A Framework for Generic Inter-Application Interaction for 3D AR Environments

Stephen DiVerdi, Daniel Nurmi, Tobias Höllerer
Department of Computer Science
University of California, Santa Barbara, CA 93106
{sdiverdi,nurmi,holl}@cs.ucsb.edu

Abstract

We present a generic software framework enabling inter-application interaction in a 3D augmented reality environment. The application interface is designed for ease of development and maximum end-user flexibility. We showcase some novel 3D applications and their interactions within the framework, demonstrating the new possibilities created by 3D interfaces. A classification is presented to categorize the applications' roles in such interactions, to help steer development. Finally, we discuss future possibilities for applications and interactions within such a framework.

1. Introduction

For the past twenty years, two dimensional desktops have been the standard of computer interfaces everywhere. Subsequently, user interfaces and application interfaces have been developed mostly with this dimensional restriction in mind. With the ever-increasing capabilities of 3D graphics cards and extensive research in the area of interactive rendering, many new avenues have been opened up in the realm of desktop and application interface possibilities. In this paper, we present a generic framework that allows 2D GUI-based and 3D applications to visually interact in novel ways, thanks to the expanded capabilities of a 3D environment. We address each of the three main components of such a framework - the system mechanisms which manage the flow of data, the interface presented to users to establish flows, and visualization of active flows.

Traditional 2D GUI-based applications have a very restrictive set of interactions. The primary emphasis in application interaction is with the user, via a number of I/O devices, and so this is where the most development has occurred. However, the field of inter-application interaction and communication is much less understood. For coarse-grained interaction, most operating systems provide some sort of clipboard functionality, which allows for the transfer of static data of a commonly understood format (text, raw image, or possibly raw audio) between applications. Drag-and-drop operations simplify copying and pasting into a gestural procedure, but the action is fundamentally the same. This user-initiated one-time interaction is sufficient for simple use cases, but it is inadequate for any interaction requiring the persistent transfer of dynamic content. Fine-grained control is made possible in these operating systems by allowing applications to export a set of functions that other processes can execute at runtime (eg. Windows DDE [6]), but these interfaces must be built in during development and must be coordinated among all the applications that will partake in the communication. Therefore, while data flows can be implemented in these systems, they have been unrealized on a large scale (outside of application suites, like Microsoft or Adobe groupware) because of the lack of a generic means to do so.

The UNIX command line interface (CLI) philosophy of small programs that are linked together to achieve more complicated tasks demonstrates a more generic means of inter-application interaction, by providing each application with standard input and standard output. Because of this system-wide design concept, application development is simplified, and users experience a greater level of control and flexibility in how they use their applications. Unfortunately, this abstraction is only maintained for CLI programs and is often discarded in 2D GUI applications. Additionally, it only allows for one stream of data coming in to, and going out from, each application, so more complicated data flows again require low-level services or custom solutions.

The framework we have developed extends on these fundamentals, providing a means for GUI applications to receive and transmit multiple distinct streams of data. Such a system has not appeared in the world of 2D interfaces, partially because of the visual limitations of such an environment. To maximize utilization of screen real estate, 2D applications are often minimized to a dock or task bar, or are "window shaded" (represented by just the title bar).

Figure 1. A typical workspace in ARWin, running a number of 2D and 3D applications in the volume above a user's desk.

They are stacked on top of each other, or tiled to cover every pixel of the display. Persistent links between applications are then unintuitive, because in such a visual environment where each mode of operation has some sort of display widget, there is simply no room for a persistent link to be presented to the user, without obscuring important data.

In 3D, we no longer have the limitations preventing effective layout and organization of application representations. Users have much greater control over size, relative placement, overlap and organization of applications in a 3D environment, than on a traditional 2D desktop. Additionally, an organization of 3D applications will almost never densely cover the workspace's volume (in fact, they will much more likely be sparse), leaving plenty of room between applications to visualize links. Therefore, it finally makes sense to allow applications to establish persistent connections with other applications as a display manager level service with an associated visualization, rather than an application level custom solution or a low-level operating system service without a visual metaphor.

As a foundation for our framework, we use exactly such an environment - ARWin [3], an augmented reality window manager (see Figure 1). As a 3D workspace designed for rapid application prototyping, ARWin provides a suitable basis for the window manager level services and custom 3D applications we have developed. The core of ARWin's functionality is derived from the ARToolkit, which provides an easy user interface mechanism to control the organization of the workspace in an intuitive fashion. By using ARWin and the ARToolkit, we were able to prototype and test our framework in a relatively short period of time.

For the rest of this paper, we describe the system we developed and the interface possibilities we realized. Section 2 examines previous work related to 3D interfaces and inter-application interaction. Section 3 is a brief description of the architecture within which we developed our framework. Section 4 lays out the classification of data transfer within applications we developed to categorize the applications described in Section 5, and their interactions in Section 6. Section 7 presents our conclusions and enumerates a few possibilities for future work.

2. Related Work

There has been extensive research in the area of 3D interfaces and workspaces, which we draw upon for our framework's design. The Information Visualizer [1] showed that a 3D workspace provides a larger space for higher density data storage and access, and that the naturally hierarchical organization of such data provides a fast means for users to search and comprehend it. Data Mountains [11] yielded additional results, showing that spatial organization is a critical component of how humans remember and access data. The Task Gallery [10] also showed that spatial organization was important for access, in particular with reference to multitasking - multiple applications organized by task and stored in a 3D environment were easier for users to navigate and interact with than the 2D GUI equivalent. An appropriately expressive 3D environment can make information access much more natural to people, as it better models the tangible environment we are used to working in.

These 3D environments have a variety of different user interaction techniques to select from. There is no shortage of work in the area of developing generic 3D widgets [2, 12] to manipulate a 3D environment in as comfortable a manner as the 2D widgets common in WIMP interfaces. Augmented reality, however, lends itself to more intuitive tangible interfaces where virtual data can be linked to physical objects, as in Windows on the World [4]. This is especially true for projects based on the ARToolkit, as the markers provide excellent physical interaction devices.

There has also been a fair amount of previous work involving inter-application interaction. Modern operating systems often provide means for applications to communicate with each other directly, at a scripting level (eg. Windows DDE [6]). These services provide extremely fine grained control over an application, usually exporting a function per application command (eg. Open, Save, Select All, Copy, Paste, etc.). Using these services always requires some programming knowledge and effort on the part of the user, as well. We provide a much coarser grained level of interaction, with a simple interface that requires no programming knowledge.

The DataTiles [8] system involves small applications which can act independently or communicate with each other at a coarse level of interaction. Some tiles act as con-

trollers of other tiles, while other interactions involve the sharing of content. The system is limited though by the rigid grid placement of applications, which lessens the flexibility of the data transfer visualization - such graphics can obscure the application content or are blocked by other non-interacting applications in the way, and so must be carefully constructed and laid out to minimize these effects.

Another ARToolkit based environment, the Tiles [7] system, links data components to markers placed on an ordinary whiteboard. Though the data tiles themselves do not interact, there are other interaction tiles, like the clipboard or trashcan, which are used to modify the contents of the data tiles, by proximity interaction. Our framework extends this interaction technique in a generic fashion, allowing application developers to create clipboard or trashcan applications, as well as any other applications that use the same style of interaction.

Finally, the IRIS Explorer [5] is a data visualization program builder that uses a visual programming language to construct rendering applications. The intuitive interface involves modules that encapsulate some piece of functionality in a rendering pipeline, which can be connected by drawing lines between them. To facilitate these connections and the development of new components, the modules conform to a generic interface for communication with other connected modules. Our communication model draws from this visual programming concept, but applies it to the more general area of task building by combining applications.

3. Architecture

We implemented our framework on top of ARWin [3], an augmented reality window manager which provides a 3D workspace for applications in a single user environment. ARWin provides the foundation for the rapid development of 3D application and interaction prototypes.

ARWin acts as both a display manager, providing a 3D display rather than X Windows' 2D display, and as a window manager, providing generic application facilities as in TWM, CDE, or Sawfish. The heart of ARWin is an event manager that polls hardware input devices and generates the appropriate events (see Figure 3). One of the core input devices is a firewire camera, which the ARToolkit uses to look for desk-affixed markers registering a global coordinate space where applications may reside, as well as floating markers for tagging to individual applications or physical objects. The user interface enabled by ARToolkit's tracking support makes operating within the environment easy and intuitive for users - applications that are attached to floating markers automatically inherit a tangible interface to control their position within the environment. At the same time, the markers affixed to the user's workspace provide a natural 3D volume to work in, that can be extended simply by

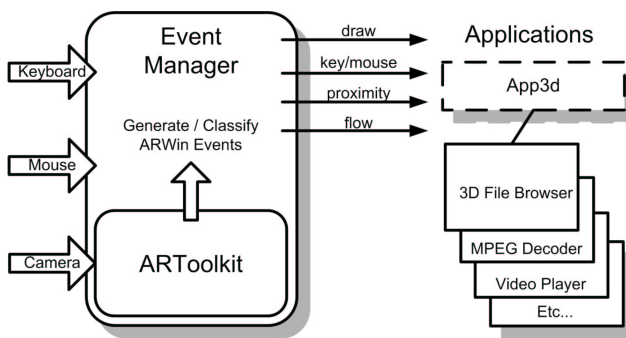

Figure 2. ARWin's Event Manager processes hardware input into events passed on to applications which extend the generic App3d superclass.

adding more markers.

Applications in ARWin extend a common superclass, App3d, which provides callbacks for all the basic events ARWin generates, as well as various control variables that applications can use to set important aspects of their behavior. Common window-manager level application interaction is implemented inside the App3d class, such as window dragging, orthographic vs. oriented view modes, or full-screen mode, while basic manipulation is provided by the tangible interface of the ARToolkit. Applications then override the callback methods to provide their own functionality for whichever events are desired. This allows for easy and rapid development of 3D applications as well as window-manager level services.

Our framework for inter-application communication was added to the App3d class (see Figure 3), providing each application with a vector of DataPort objects and a DataFlow event callback. An application that wishes to provide data streams needs only to create the appropriate DataPort objects and add them to this vector. The App3d class then handles rendering these ports correctly to the user (see Figure 6) and provides a generic interface for connecting ports between two applications to establish a data flow. When a flow is initiated, the target applications receive DataFlow events indicating a new flow being established, and the particular DataPort is specified. The application can then handle writing to or reading from the DataPort via read and write methods that interact with a ring buffer stored in the DataFlow. Thus, the application developer does not need to know anything about a DataFlow to use the port functionality - the DataPort provides the application with a simple API to communicate with the unknown entity on the other end of the connection. Because of this abstraction, DataPorts can be extended to communicate across other channels (eg. across a TCP network socket) without a need for application

modifications.

DataFlows provide a simple means for applications on either end of the flow to retrieve metadata about the flow, to control how they handle the data. This metadata is stored by the DataPort as a list of attribute-value pairs, and can be requested through the DataFlow by specifying the attribute to return the value for. Currently implemented metadata includes the name of the application, the source of the stream, the MIME type of the data being transferred, and the label of the port. A simple example of metadata in action is our raw video player application. Raw video data is a stream of frames, each frame consisting of $width * height * 3$ bytes (for RGB data), but the values $width$ and $height$ are not transmitted as part of the stream. Therefore, the raw video output port has two pieces of metadata (in addition to others), called `width` and `height`, which store the pixel width and height of the video. Then the raw video player can request these pieces of metadata and know how many bytes to read off the stream for a single frame.

Restrictions could be made within ARWin to allow flows to be connected only in particular ways - that is, only video output ports could be connected to video input ports - but we decided the UNIX philosophy of providing just the framework and allowing users and applications the final say in how to deal with data types allows the most flexibility for development and user interaction. This approach does leave open the possibility that a user may unwittingly set up a flow ”incorrectly” (eg. connecting an ASCII text output port to a raw video input port) without any feedback being provided (it is entirely up to the application to provide feedback of this nature, if it chooses), but we consider this to be a necessary evil. The main advantage to this philosophy is that the developer can focus on providing tools to manipulate data rather than trying to predict and account for all the different use cases a user might be interested in. The user then is given the flexibility to combine tools in novel ways that the developer may not have originally thought of.

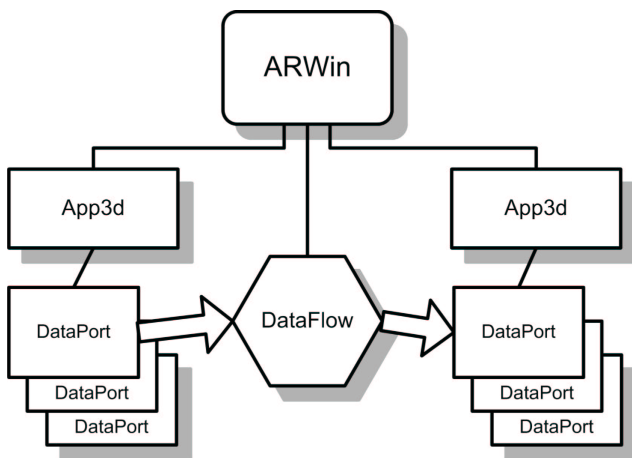

Figure 3. ARWin tracks the active applications and DataFlows. Each flow is connected to two DataPorts, opening a communication channel between the applications.

4. Classifications

To aid in the design and development of applications that take advantage of our data flow framework, we propose the following categories, classifying the various roles an application can assume while participating in a flow. The three main categories applications fall into are data *sources*, data *sinks*, and data *filters* (or mutators).

A data source is the beginning of a data flow. Data sources are applications that generate data without any input from another flow - that is, they read data from hardware (eg. a CD player) or generate data from their own computations (eg. a raytracer rendering a scene). Hardware input devices are the most common example of data sources - the mouse and keyboard are constantly generating data, and the filesystem, via the hard disk, is an extensively used data source.

A data sink is the final destination of a data flow. Data sinks are applications that receive data for processing, but then do not pass on any processed data to other applications. Hardware output devices are good examples of data sinks - a CD burner or sound card receive data and do not return it to the environment for other applications. The filesystem can also act as a data sink - when saving a file from an application, it receives the data and stores it on the hard disk, without passing the data back out for other use.

A data filter is an intermediate step along the flow of data. Data filters are applications that receive data for processing from some flow, alter the data in some way, and then pass it on to the rest of the flow. Data filters do not generate their own data without input, and do not have to have any side effects while processing the data. A good example is a gzip utility, which takes data in, compresses it, and then passes it on (usually to the filesystem).

It is important to understand that applications can generally perform all of these roles, and that the specific way they behave depends on how a user initiates a particular flow. For example an mpeg movie player may receive an mpeg encoded video stream as input and display it to the user, in which case it acts as a data sink. However, the player may also provide raw video and audio outs since it also performed the task of decoding and de-multiplexing the stream, so it can behave as a filter as well.

5. Applications

To demonstrate the capabilities of our framework, we have developed a number of ARWin native 3D applications that utilize DataFlows to extend their functionality. Most of these applications are simply 3D analogues of the traditional 2D applications users have grown accustomed to. Application development is not the focus of this paper, so our examples are fairly basic, but they demonstrate the important functionality of our framework. Of course, as people become familiar with a framework such as ours and an environment such as ARWin, much more novel applications, with more polish and attention to finer details, will be conceived and developed - what we show here is a proof of concept.

To illustrate the interaction framework we developed, imagine a scenario where a user has an MPEG encoded movie file that he wants to watch (see Figure 5). First, he uses the file browser to locate and select the movie file. Then, he connects the file browser output port to the MPEG decoder filter's input port. The MPEG decoder has two outputs, one for raw audio and one for raw video, so the user connects the audio output port to the audio device, and the video output port to the input of a raw video player application. Thus, the user combines a variety of small application modules to perform a more complicated task. Now, we will cover each application in detail.

First, the user selects a file in the file browser – a vital component in any GUI. We have developed one specially designed to work in 3D, based on the Cone Tree [9] abstraction (see Figure 5). Cone Trees are a suitable way to represent large, hierarchical datasets in a 3D environment, such as the filesystem in ARWin. The file browser provides functionality similar to any other - a user can navigate the filesystem, by opening directories and selecting files. One advantage of the cone tree representation is that it allows for easy selection of multiple files in different parts of the hierarchy - it is trivial to make one selection with files from different directories using our browser.

Next, by connecting the appropriate ports via a drag gesture, the user streams the file contents to the MPEG decoder (see Figure 6), an application that converts MPEG encoded video and audio into separate raw video and raw audio streams. Since it produces no output without input, and has no functionality other than generating output, it behaves as a data filter in our classification.

To display the raw video data, the user connects the corresponding output port to the input on our video player. This application reads one frame from the flow at a time, using the flow's metadata to determine the width and height of a frame. The image data is then written to an OpenGL texture and mapped onto a polygon. The video player is a data sink, since it acts as an endpoint for one branch of the flow.

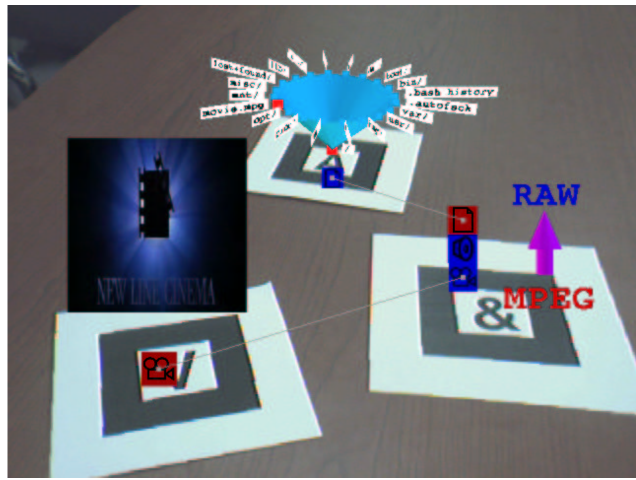

Figure 4. A DataFlow established among three applications. From back to front, the filebrowser has selected a .mpg file and streams its contents to the MPEG decoder, which streams the decoded raw video to the video player, which displays it texture mapped onto a polygon.

To listen to the raw audio data, the user connects the audio output port to the audio device "application". This application acts as a bridge between our framework and the hardware audio device, by providing an interface within ARWin that allows other applications to stream data to the sound card. The audio output device acts as a data sink, since the sound card is an output device, but other device applications can be data sources, like the CD player or keyboard. The advantage of representing hardware devices with wrapper applications within ARWin is that we gain the modularity the framework provides, when dealing with hardware I/O. That is, the MPEG decoder application could have simply written the audio data directly to the sound card itself, but this way, the user can choose among sending the audio data to the sound card, to a CD burner, back to the filesystem for writing to file, or any other data sink.

6. Interactions

Within our framework on top of ARWin, there are two methods for inter-application interactions: proximity events and DataFlows. These two methods handle two fundamentally different types of interactions. Proximity events trigger the one-time exchange of some static data and/or metadata about the application (see Figure 7). Once a proximity event has been triggered and handled, the interaction is complete. DataFlows, on the other hand, address the need for persistent interaction involving the transfer of dynamic data over

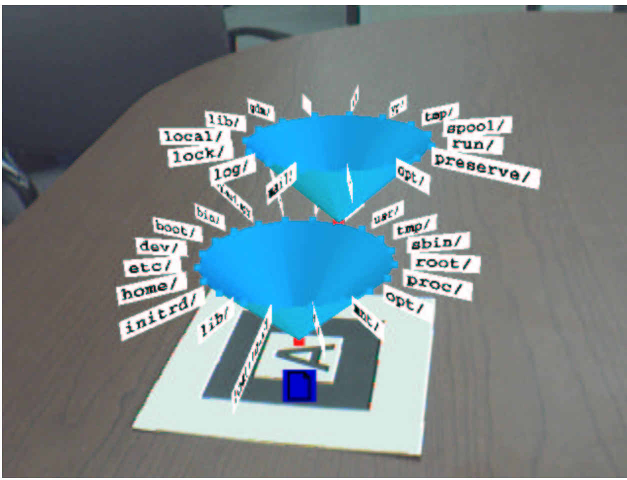

Figure 5. The FileCone file browser. Directories are represented as cones, with their contents distributed around the base of the cone.

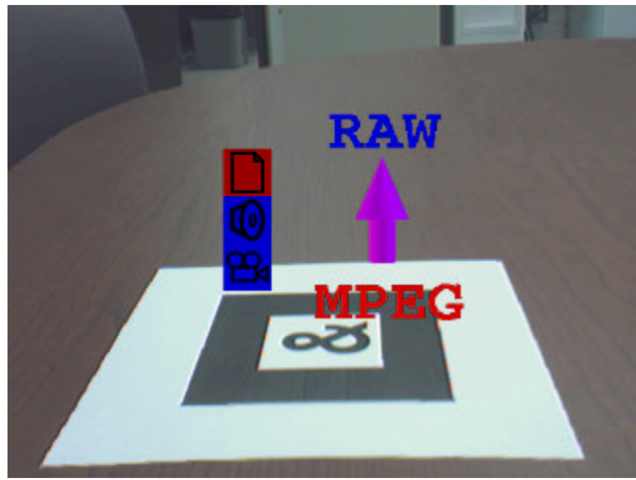

Figure 6. The MPEG Decoder filter application. It has one input port for mpeg encoded video data, and two output ports, one for raw video and one for raw video.

a longer period of time. When a DataFlow is established, it remains active until it is explicitly shut down, either by the user or by one of the interacting applications.

A proximity event is generated when the distance between two applications (calculated from the origin of each local coordinate space) drops below a certain threshold. Both applications are notified that the other is nearby and they have the opportunity to request metadata from each other. The applications are notified again when they leave each other's neighborhood, but it is up to the application developer to decide how to handle the proximity information - they may choose to store the information and continue to interact in some way with other applications even long after they have parted ways. However, the proximity event is best suited for quick one-time transfers.

As a persistent interaction technique, a DataFlow must be explicitly created and destroyed. Of course, this can be done automatically by the application rather than by direct user input - for example a CD player application could automatically establish a flow between the CD device, the player app, and the audio output device, without the user needing to specify each link. For DataFlows that are not automatic in nature, however, the user must establish them manually. To do this, applications provide a set of DataPorts, which ARWin visualizes as a row (or column) of icons floating next to the application geometry - a sort of DataPort palette. Ports displayed in red are inputs and blue are outputs, and the icon provides a graphical representation of the data type the port can transmit or receive. The user can establish a flow from one port to another by clicking on an output port on one application and dragging it to an input port on another application. This creates a DataFlow object within ARWin, and notifies each application of the new flow and the selected ports. The applications are once again free to handle this event as they choose, most likely by beginning to write data to or read data from the ports. The established DataFlow is represented by a line between the two ports. There are a variety of ways the flow can be destroyed - the user can select the flow and manually delete it with the backspace key, the user can quit one of the interacting applications, or the applications can decide they are done transferring data and close the flow automatically.

These two different modes of interaction can be used in conjunction as well - that is, the proximity event can be used to establish DataFlows between neighboring applications, which would then persist after the applications are moved away from each other. Applications can tag one input or output port as their "primary" port, so that then two applications are brought near each other, a DataFlow can immediately be established between their primary ports. This makes sense for applications that have one main data transfer capability, as well as some auxiliary functions that are much less frequently used.

It is worth noting that the DataPorts provided by an application are not static - that is, during the course of the application's execution, it can add or remove ports from its set of capabilities. This can be used to provide useful state-specific functionality. For example, in a text editor application, when the user selects some text, a selection port can be created, which transfers the selected data to a connecting application. When the text becomes unselected, the port is destroyed. By making such a selection port the primary port

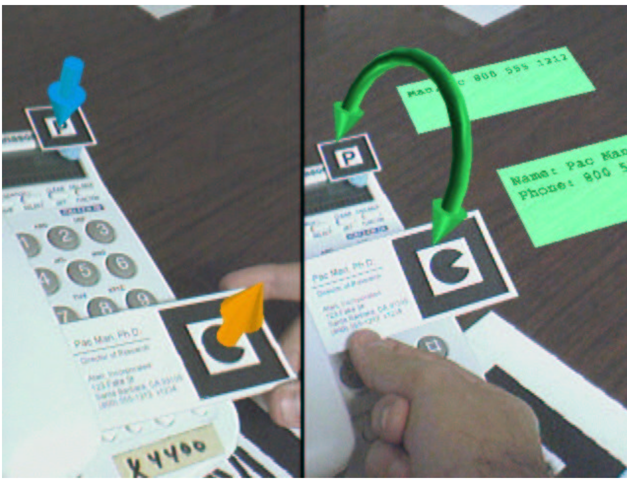

Figure 7. A proximity interaction between a business card application and a phone book application. The phone number information on the business card is transferred for storage to the phone book.

when it is created, applications have a simple way to implement clipboard functionality - the user selects some data, and then moves the application near the application to paste to, and the data will be transfered via the proximity event that establishes a connection between the applications' primary ports.

7. Conclusions and Future Work

We have developed a generic framework which allows programmers to easily create inter-application interaction functionality within ARWin using the ARToolkit. We have provided some example applications which utilize this framework in a few ways, and as application programmers become comfortable with the idea of extensive inter-application interaction, much more complicated interactions will be realized.

Our framework is limited by the fact that it only works within ARWin, which means its success depends on ARWin's success, and the success of 3D workspaces and applications in general, but hopefully with time these environments will become more commonplace. As traditional applications need to be rewritten to take advantage of 3D functionality, adoption of 3D interface paradigms will be slow, until techniques are found to easily retrofit these applications with the new capabilities. Certainly, work could be done to make integration of 3D functionality into traditional 2D applications easier. A simple interface to 3D interface toolkits that requires as little modification of legacy code as possible would encourage the augmentation of traditional 2D applications with 3D functionality.

ARWin currently has a major limitation that can be a hindrance to more sophisticated development - to allow display manager and applications to share a single OpenGL context, everything is implemented as a single monolithic process. Applications are compiled in to ARWin and everything is run in a single thread. This can make prototyping applications relatively quick, but fundamentally limits the capabilities of the entire system. We are working on extending ARWin to support multiple processes and a shared memory architecture.

This new framework leaves many avenues open for future work. The most obvious next step would be the development of more sophisticated applications that use the framework in novel ways. Our applications serve as a proof of concept, but they do not cover many of the use cases computer users regularly encounter. A wider range of applications would better show just how much inter-application interaction can enhance computer use.

For these applications to communicate, our framework currently requires the explicit connection of data ports. Some flows can be established automatically, but not in an intelligent way - the responsibility still rests with the user to supervise the proper connections between applications. However, it's possible that applications could intelligently establish flows by examining the MIME types of the ports between two neighboring applications (spatially or temporally), to find appropriate matches. Based on ports already explicitly established, the available meaningful connections an application can make are greatly diminished and so an intelligent automatic selection is possible.

An exploration of additional inter-application interaction techniques would also be beneficial. The two modes of operation we have discussed in this paper, proximity and DataFlows, are just two of the possibilities in this area of application interfaces, both concerned with the one-way transfer of data. Another interaction possibility is the transfer of generic control information. Given a generic language for controlling an application's actions, developers could write controller applications in a generic way so they could be connected with unknown applications to control their functionality. Interaction techniques like control flows would further show the powerful nature of inter-application interaction, providing a more compelling case for its adoption into daily computer use.

When using ARWin with our framework, it becomes clear that our display widget for DataFlows (drawing a line between the connected ports) is inadequate for our needs. A straight line is problematic because it often passes across an application, which can be annoying. It also provides no useful information about the type or amount of data being transferred across the link. And while some links may be interesting to be visualized by the user, not all links are, and

their display clutters the environment with information the user does not care about. So, there is significant work to be done to figure out what is the best way to visually represent these inter-application links, and how to determine when they should be displayed or hidden. This could also make the environment much more aesthetically appealing.

References

[1] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer: An information workspace. In *ACM CHI*, pages 181–188, 1991.

[2] D. B. Conner, S. S. Snibbe, K. P. Herndon, D. C. Robbins, R. C. Zeleznik, and A. van Dam. Three-dimensional widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics, Special Issue of Computer Graphics*, volume 26, pages 183–188, 1992.

[3] S. DiVerdi, D. Nurmi, and T. Höllerer. ARWin - a desktop augmented reality window manager. Poster submission, currently under review. Extended version available as UCSB-TR CSD-03-12, Department of Computer Science, University of California, Santa Barbara, April 2003.

[4] S. Feiner, B. MacIntyre, M. Haupt, and E. Solomon. Windows on the world: 2D windows for 3D augmented reality. In *ACM UIST*, pages 145–155, Atlanta, GA, Nov. 3–5 1993.

[5] D. Foulser. IRIS explorer: a framework for investigation. *ACM SIGGRAPH Computer Graphics*, 29(2):13–16, 1995.

[6] Microsoft Corporation. *Microsoft Windows Guide to Programming*. Microsoft Press, Redmond, Washington, 1990.

[7] I. Poupyrev, D. Tan, M. Billinghurst, H. Kato, H. Regenbrecht, and N. Tetsutani. Developing a generic augmented-reality interface. *Computer*, 35(3):44–50, Mar. 2002.

[8] J. Rekimoto, B. Ullmer, and H. Oba. DataTiles: a modular platform for mixed physical and graphical interactions. In *ACM CHI*, pages 269–276, 2001.

[9] G. Robertson, J. Mackinlay, and S. Card. Cone trees: Animated 3D visualizations of hierarchical information. In *ACM CHI*, pages 189–194, 1991.

[10] G. Robertson, M. van Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel, and V. Gorokhovsky. The task gallery: a 3D window manager. In *ACM CHI*, pages 494–501, Apr. 1–6 2000.

[11] G. G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich. Data mountain: Using spatial memory for document management. In *ACM Symposium on User Interface Software and Technology*, pages 153–162, 1998.

[12] D. Schmalstieg, A. Fuhrmann, and G. Hesina. Bridging multiple user interface dimensions with augmented reality. In *IEEE/ACM ISAR*, pages 20–29, Oct. 5–6 2000.