User Interface Constraints for Immersive Virtual Environment Applications

Doug A. Bowman and Larry F. Hodges
{bowman, hodges}@cc.gatech.edu
Graphics, Visualization, and Usability Center
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

Abstract

Applications of Virtual Environments (VEs) are rapidly becoming more complex and interactive. They are not restricted to tasks that are solely perceptual in nature; rather, they involve both perception and action on the part of the user. With this increased complexity comes a host of problems relating to the user interface (UI) of such systems. Researchers have produced a body of work on displays, input devices, and other hardware, but very few guidelines have been suggested for user interface software in 3D VEs. In this paper, we discuss the usage and implementation of constraints, a fundamental principle for desktop user interfaces, in highly interactive virtual environment systems. Our claims are supported with examples from the literature and from our own experience with the Conceptual Design Space (CDS) application.

INTRODUCTION

Virtual environments (VEs) and virtual reality (VR) hold great promise for a wide variety of applications in business, industry, education, and entertainment. However, very few immersive VR applications are in use outside the setting of academic research.

Surely, however, the potential exists for immersive VR applications to help us do more than walk through a planned building or play a 3D tank game. Surely some sort of "real work" - work which produces results and benefits for real-world problems - can best be accomplished in an immersive VE. Many ideas and prototypes for such applications have emerged from research centers in such diverse areas as medicine, education, CAD/CAM, and visualization. Why, then, have so few of these become competitive products among the groups for which they were developed?

Many answers to this question exist, including prohibitive cost of equipment, or slow and imprecise hardware. One important answer, however, becomes apparent when one looks at the defining characteristics of two applications that *have* made it on a commercial level. Both architectural walkthroughs and VR video games can be characterized by a very low or non-existent level of user interactivity (Note: by the term "interactivity", we mean the user's ability to create, manipulate, or change the objects in the environment, or the environment itself. The interaction of head-tracking is excluded, because it is a defining attribute of immersive VEs). In other words, the user can do little more than look around, navigate the environment in some way, and perhaps open a door or fire at an enemy tank.

These applications are in the mainstream *precisely because* they require little interactivity. More complex systems remain in research laboratories, because while their functionality is impressive, their interfaces to that functionality, and their user interaction metaphors, are inconsistent, imprecise, inefficient, and perhaps unusable.

We believe that many of these complex applications suffer from a lack of attention to the user interface (UI) software. Although a great deal of research has gone into interaction technology such as haptic devices, gesture-recognition, and displays, often fundamental principles of UI design have been ignored.

In this paper we will explore the application of one important interaction principle for desktop UIs, namely **constraints**, to immersive VE systems. We will also offer suggestions for the implementation of constraints in the most common user tasks in immersive applications. We hope that this paper will foster more discussion on the need for better UI design for virtual environments, and that it will offer a first step towards more usable and efficient VR applications through the use of constraints.

Throughout the paper, in addition to numerous examples from the literature, we will present instances from our own experiences with the Conceptual Design Space (CDS) system [24]. CDS is a highly functional, complex system offering tools for architectural design in an immersive environment. The system has been used by professional architects as well as architecture students for real design projects. Because of the high degree of interactivity as well as the ability to do real work in the system, CDS

provides an ideal platform for the study of user interface issues for immersive applications.

FUNDAMENTAL USER INTERFACE GUIDELINES

Perhaps the best-known human-computer interaction guidelines are those proposed by Donald Norman in his book *The Design of Everyday Things* [1]. Norman argues for an interface to computers that replicates all of the best things about our interaction with the physical world, while letting the machines transparently perform the computations for which they were designed.

It is important that these principles are taken from the physical world. Since a virtual environment usually attempts to mimic real-world interactions, these guidelines should be even more applicable.

Also, appropriate and usable interfaces to VE systems will become increasingly necessary as these systems move from the research laboratories into more widespread use. In a real-use setting, the developer will not always be there to coach the users, and the users themselves will not be experts. Thus, our interfaces must become more usable, requiring that closer attention be paid to principles such as those described below.

Norman proposes several general user interface guidelines. We will briefly discuss three of these guidelines: affordances, mappings, and feedback, and their application to virtual environment applications. We will focus on a fourth principle, constraints, for the remainder of the paper. These four guidelines cannot be seen separately, however: they are all related and interconnected. To obtain a usable interface, each of the principles must be considered.

Affordances

The first guideline concerns the provision of **affordances**. Affordances are those elements of an object or tool that give away its purpose and usage. As an example, consider a coffee mug. The mug's physical characteristics, especially those of its handle, naturally lead the user to hold it in a certain way. In other words, the mug **affords** this holding position. Norman argues that computer interface software should provide appropriate affordances as well, so that the user is naturally led to correct actions rather than to errors.

For virtual environments, affordances would seem to be simple. In theory, if we provide tools that look and act just like their counterparts in the real world (e.g. scissors for cutting, glue for pasting), then users will intuitively grasp their meaning and immediately begin to perform real work. In practice, things are never so simple.

Some researchers have attempted to implement VE systems where all tasks are afforded naturally. Most of this work has fallen far short of its goals, however. Smets, et al. [2] point to artificial intelligence and more advanced hardware technology as the sciences that will allow them to produce this natural interaction. For now, though, it seems that we must explore different methods for affording correct user actions.

Mappings

A second guideline proposed by Norman, and one closely related to affordances, is that good **mappings** must exist between user actions and system actions. In other words, an input by the user via the interface should produce a proportional response within the system. For example, when the user of a desktop interface clicks the mouse on an icon, she expects that the internal state of the system will change to reflect that the entity represented by that icon is now "selected".

Some good mappings exist in all immersive virtual environment systems, such as mapping user head and/or hand motion to a corresponding change in the displayed scene. It is more difficult, however, to produce good mappings at a higher level. One reason for this is the immense amount of freedom given the users of most VE applications.

Feedback

A third characteristic proposed by Norman for good interfaces is **feedback**. Feedback refers to the process of sending back information to the user about what has been done. Good feedback should follow naturally from good mappings: if the user has performed an action that triggered an internal system response, the system should let the user know what happened via feedback at the interface.

Just as they are built-in mappings, head and hand motion are inherent feedback for immersive VR systems. This feedback, however, does not help the user to perform work except by allowing her to view the correct portion of the environment. Complex systems, in terms of user interactivity, require much more feedback to keep the user informed of system state and responses. Common information relayed via feedback includes selections, modes, locations, etc.

CONSTRAINTS IN 3D VIRTUAL ENVIRONMENTS

The fourth of Norman's guidelines, and the focus of this paper, is the need for **constraints**. Constraints are the converse of affordances: they limit the possible actions of an object.

The word "constraint" itself generally has a negative connotation, since it refers to something

that limits us, but constraints are necessary in both physical and artificial situations. Consider how impossible it would be to live our daily lives without the constraints of gravity, impenetrability of solid surfaces, friction, and so forth. A world without these things would be chaos, of course.

Most virtual environment applications, though, present exactly that world: our systems lack gravity, solid surfaces, friction, and other useful constraints. Herein lies a great part of the reason why performing work in VEs is so difficult. In this section, we discuss in detail the need for constraints in various aspects of VR systems, and possibilities for their implementation.

Input Devices

The input devices used to interact with a VE system provide the all-important link between actions in the physical and virtual worlds. Because of this, badly designed input devices can render unusable the most carefully constructed software UI. Input devices in common use today for VE applications are gloves, 3D mice (with a variety of different names, shapes, and number of buttons), and in some cases, voice-recognition systems. Although this paper is mainly concerned with the implementation of software constraints for VE interfaces, we will briefly consider these most common options for input devices, with respect to their level of constraints.

When most people think of a VR input device, the glove immediately comes to mind. It is supposed to be the hardware that allows us to do anything in a virtual world that our hand can do in the real world. From the standpoint of constraints, though, gloves provide very few. There are simply too many degrees of freedom, too many possible device configurations. Better recognition algorithms are being developed, but with so few constraints, the user of a glove device is almost certain at some point to fail to produce the correct position or inadvertently trigger a response by the system [3].

Voice-recognition systems suffer from much of the same problem. There are such a variety of sounds produced by a single human voice, not to mention the variety of different voices for different users, that fast and accurate speech-recognition is very difficult to achieve. Again, algorithms are improving, but the most accurate still require training by each individual user before actual use. This is unacceptable for systems that will have a large number of occasional and first-time users. The lack of constrained input is again the problem.

Also, both glove and voice systems lack constraints from the user point of view, since nothing exists to tell users which commands are valid (a lack of knowledge in the world).

3D mice, in all of their forms, are the most constrained of the three types of input devices listed above. While the device itself still has six degrees of freedom of motion, its buttons have only one: they are either up or down. This constraint allows precise input to the system. It may not be as simple or elegant as a gesture or voice-command, but it is accurate.

Both gloves and 3D mice are usually tracked in three-dimensional space. In effect, then, a 3D cursor is created. This produces another constraints problem, in the sense that spatial input with these devices (e.g. moving an object by moving the hand) will be imprecise. Since VEs are by nature three-dimensional, we cannot constrain the motion of the input device to two dimensions in general. However, software constraints, described later, can be implemented that help to solve this problem of inexact spatial input.

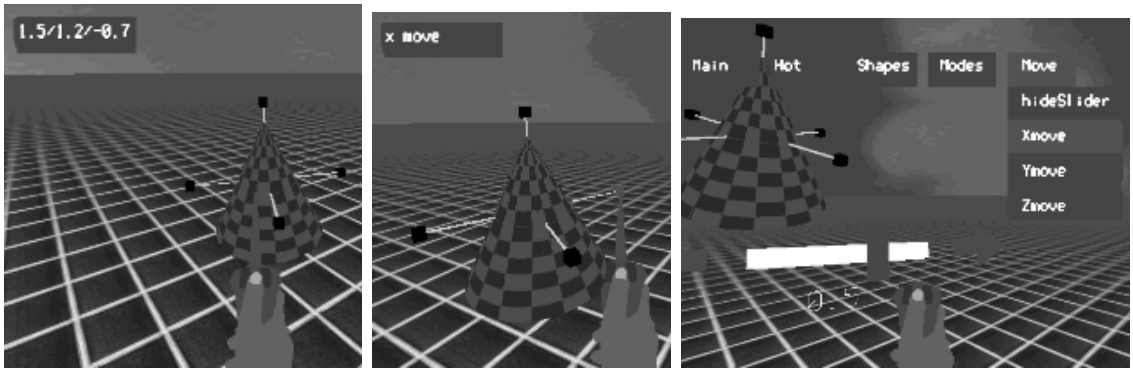
Objects

In many applications, the objects in the virtual environment, as well as the behavior of these objects, should incorporate constraints. This is a very general requirement, so let us consider some examples.

Boundaries, a special case of the general collision detection constraint, are almost always applicable, but are often ignored. By boundaries, we mean that the virtual environment should take up a defined space, out of which users should not be allowed to navigate. The authors have seen many VR applications in which the user was confused and frustrated by flying through the floor or out of the work environment altogether.

It is simple and computationally inexpensive to program boundary constraints that keep the user within the working space. General collision detection (in which no objects can pass through one another) is often the most desirable, but may not always be feasible in real-time. A simple boundary constraint, though, can greatly decrease user frustration and error.

As another example of object constraints, consider an application such as the Conceptual Design Space (CDS) described in the introduction. In our observation of users of this VR design system, an early stumbling block was the simple task of object positioning. In the first implementation of CDS, the only way to move an object was to select it (our selection mechanism is described elsewhere in this paper), which attached the object to the hand tracker (see Figure 1a), then move the hand and/or navigate through the environment before detaching the object in the desired position.



(a) (b) (c)

Figure 1. Three techniques for object motion in CDS: (a) direct manipulation, (b) constrained to a single dimension, (c) indirect manipulation using menu and slider widgets.

This method works very well for large-scale, gross motions, but is totally unusable for fine placement. Users would place the object so that it appeared in position from their vantage point, but would discover upon moving elsewhere that the position and orientation of the object were incorrect.

To solve this problem, two levels of constrained motion were added to the system. First, control handles were attached to the object on each of the three principal axes (similar to the transformation widgets described in [4]). To move the object in one dimension only, the user simply selects the appropriate handle (Fig. 1b). Subsequently, all hand motion is ignored except for translation in the selected direction, and the object is moved along with this translation until it is detached by the user.

Secondly, we provided even more precision and constraint by introducing a command system for object motion. The user first sets a translation value using a slider widget, then issues a command via a menu selection, specifying the direction of motion (Fig. 1c). The selected object is moved the specified amount in the given direction.

Taken in the order given above, these three implementations become more cumbersome, but also more accurate. There is a tradeoff between ease-of-use and precision for which a compromise must be found. We have found that redundancy (in our case, offering three methods for object motion) allows the user to choose the method which best fits the task at hand.

From these examples, then, it is clear that most, if not all, VE applications would benefit from the use of object constraints. In general, we have seen that it is helpful to:

- constrain the navigation of the user to a bounded area,
- reduce the number of degrees of freedom of an object to increase precision, and
- provide redundant methods for tasks that allow the user to choose the level of constraint.

Tools

We define **tools** in a virtual environment as those objects in the environment that assist the user in performing work with the system. They are specialized objects, in some ways not part of the environment itself, but rather the representations of methods the user may employ to perform actions on or in the environment. In this sense, we may use the terms “tool” and “interface element” interchangeably (Some would argue that “tools” consist only of those interface elements which allow direct manipulation of virtual objects, but we make no such distinction. We have found that indirect methods can be quite effective in virtual environments, even though some researchers feel that their use detracts from the “reality” of the environment [2]).

Given this definition, we can say that tool constraints are also a necessary part of a usable VE application. Tools need to be limited in their function, their number of configurations, their degrees of spatial freedom, and so on. However, we must make a tradeoff between a tool’s generality and its constraints. It is clearly desirable for VE tools to be general and reusable: just as the same library of interface elements are used in most desktop applications (menus, windows, buttons, icons, etc.), we should also be able to reuse tools in VE systems. If tools are made too general, though, a lack of constraints can affect their usability.

We have implemented several interface tools for CDS that we feel are both general and constrained, but let us consider one example at this point. In CDS, as in several other VE

applications [5,6,7], virtual pull-down menus are used to issue many system commands.

First, consider the inherent constraint and precision of pull-down menus. Unlike continuous-value tools, such as direct object dragging, menus have by nature only discrete values (the discrete entries in the menu). This property leads to increased accuracy for the user (consider the relative difficulty of placing an object at an exact coordinate vs. choosing a given menu entry). Also, precision can be enhanced further through the use of a constraint which snaps the pointer to the center of menu items [5].

A second important constraint can be implemented for menus when one considers the spatial nature of pull-down menus: they require only two degrees of freedom to operate. In other words, viewing a pull-down menu from the side or the bottom does not increase their utility (it will almost certainly make them *harder* to use), even if the menus are actually 3D objects. Furthermore, if menus are fixed in the environment or only move when the user performs a specific action (e.g. picking them up and moving them elsewhere), then the user will almost certainly lose track of their location.

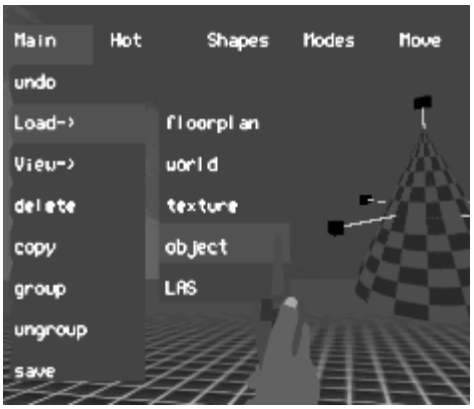


Figure 2. Hierarchical virtual menus in CDS.

Thus, there is no reason to allow users to move relative to menus. In CDS, as well as in another implementation [6], menus are normally seen simply as titles, which are bound to the user's head position so that no matter where she looks, the menu titles are in the same position in her field of view (FOV). In CDS, this is the extreme top edge of the FOV, so as to obscure as little environment information as possible. When a menu is selected, its entries appear below it, from which an item may be chosen (see Figure 2).

These two simple constraints greatly increase the usability of the pull-down menus: they are accurate and always available. Furthermore, menus are a completely general tool that can easily be reused in any highly interactive VE system. Careful consideration, then, of both the generality and constraints of interface tools for VEs is extremely important if real work is to be performed.

CONSTRAINTS FOR UNIVERSAL TASKS

Besides their application to the various physical and virtual components of a system, such as input devices, objects, and tools, we can also consider constraints for VE user interfaces from another point of view. Just as with desktop applications, there are a set of tasks that are practically universal to all interactive VE systems.

In 2D, analysis of different methods of performing universal tasks, such as pointing, dragging, selecting, and other fundamental actions, has led to efficient and effective techniques, as well as a greater understanding of some general issues for usable interfaces (e.g. [8]).

Naturally, then, if such tasks exist for immersive VEs, we should attempt to understand these tasks and compare the various techniques available. In this section, we present four "universal" actions of immersive VR applications, techniques used to perform these tasks, and an informal analysis of them based on the constraints principle.

Navigation

Probably the most common task of all in VEs is that of navigating through the space of the environment. Some artificial method must be provided for the user to move through the space, assuming that it is larger than the area that can be accurately tracked by the tracking system, and that the application is not so limited that a single user position with small head motions and rotations is sufficient. Because this task is so prevalent, there are almost as many solutions to it as there are VR applications! Here, we take only a brief look at navigation techniques in the context of constraints.

Perhaps the most natural method, though not the simplest to implement, is to use **physical user motion**. This has been implemented with treadmills, roller skates, bicycles, etc. This method contains inherent constraints, in that the user can move freely only in two dimensions (on the ground plane). This is helpful because users are less likely to become disoriented. However, it also exhibits undesirable constraints. Since navigation speed is usually limited to the physical speed of the user, a large environment is difficult to navigate in this way [23]. Also, how does the user obtain aerial views of the world, or stand on the top floor of the building? This technique is

clearly not general or flexible enough for most applications.

Another simple technique that is often implemented is artificial **flying**, usually in the direction of the user's gaze or the user's pointing [13,20]. Generally, the user simply looks or points in the direction she wants to go, and presses a button or makes a gesture to fly in that direction with a constant speed. Most often, the user is allowed to fly in any direction with complete freedom. Clearly, this technique is flexible (assuming velocity may be changed), and gives the user complete freedom over her position and orientation in the space. In the light of our guidelines, though, we can see that this method has a lack of constraints: the user can easily get lost or disoriented if given complete freedom [23].

To solve this problem, **walking** can be introduced. This is the same as the flying techniques, except that the user's head is constrained to a given height above the ground plane. Since it would be too restrictive to make this the only method of navigation, in CDS we have allowed users to toggle between flying and walking modes. This has proven to be quite useful in that the users had complete control over their position if necessary, but could also lock themselves to a certain height (e.g. to walk on a floor of a building, or to determine whether an obstacle was too low to walk underneath).

Other methods (scaling, manipulation of an iconic representation of the user, leaning, etc.) and issues (interactive velocity and acceleration changes, effect on users of constant resizing, etc.) for navigation in VEs have been discussed and implemented [9,10,12,13]. Our purpose here is not to list them all, but rather to demonstrate the utility of implementing constraints for VE navigation tasks. A successful navigation method should offer enough constraint to avoid user disorientation and to simulate physical walking, but should also be flexible for special user movement needs.

User Commands

The second universal task that we consider for virtual environments is the issuance of commands by the user. Anyone familiar with desktop computing environments will recognize command-lines and pull-down menus as two different methods of issuing commands. In this section, we would like to explore ways of specifying commands to the system while working in an immersive VE.

Some would dispute the statement that user commands are a necessary task for VEs. In fact, many have speculated that the next generation of user interfaces, not just for VR but for all computing, will be characterized as **non-command** user interfaces [14,15]. However, we do not believe that the need for user commands, in the traditional sense, will be completely eliminated.

As we have stated, not all tasks are suited for direct manipulation or for gestures. If a completely abstract, symbolic task is accomplished in one of these ways, the manipulation or gesture, in the end, is simply another command with its own syntax (not to mention that affordances and mappings will be either non-existent or negative). So, we claim that a need will always exist for appropriate command issuance techniques. Here we discuss two common methods and a hybrid technique, using the constraints principle as a guide.

A common belief is that interfaces to VE applications should be as natural (like the physical world) as possible. It is assumed that the most realistic system will also be the most usable and useful system. Even though commands seem inherently unnatural, there are physical-world parallels. Since most of these involve giving orders to others using our voices, it is not surprising that one popular method of issuing commands in a VE is through **voice-recognition**.

As we saw in an earlier section, though, voice systems suffer from a lack of constraints and affordances: there is too much freedom, and nothing tells the user what commands are valid.

In an attempt to solve this and other problems with voice, some VE developers, just as their counterparts in GUI development did, turned to menu-based commands [5,16]. Although virtual menus (as in figure 2) have been implemented in many different ways, they all provide better constraints: the number of choices is limited, and the system can distinguish the chosen command with complete accuracy. Because of this, novice or occasional users can more easily issue appropriate commands.

Experience has shown virtual menus to be both effective and efficient; however, menus will never be as efficient as speech, because they are more indirect. Also, pointing in a 3D environment is still difficult, because of its lack of constraints. We need a technique that provides the help and constraint needed by a novice, while allowing the expert to issue commands as quickly as she can think of them.

This was accomplished in desktop systems, of course, through the use of keyboard shortcuts, which allow the user to press a key combination to issue a command normally accessible only through a menu. A similar fix has been implemented by Darken [6], in which voice-recognition is used for

all commands, but the commands are organized into hierarchical menus as well. The novice user does not have to remember the names of all the commands: instead he navigates through currently available menus and submenus by speaking their names, which are continually floating just in front of him in the 3D environment. Thus, constraint is maintained while efficiency is increased (through the use of speech). Also, 3D pointing is no longer necessary for menu selection.

For experts, Darken's system allows direct invocation of a command without navigating through the menus, which is analogous to keyboard shortcuts. Also, the menu titles can be turned off altogether, for users who are intimately familiar with system commands.

This hybrid scheme exhibits the important principle of fitting the interaction method to the task. Since commands can be seen as a one-dimensional operation, a 1D input technique (voice) is used. The combination of this with the other constraints greatly increases the usability and efficiency of command issuance.

Object Selection

A third universal task in most virtual environment applications is the selection of objects in the virtual world. Because of its nature, immersive VR is used most often for the display and manipulation of 3D models that have some analog to real-world objects (as far as we know, there are no immersive word processing applications!). The objects may be direct representations of parts of the physical world at a one-to-one scale (architectural models), physical objects that cannot be directly experienced (molecular structures), or depictions of some abstract items which nevertheless are represented as 3D objects (information visualization).

In any case, immersive 3D environments all contain such objects, and unless we only wish to view them, our systems must provide some method for selection. Some of the most common uses of object selection include the specification of an object to which we wish to apply some command (the "noun" in the noun-verb model), the grouping of various related objects, or the beginning of an object manipulation (the object we wish to move).

The most obvious and trivial technique for object selection is simply to select an object when the user's hand comes into contact with it. For example, if the user is faced with a virtual control panel, she can simply move her hand to the buttons or levers to activate them. This method works well in situations such as this because it is natural for the user to use her hand in the same way that she would in the physical world to press a button, pick up an object, etc.

However, considering constraints, this method breaks down in more complex environments. First, the selection "device" (the hand), is not precise enough for the differentiation of small and/or densely crowded objects. Secondly, the user must physically navigate to distant objects in order to select them, which is an inappropriate constraint. A more general technique, then, is needed.

Extending the desktop "point-and-click" idea to a VE, several applications use **ray-casting** for object selection [4,5], including CDS. In this scheme, a ray is extended from the user's hand into the environment, and the first object intersected by the ray is selected (see figures 1 and 2 for examples of ray-casting in CDS). This eliminates the problem of precision, since the intersection is a single point, and the problem of distant objects, since any object can be selected from any location, as long as the user can point to it.

If the user cannot position the ray accurately enough to intersect the object, due to poor visual resolution or tracker noise, then distance still poses a difficulty with ray-casting. Thus, some researchers have suggested **cone-casting**, in which objects within a narrow cone cast from the user's hand are selected. Because the width of the cone expands as the distance from the hand increases, far-away objects are as easy to select as nearby ones.

In this case, then, the *relaxation of constraints* which are too stringent leads to more usable interfaces. It is important to develop, through experience or guidelines, the proper levels of constraint and freedom for the given task.

Object Manipulation

Finally, we wish to consider constraints for the task of object manipulation. As we stated above, practically all immersive VE applications have as their focus a collection of 3D objects. Once a selection method is determined, techniques must be chosen to move, transform, or otherwise modify the selected object(s). This is, of course, imperative for any VE system in which "real work" will be performed.

There are basically two choices for most object manipulation tasks, which we touched upon earlier when describing the task of object movement. These are direct manipulation and indirect, command-driven manipulation. For certain tasks, only one of these schemes will be usable. For others, both may be possibilities. In fact, these represent the two fundamental interaction paradigms available to designers of Virtual User Interfaces (VUIs).

As we have stated, the interaction method must fit the task at hand. Since object manipulation tasks vary so widely (consider scaling an object, changing its color, and adding it to a group), it is impossible to say that one of these paradigms solves object manipulation problems better than the other. We can, however, make some general comments about constraining various object manipulation tasks.

The most obvious task in this category is object movement (as well as other types of object transformations, which are similar in their need for constraints). We have already discussed this problem in the previous section on object constraints, so let us simply reiterate that it is imperative to constrain object motion for tasks requiring any degree of precision, and that various degrees of constraints are often helpful in resolving the tradeoff between efficiency and precision.

Object creation is another important manipulation task. Constraining the creation of objects is perhaps more important than constraining their transformation. To see what we mean by this, consider an implementation where 3D objects are created by sweeping out curves in space with a six degree-of-freedom tracker. This unconstrained technique will certainly result in inaccurate and unusable objects, which will be impossible to work with even if transformation schemes are well-constrained.

We have found in our experience with CDS that in many cases, it is sufficient to allow creation of a fixed set of primitive objects (via a menu command), which can then be transformed and grouped to create more complex designs. For creation tasks which are spatial in nature, such as the specification of a simple architectural building unit, we allow the use of tracker input for creation, but filter that input so that only motion in one or two dimensions is considered (e.g. specification of 2D points on the ground plane for a floor plan, and then 1D specification of building height at each of those vertices).

A third subcategory of manipulation tasks contains those actions which are more symbolic or abstract in nature, such as loading of files, selection of colors, or changes to system parameters. We believe that in almost all cases, such tasks should be constrained to one- or two-dimensional manipulations. If implemented properly, this invariably leads to greater efficiency and overall usability. Such tasks have no need for 3D spatial input, so why should the added burden be placed on the user?

Several good examples of importing 2D interfaces into 3D virtual environments can be found in the literature [16,17,18]. The "paddle" developed at Boeing, for example allows user input to desktop applications on a physical 2D surface (a clipboard), which is held in the user's non-dominant hand. Thus, input is constrained, and users can take advantage of the natural spatial referencing provided by two-handed interaction [11,19].

CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a survey of the use of constraints in user interfaces for immersive virtual environments. Many other issues for the software user interface of VE applications still need to be addressed and researched. We believe that a great deal of work is still to be done in this area.

First, we need a more widespread recognition of the problems of providing usable interfaces for VE systems. Interacting in an immersive 3D world, without the benefit of traditional input devices, is by nature a difficult problem. Certainly, hardware improvements and innovations will improve the situation somewhat. However, more technology alone will not resolve all of the issues. A more systematic implementation of constraints, as well as other software-based solutions, can help bridge the gaps where technology is limited.

Some general guidelines have been presented above, but more are needed. We feel strongly that many UI principles developed for desktop systems also apply to VEs. The guidelines of Donald Norman, on which we focused in this paper, are especially valid because they were drawn from the physical world which many VE applications seek to emulate in some way. However, many principles specific to VR must also be developed and compiled into useful forms for designers of highly interactive systems. The literature is greatly lacking in this area.

Finally, a great need exists for usability testing of interaction techniques themselves, and the complex virtual environment systems which use them. Surprisingly little work has been done in this area. First, evaluation of basic techniques for immersive interaction should be performed. This will provide a quantitative foundation on which we can build more complex systems. Second, we need to enhance or modify existing methods for application usability testing so that it is useful for VE systems. Prototyping techniques should be developed for testing early in the design cycle. We also need more research and experience in the testing of full-blown VE systems, so that our user interfaces can be evaluated on a more quantitative and controlled level.

ACKNOWLEDGMENTS

The authors wish to thank the following people for their work and comments on this research and the CDS system: Brian Wills, Tolek Lesniewski, Harris Dimitropoulos, Jean Wineman, Terry Sargent, Scott O'Brien, Hamish Caldwell, Tom Meyer, Drew Kessler, David Koller, and E.J. Lee. This work was supported in part by the EduTech Institute and the National Science Foundation.

REFERENCES

1. D. Norman, *The Design of Everyday Things*, Doubleday, New York, New York, 1990.
2. G. Smets, P. Stappers, K. Overbeeke, and C. van der Mast, "Designing in Virtual Reality: Implementing Perception-Action Coupling with Affordances," *Proc. VRST*, 1994, pp. 97-110.
3. G. Kessler, L. Hodges, and N. Walker, "Evaluation of a Whole-Hand Input Device," to appear in *ACM Transactions on Computer-Human Interaction*, December 1995.
4. M. Mine, "ISAAC: A Virtual Environment Tool for the Interactive Construction of Virtual Worlds," UNC Chapel Hill Computer Science Technical Report TR95-020.
5. R. Jacoby and S. Ellis, "Using Virtual Menus in a Virtual Environment," *Proc. SPIE, Visual Data Interpretation*, vol. 1668, 1992, pp. 39-48.
6. R. Darken, "Hands-off Interaction with Menus in Virtual Spaces," *Proc. SPIE, Stereoscopic Displays and Virtual Reality Systems*, vol. 2177, 1994, pp. 365-371.
7. J. Bolter, L. Hodges, T. Meyer, and A. Nichols, "Integrating Perceptual and Symbolic Information in VR," *IEEE Computer Graphics and Applications*, vol. 15, no. 4, July 1995, pp. 8-11.
8. S. Card, T. Moran, and A. Newell, "The Keystroke-Level Model for User Performance Time with Interactive Systems," *CACM*, vol. 23, no. 7, July 1980, pp. 398-410.
9. R. Stoakley, M. Conway, and R. Pausch, "Virtual Reality on a WIM: Interactive Worlds in Miniature," *Proc. ACM SIGCHI Human Factors in Computer Systems*, 1995, pp. 265-272.
10. R. Pausch, T. Burnette, D. Brockway, and M. Weiblen, "Navigation and Locomotion in Virtual Worlds via Flight into Hand-Held Miniatures," *Proc. SIGGRAPH 95*, in *Computer Graphics*, 1995, pp. 399-400.
11. J. Goble, K. Hinckley, R. Pausch, J. Snell, and N. Kassell, "Two-Handed Spatial Interface Tools for Neurosurgical Planning," *IEEE Computer*, July 1995, pp. 20-26.
12. K. Fairchild, L. Hai, J. Loo, N. Hern, and L. Serra, "The Heaven and Earth Virtual Reality: Designing Applications for Novice Users," *Proc. IEEE 1993 Symposium on Research Frontiers in Virtual Reality*, October 1993, pp. 47-53.
13. M. Mine, "Virtual Environment Interaction Techniques," UNC Chapel Hill Computer Science Technical Report TR95-018.
14. J. Nielsen, "Noncommand User Interfaces," *Communications of the ACM*, vol. 36, no. 4, April 1993, pp. 83-99.
15. M. Green and R. Jacob, "SIGGRAPH '90 Workshop Report: Software Architectures and Metaphors for Non-WIMP User Interfaces," *Computer Graphics*, vol. 25, no. 3, July 1991, pp. 229-235.
16. M. Ferneau, J. Humphries, "A Gloveless Interface for Interaction in Scientific Visualization Virtual Environments," *Proc. SPIE, Stereoscopic Displays and Virtual Reality Systems*, vol. 2409, 1995, pp. 268-274.
17. I. Angus and H. Sowizral, "Embedding the 2D Interaction Metaphor in a Real 3D Virtual Environment," *Proc. SPIE, Stereoscopic Displays and Virtual Reality Systems*, vol. 2409, 1995, pp. 282-293.
18. H. Sowizral, "Interacting with Virtual Environments Using Augmented Virtual Tools," *Proc. SPIE, Stereoscopic Displays and Virtual Reality Systems*, vol. 2177, 1994, pp. 409-416.
19. K. Hinckley, R. Pausch, J. Goble, and N. Kassell, "A Survey of Design issues in Spatial Input," *Proc. ACM UIST 94 Symposium on User Interface Software and Technology*, 1994, pp. 213-222.
20. W. Robinett and R. Holloway, "Implementation of Flying, Scaling, and Grabbing in Virtual Worlds," *Proceedings 1992 Symposium on Interactive 3D Graphics*, 1992, pp. 197-208.
21. R. Jacoby, M. Ferneau, and J. Humphries, "Gestural Interaction in a Virtual Environment," *Proc. SPIE, Stereoscopic Displays and Virtual Reality Systems*, vol. 2177, 1994, pp. 355-364.
22. R. Bukowski, C. Séquin, "Object Associations: A Simple and Practical Approach to Virtual 3D Manipulation," *Proc. 1995 Symposium on Interactive 3D Graphics*, 1995, pp. 131-138.
23. F. Brooks et al, "Final Technical Report: Walkthrough Project," report to National Science Foundation, June, 1992.
24. D. Bowman, "WiMP Design Tools for Virtual Environments," video proc. of *Virtual Reality Annual International Symposium*, 1995.