disappear again, so that the workspace remains uncluttered.

Visual cues for the user were also an important aspect of our design. In three dimensions, it is much more difficult to ascertain what is being pointed at by the pointing device. In order to make this process easier, and avoid mistakes, some sort of visual feedback is needed by the user. Again borrowing what works from 2-D menu design, when a menu item is selected, it changes from its normal color to a "highlight color" (Figure 2). This is immediately obvious to the user, easing the process of pointing and selecting.

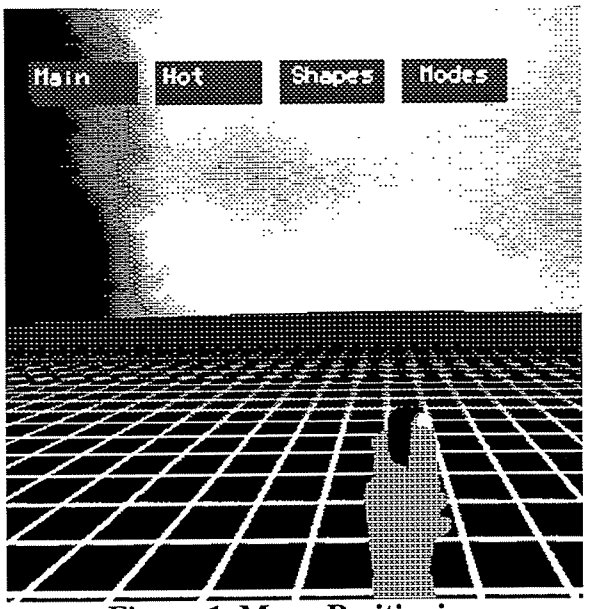

Figure 1. Menu Positioning

Creation of an application programmer interface to allow developers to create their own menu items was made straightforward by the existing structure of our SVE toolkit (Kessler, et al., 1994). SVE uses two file formats to specify the virtual world: the "world" file, which lists all of the objects in the environment, and gives their properties, and "primitives" files, which specify characteristics of individual objects.

The world file is hierarchical; that is, objects are placed in a tree format, with parent and child objects. Any change to a parent object is automatically reflected in all of its children. Thus, a menu is specified with the title bar being a top-level object, and its entries listed as children. Each child is given a position offset by a certain fixed amount in the negative Y direction, so that the menu appears as a stack of items. This also allows for the creation of submenus (menus within a menu). Submenu items are simply listed as children of the appropriate menu entry.

In the primitives files, only a few characteristics must be given to each menu object. In the case of a menu header, all that is required is the title itself (a string), and the desired color of the title bar. For an actual entry, a name, a normal color, and a highlight color (see above) are needed.

One advantage to these file formats is that they are independent of the source code for the application. In other words, changes to the world file or primitives files do not require re-compilation. Thus, many different options can be explored easily.

Finally, we provide a simple method for programmers to specify the actions their menu items will be linked to. This, obviously, is the only part of the menu system which requires additional source code on the part of the application developer. For each menu item, the programmer specifies a callback function to be executed when that menu entry is selected. The callback function may be user-defined or it may come from a library of common commands.

POINTING

Pointing, and the use of pointing to implement choice and pick operations, is intrinsically harder to do in three dimensions than in two. Positioning is harder, visual feedback is less precise, and tasks are more complex.

An effective pointing device must have both a physical and virtual manifestation. The physical manifestation is the object the user holds and operates. The virtual manifestation is the representation and the behavior of the object within the virtual environment.

Over the past two years we have experimented with several interaction metaphors and devices. Whole-hand input devices (gloves) have been promoted by some as providing a natural interface for interacting with objects in virtual spaces (Zimmerman, et. al., 1987). Despite these claims, the use of whole hand input has been limited in almost every published report to recognition of a few commands associated with the hand's posture. Most glove and other hand posture recognition devices are used for tasks that could be accomplished as well or better (and at a fraction of the cost) with a three-button mouse used in conjunction with a 3-D tracker (Kessler, et al., 1994).

Based on our own glove experience in an early prototype we decided instead to develop a three-dimensional mouse as the physical manifestation of a pointing tool. In its simplest form, this is just a common computer mouse with a tracker attached to it. The tracker allows for the three dimensional positioning of the mouse object, and events can be generated easily by the existing mouse button(s). Events may then be queued by the operating system, and the VR application can interpret them and take appropriate action.

We also built a more elegant and natural physical manifestation known to our group as the "FliteStik". This device resembles the top portion of many joysticks used by players of flight simulation games. It fits the hand much better than a mouse, and has two buttons, one in the "trigger" position, and another on top for the thumb. The device is built so as to produce the same events as the left and right buttons on a standard three-button mouse.

The virtual manifestation was more challenging. Accurate pointing in an immersive three dimension space is not as easy as pointing on a 2-D screen. To be useful, a representation of the pointing device within the virtual environment must give accurate feedback as to what is being pointed to, and must be able to reach anything that is visible.

To solve this problem, the analogy of a laser pointer is used (Figure 2). When the top button on the FliteStik is depressed, an elongated polygon resembling a ray of light is attached to the FliteStik object in the virtual

world. This causes the design system to enter selection mode. If, at any time while selection mode is active, the light ray intersects one of the menu headers, the entries associated with that menu will appear. As the user continues to hold the button down, if then the light ray intersects one of the menu entries, it changes color as a visual cue that it is being pointed at. When the button is released, a final intersection test is performed. If the light ray is intersecting one of the visible menu items, that item is selected, and the appropriate callback function is executed.

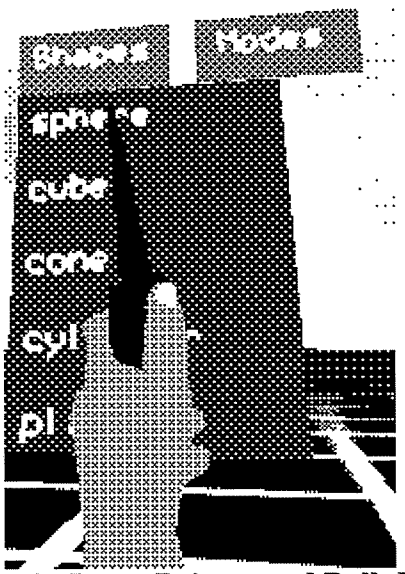

Figure 2. Laser Pointer and Pull-down Menus

There are two major advantages to this system. First, it simulates the behavior of common menu systems used by millions of people on personal computers, in that the pressing of the button begins the selection process, and the release of the button causes an item to be selected, if appropriate. Today, almost everyone has used such a system and is familiar with its operation. Thus, little training is involved.

Secondly, the system eliminates mistakes. An early problem that we faced when using a pointer was the selection of the wrong item. This occurred because selection happened when the button was pressed, not when it was released (essentially, a gun). Thus, the user was forced to guess what was being pointed at before pressing the button. At first, to solve this problem, we implemented a two-button system. One button was used as a test button. It highlighted the menu item being pointed at, but no action was taken. When the user found the proper position, the other button could be pressed to select the object. This eliminated most mistakes, but it's not unlikely that the hand could move slightly in between the test and the actual selection, causing a mistake, or that an inaccurate tracker could change the hand position and produce an error. Besides this, the system was quite cumbersome, and required the use of both buttons on the FliteStik.

In the current system, no action is taken until the user is sure the correct item will be selected, and he/she releases the button. Also, the user may cancel the selection process by releasing the button while nothing is highlighted.

WIDGETS AND DIRECT MANIPULATION

The immersive menu system is an important step towards achieving a high level of interaction, since it can provide an interface for any type of functionality desired. In some situations, however, it is preferable to use the same point and click ideas to provide alternate methods of working with objects in the virtual environment. For example, there may be a menu command to rotate an object five degrees. Even if it works perfectly and efficiently, it will be tiresome for the user to select the item eighteen times in order to perform a ninety-degree rotation. Further, the user would not be allowed to rotate the object 2.3 degrees, or at least would be required to traverse more menus in order to change the amount of rotation to an acceptable level. Clearly, there are some design tasks that are ill-suited to a menu interface. We want our virtual environments interaction tools to aid in the work to be done, rather than creating more work for the user, which could quickly overshadow the benefits gleaned.

To supplement the menu system, we use both 3-D widgets and direct manipulation of objects. Widgets and direct manipulation all use the same point and click interface that is used in the menu system, so that the integration of the two parts is seamless.

To select an object the user simply holds down the FliteStik button until the light ray intersects the desired object, then releases the button. When two or more objects are intersected, the selection algorithm consistently chooses the one nearest to the user, so that no guessing is involved. When an object is chosen, widgets such as a set of axes, proportional to the size of the object, appear within the object (Figure 3). Currently, our software supports common 3-D transformations such as translation, rotation and scaling.

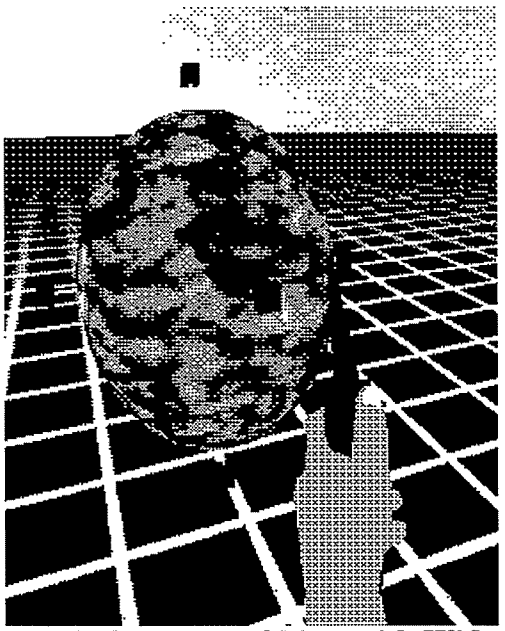

Figure 3. Geometric Object with Widgets

Our current application has several widget and direct manipulation modes that allow the user to perform a variety of tasks using the same basic actions. The mode is changed via menu commands. With the program in translate mode, when the user selects a world object, that object becomes attached to the light ray in the virtual environment. In order to translate the object, the user simply moves his/her hand, or uses the flying capability of the program to move the object farther than he/she can reach. To rotate the object, the user simply rotates the hand holding the FliteStik. When the desired transformations are complete, depressing the button releases the object. This method is also used automatically when the user creates a new object, so that it may be placed in the environment at an appropriate initial position.

Scaling is accomplished through an intuitive interface, as well. After an object has been selected, when the program is in scale mode, the user may point to one of the six widget boxes that appear on the ends of the coordinate axes. This action causes the light ray to attach to the box, and as the user moves his/her hand in the world, the object scales in proportion to the movement. Stretch or shrink mode allows scaling in a single dimension, determined by the axis the user selects (Figure 3).

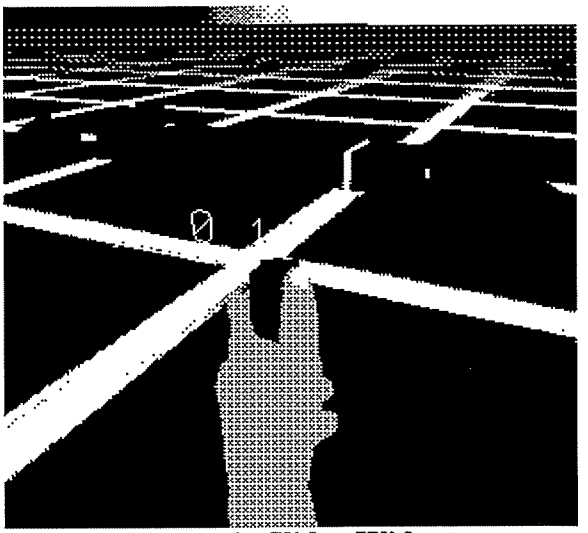

Figure 4. Slider Widget

If the user requires finer control over these transformations, each of them is accessible through a menu command as well. To determine the amount of rotation, scaling, stretching, or shrinking, a slider widget was developed (Figure 4). The slider is a bar in the virtual world which shows the current

transformation value for each of the modes mentioned. When a new mode is selected, its slider automatically appears. The value is represented graphically, with a "handle" on the slider, and with a numerical value printed below it. There are two interaction methods for changing the value. First, the user may point and click the two arrow buttons at either end of the bar to change it in set increments. Alternately, the handle may be selected and moved to a precise location on the slider.

Direct manipulation may also be used in conjunction with the menu to implement other basic actions on objects. For example, one may select an object using the FliteStik and then select the "copy" menu item to obtain an exact copy of the object, or the "delete" menu item to remove it from the environment. Also, we have implemented a "group" function that allows the user to select a number of objects. The first object chosen then becomes the parent of all subsequent objects, so that a hierarchical structure may be dynamically obtained. Later, when a transformation is applied to the parent, it automatically becomes applied to the children.

CONCLUSION

We have described an implementation that combines three-dimensional extensions of menus and pointing with widgets and direct manipulation in an immersive environment.

These tools are precise: the laser pointer metaphor allows the user to select with certainty the command or object desired, and the text of the menu items can be more descriptive than an icon representation of a command.

The system is also intuitive. While menus are not completely natural, they are understood by a large portion of the population. The use of menus is an efficient way to select commands. Because functions can be grouped in a meaningful way, and can include submenus, negotiating the menus takes little time and effort, and users can focus on the work to be performed, rather than on the use of the interface. Widgets allow simple, direct manipulation of designed objects.

In addition, the menus and other tools we describe are easy-to-use. All that is required is the pointing of a device and the pressing of a button. Both the creation of application objects and the creation of VE interface tools and objects are supported by the underlying SVE toolkit.

ACKNOWLEDGMENTS

This work was supported, in part, by a grant from the EduTech Institute and by Lord, Aeck and Sargent Architects, Inc. Many of the ideas presented were a result of discussions within the Virtual Design Working Group: Doug Bowman, Hamish Caldwell, Harris Dimitropoulos, Larry F. Hodges, Tolek Lesniewski, Tom Meyer, Terry Sargent, Andy Smith, Brian Wills and Jean Wineman.

REFERENCES

Brooks, Jr., F.P. (1986). Walkthrough - a dynamic graphics system for simulating virtual buildings. *Proc. 1986 Workshop on Interactive 3D Graphics*, pp. 9-21.

Gomez, J.E., Carey, R. , Fields, T., van Dam, A. and Venolia, D. (1994). Why is 3-D interaction so hard and what can we really do about it? Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics Proceedings,* Annual Conference Series, pp. 492-493.

Kessler, G.A., Kooper, R., Verlinden, J., Hodges, L.F. (1994). The Simple Virtual Environment (SVE) Toolkit. Georgia Tech

technical report GIT-GVU-94-34, 104 pages.

Kessler, G.A., Hodges, L.F. and Walker, N. (1994). Evaluation of a Whole Hand Input Device. Currently submitted to ACM TOCHI.

Rothbaum, B.O., Hodges, L.F., Kooper, R., Opdyke, D., Williford, J.S., North, M. (1994). Virtual reality graded exposure in the treatment of acrophobia: a case study. To appear in *Behaviour Therapy.*

Snibbe, S.S., Herndon, K.P., Robbins, D.C., Conner, D.B., and van Dam, A. (1992). Using deformations to explore 3D widget design. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31). In *Computer Graphics* 22,6, pp. 351-352.

Teller, S.J. and Sequin, C.H. (1991). Visibility Preprocessing for Interactive Walkthroughs. Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 28-August 2, 1991). In *Computer Graphics* 25,4, pp. 61-69.

Zimmerman, T., Lanier, J., Blanchard, C., Bryson, S., and Harvill, Y., (1987). A Hand Gesture Interface Device. *CHI + GI Conference Proceedings* (April 5-9, 1987) pp. 189-192 .