

Interacting with Scientific Visualizations: User-Interface Tools within Spatially Immersive Displays

John Bresnahan¹, Joseph Insley^{1,5}, and Michael E. Papka^{2,3,4}

¹ Distributed Systems Laboratory, Mathematics and Computer Science Division,
Argonne National Laboratory, Argonne, IL 60439

² Futures Laboratory, Mathematics and Computer Science Division, Argonne
National Laboratory, Argonne, IL 60439

³ Computation Institute, University of Chicago, Chicago, IL 60637

⁴ Department of Computer Science, University of Chicago, IL 60637

⁵ Department of Electrical Engineering and Computer Science, University of
Illinois at Chicago, Chicago, IL 60612

Abstract. User interfaces represent a doorknob to computer applications and directly relate to the usability of a given application. The desktop WIMP interface has been developed and refined over the past 15 years and works well for the desktop environment. The question is, What does one do in the case of today's three-dimensional immersive environments? Does one translate the desktop tools to the 3D space, and how does one interact within these spaces? In building numerous immersive scientific visualization applications, we have developed a variety of different user interface tools for use within these environments. These interfaces include methods of controlling and driving the applications, tools for navigating datasets, and underlying libraries for the development of collaborative versions of these tools. With the development of these tools, we have learned many lessons, as well as gained new insight into future user interface work.

1 Introduction

The benefits of scientific visualization to the analysis and understanding process is nothing new. Scientists have embraced its use in order to gain meaningful insight into the data that they collect, generate, and compute. As the compute resources scientists use continue to improve, the complexity of the data that they are trying to interpret increases. This situation has sparked a new wave of data analysis that moves scientific visualization off the desktop and into immersive virtual environments. What scientists need now is an intuitive way to interact with their data within these virtual environments.

Depending on the data being visualized and the type of visualization being performed, there are myriad different aspects of the visualization that one would like to be able to control. These range from changing the position and orientation of a volume of data that is being explored to adjusting the parameters of an on-going simulation whose output is being rendered in real time. One would also like to be able to control all of these aspects from within

the virtual environment, without having to return to the desktop to type in some command on the keyboard.

One solution is to borrow the familiar 2D menuing scenario from the desktop. But 2D menus don't always transition easily into 3D. Because the resolution of many virtual reality systems is somewhat limited, text-based menus are often difficult to read. In addition, it is often necessary to navigate through several levels of menus to change the modality of the task that is being performed. This process can be tedious and can make the system difficult to use. The problem can be alleviated somewhat by having multiple menus open at once, but that can lead to clutter, where the menus start to obscure one's view of the data that is being visualized. Furthermore, it is simply unnatural to use 2D menus to interact with a 3D world, whether that world is real or simulated.

As with any user interface, it is important to provide visual cues and user feedback on the state of the system. Such cues may involve highlighting an object to let users know that it has been selected or changing its color to indicate that its state has changed. These cues let users know how the system has interpreted their actions, provide a better understanding of how those actions are affecting the visualization, and enable users to make informed decisions about how to proceed.

Our experiences have been with developing scientific visualization applications primarily for the CAVE and ImmersaDesk [3]. Both are projection-based virtual reality (VR) devices that use a pair of tracked LCD shutter glasses to provide a stereo, user-centered perspective. Users interact with the environment using a tracked 3D mouse-like input device called a wand that has three buttons and a joystick. We describe here our experiences with some of the tools that we have developed for interacting with scientific visualizations in these virtual environments. In particular, we discuss a control panel interface and data selection/navigation tool, a 2D-based menuing system, and an infrastructure for connecting these tools in a collaborative session. We briefly describe the each tool, discuss some lessons learned, and present ideas on where we see the tool heading. We conclude with an overall discussion of the future of 3D user interfaces.

2 ControlPanel

In order to control different aspects of a visualization from within a virtual environment, it is often desirable to use valuators, such as sliders, for changing the value of some parameter, or a variety of buttons and toggles to change modalities or to trigger certain events. As more control over a greater number of aspects of the visualization is added, however, managing the placement of all of these controls often becomes problematic. If individual components are left floating freely in the space, the space quickly becomes cluttered and unmanageable. To help alleviate this problem, we developed the ControlPanel



Fig. 1. Image of user interacting with the CP on a Idesk.

(CP), which is a central location where all of these controls can be kept. Originally designed for the ImmersaDesk (Idesk), which is a single-screen, drafting table-sized virtual reality device, the CP sits at the bottom edge of the screen. Much like on the dashboard of a car, all of the controls are now easily accessible, yet out of the way so as to not obscure the user's view of the rest of the environment (See Figure 1). Similar to windows in a desktop environment, the CP can also be minimized to hide the controls and provide an even better view of the scene.

The ControlPanel library is implemented by using OpenInventor. The CP provides a number of generic widgets. Interaction with these widgets is managed by the CP, which uses ray-casting from a ray at the end of the wand to determine whether an object has been intersected. A pointer, which coincides with the ray, is drawn at the end of the wand to make the selection of objects easier. When an object is intersected, it changes color, to indicate that it can now be selected. The object can then be selected by pressing one of the buttons on the wand, which causes the object to change color again, letting the user know that it is now active. All of these widgets are user programmable. That is, the application developer can specify value ranges, indicate how the behavior of some widgets is affected by the value of others, and so on. Also, the widgets remain independent of the other objects in the

scene. Once the ControlPanel has been updated, the developer can get the values of the various widgets from the CP and use them to determine how or whether the rest of the scene should be altered. This approach keeps the CP somewhat generic, while giving the programmer the desired control.

Since providing users with feedback about what they are communicating to the system is so important, the CP also has a joystick object that represents the joystick on the CAVE wand. When the user pushes forward, back, left, or right on the wand's joystick, the joystick on the CP moves in the same way, to reflect the user input. If the joystick's values are not being used to control anything, then the joystick on the CP remains in the default position.

In addition to the widgets provided by the ControlPanel library, developers can add their own custom objects to the CP. Typically, one simply provides the CP with methods to get the object's root node (OpenInventor), so that it can be added to the CP's scene graph for updating it and for getting and setting its values.

2.1 Widgets

Among the widgets that the ControlPanel provides are sliders, buttons, text objects, and scrolling lists. Figure 2 shows a view of the CP.

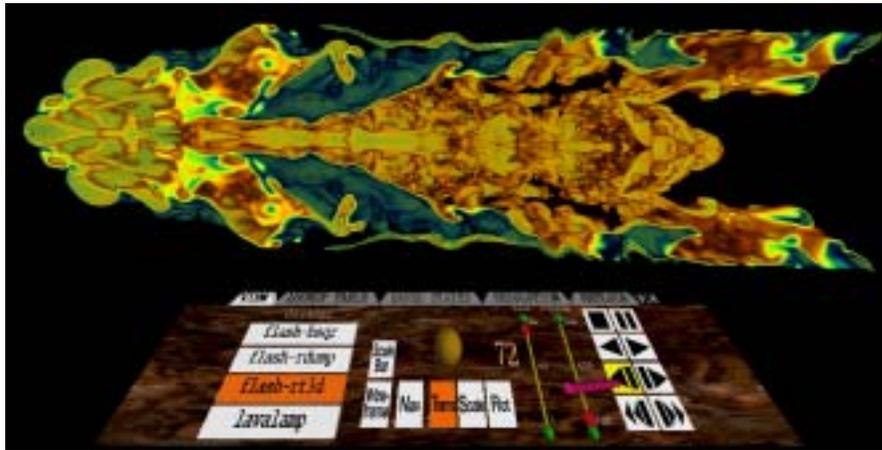


Fig. 2. Snapshot of the CP, taken from the CAVE simulator.

Sliders The sliders can be a single slider or a group of them. Their minimum and maximum values are set by the programmer, as well as whether their value should be an integer or a floating-point number.

Buttons Buttons can also be added to the CP individually or a group, and they have variety of modalities. Within a group of buttons, each button may be turned on and off independently of the others. Alternatively, it is possible to allow at most one button to be active at a time, so that when a button is activated, if another button was already active, it becomes deactivated. A third modality is much like that of a radial button, where exactly one button is active at all times; in this case, the currently active button can be deactivated only by selecting one of the other buttons.

Text Objects The text object is just a wrapper around OpenInventor's 3D text object that makes it easier to add text to the ControlPanel. As mentioned earlier, text is often difficult to read in many virtual reality systems, and the Idesk is no exception, but it is also often necessary. One needs to be careful when adding text to ensure that it is large enough to read, but that it is kept short enough to avoid cluttering up the space. Text on the ControlPanel is generally fairly easy to read because it is close to the user and on an opaque background, which helps separate it from the rest of the environment.

Scrolling List The scrolling list is a combination of the other three objects. It allows the user to scroll through a list of text strings, such as a list of available datasets that one might want to visualize, and select one of them. The programmer can also dynamically update the list of strings, if he so chooses.

2.2 Widget Management

Because the ControlPanel has a finite amount of space, it is often difficult to fit all of the desired controls on the CP at the same time. There are several ways to address this problem.

Tabs One way is to use the multiple panels available on the CP. Grouping sets of related controls onto different panels alleviates some of the congestion on a single panel and makes it easier to find a desired control. A row of tabs is displayed across the top of the CP, each with a label that describes what the set of controls on that panel does, thereby enabling the user to switch between panels easily. If there is not enough room for all of the tabs to fit across the top of the CP, the last tab becomes a pop-up list that shows all of the available panels. When the user selects the desired panel, its tab is placed at the center of the row of tabs, and as many of the adjacent tabs as will fit are also displayed.

Function Grouping Another way to free up space on a given panel is to use a group of buttons to select what the other widgets are used to control. For

example, in an application in which a group of sliders are used to manipulate the parameters of a color lookup table, there may be a slider for the red, green, blue, and transparency components of the color. For each of those components, the user may want to alter the center, width, and amplitude of a sine curve that is used to calculate the lookup table. Rather than having twelve sliders on the panel simultaneously, the user could have four sliders, one for each color component, and three buttons (which take up considerably less space) that could be used to select the particular aspect of the sine curve that the sliders are currently being used to control.

Similarly, users often want to use the joystick on the wand to control different aspects of the visualization. For instance, when visualizing a volume of data, one may want to adjust the size, position, or orientation of the volume. One may also want to use the wand to navigate to a different position in order to view the data from a new perspective. Buttons on the CP work well: they may be used to select the particular values being controlled by the wand's joystick.

2.3 Lessons Learned

In using the ControlPanel for several different applications and continue to develop it, we've observed that it has many strengths and weaknesses. We have received primarily positive feedback about the usability of the CP. One user commented that it was big and clunky, but that's what he liked about it; it made everything easy to find. In many cases the CP is quite effective in helping to organize the controls by keeping them all centrally located and by forcing users to group controls according to their function. Because of the CP's limited space, however, it was often difficult to fit all of the related controls on the same panel. Some users solved this problem by putting controls in multiple places for easier access.

One of our main observations is that the ControlPanel is better suited for some display devices than others. It works quite well on the ImmersaDesk, for which it was originally designed. It sits along the bottom edge of the screen and is easily accessible. Since the Idesk is a single-screen device, the user doesn't physically move around very much. He may take a step in any direction, but he remains in the same general location, keeping the CP comfortably in front of him. In the CAVE, however, the ControlPanel is less convenient. For one thing, the controls often get obscured by the data being displayed on the three walls and the floor. For another, since the CAVE—a ten-foot cube—allows the user to move around a lot, the user may find himself in front of the ControlPanel, where he can no longer reach its controls. Further, since the CAVE often has several observers in it at once, the ControlPanel can be obscured by one of the other people in the space.

2.4 Future Work

The ControlPanel was intended to be a generic reusable toolkit that application developers could easily customize and drop into their applications. A few developers have succeeded in doing just that, but there is much room for improvement in this area. Also, just as the visible cues are important user feedback, so are audible cues. One possible future direction is to experiment with adding sounds that indicate user interactions with the system.

3 Box-in-Box

One of the goals of visualization is to enable users to gain insight into their data. Today's simulations are generating ever increasing amounts of data, making the task of visualizing the data even more challenging. The larger amounts of data can cause a visualization that used to be interactive to slow to a snail's pace. The larger amounts of data can also overwhelm the user. The box-in-box (BIB) tool was designed to address these problems.

The BIB (see Figure 3) provides simple feedback to users on where they are in their dataset and a reference point to how much of the dataset as a whole they are looking at. The BIB consists of a box within a box. The outer box represents the dataset as a whole. The inner box represents the subset of data the user is currently viewing, and indicates where within the global data space the subset is located. This tool thus provides users with a global context of the dataset size with respect to the size of the subset of data being used.

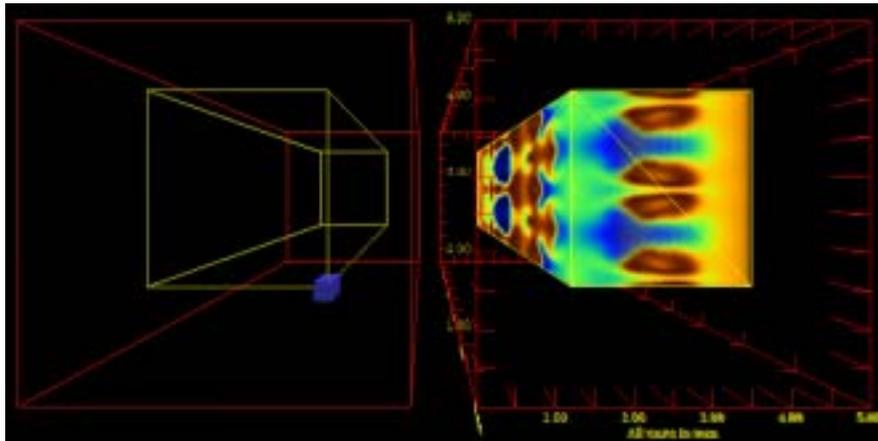


Fig. 3. Example of Box-in-Box tool selecting a subset of data to be volume rendered.

3.1 Interaction

A user can interact with both the inner box and the outer box in an intuitive manner: the intersection of the wand ray and a button press are all that is required to use the BIB. Selection of the outer box allows the tool to be moved within the virtual environment, allowing the user to place the tool in a convenient location and out of the way of the visualization results. Interaction with the inner box happens in two ways. On the one hand, a small box in the lower left-hand corner can be selected to resize the inner box, and in turn determine which data is used in the visualization. On the other hand, the inner box can be selected and moved within the outer box. Thus, the user can navigate the global dataspace while keeping the visible data down to a manageable state, be it for responsive interaction rates or for comprehension.

3.2 Lessons Learned

BIB was originally designed to be used with molecular visualization of structures constructed from millions of atoms. The large number of structures made visualization both slow and overloaded with information. The BIB allowed the user to select regions of interest within the dataset; and, in combination with a sampling selection programmed to the wand's joystick, the user was able to explore the dataset without a loss of interaction.

Since its original use, the BIB has been used to select regions of interest within volumetric datasets, passing the selected data to volume-rendering tools for visualization within the CAVE.

The BIB generally has been well received. On the positive side, it provides a rather straightforward approach to navigation and management of datasets, and most users have been able to grasp the interface relatively quickly.

On the negative side, however, the BIB is somewhat difficult to use within the CAVE simulator, but this is more a problem of interacting with 3D tools with a 2D device. Another criticism of the BIB involves the alignment of the inner box to the outer box: users would like to be able to select a region of the data that is not axis aligned to the outer box.

3.3 Future Work

Addressing the alignment of the inner box to the outer box is one of the highest priorities on our future work list. This should be rather straightforward because the BIB works on the data spatially. The challenge, however, is handing off the newly selected data to the visualization tool of choice and ensuring that that tool can handle the data. We would also like to see the BIB used with a greater variety of applications and application areas. Feedback from a larger community of users will help steer where future development on the tool goes.

4 Shared Controls - For Remote and Collaborative Use

It is often desirable for several people to share the same visualization of scientific data at the same time. By having a simultaneous view of the same dataset, researchers can exchange ideas more easily. Collaborative visualization also allows a variety of people with different expertise to study the results, thereby yielding more insight than a single person could in the same amount of time.

We believe that in addition to seeing the data collaboratively, people should be able to *control* the visualization collaboratively. Specifically, each participating member should have a view of the data and a means to control the visualization parameters (such as position and orientation). However, it is not necessary or even desirable that each participant have the same type of display capabilities or the same look and feel to the control widgets used to manipulate the visualization.

For example, it should be possible to have a collaboration in which one of the users was using an ImmersaDesk and the other participant a CAVE. All they should need to share is the same state information that controls the rendering process. The program running in the CAVE knows how to render a dataset in its environment, just as the program running on the Idesk knows how to render to its display. If a user rotates a volume ninety degrees around the x-axis, both programs need to know that the data has been rotated so that they can display it correctly, but the Idesk need not know that the CAVE has rendered part of it on the front wall and part on the floor.

Further, it should not be necessary to have all the control widgets in the collaboration share any qualities except state information. An valuator may be used to share color ranges from 0 to 255; but that valuator could be visually displayed as a vertical slider, a horizontal slider, or even a textbox where the value is entered. The look and feel of the widget should be irrelevant to the collaboration. The mode of input/interaction should fit the environment the collaborator is working in.

To this end, we developed the `CIF_Shared_State` library, written on top of Nexus, which is the communication component of Globus [5,4]. The library propagates state information to all processes participating in the collaboration [1]. Figure 4 shows a diagram of a two-user collaborative session with one user in a CAVE and one user using a desktop application.

Sun has a Java product similar to the `CIF_Shared_State` library, called the Shared Data Toolkit [2]. Since we needed access to high-end graphics hardware, however, Java was not a complete solution for us. Instead, we created a cross-platform library in both C++ and Java, and they are completely compatible with each other. In this way, we maximized the amount of graphics resources we could use.

The `CIF_Shared_State` library ensures that user-defined states are kept synchronized. A user-defined state can consist of arrays of primitive data types or can comprise combinations of different primitive data types. Such

things as rotation, translation, color mappings, and widget states were synchronized via these states.

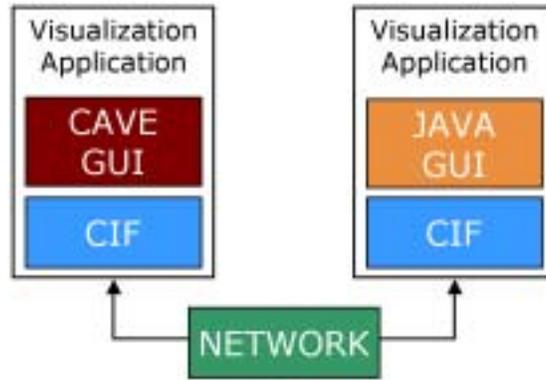


Fig. 4. Diagram of how `CIF_Shared_State` library fits into an application for collaborative or remote use.

Using the `CIF_Shared_State` library, we created an application that allowed the collaborative visualization of a volume-rendered dataset (See Figure 5). The application had a control panel (See Section 2) with several shared widgets, including buttons, sliders, and checkboxes. `CIF_Shared_State` was used to synchronize these widgets. In the case of a shared checkbox, a shared Boolean state was used. When the user checked the visual checkbox, the shared Boolean state was set to true. The `CIF_Shared_State` library then propagated this information to all programs participating in the collaboration by way of an event. When a program received an event telling it that the shared Boolean state was changed to true, it then set its visual representation of the checkbox to true. Similar shared states were used to share information about where the volume was in three-dimensional space and how to visualize it.

4.1 Lessons Learned

By decoupling the visual component from the state information and leaving the display entirely up to the rendering program, we were able to create a collaborative environment that could run on several different display technologies. Further, we were able to remotely control a rendering program. Since the shared-state information was in no way visual, we were able to have one process do nothing but render frames on a high-end graphics machine (Oynx II Infinite Reality Monster). That process provided no user interface for control. Instead, it was controlled by a two-dimensional Java GUI control



Fig. 5. Example of a collaborative application that shows both the immersive application interface and the desktop java based interface.

panel in a separate process running on a separate machine. The Java control panel was run on a machine with essentially no graphics abilities. The remotely rendered frames were then streamed to the machine controlling the visualization. We were therefore able to do reasonable high-end visualization on desktop hardware.

4.2 Future Work

In the `CIF_Shared_State` library, total order was achieved by having one of the programs in the collaboration elect itself a server. The order in which messages got to this server determined the order in which all other clients would see messages. This left the situation that the last message to the server determined the value of the state. Unfortunately, when multiple users tried to set the state at the same time, the client with the slowest connection to the server would make the determination. In the future we would like to implement floor control policies so that control could be arbitrated more fairly.

5 Conclusion

Our experience in building scientific visualization applications for immersive environments has demonstrated the need for user interfaces to allow for maximum flexibility in the application. The user interface plays a major role in determining whether an application will be useful. Experience has also led us to conclude that much more research, development, and field testing are needed on user interfaces with immersive environments. Borrowing from the desktop environment provides a start, but it is unnatural for a three-dimensional space and doesn't fully exploit the benefits of the environment. We have also learned that a given user interface solution for one immersive display device is not always the right answer for another. Finally, we believe

that group analysis of data will become increasingly popular and that user interfaces will have to meet the increasing demands of collaboration modes.

Acknowledgment

We gratefully acknowledge the members of the Distributed Systems Laboratory and the Futures Laboratory within the Mathematics and Computer Science Division of Argonne National Laboratory for help and input. This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Globus research and development is supported by DARPA, DOE, and NSF.

References

1. Bresnahan, J., Foster, I., Insley, J., Toonen, B., Tuecke, S.: *Communication Services for Advanced Network Applications*, **Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications 1999**, Volume IV, 1999, pp. 1861–1867.
2. Burrige, R.: *Java Shared Data Toolkit User Guide*, Sun Microsystems User Guide, Version 1.4, June 1998.
3. Cruz-Neira, C., Sandin, D. J., DeFanti, T. A.: *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*, **Proceedings of SIGGRAPH '93 Annual Conference**, 1993, pp. 135–142.
4. Foster, I., Kesselman, C., Tuecke, S.: *The Nexus Task-Parallel Runtime System*, **Proceedings of the First International Workshop on Parallel Processing**, 1994, pp. 457–462.
5. Foster, I., Kesselman, C.: *Globus: A Metacomputing Infrastructure Toolkit*, **International Journal Supercomputer Applications**, 11(2), 1997, pp. 115–128.