Future Generation Computer Systems 27 (2011) 914–923

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Enabling multi-user interaction in large high-resolution distributed environments

Ratko Jagodic [a,*], Luc Renambot [a], Andrew Johnson [a], Jason Leigh [a], Sachin Deshpande [b]

[a] Electronic Visualization Laboratory, University Illinois at Chicago, United States

[b] Sharp Labs of America, United States

ARTICLE INFO

Article history:
Received 7 March 2010
Received in revised form
30 July 2010
Accepted 12 November 2010
Available online 21 November 2010

Keywords:
Human factors
Input devices and strategies
Interactive systems
Distributed systems
Distributed graphics
Collaborative computing

ABSTRACT

As the amount and the resolution of collected scientific data increase, scientists are realizing the potential benefits that large high-resolution displays can have in assimilating this incoming data. Often this data has to be processed on powerful remote computing and storage resources, converted to high-resolution digital media and yet visualized on a local tiled-display. This is the basic premise behind the OptIPuter model. While the streaming middleware to enable this kind of work exists and the optical networking infrastructure is becoming more widely available, enabling multi-user interaction in such environments is still a challenge. In this paper, we present an interaction system we developed to support collaborative work on large high-resolution displays using multiple interaction devices and scalable, distributed user interface widgets. This system allows multiple users to simultaneously interact with local or remote data, media and applications, through a variety of physical interaction devices on large high-resolution displays. Finally, we present our experiences with using the system over the past two years. Most importantly, having an actual working system based on the OptIPuter model allows us to focus our research efforts to better understand how to make such high-resolution environments more user-friendly and usable in true real-world collaborative scenarios as opposed to constrained laboratory settings.

1. Introduction

In recent years, we have seen a steady increase in the resolution and the amount of collected data that needs to be analyzed or presented. Digital media produced are of higher quality and resolution. Often, the complexity of the problems that drive this increase requires that multidisciplinary teams work collaboratively towards a solution. Up until the last several years however, the predominant model for supporting such data-intensive collaborative science and engineering involved replicating and co-locating the computing resources with the storage systems, and then transmitting the highly distilled results of the computation to expensive remote visualization systems at each collaborator's site. This was because bandwidth was a scarce commodity and so it was more efficient to co-locate the data with the computation than to move the data, on-demand, to the computation. Recently, with rate of decline of the cost of bandwidth far exceeding that of computing and storage, it has become viable for scientists to connect to ultra-high-speed networks, more cheaply than they can build and maintain large computing, storage, and visualization systems. This is the fundamental premise behind the concept of shared cyber infrastructure as outlined in the OptIPuter project [1].

Furthermore, the OptIPuter project envisions that in the future it will become routine for users to work and collaborate in rooms whose walls are made from seamless ultra-high-resolution displays. Such environments empower users to better cope with the increase in scale and complexity of problems and data. We are already seeing this trend materialize as users are adopting large high-resolution displays as their visualization systems. There are several benefits of large high-resolution displays when compared to the contemporary desktop environments. The high resolution enables considerably more information to be displayed simultaneously, which reduces context switching and enables juxtaposing of relevant pieces of data for direct comparison. This allows, for example, scientists to use such displays as visualization instruments to gather a deeper understanding of their problems [2]. On the other hand, the large size of such displays makes them an excellent collaborative environment, promotes a more natural physical navigation and improves spatial performance when analyzing high-resolution datasets [3].

Such a paradigm shift requires significant changes in the supporting software as well as hardware infrastructure. Recent display and networking advances have enabled us to build such large high-resolution environments by tiling projectors (for instance, 4K digital cinema projectors) or LCD displays (full high-definition resolution or 4 megapixel panels) that are driven by

* Corresponding author.
E-mail address: rjagodic@gmail.com (R. Jagodic).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.11.018

a cluster of computers and are connected to remote computing and storage resources using high-speed networks. However, no software infrastructure exists that will fully exploit the affordances of the hardware and enable the type of distributed, collaborative working environment as envisioned by the OptIPuter project. As a first step towards this vision, we have designed and developed the Scalable Adaptive Graphics Environment (SAGE) [4]. SAGE is a middleware that turns any combination and configuration of displays driven by a cluster of computers into a seamless, fully distributed environment where windows of locally or remotely rendered applications can be moved and resized as if they were on a single local desktop. However, SAGE itself is only a middleware that manages delivery and display of remotely rendered application pixels across tiled-displays and as such is of little value without user interaction capabilities. In this paper, we present the interaction system for SAGE that enables truly multi-user and scalable user interaction with distributed data in large high-resolution display environments, effectively turning the OptIPuter vision into reality.

The contributions of our work can be summarized as follows:

First, we identify and describe fundamental design aspects of an interaction system that are the basis for the type of distributed collaborative work as envisioned by the OptIPuter project. Additionally, we highlight the reasons behind each aspect and explain why existing systems are not sufficient to fully enable these novel working environments.

Next, we present our implementation of the interaction system that satisfies these aspects. Together with SAGE, this makes it a functional instantiation of the OptIPuter model, which is ideal for empowering collaboration in distributed, large high-resolution display environments for the purposes of further research of potential applications and human factor issues. This would not be possible using existing interaction systems since they do not fully exploit the affordances.

Finally, we present our experiences with using the system over the past two years. In part, our experiences support the identified design aspects of the interaction system, but more importantly, they are a part of our ongoing research effort to better understand how to make such high-resolution environments more user-friendly and usable in true real-world collaborative scenarios as opposed to constrained laboratory settings. The existing SAGE community will gain immediate benefit from our experiences and the resulting improvements to the system however, we believe that our current and future research in human factors will provide a clearer picture of how to design and build such advanced collaborative display environments.

2. Related work

High-resolution displays are becoming more widespread in scientific laboratories [5,6], and will eventually become pervasive at the office, in the cubicle, the meeting room [7], and ultimately at home [8]. The OptIPuter project [1] explored the use of high-resolution environments for scientific discovery under the prediction that seemingly unlimited amount of network bandwidth delivered by progress in optical networking would change the way science is conducted between geographically distributed teams. Applying this concept to cinema-quality motion pictures for entertainment, science and education, the CineGrid initiative defines itself as trying to "provide media professionals access to global cyber infrastructure capable of carrying ultra-high performance digital media using photonic networks, middleware, transport protocols and collaboration tools originally developed for scientific research, visualization, and Grid computing" [9]. However, these efforts focus mostly on building the infrastructure and delivering the content and do not address multi-user interaction and human factors. Ultimately, the hardware is only half of the equation and it requires a matching software infrastructure and appropriate user interaction techniques to fully take the advantage of the bleeding-edge technology affordances.

Previous research has pointed out that traditional interaction methods do not scale well to large high-resolution environments and that new, more appropriate methods are needed [10,11]. For example, there has been significant research effort aimed towards addressing the issue of target acquisition across large distances and high resolutions [12–16]. However, the question remains whether and how these techniques work outside the controlled laboratory setting with real-world users and applications. Additionally, most of these techniques and applications were developed with single users in mind and for single-computer environments. Although several real-world applications of large displays have been demonstrated [17–20], they are highly specific to the application domain and focus on collocated work, often without true multi-user interaction capabilities.

The ubiquitous computing community on the other hand, has put more focus on distributed systems. Typically, that line of work involved integrating multiple displays and devices into a unified environment where users can interact with the system [21,22]. However, since such environments are still desktop oriented, they often impose the limitation that there can be only one active user per display or application [23]. While conceptually our goals were similar, our target environment was a seamless large high-resolution desktop where remotely rendered applications are not displayed on a single screen but rather are freely movable and resizable across any number of displays. Additionally, we must enable simultaneous multi-user interaction with all the applications on the display.

Numerous widget frameworks already exist for desktop environments, such as Qt or wxWidgets. However, these are designed for desktop operating systems and therefore inherit their limitations. For example, multi-user interaction typically is not possible and they are often tightly integrated with the operating system's device events (mouse and keyboard). This low-level coupling with the existing single-user operating systems would make it difficult to adapt the toolkits to work with multiple users simultaneously (i.e. interaction devices). Though toolkits built on top of X Window System are distributed, their typical single client–single server setup does not translate to an arbitrary number of clients and servers (e.g. a parallel application being displayed remotely on a tiled-display driven by a cluster of computers). Lastly, they are not scalable and are therefore inappropriate for displaying on very high-resolution displays where visibility and usability becomes an issue. For instance, a 100 megapixel display with a touch screen input device requires different widget parameters than a 10 megapixel display with a traditional mouse.

We found that the area of concurrent multi-user interaction in distributed large high-resolution environments is largely unexplored, even though recent trends identify the need for it. However, in order to make advances in this area, appropriate software infrastructure has to exist that will support the types of interaction scenarios as envisioned by the OptIPuter project. This gap in the available software infrastructure motivated us to design and develop SAGE and ultimately the interaction system presented in this paper upon which further research in human factors and applications can be pursued.

3. Previous experiences with SAGE

Before the direct interaction layer for SAGE, we were limited to a cross-platform desktop-based interface (SAGE UI). Every user can

Fig. 1. During our weekly technical research meetings all students show their laptop screens on the tiled-display to share ideas and promote discussion.

run a copy of the interface on their laptop and connect to the tiled-display in order to perform basic window manipulations (resize and move) and to start and stop applications. Additionally, each user can show their desktop on the display through a VNC plugin or drag-and-drop multimedia files from their laptop. Numerous complex technological demonstrations were done using this model [24], between various sites around the globe. However, the complexity of the system drove us to focus on the user perspective to make these environments successful.

3.1. Weekly meetings

For more than 2 years we have been using SAGE during our weekly technical meetings where every participant would simultaneously share their laptop screen with the group by showing it on our 100 megapixel tiled-display. At the beginning of the meeting, every user would position their small window along the periphery of the display using SAGE UI (Fig. 1).

This was by no means required but instead seemed to be a natural layout that we implicitly agreed upon not long after we started using the display in our meetings. This created a large empty space in the middle of the display that is typically used for maximizing the window of the person currently speaking. We noticed that users rarely reposition or resize other people's windows even though no such restrictions were imposed by the system. This behavior further supported our hypothesis on the importance of simultaneous multi-user interaction where every user has the ability to control the display. Since all the windows are already present on the display, switching between speakers is only a matter of bringing the correct window into focus by resizing and repositioning it using SAGE UI (as opposed to switching video cables in a more traditional single-display environment). This proved to be a significant benefit for the dynamics of the meetings because it encouraged discussion and information sharing. Often, there was a need to bring multiple windows into focus for discussion, which usually resulted in one window being maximized while other supporting windows are enlarged on the side. However, during the discussions, there is generally a need to point at the display, which had to be done by either walking up to the display or passing a laser pointer between the participants. This suggested that giving every user a more direct interaction capability, beyond the desktop-based SAGE UI, is needed.

3.2. Class study session

Besides our regular meetings, an anatomy class professor at our university wanted to hold an exam study session using the display. Numerous medical images, pertinent to that day's discussion, were preloaded and shown on the display before the beginning of class.

Fig. 2. The anatomy class study session where students were asked to answer questions using the displayed images on a 100 megapixel tiled-display.

The students gathered around the display and the professor would spur the discussion by asking questions that required the students to use the appropriate image on the screen for answering the question. The student that made a first attempt at answering usually ended up walking up to the display to point out the details and explain his/her answer (Fig. 2). Meanwhile, the professor had to remotely point at the display in order to guide the student or to raise specific questions. While this was easily accomplished using a laser pointer, manipulation of the windows on the display was still mediated for the lack of preparation time before the class and because not every student (nor the professor) had a laptop during the class. This, again, supported the need for the direct interaction since one could envision many such use cases where a desktop-based user interface is not appropriate. Furthermore, that interface had to be intuitive and with a small learning curve to encourage a more casual use of the display by non-expert users. The manipulation actions typically involved bringing a window into focus by either maximizing it or manually setting size and position. The students and the professor provided very positive feedback emphasizing the benefit of having all the materials present on the display at once, which allowed them to externalize their working memory. In a more traditional environment, the students would have to open multiple textbooks at different pages in order to view all the images that they otherwise had on the display in front of them.

3.3. SAGE community

The current SAGE user community keeps growing as more researchers recognize the potential that tiled-displays could have on advancing their own line of work. Currently this includes 40 sites with industry partners (e.g. Sharp Laboratories of America), universities (e.g. National Center for Microscopy and Imaging Research, UC San Diego) and international partners (e.g. Space Research Institute of Russian Academy of Sciences) [25]. Most partners have also expressed interest in the direct interaction capabilities based on their own experiences with the desktop-based user interface. However, since the resolution of their displays varies greatly, from 10 to over a 100 megapixels, the visibility and usability of the user interface would need to adapt to the target environment.

Some applications are already included with SAGE, such as 4K movie player (next-generation digital cinema resolution at 8 megapixels per frame), multimedia viewer, desktop-sharing tool, HD video conferencing tool, ultra-high-resolution map viewer and volume visualization tool. However, because it is reasonably easy to develop new applications or port existing ones to SAGE, the community has also contributed their own. Previously, application interaction was only possible from the application's desktop user interface that is typically displayed on the machine that renders the application. However, that is usually inconvenient

because applications are typically rendered on remote computing resources that users do not have access to. Therefore, there exists a real need to interact with these applications directly from the display regardless of whether that application is rendered locally or remotely. Furthermore, since the same application could be shown on various displays, its user interface would have to scale appropriately for the size and the resolution of that display.

4. Design aspects

Based on our preliminary experiences and the feedback from the current SAGE user community, we identified the following fundamental design aspects that we believe are necessary in order to fully exploit the affordances of the high-resolution display devices and the cyber infrastructure. Each of the aspects is called for by one the characteristics of the hardware infrastructure developed in the OptIPuter project. This grounding is clarified below.

4.1. Distributed

The central architectural element of the OptIPuter is optical networking, which provides access to remote data that is frequently faster than accessing it from a local hard drive. Furthermore, instead of replicating the data, it is often easier to have the data rendered remotely and simply stream the results to the visualization endpoints. One of the main strengths of SAGE is its transparently distributed nature that allows precisely this kind of work, which is what makes it so appealing to the growing user community. Logically, the interaction system should follow the same principle. In other words, it should allow for display of user interface elements across any number of displays driven by any number of machines. Additionally, application interfaces should be visible on any remote display without the application actually being aware of this rendering-display separation. Similarly, any physical interaction device should be able to control any remote application without being concerned with its rendering location.

4.2. Scalable

As the amount and the resolution of collected scientific data increases, users are realizing the potential benefits that large high-resolution displays can have in assimilating this incoming data. However, since such displays are assembled as tiled LCD panels or projects, their size and resolution can vary significantly depending on the target application. As the SAGE users community survey unveiled, users in fact do have displays of vastly different sizes and resolutions, ranging from displays in offices to large collaborative spaces in meeting rooms and public spaces. Therefore, it is imperative that the interaction system automatically adapts to the target display size and resolution. This adaptation is necessary from both, the visibility and the usability perspective. For instance, the physical interaction devices should adapt their speed and sensitivity (control-display gain) and the interactive objects should adjust their size to be visible and be easily target by the devices.

4.3. Multi-user

In recent years, we have seen a steady increase in the resolution and the amount of collected data that needs to be analyzed or presented. Often, the complexity of the problems that drive this increase requires that multidisciplinary teams work collaboratively towards a solution. As we personally experienced, large high-resolution displays are well suited for collaborative work since their large size easily accommodates multiple users and their high-resolution allows much more content to be displayed simultaneously. Therefore, to take advantage of these affordances, it will be necessary to support concurrent multi-user interaction. This interaction could be on the same display or even within the same application. Moreover, the applications themselves should be unaware of how many devices are currently interacting with its user interface.

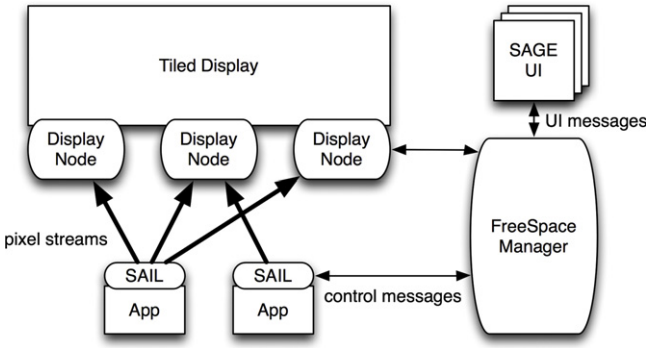


Fig. 3. Main components of SAGE.

4.4. Physical interaction device independence

The novelty of large high-resolution displays and their significantly different interaction affordances creates the opportunity for investigating various physical interaction devices and input modalities. Even though the mouse is the de facto standard for desktop systems, in most cases it is inappropriate for large high-resolution displays because it confines the user to a surface. The research community has introduced many novel and promising interaction devices however, no single device has emerged as the clear winner for the large high-resolution displays. Therefore, we deemed it necessary to have an easy way of integrating new devices into the system without requiring changes to the applications or the display interface. Furthermore, depending on the display size, resolution and room configuration, some devices may be more appropriate than others.

5. SAGE overview

In order to better understand the interaction system, this section provides a brief overview of SAGE and its components (Fig. 3). A thorough description of SAGE can be found in [4].

As mentioned earlier, SAGE allows for seamless display of various remote applications on ultra-high-resolution displays. Each application gives its rendered pixels to the SAGE Application Interface Library (SAIL) that streams them to the appropriate Display Nodes depending on the current position and size of the window on the display. Each Display Node can receive and display multiple pixel streams independently to allow multiple applications to be shown concurrently on one display. FreeSpace Manager (fsManager) is the main component of SAGE that keeps track of the current display parameters and the arrangement of the application pixels on the display. Based on the requested arrangement, fsManager directs SAIL to split up application's pixels and send each section to the appropriate Display Nodes. The only exposed fsManager commands for user interaction are moving, resizing and closing application windows, which have to be issued through a network interface. However, since fsManager already had network connections to all the components, it was leveraged for the interaction system presented in this paper.

6. Direct Interaction Manager (DIM)

DIM is the core of the interaction system that contains the following components: Device Manager to manage physical

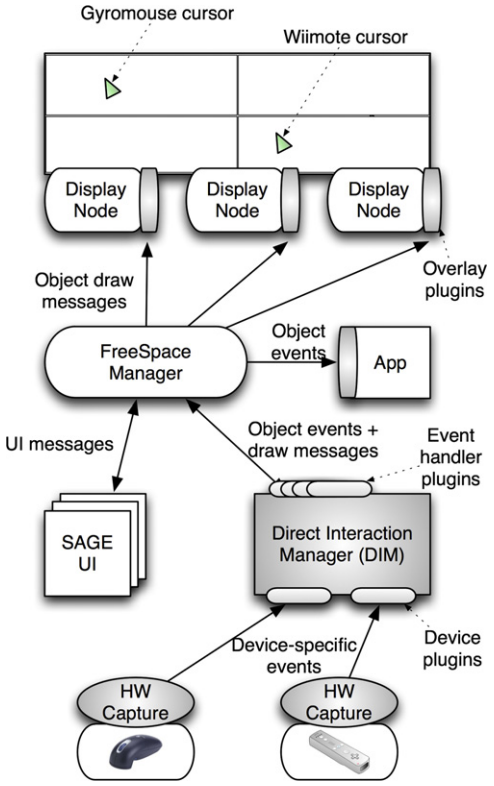


Fig. 4. Direct Interaction Manager integrated with SAGE. It manages physical interaction devices, performs event handling for events from the devices, manages the logic and drawing of objects and delivers events to applications. The components in shaded gray are all a part of the Direct Interaction Manager.

interaction devices, Event Manager for delivering device events to appropriate interactive objects and the Overlay Manager which manages the drawing of objects on the display.

6.1. Device Manager

To successfully enable multi-user interaction, the Device Manager accepts events from multiple physical interaction devices through a network interface. If multiple devices are used in a system, especially if they are of different types, it is often the case that they are connected to multiple computers receiving their events (e.g. a special machine for the camera tracker), which necessitates network-based communication. A simple Hardware Capture (HWcapture) component contains plugins for each device that captures the device-specific events, which are then delivered to the Device Manager (Fig. 4). The event delivery (messaging) library is already provided so developers only need to provide the plugin to capture device-specific events. To provide robust service, the Device Manager will dynamically add new devices and remove old ones if they fail or disconnect. Allowing devices to connect or disconnect at any point in time, essentially allows users to freely join and leave the space as they please, which mimics traditional paper-based meeting environments.

So far no single device has emerged as the best tool for large high-resolution display environments even though studies have been performed using laser pointers, hand gestures, touch screens and traditional mice. Therefore, we found it necessary to allow the use of a wide variety of devices even though they may generate different events. When these device-specific events are delivered to the Device Manager, they are passed onto a device plugin that describes the conversion of device-specific events into a generic set of events that are then further used within the system. This effectively makes all the devices appear equal in view of the interaction system and therefore allows new types of devices to be added without changing the user interface objects or applications themselves. Currently, we already include support for the following devices: mouse (and Gyromouse), joystick, Wiimote, two different touch screen implementations and a 6 DOF Ascension electromagnetic tracker. In order to add support for a new type of device, the following two steps are necessary:

- Create a HWcapture plugin that grabs device-specific events from the physical device (e.g. touches from a touch screen server).
- Create a device plugin to DIM that describes the conversion between device-specific and generic events (e.g. a touch to EVT_CLICK).

Naturally, many off-the-shelf devices actually behave like a mouse. However, the question arises as to how one captures the events from each Gyromouse independently since all the devices will fight over the control of a single operating system cursor. This means that if we wanted to use three mouse-like devices (e.g. Gyromouse), we would have to have three different computers, one device per computer. To get around this operating system limitation, we use a tool called GlovePIE [26], which can distinguish between the different physical devices connected to the same computer. However, since GlovePIE uses its own scripting language and does not offer any sort of networking capabilities, there was no direct way for us to send these captured events to DIM. While operating systems themselves cannot distinguish between different mice, they can easily support multiple joysticks. Therefore, we convert Gyromouse events to virtual joystick events and capture those from a HWcapture plugin for a standard joystick. The rest of the event conversion simply occurs as if the Gyromouse were a joystick.

6.2. Event Manager

As mentioned previously, all device-specific events are converted to a generic set of events, which in turn allows heterogeneous devices to interact with the display equally. Naturally, some devices are more powerful than others, so they can generate a larger subset of the generic events. These generic events are then put in an event queue where the Event Manager processes them in order, trying to find an appropriate event handler for each. If an event handler is found the event is passed onto the handler, otherwise the event is discarded. Event handler is the base class for any interactive object that wants to receive events. Essentially, it is a rectangular container that has size and position and depth parameters, knows how to receive events and it contains other properties common to all widgets (e.g. tooltips, labels, visibility flags). There is also an event handler for every application, which will deliver interaction events to the actual application, whether it is local or remote (for instance, a dragging event can be delivered to a map application). Since the applications can be freely resized and repositioned on the display, all the event coordinates are normalized before being sent to the applications, as they are unaware of their own window position and size.

While an event handler is processing an event, the Event Manager will lock it in order to prevent competition from different devices (for example, if a button is held down or something is being dragged). However, no such restrictions are imposed between different event handlers, which allows for true multi-user interaction.

6.3. Overlay Manager and drawing

Overlay refers to any visible object on the display. The label "overlay" is used because SAGE itself draws only application pixels on the display so any other object is actually an overlay drawn on top of the application pixels. If an event handler is drawn on

the display, it is also an overlay. On the other hand, objects such as cursors are just overlays and not event handlers because they do not react to any events. The actual drawing of each overlay is performed by the appropriate overlay plugin to the Display Nodes that describes how the overlay should be drawn in certain visual states (e.g. button is down or up). Each plugin is a subclass of a base overlay class, which contains basic draw parameters (depth ordering, transparency, size, position, visibility …) and contains a generic draw method that each object has to implement. Draw methods provided by plugins are actually just snippets of OpenGL code, which are called at the appropriate time during the draw loop. As event handlers receive events from devices, they issue drawing commands that the Overlay Manager then delivers to the appropriate overlay plugin on each Display Node.

7. SAGE widgets

With the introduction of DIM, we had a functional, though very limited, interface for SAGE. It was possible to move and resize application windows and simple events such as pan, zoom or click could be forwarded to the application (for traversing maps for example). However, there was no way for the application to present a user interface on a remote display, short of drawing it manually into its own application buffer that is streamed to SAGE. Consequently, we developed a unified widget framework, which allows all applications, and the display itself, to use the same set of widgets without dealing with the event handling, the drawing, the interaction devices being used or the distributed nature of the environment. Currently, several basic widgets have been implemented: buttons, icons, menus, labels, thumbnails, panels and sizers.

One may argue that application developers will be hesitant to redesign their applications to use these newly developed SAGE widgets. While this may be true in some cases, it is also important to note that typically, current user interfaces were developed for desktop systems and therefore are not usable on a much larger and higher resolution displays. This is primarily because they rely on the accuracy of the mouse to acquire smaller targets. However, the mouse is not appropriate anymore for these new environments since it confines users to a hard surface whereas users often want to be mobile in front of the display. Additionally, the applications themselves are often custom developed for the large high-resolution displays, again because current desktop applications rarely take advantage of the vastly increased display resolution available to them. Therefore, these custom applications are often still in development when the cost of adding a more suitable user interface is not tremendous.

7.1. The anatomy of a distributed widget

The widget framework was designed to be compatible with the existing DIM architecture where each widget is actually a collection of several distributed elements.

- An event handler plugin to DIM. This component contains the logic, the API for creating this widget and all the parameters of the widget. It also listens for device events and decides how to act on each.
- An overlay plugin to Display Nodes. This component contains the drawing code for the widget for all possible states (e.g. how to draw a button when it is pressed or not).
- Widget stubs plugin in SAIL. This small component allows the application to create the widget through a simple API (same API as the event handler plugin). It also keeps track of and initiates widget event callbacks that the application is interested in.

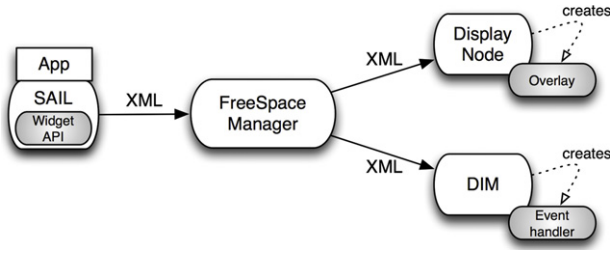


Fig. 5. The process of creating an application widget, a button in this case.

The Event Manager then ensures the proper routing of events from devices to widgets (event handler) and applications, and the Overlay Manager handles the routing of drawing commands from the widget to the proper overlay plugin on the Display Node that actually draws the widget. This plugin-oriented architecture ensures that we can easily add new widget types to the system. Therefore, in order to create a new type of widget, one has to write the three plugins described above. While it may appear complicated, the process is in fact the same process as writing a custom widget in any of the desktop-based widget toolkits. The only difference is that the code for the widget is split across three different locations as opposed to only one. Nevertheless, this has certain benefits. For instance, we can change the logic or drawing of widgets without recompiling applications, as long as the API stays the same.

7.2. Widget creation and widget types

Even though all the widgets, events and devices are managed by DIM, widgets can be created by DIM itself or by the application through a simple API that is similar to desktop-based wxWidgets framework. Widget parameters (e.g. size, position, label) are set using the API, which then creates an XML description of the widget. This XML description represents the widget in the network communication between DIM, the application and the Display Node. When a widget is created by the application, the widget stubs plugin exposed in SAIL only generates appropriate XML file and executes event callbacks as the events are received from DIM. The rest of the widget functionalities were all carried out by the event handler plugin to DIM (i.e. logic, event handling) or the overlay plugin to a Display Node (i.e. drawing), as described in the previous section. For instance, if a widget is created by the application, its XML description will be delivered to DIM, in order to create the event handler for it, and to the Display Node in order to create the overlay for it (Fig. 5). Entities that create the widgets can also register callbacks to receive widget events when they take place (e.g. button click, menu selection). These events will be automatically delivered to the appropriate event callback even if the widget was created by an application running on a distant rendering cluster.

Global widgets—created by DIM, one set of those exists for each display environment and make up the main display user interface. Examples would be some icons on the desktop or a main menu for starting applications. Global widgets are positioned relative to the display bounds and are resized appropriately for the current display size. Also, note that typical widget toolkits (and operating systems) do not allow you to place widgets on the desktop directly whereas in our framework that is a non-issue since SAGE itself takes the role of a window manager for tiled-displays.

Application widgets—created by the application. This is the most common way of using widgets in modern widget toolkits. The application developer uses these widgets for creating a user interface for the application. These widgets are always positioned

Fig. 6. Application widgets are scaled relative to the application window (player controls) while global widgets are scaled relative to the display size (maximize and close buttons).

and resized relative to the application itself in order to keep their interface consistent with the designer's intentions (Fig. 6).

Per-application widgets—created by DIM, one set of those exists for every application on the display. These are always positioned relative to the application (they follow them around). For instance, as mentioned earlier, there is an event handler (i.e. application window widget) for every application. This widget is always the size of the application pixels as drawn by SAGE and positioned so that it exactly covers them. It is completely transparent except for the borders. Then, by receiving events from devices, we are able to manipulate the application pixels as if it were an actual window in a window manager (e.g. EVT_DRAG that is delivered to this application window widget, will move the window around on the display). The size of the per-application widgets can be relative to the display bounds, as in global widgets, or relative to the application itself, as in application widgets.

7.3. Event handling scenario

Fig. 7 shows the propagation and event handling for a hardware event, in this case a mouse click. First, a mouse click occurs on some remote machine and a hardware specific event is generated. The hardware specific event is sent over the network to DIM where the Device Manager loads an appropriate device plugin for this particular interaction device, a mouse in this case. The plugin knows how to convert the hardware specific event to a generic event, EVT_CLICK. The Event Manager in DIM then attempts to find the event handler that is listening for this type of event. In this case, a button event handler plugin is found at the current event position and the event is passed onto it. The event handler determines that the click event altered the button state (into a down state) and the appropriate BUTTON_DOWN widget event is generated. This BUTTON_DOWN event is now sent to the Display Node to update the button appearance using the button overlay plugin and to the application to execute the callback that was originally set for this button.

7.4. Attaining design aspects

(1) Distributed

By decoupling the widget logic, drawing and creation, we achieved a truly distributed widget framework. Widgets can be seamlessly drawn across any number of displays driven by any number of Display Nodes. Applications located on distant rendering resources can present their user interface on any display by requesting widgets and receiving events over the network all without ever being aware of this separation. The separation means that we can develop new interaction techniques that are more appropriate for the large high-resolution displays without requiring any modifications to the applications themselves. For example, as display resolution dramatically increases, target acquisition becomes much more difficult. One approach is to dynamically resize targets as cursors get near, which is something we were able to easily implement without changing or recompiling any of the applications (Fig. 6).

(2) Scalable

During startup, DIM first collects information about the display environment and calculates the appropriate widget scale factor to adjust for usability (ease of clicking on a target with a physical interaction device) and visibility (font size visible on the display). To provide truly scalable drawing, we use dynamically resizable vector fonts and large raster images that are typically scaled down to reduce aliasing. It would be straightforward to add support for vector images as well. Since we always assume that applications are rendered remotely, we cannot assume that every image used in the application interface is already present on the display side. Therefore, the API automatically embeds necessary images in the XML widget description, which is then delivered to the Display Nodes for drawing. This gives application developers freedom to design custom interfaces.

(3) Multi-user

This design aspect was rather easy to achieve since we designed DIM to support multiple interaction devices while giving each device a separate cursor. Additionally, while rules exist to prevent multiple devices from interacting with one widget at the same time, no such rules are enforced between widgets; hence multiple users can interact with different widgets simultaneously, even within the same application.

(4) Physical interaction device independence

As mentioned earlier, it is possible that new physical interaction devices will surface that perform better on large high-resolution displays or as we have personally experienced, multiple input

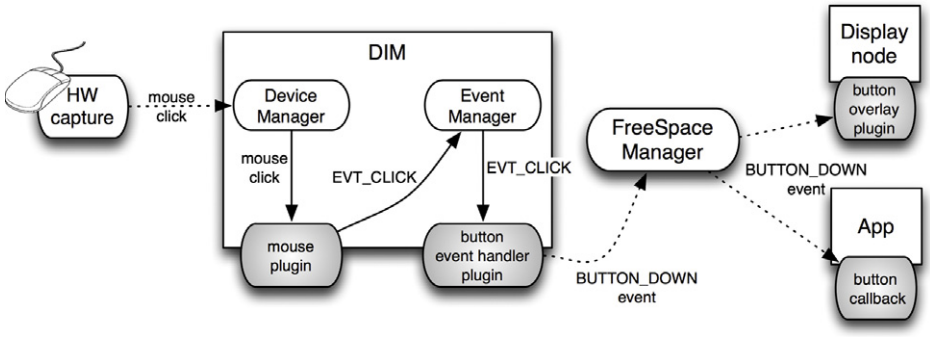


Fig. 7. Event handling scenario for a mouse click. Dotted lines denote a network connection.

modalities may be more intuitive for different use cases. New devices can be added to SAGE by simply providing a plugin that describes the conversion between device-specific events and the generic set of events. By using this generic set of events and therefore removing any specifics of each device, we can treat all of them equally for event handling purposes.

8. Experiences using the system

In order to fully exploit the affordances of large high-resolution displays, users should ideally be mobile in front of the display [27]. The mouse has been the de facto standard for desktop interaction but it is not well suited for large high-resolution display environments primarily because it confines users to a hard surface. Although, the research community has investigated various novel interaction devices [28–31], many of them require elaborate setups and equipment for which most users in the SAGE community will not have time or resources to set up or maintain (e.g. camera trackers). Therefore, the physical interaction device independence attained in our design, proved to be invaluable since it enabled us to easily experiment with various off-the-shelf devices such as joysticks, Gyromouse, trackpads, 6 DOF tracked wand and the Wiimote. Although each device had its own strengths and weaknesses, the Gyromouse seemed to be the most suitable one in the end. It was inexpensive, readily available, many could be used simultaneously and it allowed users to be mobile in front of the display because it did not tie users to a hard surface and it did not require a line-of-sight for operation. Additionally, since it was essentially a mid-air mouse, it did not require an elaborate setup. Even though it has a slight learning curve associated with it, users quickly became comfortable using it and it remained as a preferred device. However, since the Gyromouse is inherently affected by hand jitter, it may be somewhat difficult to acquire smaller targets, especially on larger displays as the accuracy of the device has to be traded for speed. Initially, application windows could be moved by clicking and dragging a small area at the bottom and top of the window to mimic the behavior on desktop operating systems. However, by allowing the windows to be dragged by clicking anywhere within the window, moving them became significantly faster and easier, and encouraged users to experiment more with the layout. Similarly, windows can be resized by dragging any of the four corners, but after allowing the windows to be "zoomed" in and out using a mouse wheel, we noticed that it became a preferred method and again, users experimented with different window sizes more often.

Although the Gyromouse was used for the bulk of the interaction, when users are at arm's length from the display, it becomes somewhat confusing to use because small movements of the mouse result in large movements on the display. In such cases, directly interacting with the wall using touch gestures is more natural. Once again, having physical interaction device independence proved valuable as integrating a touch screen device into DIM simply required converting raw touches into our generic set of events in a device plugin. The addition of touch input immediately exposed a new set of research questions. For instance, users now have the ability to interact from a distance using a Gyromouse (when focusing on the context) and interact from up-close using touch input (when focusing on detail). But, how does one automatically adapt the user interface according to the input modality being used since the affordances of each are vastly different? The question is further complicated when multiple users are interacting with the system simultaneously using different modalities. Such questions are a part of our ongoing research effort to make these environments more user-friendly and usable in true real-world collaborative scenarios.

Fig. 8. A semester-long computer science class held entirely using the new thin-bezel 20 megapixel display which in resolution equals two 4K displays placed side-by-side. In this particular case, the left half of the display was used to display lecture notes in a web browser, while the right side was used for juxtaposing relevant multimedia for easier comparison.

We continued to hold our weekly meetings in front of the display. However, we wanted to give the participants direct control of the display without using the old SAGE UI that seemed too cumbersome and full-featured for casual use. Since providing every participant with a Gyromouse was impractical and given that every participant already used a laptop in the meetings, we created a minimal desktop application called Sage Pointer that allows the users to directly control the SAGE display form their laptop, share multimedia files and share their desktop on the display. Using a global hot key, users can capture their desktop's mouse pointer and a new pointer would be created on the SAGE display that the trackpad now controls. This gives users the same ability to manipulate widgets and applications on the display as if they were using a Gyromouse, but from their personal laptops. Such interaction mechanisms proved so natural in fact, when users did not have their laptops and therefore, the ability to point at and interact with the display, certain subtle frustration became evident as they could not easily convey their ideas and thoughts relative to the existing conversation and content on the display. Clearly, this would not be possible without integrating true multi-user interaction into the design. Since new devices (i.e. pointers) can be added to DIM dynamically, users can casually switch back and forth between controlling their laptop and controlling the SAGE display. Sage Pointer application seemed to be well accepted since it was a much simpler interface to SAGE that was more powerful than the old SAGE UI. For instance, meeting participants could now point on the screen from their laptops without walking up to the screen, which was something we found to be important in our meetings using the old SAGE UI. Furthermore, the simplicity and the direct interaction features of the Sage Pointer application encouraged more impromptu meetings using the display.

Besides the experience with the new interface during our meetings, a semester-long computer science class was also taught using the new thin-bezel 20 megapixel display. The class focused on visualization and visual analytics where the focus was on teaching various visualization techniques through existing visualization examples. Because of the thin 7 mm bezels (distance between two pixels on adjacent screens), viewing text was not problematic unlike on previous generation tiled-displays where bezels potentially obscured multiple lines of text. Therefore, the professor would hold lectures using a website prepared for the class and use the SAGE display for showing images and videos pertinent to the discussion (Fig. 8). To show the media files on the SAGE display, it was sufficient to simply drag and drop them from the web browser, or local hard drive, onto the Sage Pointer

application. A Gyromouse was used to control the web browser on the computer and to control the SAGE display. The switching between the control modes was done using the Sage Pointer application, which mapped one of the auxiliary buttons on the Gyromouse to a hot key that typically did the switching. This proved to be a fairly natural and straightforward way to switch between the modes. The nature of the class required that many high-resolution images be compared simultaneously since they illustrated different solutions to the same problem. While this would be difficult using a single projector given its low resolution, it was a perfect use case for a tiled-display where the benefits of increased screen resolution immediately became apparent. At the end of the class, a survey about the classroom experience was administered to the students, which confirmed this benefit. Eighty percent of the students felt that they were learning significantly more in this classroom space than they would have in a more traditional classroom equipped with whiteboards and a single projector. The main stated reason behind this was the ability to compare multiple visualizations simultaneously, which would not have been possible in a traditional classroom.

However, as the available size and resolution of the display increases, users tend to put more and more information up and it becomes more difficult and tedious to organize it into meaningful arrangements. We have observed a similar trend during the class. For instance, every time the professor would drag media to the display, they had to be manually moved and resized for direct comparison. Furthermore, when students would use the display to present their work, they would often preload their images in order to bring them up more easily at the time of the presentation. However, as more students wanted to preload their images, the display quickly became crowded and it became more difficult to find and organize the appropriate images for each student. The problem of content organization was something we anticipated hence we developed certain layout management features to more easily organize multiple windows on the display. For instance, we added a few automatic-tiling modes that lay windows out in a grid while trying to utilize the available space as much as possible. Independently of the display resolution, at some point there will inevitably be more content than one could comfortably fit. For such cases we created a minimize feature where windows could be minimized and recalled in the future. This proved to be especially useful during student presentations where a set of images for each student could be kept until they are needed for their presentation. Furthermore, we added the ability to select multiple windows and manipulate them simultaneously (e.g. move, resize, maximize, minimize). For instance, this can be used to put in focus a remote video conference stream and a visualization shared among distributed participants (for example, a 4K visualization). Lastly, to better manage the large increase in the amount of content and to provide persistency to the uploaded content, we created a tool for easily browsing and showing a large number of multimedia files by taking advantage of the high resolution (Fig. 9). We anticipate that these features will become of even greater significance as more users attempt to use the display simultaneously which will greatly increase the amount of content.

9. Conclusion and future work

We have presented this interaction system as a first instantiation of the OptIPuter's interaction model while focusing on the design aspects which are drawn from the basic premise of the OptIPuter: collaborative large high-resolution displays interconnected with ultra-high-speed networks. Having an actual working system allows us to study such environments in real-world settings. By introducing the direct interaction capabilities, we have encouraged a more casual and consistent use of our high-resolution display environments as evidenced by more frequent impromptu meetings. Further simplifying the interaction techniques (e.g. resizing and moving windows) brought additional benefits. We have also noticed that giving the power to every user to simultaneously control the display is extremely beneficial in encouraging participation and conveying of ideas and thoughts during collaborative work. But, perhaps most importantly, increased casual use exposed many new research questions, which will need to be addressed before we can reap all the benefits of such environments. Currently, we are focusing our research efforts on bridging multiple input modalities for smooth interaction and content organization issues in collaborative work.

Fig. 9. The media browser, developed using SAGE widgets, is a tool for easily browsing and showing multimedia files on large high-resolution displays.

We will continue to improve the user interaction system through experiences with real-world users, such as classroom deployments, distributed scientific workspaces, ultra-high-resolution media production and content delivery for science, education and entertainment. Besides opening up the opportunities to study distributed collaborative work and new applications, we believe more useful feedback can be gathered through our planned release of the new interaction system to the SAGE users community.

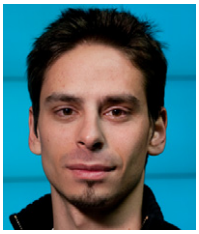
Acknowledgements

The Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago specializes in the design and development of high-resolution visualization and virtual reality display systems, collaboration software for use on multi-gigabit networks, and advanced networking infrastructure. These projects were made possible in part by the National Science Foundation (award SCI-0225642) and Sharp Laboratories of America.

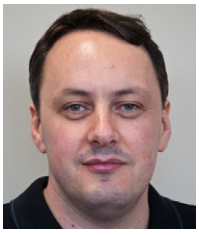
References

[1] L. Smarr, A. Chien, T. DeFanti, J. Leigh, P. Papadopoulos, The OptIPuter, Communications of the ACM 46 (11) (2003) 58–67.
[2] J. Leigh, L. Renambot, A. Johnson, B. Jeong, R. Jagodic, N. Schwarz, D. Svistula, R. Singh, J. Aguilera, X. Wang, V. Vishwanath, B. Lopez, D. Sandin, T. Peterka, J. Girado, R. Kooima, J. Ge, L. Long, A. Verlo, T.A. DeFanti, M. Brown, D. Cox, R. Patterson, P. Dorn, P. Wefel, S. Levy, J. Talandis, J. Reitzer, T. Prudhomme, T. Coffin, B. Davis, P. Wielinga, B. Stolk, G. Bum Koo, J. Kim, S. Han, J. Kim, B. Corrie, T. Zimmerman, P. Boulanger, M. Garcia, The global lambda visualization facility: an international ultra-high-definition wide-area visualization collaboratory, Future Generation Computer Systems 22 (8) (2006) 964–971.
[3] R. Ball, C. North, The effects of peripheral vision and physical navigation on large scale visualization, in: GI'08: Proceedings of Graphics Interface 2008, 2008.
[4] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, J. Leigh, High-performance dynamic graphics streaming for scalable adaptive graphics environment, in: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC'06, Tampa, Florida, November 11–17, 2006.

[5] B. Stolk, P. Wielinga, Building a 100 Mpixel graphics device for the OptIPuter, Future Generation Computer Systems 22 (2006) 972–975.
[6] T. DeFanti, et al., The OptIPortal, a scalable visualization, storage, and computing interface device for the OptiPuter, Future Generation Computer Systems 25 (2) (2009) 114–123. doi:10.1016/j.future.2008.06.016.
[7] J. Leigh, M. Brown, Cyber-: merging real and virtual worlds, Communications of the ACM 51 (1) (2007) 82–85.
[8] L. Renambot, B Jeong, H. Hur, A. Johnson, J. Leigh, Enabling high resolution collaborative visualization in display rich virtual, Future Generation Computer Systems 25 (2) (2009) 161–168. doi:10.1016/j.future.2008.07.004.
[9] http://CineGrid.org.
[10] R. Ball, C. North, Analysis of user behavior on high-resolution tiled displays, in: Human–Computer Interaction—Interact 2005, 2005.
[11] M. Czerwinski, G. Robertson, B. Meyers, G. Smith, Large display research overview, in: Conference on Human Factors in Computing Systems, 2006.
[12] R. Ball, M. Szwedo, C. North, Dynamic size and speed cursor for large, high-resolution displays, 2006.
[13] C. Forlines, D. Vogel, R. Balakrishnan, HybridPointing: fluid switching between absolute and relative pointing with a direct input device, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2006.
[14] M. Kobayashi, T. Igarashi, Ninja cursors: using multiple cursors to assist target acquisition on large screens, in: CHI'08: Proceeding of the Twenty-sixth Annual SIGCHI Conference on Human Factors in Computing Systems, 2008, pp. 949–958.
[15] E. Tse, M. Hancock, S. Greenberg, Speech-filtered bubble ray: improving target acquisition on display walls, in: ICMI'07: Proceedings of the 9th International Conference on Multimodal Interfaces, 2007.
[16] D. Vogel, R. Balakrishnan, Distant freehand pointing and clicking on very large, high resolution displays, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2005.
[17] G. Fitzmaurice, A. Khan, G. Kurtenbach, G. Binks, Cinematic meeting facilities using large displays, Computer Graphics and Applications, IEEE 25 (4) (2005) 17–21.
[18] W. Buxton, G. Fitzmaurice, R. Balakrishnan, G. Kurtenbach, Large displays in automotive design, Computer Graphics and Applications, IEEE 20 (4) (2000) 68–75.
[19] J. Richards, P.E. Mantey, Large classroom experience with an interactive tiled display mural, in: Frontiers in Education Conference, 36th Annual, 27–31 October, 2006, pp. 15–20.
[20] B.I. Olsen, S.B. Dhakal, O.P. Eldevik, P. Hasvold, A large, high resolution tiled display for medical use: experiences from prototyping of a radiology scenario, in: Studies in Health Technology and Informatics, 2008.
[21] M. Roman, C.K. Hess, R. Cerqueira, A. Ranganathan, Gaia: a middleware infrastructure to enable active spaces, IEEE Pervasive Computing (2002).
[22] P. Tandler, Software infrastructure for ubiquitous computing environments: supporting synchronous collaboration with heterogeneous devices, in: Proceedings of UbiComp 2001: Ubiquitous Computing, 2001, 2001.
[23] B. Johanson, A. Fox, The event heap: a coordination infrastructure for interactive workspaces, in: Mobile Computing Systems and Applications, 2002, Proceedings Fourth IEEE Workshop on, 2002, pp. 83–93.
[24] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, J. Leigh, High-performance dynamic graphics streaming for scalable adaptive graphics environment, in: Proceedings of SC06, November, 2006.
[25] http://www.evl.uic.edu/cavern/optiplanet.
[26] GlovePIE. http://glovepie.org/glovepie.php.
[27] R. Ball, C. North, D.A. Bowman, Move to improve: promoting physical navigation to increase user performance with large displays, in: CHI'07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2007.
[28] B.A. Ahlborn, D. Thompson, O. Kreylos, B. Hamann, O.G. Staadt, A practical system for laser pointer interaction on large displays, in: VRST'05: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, 2005.
[29] P.B. Baudisch, M. Sinclair, A.D. Wilson, Soap: a pointing device that works in mid-air, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2006.
[30] P. Forbrig, F. Paternò, A. Pejtersen, Multitouch sensing for collaborative interactive walls, in: Proceedings of the 1st Tc 13 Human–Computer Interaction Symposium, Hcis 2008, September 7–10, 2008, Milano, Italy, 2008, p. 272.
[31] S. Malik, A. Ranjan, R. Balakrishnan, Interacting with large displays from a distance with vision-tracked multi-finger gestural input, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2005.



Ratko Jagodic is a Ph.D. candidate in the Department of Computer Science at the University of Illinois at Chicago and a research assistant at the Electronic Visualization Laboratory. His research interests include human–computer interaction and computer supported cooperative work in large high-resolution display environments.



Luc Renambot received a Ph.D. from the University of Rennes 1 (France) in 2000, conducting research on parallel rendering algorithms for illumination simulation. Then holding a Postdoctoral position at the Free University of Amsterdam, till 2002, he worked on bringing education and scientific visualization to virtual reality environments. In 2003, he joined EVL/UIC first as a PostDoc and is now a Research Assistant Professor, where his research topics include high-resolution displays, computer graphics, parallel computing, and high-speed networking.

Andrew Johnson is an Associate Professor in the Department of Computer Science and member of the Electronic Visualization Laboratory at the University of Illinois at Chicago. His research focuses on the development and effective use of advanced visualization displays, including virtual reality displays, auto-stereo displays, high-resolution walls and tables, for scientific discovery and in formal and informal education.

Jason Leigh is an Associate Professor of Computer Science and director of the Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago. Leigh is a co-founder of VRCO, the GeoWall Consortium and the Global Lambda Visualization Facility. Leigh currently leads the visualization and collaboration research on the National Science Foundation's OptIPuter project, and has led EVL's Tele-Immersion research agenda since 1995. His main area of interest include developing collaboration technologies and techniques for supporting a wide range of applications ranging from the remote exploration of large-scale data, education and entertainment.

Sachin Deshpande is a Senior Researcher at Sharp Laboratories of America, where he has been a member since January 2000.

He received his B.E. from Government College of Engineering Pune, his M.Tech. from IIT Mumbai and his Ph.D. from University of Washington, Seattle.

His research interests include video streaming, networking, video and image coding, multimedia systems, and user interfaces.

He has more than thirty-five publications in reputed peer reviewed journals, conferences, books. He holds twenty-five US patents and has several applications pending. He has received Sharp Laboratories Department Inventor of the Year award twice. He was Short Courses Co-Chairman at IS&T/ SPIE Electronic Imaging 2008 and 2009 conference.