Olwal, A. and Feiner, S. Unit—Modular Development of Distributed Interaction Techniques for Highly Interactive User Interfaces. In *Proceedings of GRAPHITE 2004 (International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia)*, Singapore, Singapore, June 15-18, 2004, p. 131-138.

Unit: Modular Development of Distributed Interaction Techniques for Highly Interactive User Interfaces

Alex Olwal[1,2] Steven Feiner[1]

[1] Department of Computer Science
Columbia University
New York, USA

[2] Department of Numerical Analysis and Computer Science
Royal Institute of Technology
Stockholm, Sweden

alx@kth.se, feiner@cs.columbia.edu

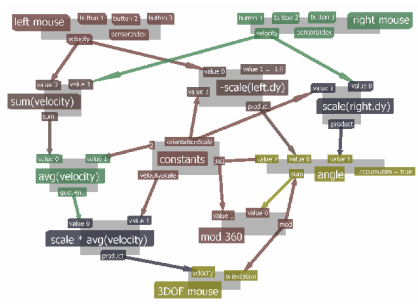

Figure 1. The Unit User Interface, displaying the Unit Graph that specifies the interaction for the three–degree-of-freedom mouse.

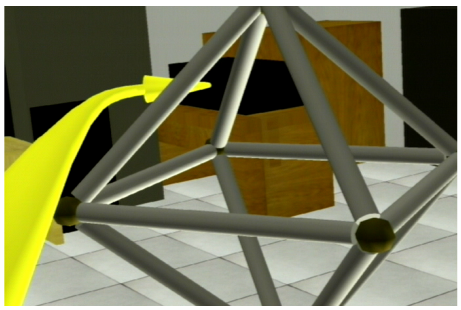

Figure 2. The flexible pointer selecting an obscured object, without visually interfering with the occluding object.

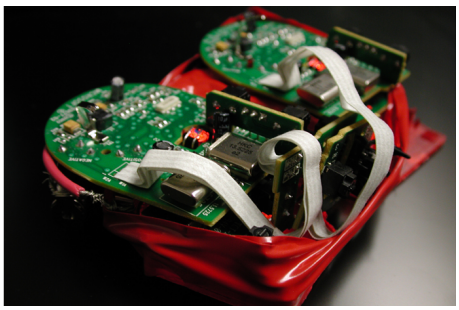

Figure 3. The 3DOF mouse, a simple composite input device consisting of two wireless optical mice. A Unit Graph extracts the rotational acceleration in the plane of the surface.

Abstract

The Unit framework uses a dataflow programming language to describe interaction techniques for highly interactive environments, such as augmented, mixed, and virtual reality. Unit places interaction techniques in an abstraction layer between the input devices and the application, which allows the application developer to separate application functionality from interaction techniques and behavior.

Unit's modular approach leads to the design of reusable application-independent interaction control components, portions of which can be distributed across different machines. Unit makes it possible at run time to experiment with interaction technique behavior, as well as to switch among different input device configurations. We provide both a visual interface and a programming API for the specification of the dataflow. To demonstrate how Unit works and to show the benefits to the interaction design process, we describe a few interaction techniques implemented using Unit. We also show how Unit's distribution mechanism can offload CPU intensive operations, as well as avoid costly special-purpose hardware in experimental setups.

C.R Categories and Subject Descriptors: D.1.7 Visual Programming; D.2.2 Design Tools and Techniques—*User Interfaces*; D.2.6 Programming Environments—*Graphical environments*; D.3.2 Language Classifications—*Data-flow languages*; H.3.4 Systems and Software—*Distributed Systems*; H.5.1 Multimedia Information Systems—*Artificial, augmented, and virtual realities*; H.5.2 User Interfaces—*Graphical user interfaces (GUI), Input devices and strategy, prototyping*; I.3.4 Graphics Utilities—*Virtual device interfaces*

Keywords: interaction techniques, dataflow programming, visual programming, augmented reality, mixed reality, virtual reality.

1 Introduction

Despite tremendous improvements in computer systems over the past several decades, designing and developing interaction techniques is still a difficult task, especially for highly interactive immersive 3D environments, such as Augmented Reality (AR), Mixed Reality (MR) and Virtual Reality (VR). Interaction in immersive environments involves many different types of user input and many devices with which that input is provided, such as position and orientation trackers, voice input, and haptic devices, in addition to conventional mice, trackballs, touch screens, and keyboards. While there is an increasing number of well-known metaphors for immersive interaction, such as "the virtual hand" [Bowman and Hodges 1997], "ray pointer" [Bowman and Hodges 1997], and "flashlight pointer" [Liang and Green 1993], there is still much variation in how these metaphors are implemented.

Interaction techniques involve the mapping of data from input devices to application semantics. Therefore, we find it particularly attractive to use a dataflow approach to the design of interaction techniques, in which data is processed through a customizable network. We introduce the Unit framework [Olwal 2002], which allows users to specify 2D and 3D interaction techniques as data-

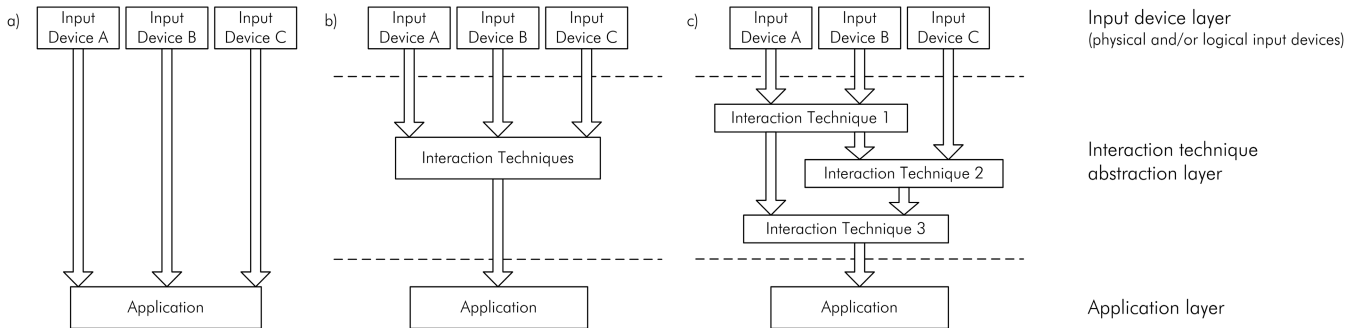

Figure 4. a) Traditional input device interface for an interactive application. Input devices are mapped to functionality on the application level, making it hard to change devices and behavior. b) Unit places interaction techniques in an abstraction layer between input devices and the application. This layer can be changed for different combinations of interaction devices and behavior. c) Unit's modularity also makes it possible to abstract interaction techniques from each other.

flows and to modify them in running programs. We use our framework to abstract interaction techniques from applications that use them, as well as from input devices that control them. This allows users to flexibly configure and dynamically change interaction technique behavior, independent of both input devices and applications. Furthermore, in our framework, a dataflow can be easily distributed over multiple machines to create distributed interaction techniques, as well as distributed applications. As shown in Figure 1, we use a direct-manipulation, visual-programming representation to specify the behavior of the dataflow in the Unit User Interface (Unit UI), which is itself implemented with Unit.

In the remainder of this paper, we first present related work in Section 2, followed by brief introductions to the Unit framework and the prototype Unit UI in Sections 3 and 4. To explain how Unit can be used, we describe some example interaction techniques that we have developed with it in Sections 5, 6 and 7: a novel flexible pointer for selecting objects in 3D environments (Figure 2) [Olwal and Feiner 2003a], an experimental setup for analyzing non-verbal features of speech [Olwal and Feiner 2003b], and a quickly prototyped rotationally sensitive mouse (Figure 3), created from a pair of conventional mice. We describe our implementation in Section 8, and present our conclusions and future work in Section 9.

2 Related Work

Data flow programming and directed-graph–based visual programming languages have been used together by a number of researchers to design 2D UIs and interaction techniques. Projects that take this approach have included Smith's InterCONS [Smith 1988], Borning's ThingLab [Borning 1981], and Maloney and Smith's Morphic user interface framework [Maloney and Smith 1995] for Self [Ungar and Smith 1987]. A key issue here is the recognition that interaction techniques essentially map the outputs (and inputs) of interaction devices to the inputs (and outputs) of applications; this observation has long been an underlying theme of work on building formal models of abstract graphical input devices, which in turn can be composed together in graphs to create hierarchical input devices [Anson 1982, Duce et al. 1990].

Most 3D interaction techniques can be conceptualized this way, and 3D toolkits that embody these techniques (e.g., [Kessler et al. 1997, CaveLib 2003, VRJuggler 2003, VRPN 2003]) typically use abstract input devices. We have chosen to abstract the interaction technique components in a similar fashion to BodyElectric [Lanier et al. 1993], ICON [Dragicevic and Fekete 2001] and InTml [Figueroa et al. 2002]. (See Figure 4.) As in these systems, our components are dataflow graphs, which are assembled into customized interaction techniques. In contrast to InTml, whose developers emphasize their XML-based specification language, we have chosen to focus on an interactive design process in which interaction techniques can be modified at runtime. We have also been more concerned with design issues that are typical for highly interactive distributed environments, such as AR/MR/VR, which distinguishes our framework from systems targeted for a user on a single computer, such as ICON. While BodyElectric also addressed 3D virtual environments, its dataflow operated on only a single machine (a Macintosh that controlled one or more SGI workstations). In contrast, Unit's dataflow graphs can be spread across multiple machines.

Our approach allows the development of flexible interaction techniques, portions of which can be distributed, as well as replaced and remapped at runtime, and we provide a user interface for visual dataflow programming of these behaviors, as well as a programming API. While the direct-manipulation creation of 3D widgets is an appealing approach [Zeleznik et al. 1993], we have

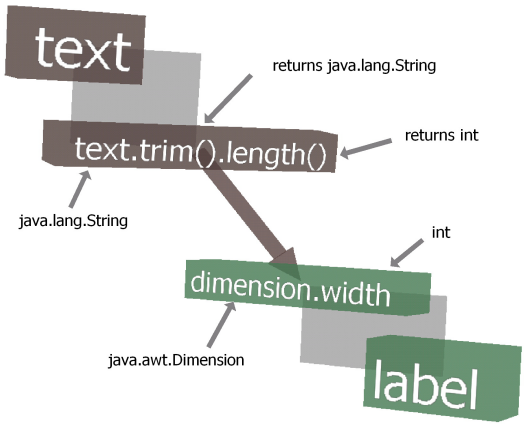

Figure 5. Two units with two connected properties in the Unit UI. As the *text* property changes, the *width* field of the *dimension* property of the *label* is updated.

chosen a dataflow language for its clear depiction of the mapping between device outputs and application inputs. While our work on Unit primarily applies this approach to the design of experimental 3D interaction techniques, we also believe it is of significant relevance to the design of new input devices (e.g., [Fitzmaurice et al. 1995, Greenberg and Fitchett 2001, Suzuki and Kato 1993, MacKenzie et al. 1997, Resnick 1993, Hinckley and Sinclair 1999]).

3 The Unit Framework

The Unit framework uses the concept of *units* to represent the nodes in the data flow. Each unit has any number of *properties* and any number of connections to properties in other units, as shown in Figure 5. Two units with two connected properties in the Unit UI. As the *text* property changes, the *width* field of the *dimension* property of the *label* is updated.. When a property is updated, a special method is called, which by default updates all connected properties. Customized units typically override this method with their own data processing, and when done, typically redirect to the default method. These connections can also be made over the network, allowing each unit in the dataflow to be able to share its properties and listen to property changes, anywhere on the network.

These simple rules allow the design of flexible and customizable units that specify the desired behavior through their combination into *Unit Graphs*, much like electrical circuits.

Units

As mentioned above, the key components of a unit are its properties and the ability to maintain and update connections to properties in other units.

Properties

Properties are attribute-value pairs, in which the value can be a pointer to any Java Object.

Connections

Connectivity is peer-to-peer, where only the involved units are aware of their connections and are solely responsible for administering their relationships. Connections between two units are typically accomplished by reference, with the data pointer of the source property initially copied to the target property. When an update is made, the target unit is notified that the data has changed. We also provide the ability to make connections by value, in which each update replicates the data in the source unit; however, connections by reference are typically preferred for efficiency. In addition, connections can be created through a chain of references to fields and methods in the property value, provided that the resulting source and target values are of the same data type. These references are more expensive, since they require dereferencing, evaluation, and replication.

Figure 6 shows a simple example in which two mice are used to provide six–degree-of-freedom control of an application.

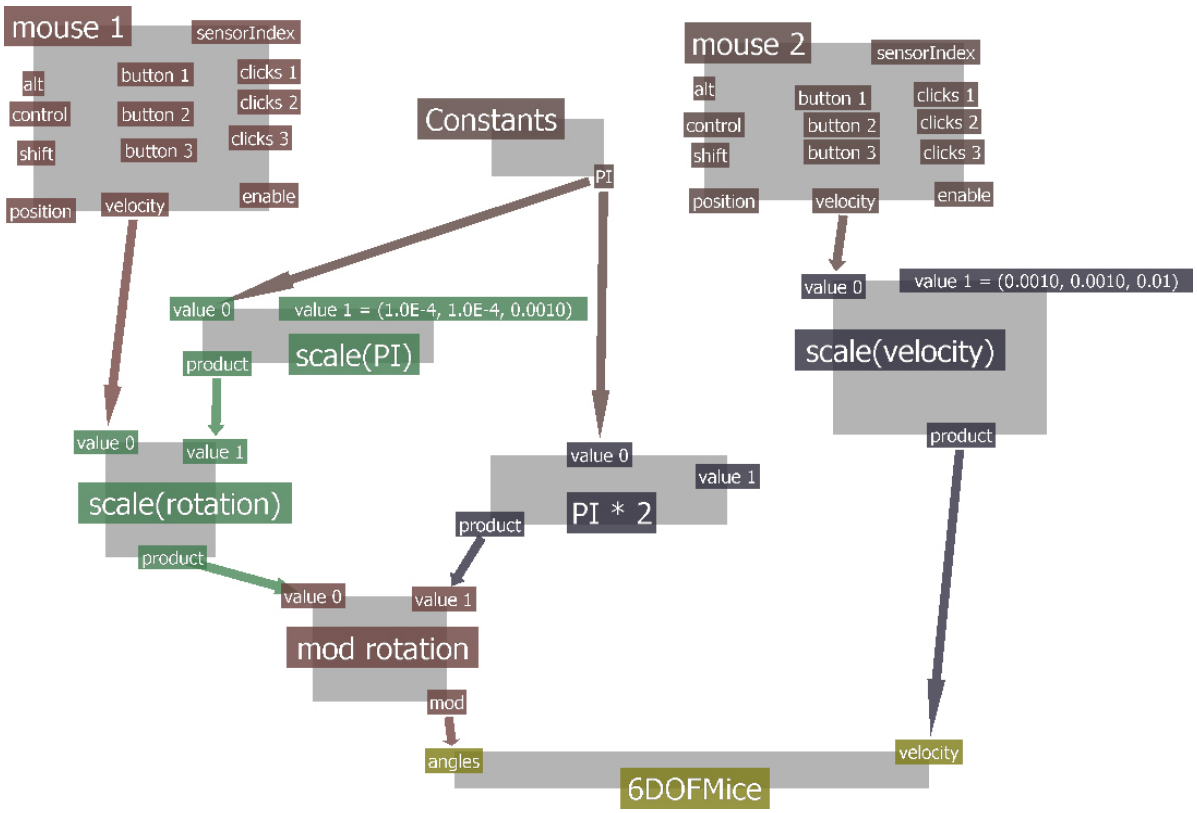

Figure 6. A screenshot from the Unit UI, in which a dataflow graph is being specified for direct control of six degrees of freedom through the use of two mice. (We use the scroll wheel to provide a third degree of freedom from each mouse.)

Distribution

Connections can be transparently distributed over Java Remote Method Invocation (RMI), with the addition of the hostname in the specification of the connection. Following the peer-to-peer approach, each unit is individually distributed through the RMI registry and directly accessible to other units.

This distribution approach allows parts of Unit Graphs to be distributed over an arbitrary number of applications running on an arbitrary number of machines. A common problem in immersive environments is the use of hardware, such as six–degree-of-freedom trackers, that have a permanent physical connection to a single machine. Unit not only allows cross-platform access to platform-specific devices, but also simplifies the sharing of machine-specific devices. Our framework allows several Unit Graphs, running on different machines or within different programs on the same machine, to communicate as a single graph, providing transparent access to data and flow control from anywhere.

Core components

Unit, the core class, which is the superclass of all units, provides the general unit functionality, which most importantly is the handling of properties, connections, and distribution. Two units are connected by specifying the source unit, source property, target unit, and target property, and an optional host name for remote connections.

We have also implemented a set of core units that provide additional functionality. These include units for flow control, such as switches and multiplexers, units for scalar and vector operations, and units for I/O (multiple mice, keyboards, six–degree-of-freedom sensors, and speech recognition/synthesis).

The units are arranged in a class hierarchy under the *Unit* superclass. It is easy to implement new units, which typically involves overriding the *changeProperty* method that is called on every property update. Most core units have a set of reserved property names that are used for their specific input and output properties.

4 The Unit User Interface

We created the Unit UI, shown in Figures 6 and 7, as a complement to our programming API, to allow users to design, manipulate, and visualize the dataflow in a Unit Graph.

The Unit UI lets the user add, modify and delete units, properties and connections, as well as load and save Unit Graphs. Besides visually displaying the dataflow as directed graphs with units, properties and connections, live data propagation is visualized by highlighting a property for a short time after it is updated. The user can also switch to live views of values of interest.

The Unit UI was implemented with units, using the same dataflow approach as the interaction techniques it manipulates—demonstrating that our framework does not restrict itself to interaction technique specification, but also applies to traditional application logic. The Unit UI is a 3D application, and can thus

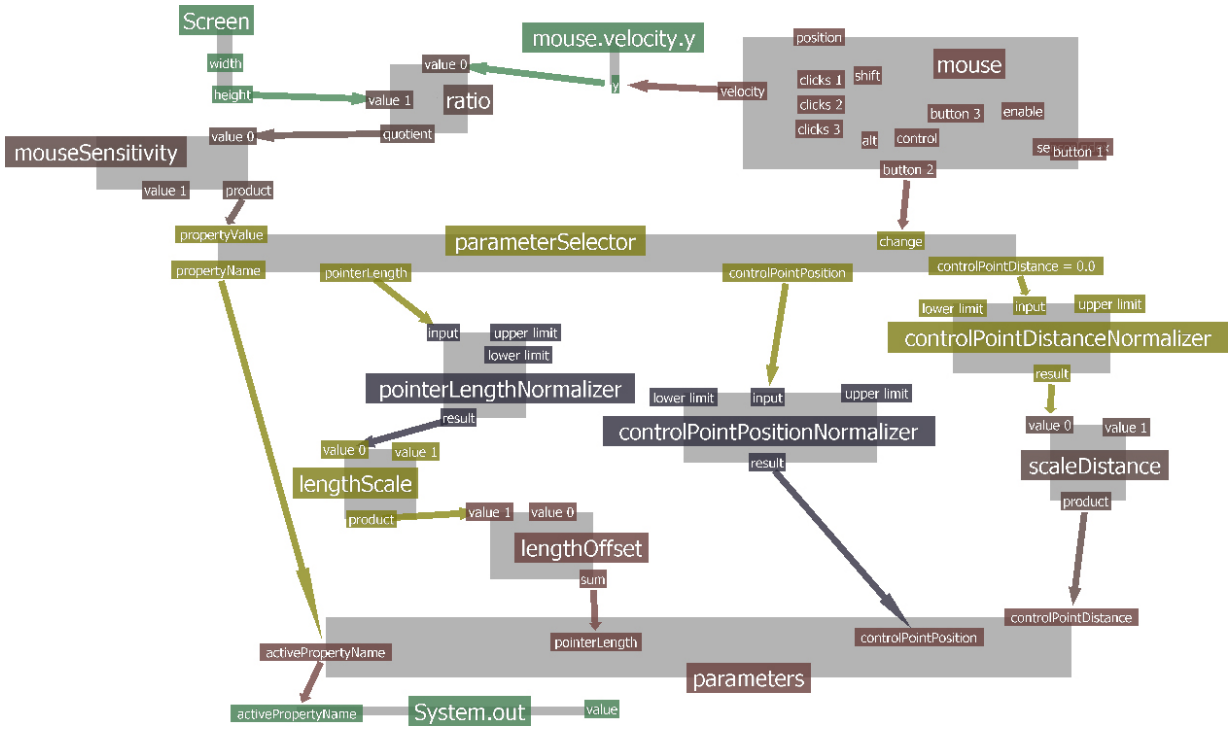

Figure 7. The dataflow for the tweaking code that manipulates parameters in the flexible pointer through a mouse with a thumb-controlled joystick. Clicking the second mouse button alternates the parameter to be modified, while the movement of the joystick changes the value. The result is accessible in the unit named "parameters" at the bottom, which is remotely connected to the graph for the flexible pointer. (Screenshot from the Unit UI.)

Figure 8. The flexible pointer interaction technique provides easier selection and clearer indicative pointing in collaborative environments, in addition to the ability to select fully or partially obscured objects.

coexist in the immersive environment, side-by-side with the interaction techniques whose behavior it controls.

However, because of the limited field-of-view and low (800×600) resolution of our head-worn displays, we find it more productive to interact with the Unit UI in 2.5D on high-resolution (1920×1200) 24" desktop displays. In the following sections, we describe our experience with Unit by presenting some of the experimental interaction techniques that we have developed with it.

5 The Design of an Interaction Technique for Immersive Environments

We have implemented a novel interaction technique, called the *Flexible Pointer* [Olwal and Feiner 2003a], which is an extension of existing ray-casting techniques for selection in immersive environments. The flexible pointer allows the user to point around objects, with a curved arrow, for selection of fully or partially obscured objects, as well as to more clearly point out objects of interest to other users in a collaborative environment. The flexible pointer, shown in Figure 8, reduces ambiguity by avoiding obscuring objects, which would have been selected with traditional ray-casting techniques. The flexible pointer also has a visual advantage in situations in which it is easy to point out an object, without obstructing the object of interest, while still providing a continuous line from the user to the target.

The problems that we address with the Unit framework are how users can control the pointer, and how we can interactively modify and tweak this mapping, at runtime and during the design phase.

Implementation

First, we have to decide on a representation for the geometry of the flexible pointer. We choose a Quadratic Bézier spline, where position, length, and curvature of the pointer are controlled by three points in space.

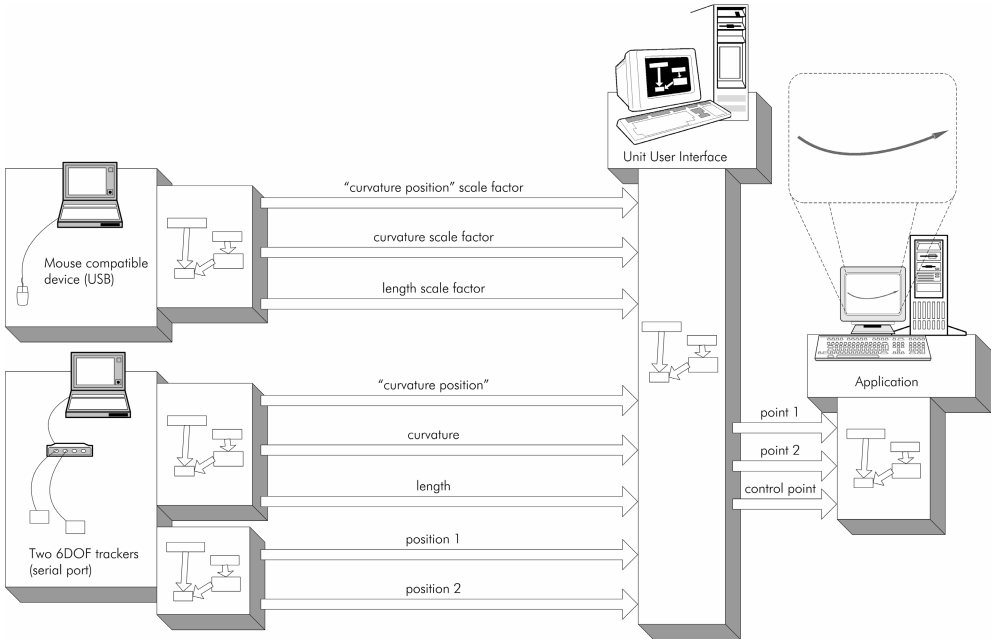

Figure 9. Overview of the flexible pointer interaction technique during the design phase. From right to left: The application implements the geometry for a Bézier curve pointer and uses a distributed unit to listen to changes in its properties. The Unit UI is run on a second computer on the network. The graph manipulated by the Unit UI outputs the three parameters to the Bézier curve, while taking the five curve parameters as input. Two six–degree-of-freedom trackers provide the position properties and length, and the curvature characteristics are derived from their orientation. The behavior of the flexible pointer can be tweaked by a graph that lets the user manually adjust the constants with a mouse-compatible device. Interaction technique abstraction occurs at several stages here, most clearly between the application, the Unit UI, and the sensors.

Secondly, we implement a corresponding, customized unit that listens to changes in its *position, end point,* and *control point* properties, and updates the geometry accordingly. We now have a mechanism for listening to, and updating the values of this unit, both locally and over the network. Any component in our framework is thus able to listen to changes or update the geometry of the pointer, by accessing these properties. For increased precision, our prototype flexible pointer utilizes a two-handed approach, where the hands are tracked with two six–degree-of-freedom trackers, the distance between the hands map to the length of the pointer, and the relative bending of the hands determines the curvature characteristics of the pointer. We implement this control behavior as a separate Unit Graph that updates the properties of the above-mentioned unit that is controlling the geometry.

Design process

One of the hardest tasks in interaction technique design is the assignment of appropriate values to constants, and as with most interaction techniques, there are several such constants for the flexible pointer (e.g., the scale factor for the mapping of the distance between the user's hands to the length of the pointer).

Thanks to Unit's modularity, separate Unit Graphs can be used for interactive tweaking and debugging of the running interactive technique. We constructed a new graph that takes input from a small handheld presentation mouse with a thumb-controlled joystick. A button click alternates between the constants that are modified and pushing the joystick up/down increases/decreases the value of the current constant, as shown in Figure 7. Although we could place the graph in the same program as the flexible pointer, avoiding the mix of interaction technique and tweaking code seemed reasonable, and we found it more advantageous to run it in a separate program. In fact, the ease of distribution made us place it on a separate machine, which gave us an exclusive environment for developing the tweaking code, as shown in Figure 9. The behavior of our interaction technique can be modified in real time as soon as the graph is connected to the flexible pointer. More importantly, we can have the flexible pointer running constantly, while modifying, recompiling, and restarting the tweaking code. When satisfied with the behavior of the interaction technique, the tweaking code is removed, simply by not running it. This example shows how we can use Unit to abstract the interaction techniques from the input devices and the application, and also how two interaction techniques (the flexible pointer and the tweaking code) can be abstracted from each other.

6 The Development of Interaction Techniques Using Distributed Speech Recognition, Analysis, and Localization

We found Unit very useful in a recent experimental setup for a user interface based on speech analysis and audio localization [Olwal and Feiner 2003b]. We intended to explore the use of non-verbal features of the user's speech for implicit or explicit program control. Additionally, we planned to use multiple microphones to approximate the user's head position, by comparing the audio from the different microphones.

Running CPU-intensive speech recognition on multiple microphones

First, we needed a mechanism for getting input from multiple microphones, so we considered the following approaches:

1) Using multiple general-purpose sound cards on one computer. One would have to be careful to not run into hardware conflicts, since an ordinary PC is not designed to have many simultaneously active sound cards.

2) Using a special-purpose sound card with multiple audio inputs. One of these cards would be too expensive for our low-budget experimental setup.

3) Using a special-purpose array microphone for audio localization, where the signal processing is done in hardware. The few such inexpensive consumer-level microphones we found did not provide programming API access to inferred positional data. These microphones also put restrictions on the setup, limited by the characteristics of the microphone, and we found it neither feasible nor cost effective to build our own microphone.

Second, speech recognition is CPU intensive, and running several instances of speech recognition software on the same machine used for the visualization would significantly affect the frame rate.

A distributed approach

Realizing that we had many available machines in our lab, equipped with standard sound cards, we decided to take advantage of Unit's distribution mechanism to offload the CPU-intensive speech recognition to other machines on the network. Each of these machines could then support one microphone, without the need for any special-purpose hardware or alteration of the hardware configuration.

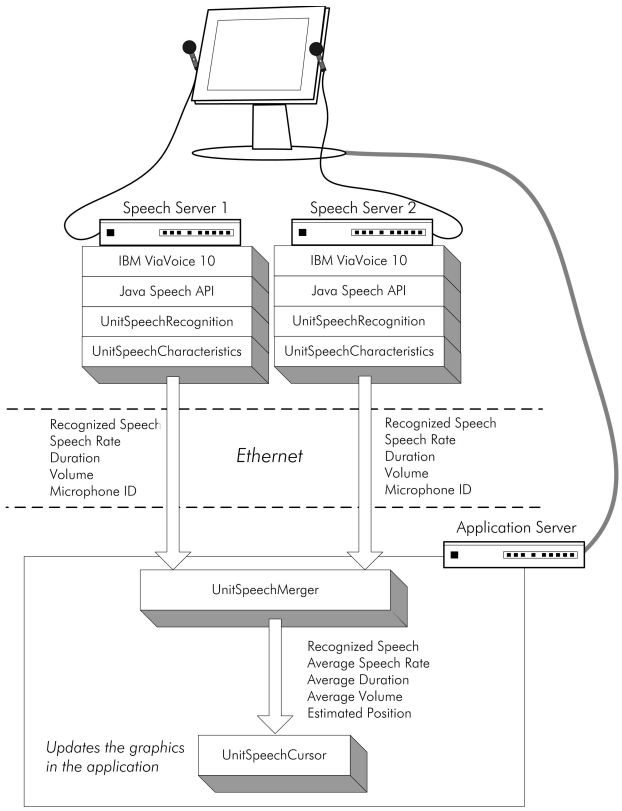

Figure 10. The architecture for the experimental non-verbal speech and audio localization setup.

We designed our Unit Graph such that the speech is analyzed locally on each speech server, with the recognized speech and the extracted speech features communicated over Ethernet to the application server. The Unit dataflow in the application server fuses the input and adjusts the behavior of the application accordingly. Our experimental setup is shown in Figure 10.

It might sound contradictory that we find it more cost-efficient and convenient to use a separate computer, instead of a special-purpose sound card, to host a microphone. However, the important point here is that Unit allowed us to use our currently available general-purpose hardware for rapid prototyping of an experimental user interface, without having to deal with the hardware-related issues that would play a central role in designing a practical product. While Unit made it possible to easily develop a distributed dataflow for our purposes, its transparent distribution mechanism also makes it straightforward and simple to reconfigure the application to run on a single machine (e.g., with multiple sound cards or a multi-input sound card).

7 Composite Input Devices

The Unit framework has made it easy for us to develop rudimentary prototype input devices, assembled from arrangements of two or more input devices. Figure 3 shows one of the simplest examples of a composite input device: a three–degree-of-freedom mouse created from two off-the-shelf wireless optical mice that are rigidly attached to provide an additional degree of freedom (rotational acceleration in the plane of the surface on which they are used). Unit provides simple means for specifying the relations between the two mouse sensors, and thus allows the behavior of this composite input device to be visually programmed, completely in software, as shown in Figure 11.

Unit thus makes it possible to build composite input devices that consist of hierarchies of different input devices and interaction techniques, while providing unified application-level APIs to these devices.

8 Implementation

The Unit framework is implemented with Java and Java3D, and therefore runs across multiple platforms. Unit's current implementation supports conventional pointing devices (e.g., mice, trackballs, touchpads, trackpoints, and touchscreens) and keyboards, as well as several six–degree-of-freedom sensors (Ascension Flock of Birds, InterSense IS600 Mark 2 Plus, and InterSense IS900) and speech recognition and speech synthesis (through the Java Speech API and IBM ViaVoice). RMI is used for distribution over TCP/IP. We have used a heterogeneous machine pool during development, with machines ranging from an Intel Celeron 400 MHz, with 192 MB RAM, running Windows 98, to a Dual Intel Xeon 2.8 GHz, with 1 GB RAM, running Windows XP. The low-end machines can be used for running Unit Graphs and input device handling, while the more powerful machines with 3D acceleration hardware are needed for 3D graphics.

9 Conclusions and Future Work

As we have showed, Unit allows the flexible specification of interaction techniques, while effectively avoiding problems related to specific hardware setups in experimental systems through a peer-to-peer distribution mechanism. Besides abstracting interaction techniques from input devices and applications, Unit's modularity has also proven convenient, since it allows debugging components to be developed in a stand-alone fashion outside the interaction technique of interest.

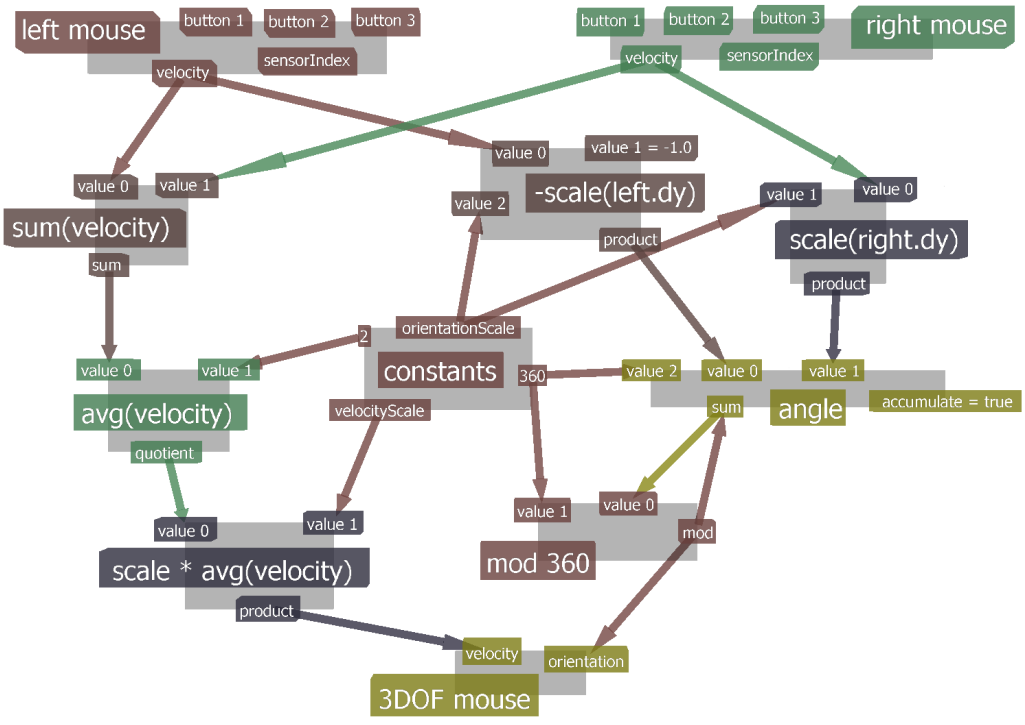

Figure 11. Rotational acceleration can be extracted from two rigidly attached mice with this Unit Graph. The resulting unit (3DOF mouse) can be interfaced as a "new" input device, providing three degrees of freedom. (Screenshot from the Unit UI.)

While our framework is implemented in Java and Java3D, interactive 3D graphics systems are more often developed in OpenGL and C/C++ today. Therefore, to make our framework accessible from a non-Java environment, we plan to write a C/C++ version of the Unit base class to allow native code to communicate with, and take advantage of, the Unit framework.

We intend to extend Unit's interoperability by writing a bridge to VRPN [VRPN 2003], which would provide the Unit framework with support for an even wider range of trackers. Bindings to other popular VR frameworks, such as CaveLib [CaveLib 2003] and VRJuggler [VRJuggler 2003], are also of interest. We are generally interested in adding more input and output options to Unit, such as haptics, tablets, joysticks, audio/MIDI and phidgets [Greenberg and Fitchett 2001]. Some of these can be achieved with little effort by writing bindings for the abovementioned libraries.

The Unit UI also needs to be improved if it is to be able to support the design process of increasingly complex Unit Graphs. Besides missing functionality (e.g., copy/paste), we believe that visualization approaches, such as encapsulation, explosion views, layers and fisheye lenses, would be advantageous to the Unit UI. It would also be useful to extend the visualization of the changing data in the graphs, as well as the actual data flow.

Acknowledgments

This research was funded in part by Office of Naval Research Contracts N00014-99-1-0249 and N00014-99-1-0394, NSF Grants IIS-00-82961 and IIS-01-21239, and gifts from Intel, Microsoft Research, and Alias Systems.

References

ANSON, E. 1982. The Device Model of Interaction. *Proc. SIGGRAPH '82* (*ACM Comp. Graph.*, 16(3), July 1982), Boston, MA, July 26–30, 107–114.

BORNING, A. 1981. The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. *ACM Trans. on Prog. Langs. and Sys,* 3(4), October 1981, 343–387.

BOWMAN, D., HODGES, L.F. 1997. An Evaluation of Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments. *Proc. Symp. on Interactive 3D Graph.*, 35–38.

CAVELIB. 2003. http://www.vrco.com/products/cavelib/cavelib.html. Virtual Realty Consulting (VRCO) Inc. Virginia Beach, VA.

DUCE, D., VAN LIERE, R., AND TEN HAGEN, P. 1990. An Approach to Hierarchical Input Devices. *Comp. Graph. Forum*, 9(1), 15–26.

DRAGICEVIC, P. AND FEKETE, J.D. 2001. Input Device Selection andInteraction Configuration with ICON. *Proc. IHM-HCI 2001.* Frontiers, Lille, France, Springer Verlag, 543-448.

FIGUEROA, P., GREEN, M., HOOVER, H.J. 2002. InTml: A Description Language for VR Applications. *Proc. 3D Web Technology*. 53-58.

FITZMAURICE, G.W., ISHII, H., BUXTON, W. 1995. Bricks: Laying the Foundations for Graspable User Interfaces. *Proc. Human Factors in Comp. Sys.* (*CHI '95*), 442–449.

GREENBERG, S. AND FITCHETT, C. 2001. Phidgets: Easy Development of Physical Interfaces Through Physical Widgets. *Proc. ACM Symp. on User Interface Software and Tech.* (*UIST '01*), Orlando, FL, 2001, 209–218.

HINCKLEY, K. AND SINCLAIR, M. Touch-Sensing Input Devices. 1999. *Proc. Conf. on Human Factors in Comp. Sys.* (*CHI '99*), 223–230.

KESSLER, G.D., KOOPER, R., VERLINDEN, J.C. AND HODGES, L. 1997. The Simple Virtual Environment Library, Version 2.0, User's Guide, http://www.cc.gatech.edu/gvu/virtual/SVE/docV2.0/sve.book_1.html. Technical Report, Graphics, Visualization, and Usability Center, Georgia Institute of Technology.

LANIER, J. GRIMAUD, J-J, HARVILL, Y., LASKO-HARVILL, A., BLANCHARD, C., OBERMAN, MARK., TEITEL, M. 1993. Method and system for generating objects for a multi-person virtual world using data flow networks. United States Patent 5588139.

LIANG, J., GREEN, M. 1994. JDCAD: A Highly Interactive 3D Modeling System. *Comp. and Graph.,* 18(4). 499–506.

MACKENZIE, I. S., SOUKOREFF, R. W., PAL, C. 1997. A Two-ball Mouse Affords Three Degrees of Freedom. *Extended Abstracts of Human Factors in Comp. Sys.* (*CHI '97*), 303–304.

MALONEY, J. AND SMITH, R. 1995. Directness and Liveness in the Morphic User Interface Construction Environment. *Proc. ACM Symp. on User Interface Software and Tech.* (*UIST '95*), Pittsburgh, PA, 1995, 21–28.

OLWAL, A. 2002. Unit—A Modular Framework for Interaction Technique Design, Development and Implementation. MS Thesis, Department of Numerical Analysis and Computer Science, Royal Institute of Technology, Stockholm, Sweden.

OLWAL, A AND FEINER, S. 2003a. The Flexible Pointer—An Interaction Technique for Selection in Augmented and Virtual Reality. *Conference Supplement of ACM Symp. on User Interface Software and Tech.* (*UIST '03*), Vancouver, BC, 2003, 81–82.

OLWAL, A. AND FEINER S. 2003b. Using Prosodic Features of Speech and Audio Localization in Graphical User Interfaces. Technical Report CUCS-016-03, Department of Computer Science, Columbia University, New York, NY.

RESNICK, M. 1993. Behavior Construction Kits. *Communications of the ACM,* 36(7). 64–71.

SMITH, D.N. 1988. Building Interfaces Interactively. *Proc. ACM SIGGRAPH Symp. on User Interface Software,* Banff, Alberta, October 17–19, 1988, 144–151.

SUZUKI, H., KATO, H. 1993. AlgoBlock: A Tangible Programming Language, A Tool for Collaborative Learning. *Proc. 4th European Logo Conf.,* August 1993, 297–303.

UNGAR, D. AND SMITH, R. 1987. Self: The Power of Simplicity. *Proc. OOPSLA '87,* Orlando, FL, October 1987, 227–241.

VRJUGGLER. 2003. http://www.vrjuggler.org/. Virtual Reality Applications Center, Iowa State University, Ames, IA.

VRPN. 2003. http://www.cs.unc.edu/Research/vrpn/. Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC.

ZELEZNIK, R. C., HERNDON, K. P, ROBBINS, D. C., HUANG, N., MEYER, T., PARKER, N., HUGHES, J.F. 1993. An Interactive 3D Toolkit for Constructing 3D Widgets. *Proc. SIGGRAPH '93,* 81–84.