Toolsets for the Development of Highly Interactive and Information-Rich Virtual Environments

Doug A. Bowman and Larry F. Hodges

College of Computing

Graphics, Visualization, and Usability Center

Georgia Institute of Technology

801 Atlantic Drive

Atlanta, GA 30332-0280

(404) 894-8787

{bowman,hodges}@cc.gatech.edu

ABSTRACT

This paper describes a unique set of programming toolsets which are designed specifically to aid in the authoring of immersive virtual environment (VE) applications. The first toolset provides a wide range of user interface options through a pen-based metaphor, and benefits applications which have a high degree of interactive complexity. The second toolset allows high-level control of audio annotations, and benefits applications which provide information to the user via the audio modality. The toolsets are designed to provide abstraction from the basic VE software system, to allow rapid prototyping and experimentation, and to be reusable across multiple applications. Two applications which have made extensive use of both toolsets are also presented.

Keywords

Virtual environments, three-dimensional user interfaces, user interface software, pen-based computing, audio.

INTRODUCTION

The development of immersive virtual environments is a difficult and time-consuming task, requiring large amounts of resources in several different areas. First, one must create the models to be used in the virtual world. This includes the specification of polygons and their formation into higher-level objects, the selection of appropriate colors and textures for the models, and consideration of the lighting to be used in the scene. Once the models are created, software must be implemented to allow for interactive, real-time viewing of the virtual environment. Here we must be concerned with the display device, such as a head-mounted display (HMD), the tracking devices, the rendering engine, viewpoint specification, and so on. Finally, the user interface (UI) for the VE application must be produced. This UI must allow the user to perform the tasks required by the application, such as movement through the world (travel [6]), selection and manipulation of virtual objects [5], and system control.

The fulfillment of these three requirements for the development of immersive VEs can benefit from the help of software support tools, and indeed the first two requirements have been the focus of such tools. In the area of modeling, many three-dimensional (3D) modeling packages have been developed which allow the fast creation of polygonal models and the application of color, texture, and lighting properties. There has also been research into the use of immersive modeling [8,19] - that is, modeling within a VE - so that the effects of changes can be seen immediately and interactively. It should also be noted that many virtual environments can be built from libraries of pre-defined 3D shapes, as in the case of a simple home or office walkthrough.

Software that aids the developer in creating a real-time viewing application for the virtual world and in interfacing to various input and output devices (VE system support software) is also available. Products such as WorldToolKit™ [25] and Alice [22] allow the rapid creation and prototyping of VEs with a minimum of programming, and include the ability to communicate with a range of devices commonly used in such applications. There is also research into such system software. For example, the Simple Virtual Environment (SVE) library [17] allows the development of a basic VE, in which the user can look around and move through the virtual world, with only 3 lines of C code. It hides the details of input and output devices, model maintenance, and rendering from the programmer, while still allowing changes to the default behaviors, enabling more complex applications, if the developer so desires.

However, the third requirement - the creation of the user interface to such virtual worlds - has been the focus of few software support efforts. Instead, developers have generally relied on ad hoc techniques to produce the software necessary for users to interact with the system and the objects in the virtual world. For some applications, this is sufficient, as in cases where the user's sense of presence [3] within the VE is the most important factor, and little interaction besides head-tracking and perhaps some travel technique is required. This is true of such applications as virtual exposure therapy [15] or architectural walkthrough [9].

On the other hand, in order to create interactively complex virtual environments, programming toolsets are highly desirable. An application such as immersive modeling (see above) requires techniques for creating new objects, positioning objects, changing the shape of objects, changing the color or texture of objects, setting the viewpoint, setting system parameters, and so on. One can, with a lot of hard work, produce this UI from scratch, without the help of any tools, but this results in interfaces that are difficult to change, that are unlikely to be reusable elsewhere, and that do not engender consistency across applications.

Therefore, high-level toolsets for the specification and development of UIs for immersive virtual environments are necessary. These tools must provide abstraction away from the lower-level graphics or virtual environments software; they should allow rapid prototyping and experimentation; and they should be reusable across multiple applications. Furthermore, if such toolsets are used widely, they can encourage interface consistency across applications.

For projects in our lab, we have developed two such toolsets: one for audio annotations and one for pen-based interaction. Both of these toolsets are accessed at the programming level, but they provide high-level, flexible and generalized support for the creation of a variety of UI components. In the remainder of this paper, we will discuss the implementation and structure of these toolsets, and present two applications which have used the tools extensively.

RELATED WORK

Toolsets to aid in the development of user interfaces are common for 2D graphical user interfaces. They range from lower-level standards such as Motif [12] to object-oriented toolkits like Artkit [14] and systems such as UIDE [26] that attempt to understand something of the intentions of the designer so that models can be created and some portions of the UI may be generated automatically.

For three-dimensional user interfaces, most work is much more unstructured and inconsistent, and this is even more true for immersive VEs. Most of the work has focused on specific interaction techniques, rather than programming tools that aid the developer of virtual worlds. Techniques have been characterized for tasks such as viewpoint motion control [6,20,27] and object manipulation [5,29].

Our stylus toolset addresses the task of *system control*, and several metaphors for system control in VEs have been discussed. The most common idea is that virtual environments should act and respond as does the physical world, so that interaction can be performed in a natural and intuitive manner [e.g. 21]. However, others have seen the need for more abstract system control interactions [24], and have developed virtual pull-down menus [8,16], a virtual "tricorder" (a single unified VE control tool) [28], or voice-controlled menus [10].

The use of 2D input in a 3D virtual environment has also been explored to some degree. The "pen & tablet" metaphor has been used in systems which require both presence within a virtual world and more abstract representations such as maps and text descriptions [2].

Audio annotation systems have also been previously implemented [e.g. 13,18]. These systems have focused, however, on allowing VE users to record and place annotations within a collaborative design or visualization environment. Our toolset, on the other hand, is designed to allow developers to include audio annotations of various types within a 3D space, and to provide techniques with which users may access these annotations.

PEN-BASED INTERACTION TOOLS

One important consideration in interactively complex VE applications is a technique for controlling the system. This may include such tasks as the setting of system mode, the selection of commands, the specification of audio volume, and so on. These types of tasks do not generally lend themselves to a purely 3D implementation, which is the normal type of interaction found in immersive VEs (based on 3D position and orientation trackers). A 6 degree of freedom (DOF) input device is not appropriate for most such tasks, since there are too many unnecessary DOFs, making the task overly complex.

Such tasks, however, are commonplace in traditional 2D graphical user interfaces (GUIs), and 2D techniques using menus, toggle buttons, sliders, and the like are well-understood.

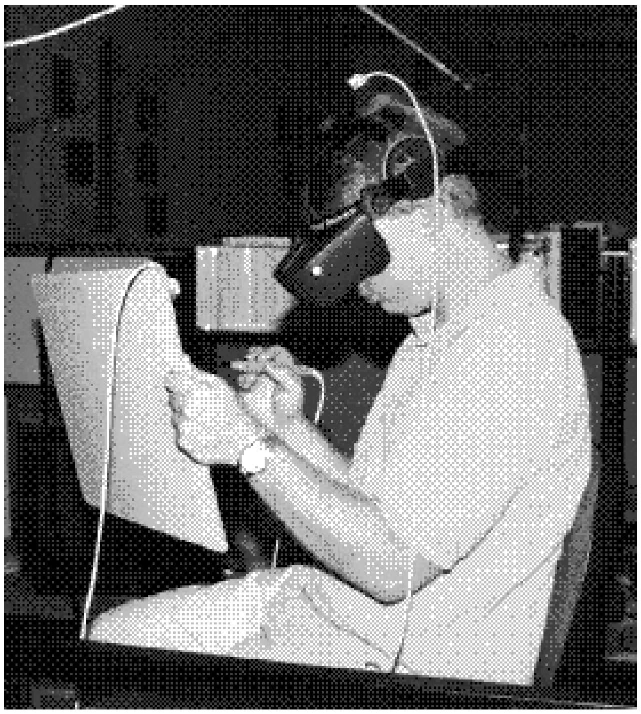
Figure 1. Physical devices used in a pen & tablet interface.

Therefore, we have developed a toolset which allows 2D interaction within a 3D virtual environment, using a "pen & tablet" metaphor. This is based on the stylus tool which is available for use with the Polhemus Fastrak™ electromagnetic tracking system, but can be used with any single-button, tracked input device.

To implement a 2D UI within the VE, one can place interface objects, such as menus, buttons, and sliders, on the surface of a virtual tablet. By combining a physical tablet and a tracker, and attaching the virtual tablet to the physical one, we obtain a 2D user interface that the user can carry with him through the VE (see Figure 1).

Such an implementation has several advantages. First, the user interface is always available to the user, since it is carried in the user's hand. No time is lost searching for the interface when the user wishes to perform an action. Secondly, unlike interface objects that remain attached to the user's view, the UI can be put away if desired, so that the virtual world itself is not obscured. The user has only to place the tablet out of her field of view. Finally, and most importantly, the physical tablet provides physical feedback and a constraint on input. The user knows that interface elements on the tablet cannot be selected unless the pen is touching the physical tablet, and the physical surface can be used as a guide during operations such as dragging an icon. The stylus can be used as a 2D mouse when on the tablet surface, and also for separate 3D operations at other times.

Figure 2 gives an overview of the stylus toolset we have implemented. The toolset is an extension of the SVE library, and uses its routines for model maintenance, callback specification, object intersections, tracking, and device I/O.

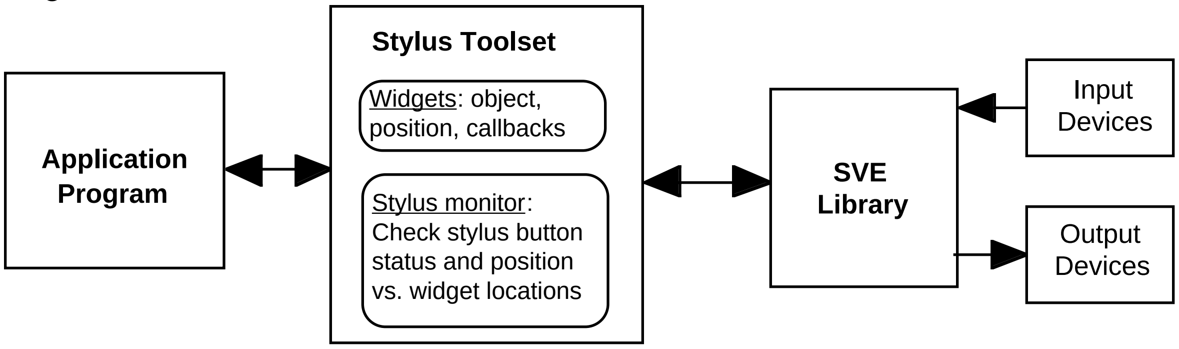
Application Program
Stylus Toolset
Widgets: object, position, callbacks
Stylus monitor: Check stylus button status and position vs. widget locations
SVE Library
Input Devices
Output Devices

Figure 2. Structure of stylus toolset and related components.

From the programmer’s perspective, the toolset is built upon the concept of “widgets,” which refer to any selectable interface elements. These widgets each have an associated precision (how close does the stylus have to come to the widget before it is activated). This precision may be specified as a single value which is used in all three dimensions, or by a separate value for each of the three dimensions. This is useful for pen & tablet implementations, when the (x,y) position of the stylus on the tablet needs greater precision than the z position of the stylus (its depth). Each widget also has associated callback routines, which implement the widget's functionality. Application routines can be called when the stylus enters or exits a widget, or when the stylus button goes up or down within a widget. This matches the general model followed by most 2D UI toolkits.

In the application program, the programmer is responsible for loading objects into the world which will serve as widgets. He must also write the callback routines, which are generally short functions that implement the widget functionality. To activate the stylus toolset, he simply calls an initialization routine and then “registers” each widget, specifying the object to use, the precision of measurement required, and the callback routines. The toolset then manages the operation of the interface via the “stylus monitor” routine, which is called before each frame is drawn to the screen. This routine checks the position of the stylus tip relative to all registered widgets and the status of the stylus button, and then calls any appropriate callbacks.

The stylus toolset also loads, by default, a visual representation of the stylus (a model of the Polhemus stylus device) into the VE. The model provides visual feedback as to the absolute position and orientation of the stylus at all times, as well as its relative position with respect to interface widgets.

From this simple structure, practically any 2D user interface can be constructed within a 3D VE, including those with draggable objects. We have implemented simple menus, toggle switches, and icons which can be dragged to new locations. Pull-down menus, 1D or 2D sliders, and drawing tools are some of the other possible interface elements which could be produced.

Many novel 3D interfaces can be created as well, although these will generally lack the physical feedback afforded by the tablet in our 2D interface. For example, a simple object manipulation technique could be created by specifying all movable objects as widgets, and writing a callback routine which attaches the object to the stylus when the button goes down within the object, and detaches it when the button is released.

The stylus toolset also contains two important utility routines. The calibration utility allows the user to interactively calibrate the position of widgets so that they work in the desired manner (e.g. so that the physical and virtual tablet surfaces are aligned). Each widget is highlighted in turn, and the user presses the stylus button at the desired location of that widget. This is important because of inaccuracies in tracking devices which might cause a widget's position to fall elsewhere than on the surface of the physical tablet.

Finally, a routine is included that returns the relative position of the stylus tip to the origin of a widget. This can be useful for interfaces such as image maps, where a single texture serves as an interface, and the callback function is determined by the position selected within the image. This utility may also be used in three dimensions. For example, a cube representing RGB color space could be defined as a widget, with the selected color depending on the relative location selected within the cube.

Because of their simplicity, generality and flexibility, the stylus tools allow designers to focus on the visual appearance of the interface and the core functionality, rather than on low-level processing of events.

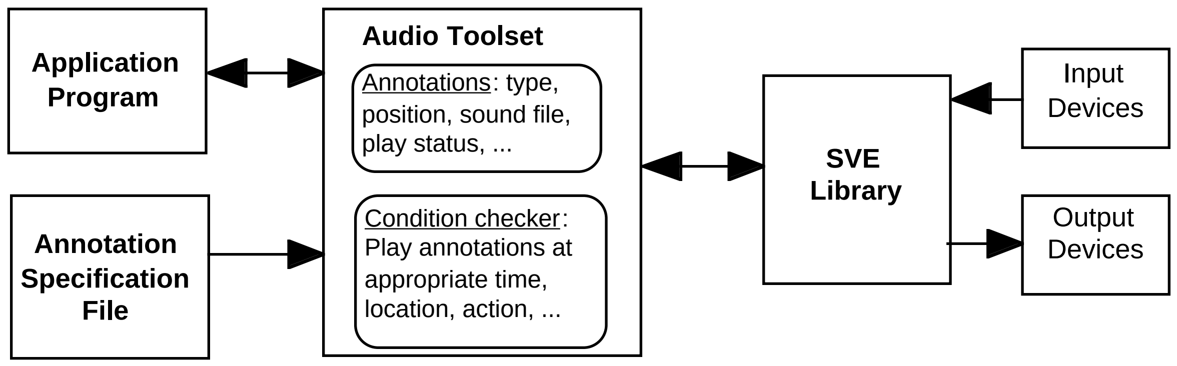
Application Program
Annotation Specification File
Audio Toolset
Annotations: type, position, sound file, play status, ...
Condition checker: Play annotations at appropriate time, location, action, ...
SVE Library
Input Devices
Output Devices

Figure 3. Structure of audio annotation toolset and related components.

AUDIO ANNOTATION TOOLS

There have been a relatively small number of applications of immersive virtual environments which have been found useful and usable for real-world tasks. Applications such as virtual exposure therapy [15] and architectural walkthrough [9] are well-known and have received practical use outside the laboratory. Another application category which is gradually gaining acceptance is the use of VEs as *information spaces*, for educational purposes [e.g. 7,11].

VEs with embedded information move beyond the typical paradigm of presenting a virtual world experience by itself; rather, they augment the perceptual information of the virtual world (objects, environmental sound, etc.) with abstract or symbolic information [4]. This information may be text, audio, images, or animation, and may be used to explain features of the environment, tell why a certain phenomenon has occurred, or give contextual help to the user.

Audio is perhaps the most important modality in which abstract information can be presented in VEs. Unlike the desktop, where textual information dominates, immersive VE displays generally have low resolution, making small text unreadable, while text large enough to be legible obscures the user's field of view.

Furthermore, there is the problem of determining where text should be displayed, since the familiar flat desktop environment does not exist. Image information can be extremely useful in certain situations, but images almost always need some further explanation to be completely understood. Animation can also be a powerful tool for dynamic information presentation, but many types of information are not appropriate for animation.

Audio, on the other hand, is ideal for many types of information presentation in VEs. It does not obscure the display, allows parallel information gathering (visual information from the environment and abstract audio information), and can present a richer set of cues than written information (e.g. tone of voice, speed of speaking). The fact that audio information is time-based and linear in nature may be a disadvantage, but there are many cases in which the message may be quite short, such as identifying the names of buildings, and the user can continue to look and move around during longer audio segments. Therefore, because we perceived a need for a robust and general set of tools for the presentation of audio in immersive VEs, our second toolset has focused on audio annotations. Figure 3 gives an overview of this toolset, which also relies on the SVE library. The basic construct in this toolset is an *annotation*, which refers to any informational audio that is to be presented within the VE. Unlike previous tools, which focused on the creation of annotations by users of the system, our toolset is geared toward the system developer.

The most important feature of our audio annotation toolset is its support for many different annotation types. The simplest type is an audio clip that plays continuously in the environment. There are also several types that play automatically when certain conditions are met. These annotation categories are similar to those in the Virtual Reality Modeling Language (VRML) 2.0 specification, but our system is designed for use with immersive virtual environments (those viewed using head-tracking and a head-mounted display (HMD) or spatially immersive display (SID)).

First, an annotation may be set to play when a given object (including the user) comes within a certain radius or enters a specified bounding box of another object or any 3D point. This might be used, for example, to inform the user that he was approaching an interesting location. Secondly, annotations may be based upon the user's point of view - an annotation plays when the user looks in a given direction or at a specified object. This allows the developer to indicate points of interest without requiring the user to go exploring first. Third, annotations may be set to play when the user passes through a given planar surface in a certain direction. This is useful to indicate state changes, such as the transition from the inside of a building to the outdoors. Fourth, annotations may be triggered by a specific event generated within the VE application. Such annotations are often used when the audio needs to play at a certain time in a simulation or due to some user action. Finally, object annotations play when the user selects a specified object in the environment. Object annotations are generally used when the information relates directly to a specific object, such as the name of the artist of a painting.

For this last type of annotation, the toolset provides a default interaction technique for object selection called ray-casting [20]. A virtual light ray extends from the user's hand when a button is pressed, and objects with annotations attached to them highlight when intersected by the ray. If the button is released while an object is highlighted, the object is selected and the annotation is played. Although ray-casting is known to be sub-optimal for object manipulation, it is nearly ideal for object selection tasks, since it allows the user to select any object that can be seen, no matter its distance from the user [5].

The toolset also allows the user to specify that a default annotation icon (see Figure 4) should be used instead of an object already in the environment. This is often desirable, because the user can be certain that the icon contains an annotation, while annotations attached to objects in the virtual world may be difficult to find.

Specification of audio annotations is extremely simple and flexible. The developer simply gathers the desired sound files (we use AIFF format) and creates a simple text file listing each of the annotations and their characteristics. For example, an entry for a positional annotation contains a 3D location and a radius, while an object annotation entry lists the name of the object to which the annotation is tied. In the application program, the programmer is only required to call an initialization routine for the audio toolset. This allows rapid prototyping of audio information spaces, since the text file can be changed without recompiling the program. All annotations can be selectively enabled or disabled, and all annotation attributes can be changed at any time during application execution. The toolset continually monitors the positions, objects, and events that have been specified and plays annotations at the appropriate times.

APPLICATIONS

Both the stylus and audio annotation toolsets have been used extensively in two major projects in our laboratory: the Virtual Venue [7] and the Virtual Reality Gorilla Exhibit [1]. The applications were diverse enough to test the range of functionality offered by the toolsets, and development was aided a great deal due to this high-level functionality.

The Virtual Venue

The Virtual Venue was our first attempt at what we have termed an “information-rich” virtual environment. Within a virtual model of the Georgia Tech Aquatic Center, we embedded many different types of perceptual and symbolic information so that the user not only experienced the feeling of being within the space, but also obtained information about the building and the sports of swimming and diving.

Both the stylus and audio annotation toolsets were developed in order to facilitate the development of the Virtual Venue. It was clear from the beginning that audio annotations would play a major role in the presentation of abstract information. However, we did not anticipate the need for the user to issue a large number of commands to the system. When we found that such an ability was necessary, the stylus toolset was developed in order to allow the creation of a virtual menu system.

Audio Annotations in the Virtual Venue

Audio annotations were used for three major purposes in the Virtual Venue. First, object annotations were used to identify salient features in the Aquatic Center, such as the diving platforms, seating areas, and pools. The ray-casting technique was used to select objects, and the user would then hear the name of the object spoken aloud. Some objects also had related textual information, which would be displayed simultaneously. Second, 3D location annotations were used to present environmental sounds at appropriate positions. A cheering crowd sound was played in the seating area, and the sound of splashing water could be heard in the pool. Finally, annotations were used to provide help to the user. Several annotation icons (see Figure 4) were placed within the environment in appropriate positions. For example, the icon next to the diving platform gave instructions on how to cause the simulated diver to perform a dive. Also, help was given for some menu items, with a brief audio explanation of the effects of the selected command.

The audio annotation toolset allowed rapid development and testing of the audio information content of the Virtual Venue. In particular, the use of the default object selection interaction technique and the default audio icons saved time in programming and allowed the developer to focus on the more important aspects of the environment for end users: the information content.

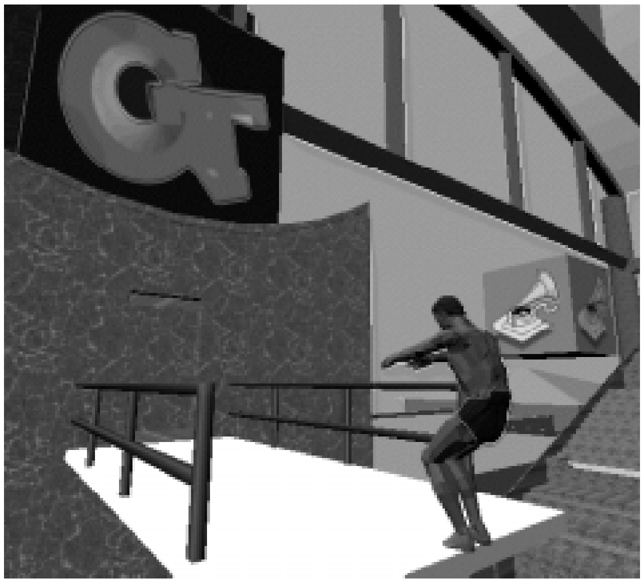

Figure 4. Audio annotation icon used to obtain information about the simulated diver in the Virtual Venue.

Stylus Interaction in the Virtual Venue

As noted above, the stylus toolset was used to implement a virtual menu system for the Virtual Venue. In previous work, we had experimented with virtual menus that were attached to the top of the user's field of view [8]. However, these tended to obscure relevant parts of the environment, and selection of menu items was sometimes difficult. Therefore, we decided to implement a pen & tablet metaphor for the Virtual Venue project. This would allow the menu to be put away for an unobscured view, and would ease selection due to the proximity of the menu and the physical feedback afforded by the tablet. The menu system developed for the Virtual Venue was a simple hierarchical scheme in which only a single level of the menu was visible at any time. Although this might have hindered the user in menu navigation, we chose this technique for simplicity and so that the text labels on menu items could be read easily. Moreover, the menu was only 3 levels deep at the maximum, and 2 levels in most places.

Use of the menu system was quite simple: users pointed to the desired item, using the physical feedback provided by the tablet, and pressed the stylus button. To return to the previous level of the menu, a back arrow was provided. The main menu is shown in Figure 5.

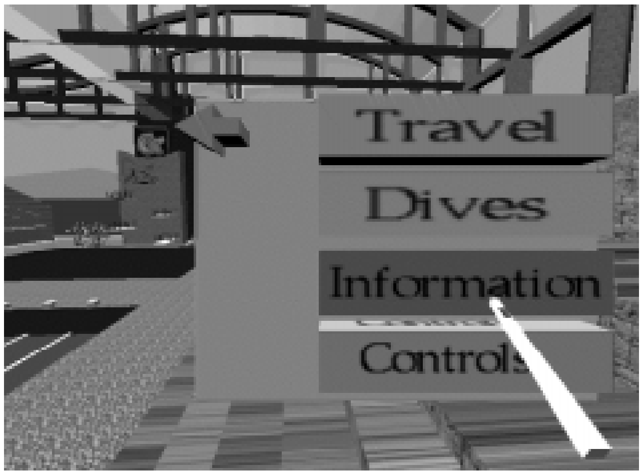

Figure 5. The main menu and the visual representation of the stylus in the Virtual Venue.

The use of the stylus toolset in development of the menu system for the Virtual Venue was beneficial. The callback mechanism allowed separation of functionality into small, easily modified subroutines. Also, the developer did not have to focus on the interface objects themselves, the status of the stylus button, or the position of the stylus in the VE. Again, the focus could be on the core functionality of the system rather than the low-level details of the interface.

The Virtual Reality Gorilla Exhibit

The VR Gorilla Exhibit is actually two separate VE applications, both of which have educational goals, and both of which use the same virtual world: the main gorilla habitat at Zoo Atlanta. The original version is aimed at middle-school students, and teaches them about gorilla behaviors and social structure. The user "becomes" a juvenile gorilla, the lowest member of the family group hierarchy. Students can explore the environment and interact in real time with virtual gorillas in the habitat.

The other version of the VR Gorilla Exhibit has a different purpose and audience: it is intended to teach students of architecture about the design of zoo habitats in general and this gorilla habitat in particular. Users can explore the environment in a more unconstrained way than in the middle-school version; they can obtain information about the habitat and its design; and they can interactively change features of the habitat, such as the position of trees, the location and orientation of visitor viewpoints, and the slope of portions of the terrain.

The audio annotation tools have been used in both versions of this system, and the stylus toolset has been utilized in the development of the zoo design application.

Audio Annotations in the VR Gorilla Exhibit

Both versions of the system require information to be presented at an abstract level. In the middle-school application, students may learn by experience, but they may also be confused or draw incorrect conclusions about the gorilla behavior they have witnessed. Therefore, we have augmented the experience with brief audio clips that explain why the virtual gorillas act the way they do and the appropriate response the student should make. For example, when a gorilla is becoming annoyed, he may begin to “cough” at the offender and avert his gaze. Since this does not correspond to any human experience, an event annotation plays: “Coughing and gaze aversion are signs that this gorilla is becoming annoyed because you are too close to him. You should quickly look away and move away.” Annotations have also been used in this application to give instructions to the students on the use of the input device, to explain the gorilla dominance hierarchy, and to tell users about various parts of the environment when they approach them.

In the zoo design application, audio annotations are used in a similar fashion as in the Virtual Venue. They present relevant information about locations in the environment and the design of parts of the habitat. Both location-based and object annotations have been placed in the gorilla habitat, explaining such features as the moat which surrounds the habitat, the large rocks, and the fallen logs.

In both cases, we did not want the audio information to be repeated over and over, and we did not wish the user to be overwhelmed with sound. The audio annotation toolset allows the programmer to set a flag selectively for each annotation which determines whether the annotation will be played only once, or each time the correct conditions are met. It is also quite possible that two or more annotations might be triggered at once, such as when two gorillas begin to interact with one another. The annotation tools also allow the developer to specify whether more than one audio clip can be played simultaneously. Finally, the zoo design version allows the user to selectively enable or disable annotations by using the stylus to select an annotation icon (see below). Using these techniques, information “clutter” can be reduced, while programmers still retain control and flexibility.

Stylus Interaction in the VR Gorilla Exhibit

The interactive requirements for the zoo design version of the VR Gorilla exhibit are quite extensive. Users need to be able to move about and navigate the space easily, find information in the habitat, position objects such as trees and rocks, change the terrain, and set the visitor viewpoints. A menu system such as the one used in the Virtual Venue might work, but would likely be confusing and difficult to use due to the complexity of interaction.

Therefore, although we are still using the pen & tablet metaphor as before, we have in this case used the stylus toolset to create a single unified interface for all of the user tasks in the zoo design application. The elements that appear on the tablet remain there, and the user manipulates the elements in order to control the system.

A view of the tablet interface is presented in Figure 6. Its main feature is a large map of the gorilla habitat. On this map are presented various icons, notably one representing the user's current location. Other icons represent the locations of audio and text annotations, the locations of trees, rocks, and tufts of grass, the current state of the terrain, and the location of visitor viewpoints. On the right side of the map are four toggles, which turn on or off the display of the icons representing information, movable objects, terrain, and viewpoints.

The toggle buttons, as well as some icons which are not intended to be movable, are implemented exactly like the menu items in the Virtual Venue. For example, the icons representing audio annotations can be clicked to enable or disable the annotation. Icons representing the user, movable objects, and visitor viewpoints, on the other hand, can be dragged to new locations on the map, which causes the item they represent to be moved as well. The user icon is special, in that the user's viewpoint is not moved as the icon is dragged, but is animated smoothly from the previous location to the new one after dragging is completed.

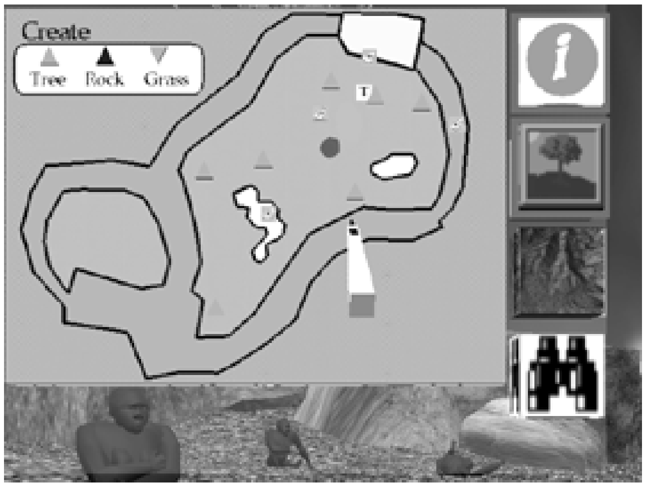

Figure 6. The tablet interface used in the zoo design version of the VR Gorilla Exhibit.

Due to the flexibility and generality of the stylus toolset, development of this complex user interface (compared to other immersive VE applications) progressed quickly. Dragging interactions were simple to implement, due to the availability of button up/down and enter/exit events. Again, the developer could focus on the visual appearance of the interface and the core functionality, rather than on lower-level event processing or object intersection routines.

In this application, the stylus tools were also used for some three-dimensional aspects of the interface. We wanted to allow users to move trees and other objects by direct manipulation of the objects as well as by using the icons on the map. Using the stylus tools, we simply identified the trees, rocks, and tufts of grass as widgets and wrote callbacks that moved the objects along the ground based on the stylus position. In order to help the user reach objects that were far away, we employed the Go-Go technique [23], which maps physical hand position to virtual hand position using a non-linear increasing function.

CONCLUSIONS AND FUTURE WORK

In this paper, we have presented two programmer-level toolsets that aid in the development of immersive virtual environment applications. Our focus has been on information-rich virtual environments which also have a large degree of interactive complexity. The audio annotation tools have allowed us to quickly develop two environments with a wide variety of audio being used for informational purposes. Stylus tools have supported the rapid implementation of two different user interfaces based on the pen & tablet metaphor, which have proven robust and usable in user testing (see e.g. [7]). The use of these toolsets and others like them will decrease development time, allow programmers to focus on high-level functionality and content, and promote reusability and consistency across applications.

In the future, we plan to continue development and usage of both of these toolsets. Educational virtual environments are an important new application category, and robust tools for the development and management of audio annotations for these applications are necessary. The pen & tablet metaphor has some very desirable characteristics for immersive VEs, and our stylus toolset should allow us to explore many different variants of this metaphor quickly to determine the most efficient and usable combinations.

Furthermore, we plan to develop and test novel 3D interfaces using the stylus toolset. Although these will lack the physical feedback afforded by the tablet, such interfaces may be appropriate and efficient for tasks which inherently require three dimensions.

ACKNOWLEDGMENTS

The authors would like to thank Don Allison and Jean Wineman for their help and suggestions on the use of the toolsets for the Virtual Reality Gorilla Exhibit. This work was supported in part by the National Science Foundation.

REFERENCES

1. Allison, D., Hodges, L., and Wineman, J. Gorillas in the Bits. In *Proceedings of the Virtual Reality Annual International Symposium*, 1997, 69-76.

2. Angus, I., and Sowizral, H. Embedding the 2D Interaction Metaphor in a Real 3D Virtual Environment. In *Proceedings of SPIE, Stereoscopic Displays and Virtual Reality Systems*, *2409*, 282-293.

3. Barfield, W., Zeltzer, D., Sheridan, T., and Slater, M. Presence and Performance Within Virtual Environments. In Barfield, W. and Furness, T. (Eds.) *Virtual Environments and Advanced Interface Design*. Oxford University Press, 1995.

4. Bolter, J., Hodges, L., Meyer. T., and Nichols, A. Integrating Perceptual and Symbolic Information in VR. *IEEE Computer Graphics and Applications*, *15*(4), 8-11.

5. Bowman, D. and Hodges, L. An Evaluation of Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments. In *Proceedings of the Symposium on Interactive 3D Graphics*, 1997, 35-38.

6. Bowman, D., Koller, D., and Hodges, L. Travel in Immersive Virtual Environments: An Evaluation of Viewpoint Motion Control Techniques. In *Proceedings of the Virtual Reality Annual International Symposium*, 1997, 45-52.

7. Bowman, D., Hodges, L., and Bolter, J. The Virtual Venue: User-Computer Interaction in Information-Rich Virtual Environments. Graphics, Visualization, and Usability Center Technical Report GIT-GVU-96-22.

8. Bowman, D. Conceptual Design Space - Beyond Walk-through to Immersive Design. In Bertol, D. *Designing Digital Space*. John Wiley & Sons, New York, 1996, 225-236.

9. Brooks, F. Walkthrough Project: Final Technical Report to National Science Foundation. University of North Carolina Computer Science Technical Report TR92-026, 1992.

10. Darken, R. Hands-off Interaction with Menus in Virtual Spaces. In *Proceedings of SPIE*, *2177*, 1994, 365-371.

11. Dede, C., Salzman, M., and Loftin, R. ScienceSpace: Virtual Realities for Learning Complex and Abstract Scientific Concepts. In *Proceedings of the Virtual Reality Annual International Symposium*, 1996, 246-252.

12. Ferguson, P. and Brennan, D. *Motif Reference Manual*. O'Reilly and Associates, 1993.

13. Harmon, R., Patterson, W., Ribarsky, W., and Bolter, J. The Virtual Annotation System. In *Proceedings of the Virtual Reality Annual International Symposium*, 1996, 239-245.

14. Henry, T., Hudson, S., and Newell, G. Integrating Gesture and Snapping into a User Interface Toolkit. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1990.

15. Hodges, L., Rothbaum, B., Kooper, R., Opdyke, D., Meyer, T., North, M., de Graff, J., and Williford, J. Virtual Environments for Treating the Fear of Heights. *IEEE Computer*, *28*(7), 1995, 27-34.

16. Jacoby, R., and Ellis, S. Using Virtual Menus in a Virtual Environment. In *Proceedings of SPIE, Visual Data Interpretation*, *1668*, 1992, 39-48.

17. Kessler, D., Kooper, R., Verlinden, J, and Hodges, L. The Simple Virtual Environment Library Version 1.4 User's Guide. Graphics, Visualization, and Usability Center Technical Report GIT-GVU-94-34, 1994.

18. Loughlin, M., and Hughes, J. An Annotation System for 3D Fluid Flow Visualization. In *Proceedings of IEEE Visualization*, 1994, 273-279.

19. Mine, M. ISAAC: A Virtual Environment Tool for the Interactive Construction of Virtual Worlds. University of North Carolina Computer Science Technical Report TR95-020, 1995.

20. Mine, M. Virtual Environment Interaction Techniques. UNC Chapel Hill Computer Science Technical Report TR95-018, 1995.

21. Nielsen, J. Noncommand User Interfaces. *Communications of the ACM*, *36*(4), April 1993, 83-99.

22. Pausch, R., Burnette, T., Capehart, A., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, G., Koga, S., and White, J. Alice: A Rapid Prototyping System for 3D Graphics. *IEEE Computer Graphics and Applications*, *15*(3), May 1995, 8-11.

23. Poupyrev, I., Billinghurst, M., Weghorst, S., and Ichikawa, T. The Go-Go Interaction Technique: Non-linear Mapping for Direct Manipulation in VR. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1996, 79-80.

24. Rygol, M., Dorbie, A., Worden, A., Ghee, S., Grimsdale, C., and Harvey, J. Tools and Metaphors for User Interaction in Virtual Environments. In Earnshaw, R., Vince, J., and Jones, H. (Eds.), *Virtual Reality Applications*, Academic Press, 1995, 149-161.

25. Sense8 Corporation. WorldToolKit. Available at http://www.sense8.com/products/wtk.html.

26. Sukaviriya, P., Foley, J., and Griffith, T. A Second Generation User Interface Design Environment: The Model and the Runtime Architecture. In *Proceedings of INTERCHI*, 1993, 375-382.

27. Ware, C. and Osborne, S. Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. In *Proceedings of the Symposium on Interactive 3D Graphics*, in *Computer Graphics*, *24*(2), 1990, 175-183.

28. Wloka, M. and Greenfield, E. The Virtual Tricorder: A Unified Interface for Virtual Reality. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1995, 39-40.

29. Zhai, S. and Milgram, P. Human Performance Evaluation of Manipulation Schemes in Virtual Environments. In *Proceedings of the Virtual Reality Annual International Symposium*, 1993, 155-161.