

# **The Virtual Panel Architecture: A 3D Gesture Framework**

S. Augustine Su  
Richard Furuta

Hypermedia Research Lab

JULY 1993  
TAMU-HRL 93-004

***The Virtual Panel Architecture:  
A 3D Gesture Framework***

S. Augustine Su  
Richard Furuta

July 1993  
TAMU-HRL 93-004

# The Virtual Panel Architecture: A 3D Gesture Framework\*

S. Augustine Su  
Computer Science Department  
University of Maryland  
College Park, Maryland 20742  
su@cs.umd.edu

Richard Furuta  
Hypermedia Research Laboratory  
Dept. of Computer Science  
Texas A&M University  
College Station, Texas 77843-3112  
furuta@cs.tamu.edu

## Abstract

A control panel is a commonplace means for controlling devices in the physical world and also for controlling interactive computer applications. Sparked by this metaphor, we designed the framework—*Virtual Panel Architecture* (VPA) to help implement an intermediate abstraction with elements of both the physical panel and the computer-based panel. This abstraction, handling 3-D point-based gesticulative interaction with modeled object hierarchies, is suitable for both Virtual Reality environments and traditional environments. Based on the VPA, we demonstrate an environment of virtual panels, which can be incorporated into keyboard-and-mouse-based computing. Users are able to turn virtual knobs, adjust virtual sliders, and point on virtual screens on these virtual panels in this environment. Therefore, the input interactions are not only multiplied and diversified, but also conceived and implemented easily.

## 1 Introduction

A control panel is a commonplace means for controlling devices in the physical world and also for controlling interactive computer applications. In computer applications, the general familiarity of panel components (knobs, switches, sliders, buttons, etc.) provides an intuitive connection to the physical world.

In this paper we describe a framework designed to help implement an intermediate abstraction with elements of both the physical panel and the computer-based panel. While the computer-based panel represents a powerful abstraction, displaying it on a screen and manipulating it with a mouse-like device removes some of the ease-of-use found in the physical analogue. Particularly when the panel becomes large or when multiple instances of the panel are active simultaneously (perhaps to control multiple devices), the mechanics of mouse positioning and of hiding and displaying different panel instances becomes a major factor in the ability to adjust controls in a timely fashion.

---

\*This paper is based upon work supported by DVP, Inc., in cooperation with the Maryland Industrial Partnerships program, and upon work supported by the National Science Foundation under grant number IRI-9007746.

The framework, the *Virtual Panel Architecture* (VPA), supports the creation of “Virtual Panels”—easily reconfigurable panels that both have a physical manifestation and that also are intended for controlling computer-based processes. Such panels share strong points of both the physical but also the computer-based implementations. As with the physical panel, virtual panels can be arranged to permit use of “muscle memory” in finding and altering the controls. As with the computer-based panel, virtual panels can be device independent and can be easily and flexibly designed and customized.

The VPA supports not only Virtual Panels but more generally many kinds of interfaces based on 3-dimensional gesticulative input. When a hand-tracking device is incorporated into a computer environment, the users gain the freedom to explore their virtual surroundings using gesticulative input. There are at least three approaches to use this freedom: [KL87] and [Fel90] used sign language approaches; [WGW90] and [PEF+90] used object-based manipulations; [Bol80] and [FMHR86] used voice-assisted methods. Because we believe Direct Manipulation [Shn92] is a good paradigm for data manipulation, we have focused on the second approach. However, the VPA still provides enough room to contain other approaches.

The VPA consists of: (1) the Gesture Server, which processes the data from hand tracking device(s), defines “point-based” gestures, and reports the gestures and hand positions, (2) the Panel Server, which maintains a database of objects, decides which objects the gestures are manipulating, and sends out events to filters, (3) filters, which cooperatively abstract information from events to serve applications, where they communicate by events via pipes, and (4) applications. In section 2, we will discuss the underlying concepts of the VPA.

In our current implementation, the gestures and objects in the VPA are simplified to largely relieve the computational cost. We will introduce some interesting virtual devices encapsulated in virtual panels to demonstrate our ideas in section 3.

The VPA is suitable for the front-end processing of Virtual Reality (VR) environments. In addition, it can be incorporated into traditional keyboard-and-mouse-based environments as shown by our demonstration in section 4. We use a VPL DataGlove [ZLCBH87] to implement our first version. Of course, there are some inborn obstacles confronted by this simple VPA, e.g., feedback problems and performance problems. We will discuss possible improvements to relieve these issues. And finally, section 5 will be conclusions.

## 2 Virtual Panel Architecture

### Gesture Server

Our initial studies of 3D-gesture-based interfaces suggest that most of easily remembered gestures are those that are simple and natural, for example, Point and Pick. Modeling such simple gestures would not require great complexity. However, we also wished to retain the flexibility to extend the modeling techniques to represent more complex gestures, which could invoke a richer set of inputs in order to capture the more exacting requirements.

With these goals in mind, we elected to define gestures as the temporal-spatial loci of points of interest, where these points are attached to specified areas of the user’s hands, rather than using the more traditional descriptions based on bending angles of the joints.

According to the definition, the simplest case is the one-point gesture, Point, whose position calls the Pointer. The Pointer can be attached to the tip of a stylus, to the user’s index fingertip, or to the central back of the user’s hand, as with the DataGlove.

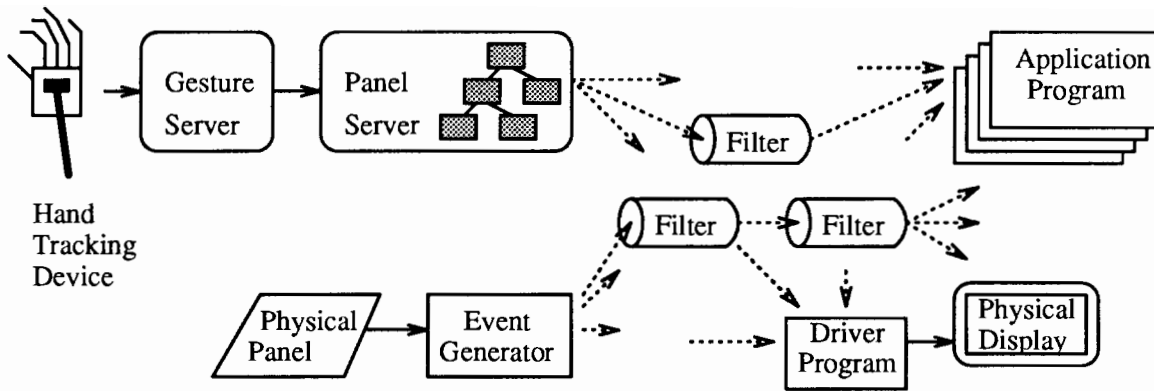


Figure 1: The Virtual Panel Architecture (pipes are indicated as dashed lines.)

A well-known obstacle for the gestures based on a unique tracked point is that they cannot signify status changes naturally (like a mouse button press) while they are traversing in space. Introducing a second tracked point not only overcomes the obstacle, but also allows one to define interesting natural gestures. For instance, if we attach a point to each tip of the index finger and the thumb, then the gesture Pick can be defined when the two points are within a specified distance. The Pick gesture can represent a mouse-button press to select objects. But it is more attractive than the button press because it is more conforming to the cognition of select. Furthermore, if the two points in Pick are 6-D, then various Picks can be defined: Tip Grip, Pinch Grip, and Lateral Grip [KKKE90]. Different Picks can signify different status just as different mouse-button presses.

In general, six 6-D points of interest on one hand are sufficient to describe any static hand shape [Su93]. Therefore, six temporal-spatial loci are sufficient to describe any dynamic gesture of a hand. Partial gestures can be specified by less than six loci, like gestures above.

The Gesture Server is in charge of (1) accepting inputs from hand tracking device, (2) outputting 3-D or 6-D coordinates of some points of interest, and (3) outputting gesture changes, which are recognized by the Server (see figure 1).

### Panel Server

The Panel Server is responsible for (1) maintaining a database of 3-D hierarchical objects, (2) accepting inputs from Gesture Server, and (3) outputting events about the manipulation of objects by point-based gestures after traversing the object hierarchies.

Equipped with tracked gestures, users are able to directly manipulate objects surrounding them at will. For example, the Pointer is able to “enter” objects, to “move” inside objects, “leave” objects, and “pick” objects, similar to that the cursor moves around hierarchical windows in the X window system [SG86]. Each panel is a top-level object. Since the number of intended gestures can be small, the semantics of manipulations of objects by gestures has to mostly rely on the objects. The diversity of meanings of objects is the first design space in the VPA.

### Filter Processing

To gain more flexibility, we defined filters. Each filter is an independent module that receives events via pipes, does any processing that is needed to accomplish data abstraction, and sends out new events via pipes. Since the jobs done here are sometimes very similar, we can provide some

“canned” filters as black boxes for code re-usability. The VPA has provided necessary procedures for sending events, receiving an event, extracting information from a received event, and forming a new event. The second design space in the VPA is due to the filters.

Pipes are the means of communication for events and they do nothing else. Therefore, the functionalities of individual filters and manipulated objects have been separated completely by pipes. The interconnections of pipes among objects, filters and applications are another important design space in the VPA. Finally, applications handle the events from filters and/or objects via pipes.

More nicely, since the modules of the VPA communicate by events only, if we wish to provide additional information about the system’s state and provide specialized user functions we can insert extra physical control panels and special-purpose displays as desired.

### **3 Virtual Panels**

Supported by the VPA, users are allowed to simultaneously manipulate multiple virtual panels located in the surrounding space. A device object on a virtual panel can be a button that outputs a command, a linear slider or a valuator, which reports a floating number, or a data tablet, which reports 2-D information. Diversified input device objects for a single application can be included in a single virtual panel or in multiple virtual panels.

Therefore, a user is able to use the virtual panels to input all of an application’s data, and can switch from panel to panel easily and directly to switch between the working spaces for different applications. We will show a few examples only to give readers a flavor of how to describe virtual devices in the VPA. An example of virtual panels is shown in figure 2.

#### **Virtual Keyboards**

A virtual panel can model a keyboard: a flat spread-out container object, on which each key is modeled as a small non-overlapping object. Whenever users press a key object (e.g., the Pointer “enters” the key object), this object reports an Enter event to an affiliated filter. Thereupon, the filter labels the event with key ID and then sends it to a pipe of destination, which is responsible to collect labeled events from the virtual keyboard. Therefore, applications are able to receive string inputs from the pipe.

To provide auditory feedback, Enter events are sent to a filter to trigger clicks. Furthermore, two-hand keyboarding is possible if Gesture Server are able to track ten tips of fingers and thumbs of both hands. Otherwise, the number of fingers and thumbs allowed to strike virtual keyboards will be limited.

#### **Virtual Screens**

A virtual panel or its portion is able to be used to simulate a mouse pad, or even the whole display of a touchscreen. Users can point into this panel and a corresponding cursor appears on the screen in the meantime. That is, the virtual panel reports Move and Pick events with 3-D position information to a filter that calculates the coordinates and warps the cursor on the screen. However, this method is more desirable than the use of mouse since pointing directly is intuitively more straightforward than dragging.

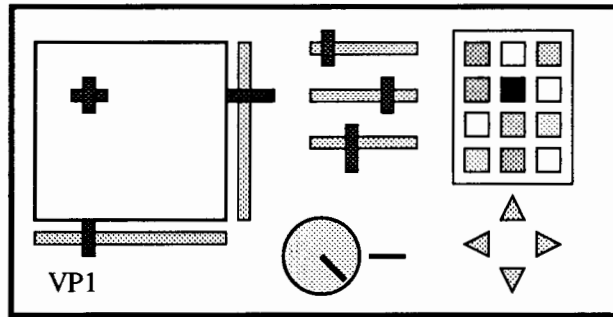


Figure 2: The layout of an instance of a virtual panel

### Other Virtual Devices

A Virtual Knob is a useful device incorporated into virtual panels. Users can *turn* clockwise or counterclockwise when holding a knob object, which reports a series of 3-D coordinates of the Pointer to a filter. Then the filter computes angular displacements relative to a hypothetical axis of the knob.

For 3-D interaction tasks, the most natural way is to specify inputs in 3-D environments. For example, given a color model RGB, CMY, or HSV [FvFH90], the best interaction method is not to decompose this instinctive 3-D structure into two or three 1-D or 2-D interaction scales on a 2-D screen. Instead, we may model it as a cube object and let users pick the colors they want. There is no limit to conceive hand-operated virtual devices in the VR environments based on the VPA.

## 4 Early experiences and Discussions

The VPA is able to accommodate sign-language approaches and voice-assisted methods. For the former, the Gesture Server is enhanced to recognize new point-based gestures; for the latter, the output of the voice recognizer can be processed to produce events to be fitted into the VPA.

Let us take a look from the user's side. In VR-styled environments, each manipulable object is drawn in a virtual world displayed on head-mounted displays. Thereupon, users are able to see static or dynamic objects, manipulate the objects by their gestures, and even perceive the tactile or auditory feedback when they touch or penetrate the objects, if the induced events are sent to trigger the proper hardware with the help of proper filters.

In keyboard-and-mouse-based environments, however, the tactile feedback and visual feedback are not provided naturally outside the screen and keyboard. To maintain the feedbacks we have to carefully code into working environments. For these, we model input-only panels as virtual panels, whose layouts are edited on the screen and printed out on paper or any mockup to provide proper visual feedback. With careful design, the auditory feedback enhances the sense of touching panels with various sounds triggered while gestures are operating virtual devices. Initially, virtual panels are instantiated by users (Pick three points to specify the plane of a virtual panel) and are put on a real desktop or pinned on the wall. Afterwards, users can pick and move any virtual panel. The environment that we are implementing is incorporated into the X window system.

Let us take a look from the system's side. We should remind readers that DataGlove is an angle-based physical device (not counting the Polhemus 3SPACE ISOTRAK in it), which tracks each finger's and thumb's bending. Since we can transform the angle-based hand model into our

point-based hand model, or vice versa [Su93], provided that enough degrees of freedom have been measured, the DataGlove can still be the physical device for the VPA, since it can be used to at least recognize partial gestures.

In our first implementation, two gestures, Point and Pick are recognized. Despite the simplest of these gestures, this permits non-trivial experimentation. The Point gesture is very useful: it can adjust sliders and turn rotary knobs. The Pick gesture is used to signify picking. We use Macintosh-based gesture acquisition running VPL's Body Electric to accept data from the DataGlove and the ISOTRAK. The data is then sent to a DECstation where the servers and applications are running.

The easiest way to implement filters and pipes in Unix is to fork a process for each filter and to use a named pipe for each pipe [Ste90]. However, because there are many small filters and a huge amount of events to and from pipes, this method is bound to suffer from process overhead and read/write I/O contention, which will affect real-time performance. Therefore, in the future, we intend to optimize the routes of event flow among filters and pipes in order to reduce the number of processes.

Furthermore, the filters have to be carefully coded, otherwise the input events from them will be deferred. Most of the time in our implementation, the interconnections among filters and pipes are simple. If the computation of filters is too intensive, additional remote machine(s) may be used to implement the Gesture Server, the Panel Server and some filters. Remote procedure calls can be used in this case to permit communication between the components.

## 5 Conclusions

The ideas of Virtual Reality are fascinating. The growing technology is resulting in a fundamental change to the future of computer environments: multi-modal I/O and full immersion in artificial reality. However, Shneiderman [Shn92] has pointed out that for most applications, *looking-at* may be more desirable than *being-in*.

We believe it advantageous to re-evaluate existing computer uses to determine in which situations being-in is more desirable, in which situations it is not, and in which situations partial being-in and partial looking-at is most desirable. We are also concerned about the smooth transition from traditional applications to VR-styled applications, and are interested in the technology applied to both.

Based on this line of thinking, we proposed the VPA, which is suitable for traditional and VR-styled applications. In the VPA, we identified the Gesture Server, the Panel Server, filters, pipes and applications; we took a new approach to look at gestures: point-based, not angle-based; we modeled the gesticulative manipulations on 3-D object hierarchies as events; and used filters to abstract higher level data for input-handling. Also, we presented an environment of virtual panels, which is incorporated into keyboard-and-mouse-based computing.

With our work's creation of virtual control panels, input interactions are able to be greatly multiplied and diversified. Additionally, with the abstraction of objects and filters, these input interactions are able to be conceived and implemented easily.

## Acknowledgement

We wish to thank Greg Drew for his programming assistance on this project.

## References

- [Bol80] Richard A. Bolt. Put-That-There: Voice and Gesture at Graphics Interfaces. *ACM Computer Graphics*, 14(3):262–270, 1980.
- [Fel90] S. Sidney Fels. Building Adaptive Interfaces with Neural Networks: the Glove-Talk Pilot Study. Technical Report CRG-TR-90-1, Dept. Computer Science, U. Toronto, February 1990.
- [FMHR86] N. I. Fisher, M. W. McGreevy, J. Humphries, and W. Robinett. Virtual Environment Display System. In *ACM Workshop on Interactive 3D Graphics*, pages 77–87, 1986.
- [FvFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: principles and practice*, chapter 13. Addison-Wesley, 2 edition, 1990.
- [KKKE90] K. H. E. Kroemer, H.J. Kroemer, and K. E. Kroemer-Elbert. *Engineering Physiology: Bases of Human Factors/Ergonomics*, page 22. van Nostrand Reinhold, New York, 2 edition, 1990.
- [KL87] James Kramer and Larry Leifer. The Talking Glove: An Expressive and Receptive Verbal Communication Aid for the Dead-Blind, and Nonvocal. In *Proc. 3rd Ann Conf Computer Technology/Special Education/Rehabilitation*, pages 335–340, 1987.
- [PEF+90] A. Pentland, I. Essa, M. Friedmann, B. Horowitz, and S. Sclaroff. The ThingWorld Modeling System: Virtual Sculpting by Modal Forces. *ACM Computer Graphics*, 24(2):143–144, March 1990.
- [SG86] Robert W. Scheifler and Jim Gettys. The X Window System. *ACM TOG*, 5(2):79–109, April 1986.
- [Shn92] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 2 edition, 1992.
- [Ste90] W. Richard Stevens. *Unix Network Programming*. Prentice-Hall, Inc., 1990.
- [Su93] S. Augustine Su. Hand modeling in Virtual Environments. Technical report, Computer Science Center, University of Maryland, College Park, 1993.
- [WGW90] Andrew Witkin, Michael Gleicher, and William Welch. Interactive Dynamics. *ACM Computer Graphics*, 24(2):11–21, March 1990.
- [ZLCBH87] T. Zimmerman, J. Lanier, S. Bryson C. Blanchard, and Y. Harvill. A Hand Gesture Interface Device. In *Proc. CHI+GI 1987*, pages 189–192, Toronto, Ontario, April 1987.