# Multi-User Interface for Scalable Resolution Touch Walls

BY

ARTHUR NISHIMOTO
B.S., University of Illinois at Chicago, 2010

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2014

Chicago, Illinois

## ACKNOWLEDGEMENTS

I would like to thank my committee members Andrew Johnson, Jason Leigh, and Luc Renambot for their support and guidance through all the brainstorming, planning, and refinement leading to the completion of this thesis. I would also like to thank Maxine, Dana, Lance, Jonas, Dan, Alessandro, Khairi, Victor, Jillian, Tommy, Krishna, Alex, Huy, Galen, Bazz, Chihua, Shi, Dennis, and Ratko at the Electronic Visualization Lab for their continuing encouragement and enthusiasm for my various projects.

I would also like to thank the fifteen participants in my user study who volunteered during spring break to provide valuable insight for the evaluation chapter of this thesis. I would like to thank Jason Leigh again for providing me the opportunity to become part of the EVL family as an undergrad and showing me a path to explore the more visual and creative side of computer science in graduate school - a path that I had not previously considered and looking back I'm not sure where I'd be without it.

Finally I would like to thank my parents for their constant support and encouragement for reaching toward the stars and my brother and cousins whose own academic endeavors encourage me to continue as much as perhaps I encourage them.

AN

# TABLE OF CONTENTS

**TABLE OF CONTENTS (continued)**

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| API | Application Programming Interface |
| CGLX | Cross Platform Cluster Graphics Library |
| EVL | Electronic Visualization Laboratory |
| GUI | Graphical User Interface |
| HCI | Human Computer Interaction |
| NUI | Natural User Interface |
| RFID | Radio-frequency identification |
| SAGE | Scalable Adaptive Graphics Environment |
| SDK | Software Development Kit |
| SRDW | Scalable Resolution Display Wall |
| SRTW | Scalable Resolution Touch Wall |
| UI | User Interface |
| WIMP | Window, Icon, Menu, Pointer |

## SUMMARY

Multi-touch is becoming a common technology for providing direct natural user interaction for smartphones and tablets. However, limited work has been done in designing multi-touch interfaces for large scale display environments where multiple people can interact with the display simultaneously.

A common method of handling multiple user interaction in large high-resolution display environments is using a mobile device to interact with the display while allowing the device to keep a notion of user identity. For large touch screen environments, user distinction becomes more difficult as the current touch technology allows the tracking of several touch points, but is not capable of associating a particular touch to a specific user without additional sensors such as camera tracking or radio-frequency identification (RFID).

What I propose is a multi-touch user interface design for multi-user interaction on large display environment. This methodology will allow multiple users to open a menu in their own personal space and select content without conflicting with other users. As these scalable wall environments get larger, this will also allow access to the menu anywhere on the wall instead of physically moving to a fixed position menu. This type of interface would allow multiple users to more efficiently select content and control local 'personal' space on high-resolution display walls.

# 1. INTRODUCTION

Multi-touch is becoming a common technology for providing direct natural user interaction for smartphones and tablets. However, limited work has been done in designing multi-touch interfaces for scalable resolution display walls where multiple people can interact with the display simultaneously. Middleware designed for multi-user interaction like Scalable Adaptive Graphics Environment (SAGE) supports direct multi-touch interaction, however in its current implementation the user interface is primarily designed for indirect pointer interaction. Indirect pointer interaction allows users to control large displays much like a desktop environment. The user is far enough away from the screen to get an effective overview and the indirect interaction allows the pointer to quickly traverse the display. In a direct touch environment interaction is limited to the space near the user. User interface components in fixed locations on the wall that were easily accessible using a pointer now require the user to physically walk to that location.

This thesis presents a methodology for a multi-user interface designed for scalable resolution touch walls. In comparison to SAGE where the menu containing the media browser is tailored to a single user interacting from a distance, this methodology allows multiple users to access a menu in their own personal space and select content without conflicting with other users. As these scalable wall environments get larger, this will also allow access to the menu anywhere on the wall instead of at a fixed position. This type of interface should allow multiple users to more efficiently select content and manage a personal space on a shared display wall. A prototype of the interface was implemented and evaluated using a simulated search task to understand how to effectively design a localized menu interface targeted for scalable resolution touch walls (SRTW).

# 2. BACKGROUND

This chapter will describe three major background topics this work builds on. The first section will describe the user interface paradigms that provide the basis this thesis. The next two sections will describe two hardware components that drive the methodology and implementation of this thesis: scalable resolution display walls and multi-touch interfaces.

## 2.1 User Interface Paradigms

An interaction paradigm commonly used in user interface design since the 1980s is "Windows, Icons, Menu, Pointer" (WIMP) which can be described as follows:

1. A window contains a program that runs independent but simultaneously with other windows.
2. Icons act as a shortcut for executing a program or task.
3. Menus are a text or icon based selection system for programs or tasks.
4. A pointer is an onscreen symbol representing the movement of the device used to select icons or other GUI elements.

The WIMP paradigm continues to be used in modern desktop applications but does not work well in large display environments because of the increasing distance a pointer has to travel as the display gets larger. This can be offset by increasing the sensitivity of the pointer but at the cost of precision needed to select icons and menus. For multi-touch enabled displays, the touch interface lacks the precision of pointer devices for interacting with GUI elements.

Andres van Dam [3] describes a post-WIMP interface as one not dependent on traditional 2D widgets like menus and icons but one which involves multiple senses in parallel, natural language communication and multiple users. In many ways a post-WIMP is similar to natural user interfaces (NUI) where the end goal is to provide a user interface that is easy to learn and use. Other examples of post-WIMP interfaces would be digitizer pens for tablets, using a steering wheel verses using a gamepad or keyboard for a driving game, or using a motion controller to swing an object rather than clicking a button. Multi-touch interfaces can be used in either WIMP or post-WIMP interfaces, but the latter is capable of proving more meaningful interaction beyond turning a touch into a mouse pointer.

## 2.2 **Scalable Resolution Display Walls**

Scalable resolution display walls (SRDW) are an emerging technology for displaying large amounts of information at a high enough resolution that it is possible to connect data through spatial recognition. One of the advantages of SRDWs is allowing multiple users to simultaneously examine the data. Depending on the resolution and type on content, users may be standing directly in front of the wall or sitting at a distance. Leigh et al. [13] describe four major interaction zones for SRDWs. Zones three and four use typical interaction for SRDWs which essentially treat the display as a giant desktop environment using a mouse and keyboard. Zone two which is closer to the display wall corresponds to a user standing near the wall using a wireless devices such as a Gyromouse and tablets or being tracked using a device like the Microsoft Kinect. Zone one interaction is for direct touch and gestures using a multi-touch screen.

## 2.3 **Touch Interfaces**

Touch is often associated with natural user interfaces (NUI). The goal of such an interface is to have users quickly become experts at using the user interface in a short amount of time with little training. Early touch screen interfaces were often on tabletop environments which mimicked familiar paper-based interactions such as sliding and rotating across the table. One of the most common touch devices today is the mobile phone or the tablet. The concept of sliding a finger to scroll or using a pinch gesture to zoom is a learned technique that mobile devices have impressed on a large population of users. The touch wall is a relatively recent extension of touch interfaces that runs into several potential problems of scalability. In such environments the whole interface is no longer easily visible to the user depending on the size of the display wall requiring a different interface design from desktop or mobile devices.

# 3. STATE OF THE ART

As information grows beyond the scope of a desktop display, scalable resolution display walls are used to aid users in organizing and analyzing data much like they would with a whiteboard or collection of notes on a wall. SRDWs have employed multiple techniques for interacting with the content on the wall. A common technique is treating a display wall as a large desktop using a keyboard and mouse. This type of interaction works well for a single user experience, but does not utilize the multi-user collaboration the SRDWs can leverage. This chapter examines some of the collaborative applications of large display walls and how multi-touch interfaces have been integrated into many of them.

## 3.1 PWIG Interaction Paradigm

The Page, Widget, Icon, and Gesture paradigm was proposed in 2013 by Wang, Fan, Deng, and Yin [17] as a post-WIMP interaction paradigm based on observations of direct touch interaction on mobile devices. Page/Frame is similar to the Window component of the WIMP paradigm as it serves as a container for applications as well as other components of the user interface. Widgets go beyond the WIMP menu and add additional user interface components such as text boxes, select lists, and sliders. Icons/Buttons are described the same as their WIMP counterpart but are still important user interface components. The final component Gesture is divided into two types: Touch and Slide and serves the primary method for interaction in touch interfaces.

## 3.2 **<u>Methods of Multi-Touch Integration</u>**

Multi-touch interaction has been implemented using both WIMP and post-WIMP paradigms. When multi-touch interfaces are integrated into a desktop or large display environment, one of three techniques is commonly used.

### 3.2.1 **<u>Direct mapping of touch to pointer</u>**

The first involves the direct mapping of multi-touch points or gestures to mouse pointer events. This results in touch drags becoming mouse drags or touch taps becoming mouse clicks. This technique is common many application programming interfaces (API) distributed by multi-touch manufacturers or natively integrated into the software such as modern Windows and OSX operating systems. While this is the easiest and most straightforward method of adding a touch screen to any application, the interaction is compressed into the WIMP paradigm which limits the capability of what direct touch interaction can accomplish as well as have several drawbacks which are described in the next section.

### 3.2.2 **Multi-touch gestures augmenting traditional menus**

Another method of multi-touch integration relies on using gestures to interact with traditional WIMP elements like menus rather than using mouse-like touch movements. Bailly, Lecolinet, and Guiard [1] describe five major drawbacks to using a traditional WIMP menu on interactive surfaces:

1. Occlusion by the finger/hand interacting with the menu
2. Inaccuracy due to finger verses menu size
3. Lack of keyboard menu shortcuts
4. Accessibility of the menu bar
5. Lack of user identity without specific technology

The authors presented two gesture based shortcuts to alleviate these drawbacks. The first counts the number of fingers placed on the surface by the non-dominant hand. Each menu on the menu bar is associated with a particular finger count. The finger count from the dominant hand is then used to select the item within the selected menu. Similarly the second shortcut uses a two-finger directional gesture to determine the selected menu and a single-finger gesture to determine the selected menu item. In this case each menu or item is associated with a finger swipe in a particular direction like up, left, or diagonally down-right. Another example of this type of technique was proposed by Kentaro Go and Hiroki Kasaga [7] using the number of touch taps rather than held finger counts.

Both of these approaches provide a method of augmenting a touch interface to traditional menus, but are limited to single user interactions on smaller displays. Traditional drop-down menus themselves do not scale to larger wall sized displays as the user's focus is on a localized area on the display rather than an overall desktop view where such menus are fixed to a particular region of the display.

### 3.2.3 **General goals for touch-centric interface**

The third method of touch integration is designing the user interface specifically to utilize multi-user multi-touch interaction. The goal of such an interface is to provide every user with the necessary to perform the required task anywhere on the wall. Important aspects of such an interface include:

- Accessibility to the menu anywhere on the wall
- Finding a balance between physical and virtual navigation
- Identification of users and defining a user space
- Management of multiple user spaces

As scalable resolution touch walls can be large enough that a user may not be able to easily see the entire wall at once, the menu used to interact with the wall cannot be in a fixed location and must be able to be open by the user from any location on the wall. Depending on the task being performed on the wall it may be important to focus on a particular section of the wall and use virtual navigation rather than having to physically navigate. Conversely if the task involves a large amount of spatial information, it may be more important that the user be able to physically move around to examine the data and still have the interface accessible when the user needs it. User identification on multi-touch walls can be challenging without external sensors, but a level of identification is needed in order to provide more meaningful gestures for interaction. As scalable resolution touch walls can be a multi-user environment, the ability to identify and manage user spaces is important to aid users in identifying content ownerships, where user space boundaries lie, and how overlapping spaces are handled.

## 3.3 **Sense making**

Information visualization examines how the visual representation and interaction techniques can improve the understanding and analysis of abstract data. Jakobsen and Hornbæk [12] examined collaboration on a wall sized multi-touch display using an interface similar to Cambiera, a multi-touch collaborative visual analytics tool for multi-touch tabletops. The participants in the exploratory study worked on the "Stegosaurus" scenario from the IEEE Visual Analytics Science and Technology (VAST) 2006 challenge. The data set consist of 230 news articles, 3 images, 1 map, 1 spreadsheet, and 3 reference documents. Pairs of users had to read the documents, make hypotheses, gather evidence, filter irrelevant data, and connect the data. The interface consists of four search buttons on the bottom of the display. Clicking on a button opens a search bar and an on-screen keyboard. Documents matching the search query are identified in a tile bar with colored rectangles highlighting where in each document the query appears. Documents can be opened by dragging the document from the tile bar. Also holding a finger on the background will open a note window [10].

The results of the study found that user's spent 56% of the time within 46 - 120 cm of each other, 18% of the time closer than 46 cm, and 22% of the time at 120 – 370 cm. In terms of distance from the screen, users spent 60% of the time within 46 cm and 31% within arm's length at 76 cm. The study also found that participants preferred working in the center of the display, although the authors attribute this to the initial documents being at the center of the screen.

The author's do not mention where the four search buttons are located horizontally across the display as this may had an effect on where user's tend to work. If this task were to be scaled up to include more users, the interface would likely need more than 4 static buttons to accommodate a larger region of the display being used simultaneously.

## 3.4 **CGLXTouch**

A multi-touch table and portable devices are used in CGLXTouch to interact with large display walls running the Cross Platform Cluster Graphics Library (CGLX). Designed for high performance large scale visualization, CGLX serves as a middleware managing the rendering and input synchronization across a cluster [15]. The multi-touch table and mobile devices allow users to interact with the visualizations displayed on the main display wall. The head node of the cluster manages the multiple input devices as well as streaming display information back to the mobile devices as they lack the computational power to do their own rendering.

One of the applications CGLXTouch presents is a common multi-touch sorting application where two-dimensional content can be translated, rotated, and scaled using simple gestures. One of the challenges of this application was interacting with objects on smaller touch screen devices where scaling and rotation gestures become more difficult to perform.

Using mobile devices allows multiple users maintain a unique identity on the wall and control content from a distance through their own personal view of the display wall. When content controlled by different users begin overlapping, this type of environment makes it harder to define a personal space compared to using a direct touch screen approach where personal space can be implied using the user's physical position in front of the display wall.

## 3.5 **Magic Input**

Magic Input for SAGE proposed a multi-user interaction system for desktop applications displayed on a large tiled-display environment [14]. Magic Input consists of three major components: Input Manager, Input Client and the Application layer. The Input Clients run on computers with input devices to be sent to SAGE. These clients connect and stream input events to the Input Manager which controls all SAGE Pointers representing each of the Input Clients. If the SAGE Pointer is above an application window, Input Manager will also handle forwarding and prioritization of multiple SAGE Pointer events to the application. The application layer receives input events and generates the appropriate mouse and keyboard functions on the application allowing interaction through the SAGE Pointer to a target application without any modification to the application source code.

The Input Manager and Input Client presented in Magic Input share similarities to the input manager used by the methodology presented in this paper. Both provide a middleware between various input clients and the window manager running of the large display. Magic Input works well for expanding pointer based interaction on large displays and does feature a novel method of interacting with non-customizable commercial software.

## 3.6 <u>**CubIT**</u>

The Cube is a multi-user interactive facility consisting of 48 multi-touch screens and large projector displays. CubIT is a multi-user presentation and collaboration framework allowing users to upload, interact, and share content across a wide display environment [16]. Using the multi-touch interface, users can drag, rotate, and scale media content across the display. CubIT's user workspace is a personal window on the shared display capable of displaying the user's information as well as saved media content. Through the user workspace, the user can browse through thumbnails of content, open content on to the workspace, and also reorganize and delete content as needed.

CubIT has the ability to distinguish user identity visually through the workspace window. Using RFID readers along the display users are able to bring of their personalized workspace at their location. This is one of the few examples of a touch-centric user interface for a touch wall.

## 3.7 <u>**SAGE**</u>

The current version of the Scalable Adaptive Graphics Environment (SAGE) supports multi-user interaction through the Direct Interaction Manager (DIM) [11]. DIM is divided into three components: the Device Manager, Event Manager, and the Overlay Manager. The Device Manager handles the conversion of various input devices into a generic set of events though a device plugin. This allows the rest of SAGE to use a standardized event structure regardless of the original input device. The Event Manager processes and queues the events to be sent to the appropriate event handler. The event handler serves as the base class for all interactive objects.

The Overlay Manager receives draw commands from event handlers and directs the overlay plugin on each display node to perform the rendering.

One of the major components of the SAGE user interface is the media folders which hold copies of all of the media that have been placed onto the wall. The media folders consist of different file types represented by button widgets. These buttons are typically attached to the left side of the display canvas. Selecting a button will open up the file directories containing subdirectory buttons representing collections of related media and thumbnail icons representing the individual media files.

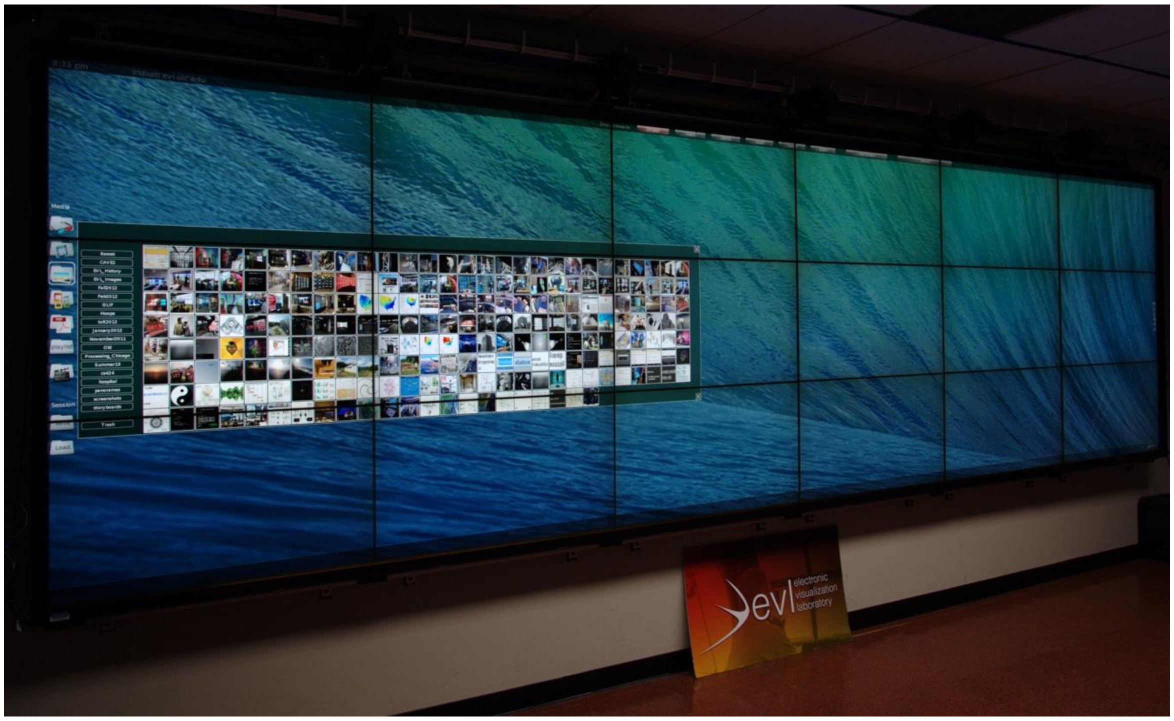

Figure 1: SAGE media folder on the EVL Cyber-Commons wall

The SAGEPointer is one of the primary interaction methods within SAGE. Typically used as an extension of a mouse, the SAGEPointer allows multiple users to click, drag, and scale content across the SRDW simultaneously. When the touch plugin was added to the Device Manager, the touch events were converted for use within SAGE. For example touch clicks became analogous to SAGEPointer clicks, touch zoom gestures became SAGEPointer scroll wheel events, and touch drags became pointer drags. Using this direct mapping from touch to SAGEPointer events allowed the touch overlay to make use of the multi-user capabilities of the SAGEPointer. Some of the drawbacks of this technique stem from the fundamental differences in direct and indirect interaction devices. For the SAGEPointer and SAGE Touch this can be broken down into user identification and physical verses virtual movement.

As an indirect interaction method, the SAGEPointer has the advantage of having a specific identifier attached to the pointer. In this case it would be a unique IP address of the device controlling the pointer. The events provided by multi-touch overlays give the position and size of the touch with a unique finger ID. While this ID is useful for distinguishing individual points on the screen, there is no direct connection between the touch point and the particular user that created it. Assumptions can be made about closely clustered points belonging to the same user or the use of external camera tracking to distinguish user positions in front of the display wall.

An indirect interaction method such as the SAGEPointer has the advantage of virtually moving across a large SRDW quickly by exaggerating the motion of the pointer. This works well as indirect interaction is typically done further away from the display where the user can get an overview of the entire display. In the current SAGE touch implementation, the dragging motion is similarly exaggerated allowing the user to drag a window across the screen without walking along with it. Where this interaction breaks down is the media folders. The SAGEPointer can easily traverse the SRDW to get to the media folder icons on the left of the display. The SAGE touch interface requires the user to walk over to the left side of the wall to interact with the media folders. For a true multi-user direct touch interface, the media folder needs to be uncoupled from a fixed position on the screen and be able to open where a user at the wall is currently located. If multiple of these can be opened across the wall, this would allow multiple users to simultaneously browse and select content which cannot be as easily done using the current SAGE interface.

## 3.8 **Touch Menus**

User interface menus all have design considerations that affect usability of the interface. Most menu designs have to consider the size and layout of the interaction widgets. Depending on the input type, the size of the widget can affect the accuracy of buttons and the layout can affect performance based on how long it takes to move from one part of the menu to the next. Touch interfaces have an additional consideration of occlusion of the user’s hand and arm when interacting with the touch screen. Many related works have examined menu design for multi-touch environments. One of the common designs is the pie or radial menu.

### 3.8.1 **Pie Menu**

The pie or radial menu design consists of menu buttons arranged in a circular fashion compared to traditional linear drop down menus. As far back as 1988 the effectiveness of pie menus was being compared to linear menus. Callahan et al. [2] showed that due to the smaller distance needed to select menu items in a circular arrangement the selection speed was 15% faster than linear menus. One of the issues cited with radial menus was the limit no how many menu items can be placed in a circular arrangement before the menu size becomes impractical.

### 3.8.2 **Stacked Half-Pie Menus**

Similar to how linear menus can have multiple sub-menus, Hesselmann, Flöring, and Schmitt [9] presented a stacked pie menu for handling several menu selections in multiple hierarchic levels. To manage occlusion by the user's hand while using the touch menu, a half circle design was used and placed at the bottom of the touch screen. The design has the activation button at the center of the menu. Opening the menu displays the first sub-level from the root as a ring of icons around the center of the menu. The ring can be scrolled to reveal additional menu icons at that level from off screen. Selecting another sub-menu will add an additional ring to the menu with further menu options. This design does handle menus with several items across multiple depth levels, although a menu with several depth levels can become unwieldy as the menu size increases. Also, the half ring attached to the bottom of the screen may not translate well from a tabletop to a wall display.

### 3.8.3 **pieTouch**

The pieTouch prototype by Ecker, Broy, Butz, and Luca [4] was designed for a 7" touch screen for use in cars. The major consideration was designing a menu interface that would be quick and easy to use with minimum glance time. Audio cues on each touch event allowed the interface to be used eyes-free. PieTouch uses an open circle menu where there are no buttons placed from 110 to 190 degrees. This was the result of an informal user study to determine where most users hand occluded the pie menu. The methodology presented in this paper uses a similar notion to manage hand and arm occlusion is on SRTWs.

# 4. METHODOLOGY

This chapter presents the methodology behind SAGETouchUI a multi-touch, multi-user interface prototype for the Scalable Adaptive Graphics Environment (SAGE). First the design goals are defined and the rationale behind them is examined. Next, the hardware system and backend architecture supporting the application is described. Finally, the user interface is described in detail focusing on the menu design and how the interactions are implemented.

## 4.1 <u>**Design Goals**</u>

The current user interface for SAGE utilizes media folders attached to a fixed position on the left side of the screen. While multiple pointers can simultaneously interact with content on the screen, the browser itself remains a single user experience as there is only one instance of the browser. As the media folders are in a fixed position on the screen, users using the touch interface will have to always have to walk to a specific part of wall to open content and then organize content elsewhere on the wall. A new methodology is needed to allow multiple users to access the media folders as well as allow users to open the browser anywhere on the display wall.

Within the existing SAGE content browser, all previously loaded files are displayed simultaneously as thumbnails in a large window. Another design goal is to condense the thumbnail browser into a smaller personal space window which still allows a user to navigate through all the available content.

Many of the interaction widgets within SAGE are attached to applications such as the PDF viewer. Often when scaling or moving these windows, the button widgets will fall off the edge of the screen requiring the user to re-orient the window in order to use the button. One of the design goals of this work is creating screen aware interaction widgets that are capable of intelligently relocating such that the widget is still within the usable screen space, but not covering up any content under it like text.

The user interface design presented here is tailored to touch walls large enough for multiple users to simultaneously interact directly on the wall. However one potential drawback in the localized menu design might be on smaller touch walls where limited space considerations may dictate that a fixed, always visible menu may be more effective than a movable, on demand menu. When every user has defined their own personal space on the large touch wall, organization of that space and handling overlapping zones becomes a problem that needs to be resolved by the users, the system, or perhaps a combination of both.

## 4.2 **Localized Menus**

The menu system of SAGETouchUI is designed to take advantage of large display environments where multiple users can interact with the wall simultaneously. Using a three-finger gesture, users can bring up a radial menu anywhere across the display. This allows multiple users to have access to the content browser and control menus in their own personal space. The menu takes into account the size of the gesture and adjusts its size to prevent icons from being occluded by the user's hand. The menu also takes in account the handedness of the three-finger gesture. If the gesture was calculated to be right-handed, the menu will open windows to the left of the gesture and thus in front of the user's field of view. Likewise a left-handed gesture will open content to the right.

The icons within the menu are spaced slightly farther than the radius generated by the hand gesture so that the icons are close to the initial position of the user's hand, but not occluded. The icons are also limited to an arc starting near the user's thumb and arching around the hand to the little finger. This prevents the icons from be occluded by the user's arm. Having the flexibility to design a menu around the user's hand position was a deciding factor in using a pie or radial representation opposed to using a grid or more linear layout.

## 4.3 **Scrollable Thumbnail Window**

Similar to the media folders in SAGE, most of the icons in radial menu are associated with different content types such as images, PDF documents, and videos. Selecting one of the icons will open a thumbnail window displaying the currently available content of that type. Keeping with the notion that the radial menu is part of a localized personal space, the window is scaled to fit within arm's reach of the initial menu position.

## 4.4 **Screen Aware Widgets**

One of the potential drawbacks of using an interface where the user can decide the position of the menu is that the interaction buttons are not necessarily guaranteed to be on screen. This is true for any interface that has movable windows with control icons in fixed layouts. In the PDF viewer in SAGE, there are a number of interaction widgets fixed to the bottom of the window allowing the user to select the first, previous, next, or last page. Moving or resizing the window such that the buttons are off screen and inaccessible without adjustment is a common occurrence. SAGETouchUI resolves this by allowing the button widgets to be aware of their position relative to the screen, and lock to the bottom before they can go off screen. This allows the buttons to be accessible at all times. To avoid occluding the content of the window in this case, the buttons become semi-transparent after a few seconds of inactivity.

The menu does a similar adjustment if the user attempts to open a left-handed menu on the far right side of the screen. In this case the menu will open in using the right-handed layout so that the content will appear on the left side of the menu and still on screen.

# 5. IMPLEMENTATION

This chapter describes the implementation of SAGETouchUI. First the hardware setup will be described followed by several sections explaining the software components of SAGETouchUI and how they communicate with other parts of the system. This chapter will also provide the rationale behind some of the software design decisions.

## 5.1 **System Overview**

This user interface design was developed for the Cyber-Commons touch wall at the Electronic Visualization Laboratory at the University of Illinois at Chicago. The Cyber-Commons wall consists of eighteen ultra-thin bezel LCD displays connected to either a Linux-based single machine or a cluster of six machines. The system uses an additional Windows machine to drive a PQLabs Multi-Touch Wall overlay which creates a grid of infrared light across the twenty foot display and provides the position and size of 32+ touch points breaking the beam.

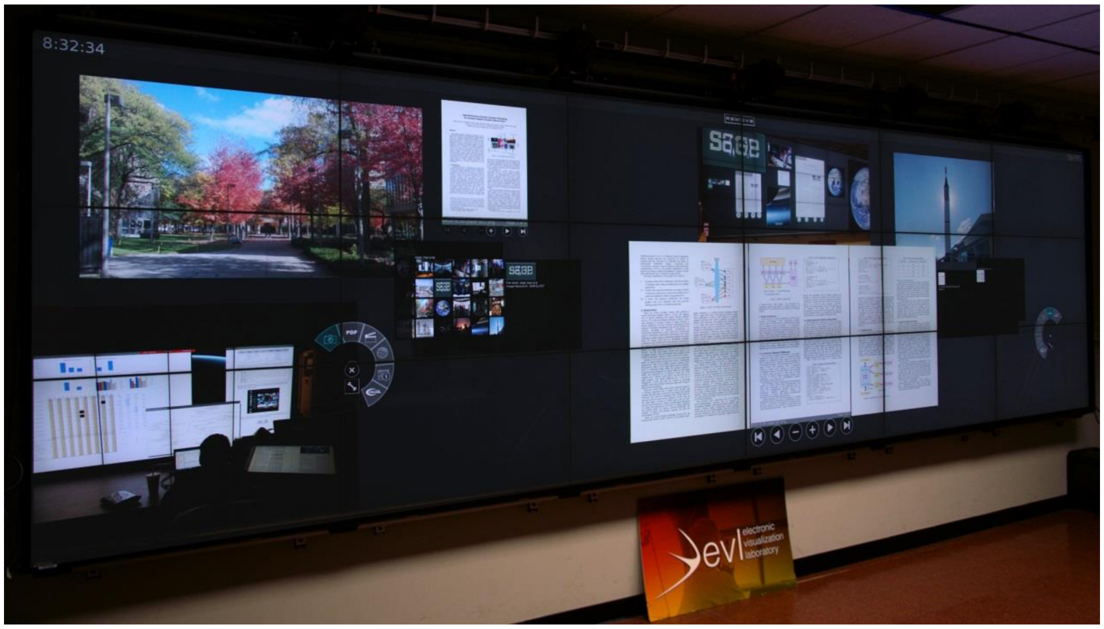

Figure 2: SAGETouchUI

## 5.2 **OmicronSDK**

The OmicronSDK is a C++ input abstraction and utility library developed at the EVL to receive and process input events and stream the events to clients connecting to the Omicron input server [5]. Omicron receives raw touch data using the PQLabs API and generates an Omicron event. Omicron events are abstract data containers which can store different types of input data such as pointers, motion capture positions, and controller buttons. This abstraction allows any application to connect to an Omicron input server, receive different inputs in a common format, and use the events on a per-application basis. Hansen, Hourcade, Virbel, Patali, and Serra [8] used a similar event model in their multi-touch toolkit. Previous applications using Omicron have received the raw input data directly and processed the event internally. As the SAGETouchUI relies on multi-touch gestures, a new touch gesture manager was added to the Omicron input server to allow future applications to take advantage of the touch gestures implemented for this methodology.

## 5.3 **Software Development Environment**

SAGETouchUI was developed using Processing, a prototyping and visualization tool started by Ben Fry and Casey Reas [6]. Processing was originally designed as a tool for visual designers and artists. Much of the language is designed to simplify drawing and displaying interactive images which made it an ideal environment for developing and prototyping a user interface. Also since Processing is based on Java, this allows SAGETouchUI to be cross-platform encouraging development on Windows, OSX, and Linux.

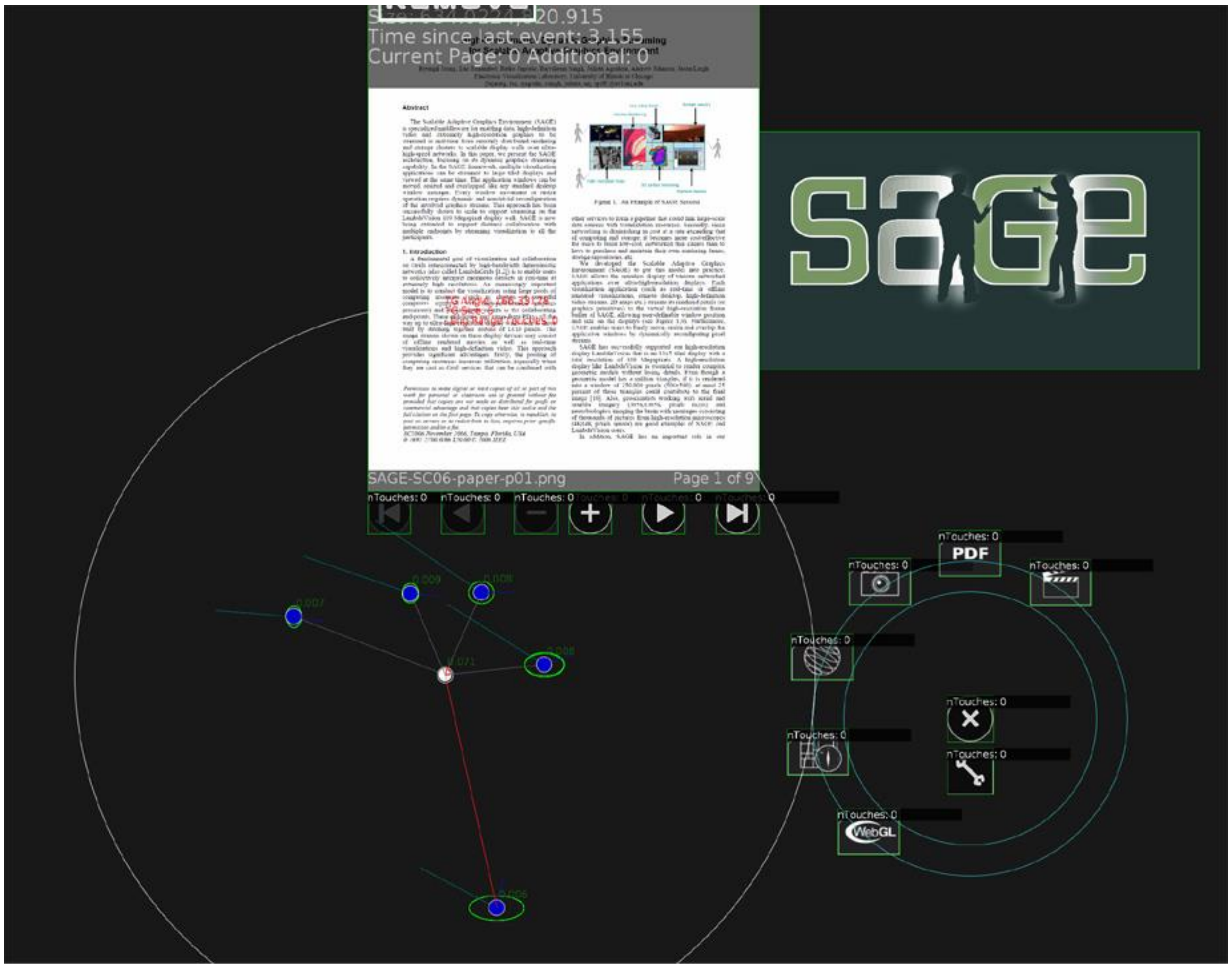

Figure 3: Debug mode showing bounding boxes and detailed touch info

Some of the drawbacks of using Processing include the lack of flexibility in memory management that SAGE is capable of doing. Compared to the C++ backend driving SAGE, SAGETouchUI is limited in the number and size of content it can open. This drawback was not a large concern as the point of SAGETouchUI is to prototype the user interface, not to fully re-implement the capabilities of SAGE.

## 5.4 <u>**Touch Gestures**</u>

The use of touch gestures within SAGETouchUI was implemented in two stages. The first stage had the touch gesture manager integrated within the SAGETouchUI. The second stage extracted the gesture manager into an external application which streamed gesture events to SAGETouchUI. The first method allows all user interface widgets to have access to all available touch data. The second has the advantage of allowing the gesture manager to act as a server to stream gesture events to other applications.

## 5.5 <u>**Structure of SAGETouchUI**</u>

SAGETouchUI was designed to mimic the most basic SAGE features so that new user interface could be prototyped and evaluated relatively quickly. The interactive user interface elements are Content Widgets, Content Windows, and the Mediabrowser. SAGETouchUI also consists of multiple managers which handle various components of its operation:

- Omicron Manager
- Touch Gesture Manager
- UI Manager
- Content Manager
- Cluster Manager (Event Server and Cluster Client)

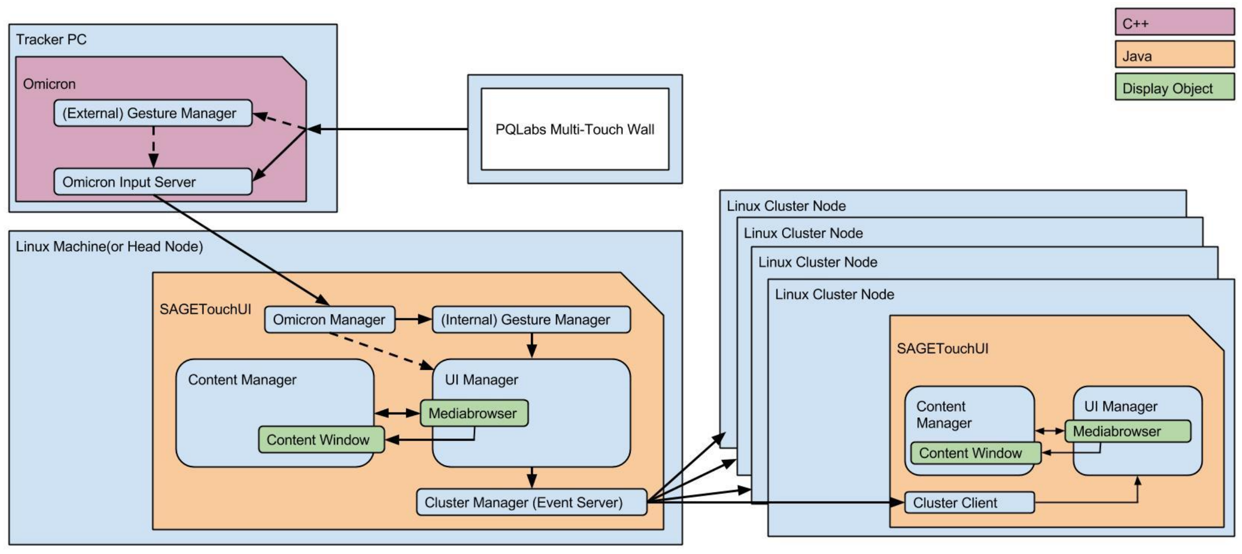

Figure 4: Input event flow diagram

Figure 4 shows the flow of input events from the PQLabs multi-touch hardware to the user interface display objects. The dashed line shows an alternate flow if the C++ external touch gesture manager is used instead. In this case gesture events are generated on the input server, streamed to Omicron Manager, and then passed directly to the UI Manager.

### 5.5.1 **Content Widget**

Content Widgets provide the base class for all interactive user interface elements in SAGETouchUI. Each Content Widget keeps track of how many inputs event there are inside it, how long those events have been there, and maintains what its current state is such as idle, pressed, and clicked. The Button Widget class builds off the Content Widget class by adding image support as well as a selected or unselected state. The buttons on the menu are made of these widgets. Thumbnail Widgets are derived from the Button Widget class and have the additional functionality of sharing information with the Content Manager to display metadata and open the full resolution content as a Content Window.

### 5.5.2 **Content Window**

The Content Window is the base class for displaying image or PDF content on the shared canvas. The class processes the input events that translate and scale the window across the canvas. The derived Image Window and PDF Window classes support the display of their specific content types and additional interaction features such as the button widgets used to select different pages in the PDF Window.

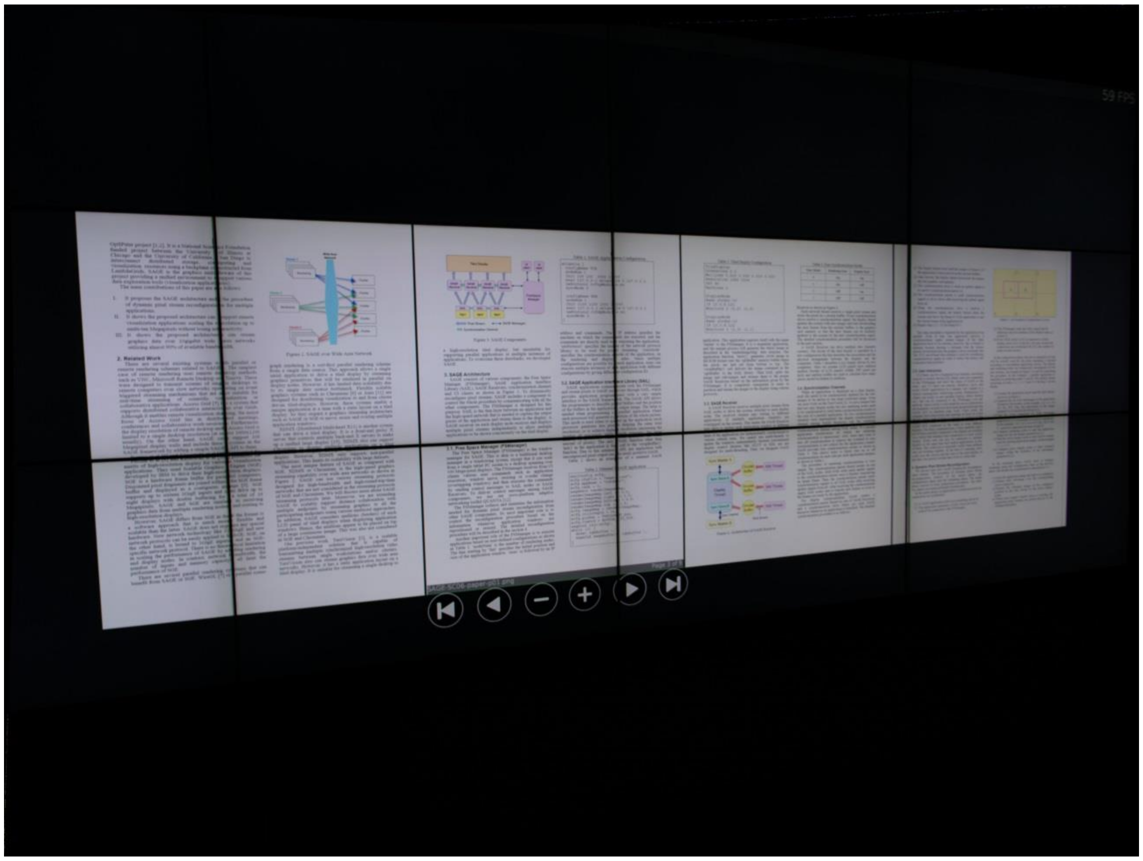

Figure 5: Multi-page PDFWindow

Figure 5 illustrates the multi-page PDF window. As scalable resolution touch walls can have the additional screen space to view large amounts of data, the PDF window presented here expands on the single page format of SAGE and allows multiple pages to be viewed simultaneously. The button widgets attached to the PDF window are from left to right: first page, previous page, collapse multi-page view, expand multi-page view, next page, and last page. Following the same concept of the localized radial menu, the button widget bar will switch to the page where the last touch occurred. This allows the user to bring the widget bar to their location anywhere in front of the PDF window.

### 5.5.3 **Mediabrowser**

The Mediabrowser is divided into two components: the radial menu and the thumbnail window. The radial menu is the starting point for all interactions in SAGETouchUI. Using a three-finger gesture, the radial menu opens at that location providing a list of buttons for opening up windows around the menu. There are three layouts for the menu: Left-handed radial, centered, or right-handed radial. If the Gesture Manager detects a left-handed, three-finger gesture, a left-handed radial menu will appear. For both radial menus, the buttons are arranged such that there are no major buttons in where the user's arm is likely occluding the view of the screen. Depending on the size of the user's hand, the radius of the menu will adapt so that the buttons are just beyond the user's hand radius to prevent accidental button clicks. The menu also has a minimum size to prevent the icons from getting too small.

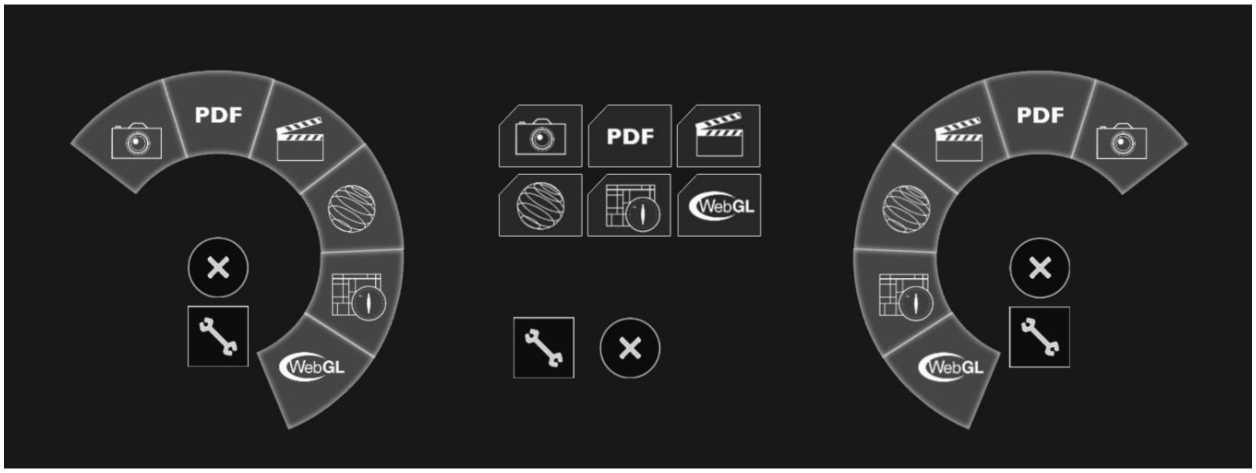

Figure 6: Menu Layouts (Left-Handed Radial, Centered, Right-Handed Radial)

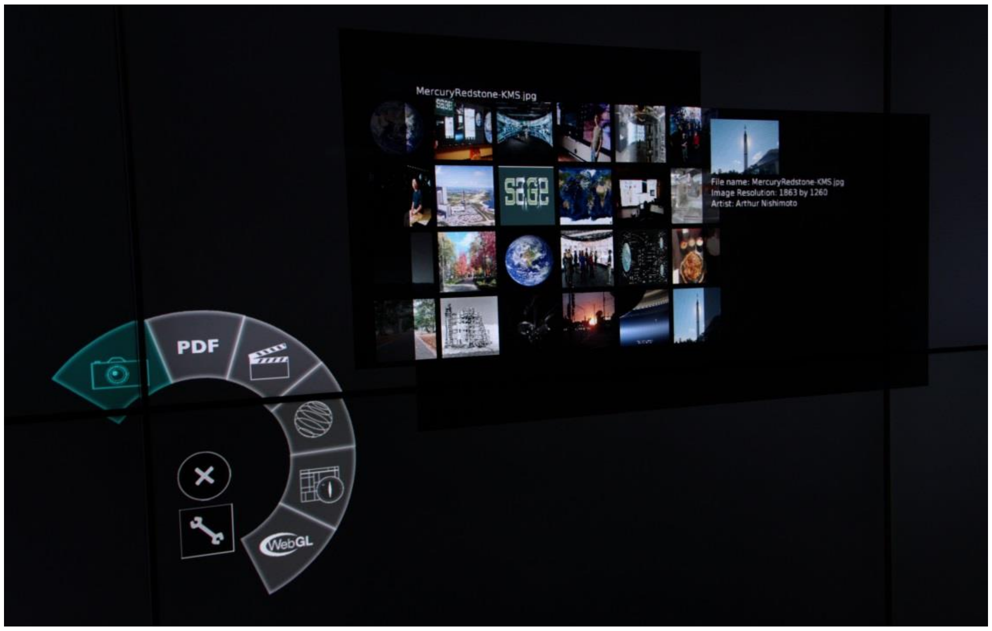
MercuryRedstone-KMS.jpg
File name: MercuryRedstone-KMS.jpg
Image Resolution: 1863 by 1260
Artist: Arthur Nishimoto
PDF
WebGL

Figure 7: Radial menu and image thumbnail window

For the current prototype, there are four functional buttons on the radial menu: images, PDFs, options, and close menu. The remaining icons are placeholders to fill out potential layouts of the menu. The image and PDF windows are used as scrollable thumbnail viewers for displaying hundreds of thumbnails in a constrained space. These windows also have a side window for displaying additional metadata information and a preview image of the media content. Clicking on any of the thumbnails will open the media content as a content window on the shared canvas.

The options menu contains controls primarily for debugging and testing purposes. This can be visual debugging such as displaying widget bounding boxes and verbose visual touch information or adjusting the layout and behavior of the interface. Many of the layout options are used as part of the user study in Chapter 6. All of the windows are attached to the radial menu so that the user can move the menu and window together to any desired location on the screen.

Similar to how button widgets on the PDF window are aware if they are off-screen, the thumbnail window can check if the edge of the window is off-screen or overlapping with another user's thumbnail window. In these cases the window will resize or move to avoid the overlap.

### 5.5.4 **Omicron Manager**

The Omicron Manager handles incoming communication with an external Omicron input server which aggregates various input devices into a generalized Omicron event. Within the current Omicron event system, Pointer events can be a mouse, SAGEPointer, or touch overlay events. Pointer events consist of raw input information. All events have an ID for that pointer to differentiate between different SAGEPointers or touch fingers. Events follow the same basic flow which consists of an initial or down event followed by move or dragging events to finally an up or release event when the pointer has finished. Depending on the pointer type there are slight variations of this flow. For example a mouse or SAGE pointer can hover freely over the display wall and click and drag when needed. In a touch event there is no notion of hover so every basic touch event is a click, drag, and release. These basic down, move, and up events are passed to the Touch Gesture Manager to create more advanced interactions. In a later version of SAGETouchUI, this manager also accepts touch gesture events generated from an external server and passes them directly to the User Interface or UI Manager.

### 5.5.5 <u>**Touch Gesture Manager**</u>

The touch gesture manager receives raw touch data information consisting of a touch ID, x and y position, and an event state of either down, move, or up. The gesture manager will group touch points close to one another into a TouchGroup object. Each TouchGroup object has an ID based on the first touch that created it, any touch points that were created in its radius, as well as its position based on the average position of all touch points in the group. This ID essentially becomes a user ID or at the least a user's hand ID. Each touch point also contains its last position and if the point is stationary and idle. The TouchGroup will also use this information to determine the behavior of the touch points and in turn what gestures to generate. Some gestures will use basic information about the TouchGroup such as the five-finger 'close content' gesture which simply checks if there are at least five stationary touch points within the group. Gestures like zoom will look at the direction of two touch points to determine if they are moving apart in opposite directions within an arc.

| **Gesture** | **Usage** |
|---|---|
| Down/Press, Move/Drag, Up/Release | Selecting / Dragging |
| Click | Button Interaction |
| Double Click | Maximizing Content |
| Three-Finger Hold | Open Menu |
| Five-Finger Hold | Closing Content |
| Big / Fist | Push to Back |

Table I: List of current gestures

Some gestures will look at the arrangement of touch points to determine the handedness of the gesture. To determine if a gesture is left-handed, the TouchGroup will look for the farthest touch point from the center of mass of a gesture. If the point is clearly farther than the other points and to the right of the center of mass, then that point is assumed to be the thumb of a left hand. Likewise if the finger is to the right of the center of mass, that point is assumed to be the thumb of a right hand. After new gesture events are generated, these events are passed either directly or through the cluster Event Server to the UI Manager.

### 5.5.6 <u>**UI Manager**</u>

The UI Manager primarily keeps track of all currently open Mediabrowser menus and streams input events to those menus. It also handles the creation of the menus and handles overlapping between menus or the screen edge. Each Mediabrowser handles input to all buttons and thumbnails inside the menu. When the thumbnail window is generated or when a thumbnail is selected, the Content Manager takes over to open the content.

### 5.5.7 <u>**Content Manager**</u>

On startup, the Content Manager loads all saved media content and generates a thumbnail collection that the UI Manager will use to create the thumbnail windows for the media browsers. In addition to the image and PDF content, the Content Manager will also collect and store metadata information which the Mediabrowser can display for each file. When a Mediabrowser’s thumbnail icon is selected, the Content Manager will then open the full resolution image as a Content Window on the shared canvas. The Content Manager also will handle passing input event to Content Windows.

### 5.5.8 **Cluster Manager**

The Cluster Manager consists of two components: the Event Server and Cluster Client. Both these components provide the infrastructure needed to run SAGETouchUI on a cluster of computers rather than the initial single machine design.

The Event Server runs on the instance of SAGETouchUI designated as the master node by a configuration file. When cluster mode is enabled, the UI Manager will send additional commands to the Event Server which will then be send to the client nodes. These commands include events such as opening a new menu with ID 2 at position x, y, thumbnail ID 42 on menu 3 was pressed, or content window ID 37 was moved to position x, y. Only the master node will receive input events from the Omicron Manager while the clients will receive these UI events from the Event Server.

The Cluster Client receives commands from the master node instance and passes them to the UI Manager to synchronize the state of all menus and Content Windows. The Cluster Client also reads the configuration file to determine what part of the shared canvas the client instance should display. The client receives data on either a TCP or UDP stream. The TCP stream is used for critical single instance events such as a new menu opening or button presses. The UDP stream is used for constantly streaming events such as dragging a Content Window or scrolling the thumbnail viewer.

# 6. EVALUATION

This chapter evaluates SAGETouchUI by first examining the performance of the software components of the system followed by a user study designed to evaluate the usability and user experiences with the system. This chapter will end with the results and insight gained from these evaluations and what the future design implications are.

## 6.1 System Evaluation

As a user interface by design must receive user input, process the input, and respond in a real-time, the SAGETouchUI prototype's various managers were evaluated to determine how fast the system was transmitting data, processing user input, and finally rendering a response on screen. Due to the iterative design of SAGETouchUI, three versions of the software were tested. The first runs on a single machine and receives basic touch point information and does all gesture recognition. This will be referred to as using the internal Gesture Manager. The second also runs on a single machine, but receives gesture events from the external Gesture Manager running on the Windows machine directly attached to the touch overlay. The third runs on a six node cluster and measures the performance of the Cluster Manager.

### 6.1.1 System Hardware

The PQLabs touch overlay is connected via USB 2.0 to a Windows 8 Core i7 machine which streams touch events on 100 Mbps Ethernet connection to the SAGETouchUI application running on either a single OpenSUSE 12.1 Dual Intel Xeon X56560 2.66 GHz using three NVIDIA GTX480s or a cluster head node using Dual Intel Xeon X5675 3.06GHz, an NVIDIA GTX 580, and a 10G network connection to the cluster. For the user study described later, the cluster configuration was used.

### 6.1.2 **Internal Gesture Manager**

To evaluate the internal gesture manager, the timestamp of when an event would enter and leave one of the managers was saved to a log file. Table 2 shows the flow of input events of a three-finger gesture from individual touch points to the menu appearing on screen.

| | Time (milliseconds) | | |
|---|---|---|---|
| | Minimum | Maximum | Average |
| Touch overlay to Omicron Input Server (C++) event sent | 0 | 10 | 2.19 |
| Network from Omicron Input Server to SAGETouchUI (Java) | 0 | 1 | 0.01 |
| SAGETouchUI receives event, passes to Touch Gesture Manger | 0 | 1 | 0.07 |
| Internal Gesture Manager identifies and generates gesture | 33 | 1083 | 154.00 |
| Gesture passed to UIManager and displayed on screen | 1 | 17 | 16.02 |
| Total | 34 | 1112 | 172.28 |

Table II: Menu open speed using the internal gesture manager

### 6.1.3 **External Gesture Manager**

The external gesture manager is nearly identical to the implementation of the internal gesture manager. The key difference is the external gesture manager is written in C++ rather than the internal gesture manager's Java implementation. The external gesture manager is a module within the OmicronAPI's input server. Once the touch events are received by Omicron, they are passed to the gesture manager which in turn generates a new gesture event which is sent to SAGETouchUI. The performance of this is shown in Table 3.

| | Time (milliseconds) | | |
|---|---|---|---|
| | Minimum | Maximum | Average |
| Touch overlay to Omicron Gesture Manager | 0 | 0 | 0.00 |
| Gesture Manager identifies and generates gesture | 5 | 16 | 8.63 |
| Gesture event send out | 0 | 1 | 0.04 |
| Network from Omicron Input Server to SAGETouchUI | 0 | 1 | 0.01 |
| Gesture event passed to UIManager and menu displayed | 1 | 18 | 16.27 |
| Total | 6 | 36 | 24.95 |

Table III: Menu open speed using the external gesture manager

### 6.1.4 **Gesture Manager Analysis**

Comparing the performance of the internal and external gesture managers, the most notable measurement is the time for identifying and generating the three-finger gesture. The internal Java-based manager on average took 154 milliseconds while the C++ manager took 8.63 milliseconds. The external C++ gesture manager has the advantage of performing faster as well as providing gesture events to any application connecting to the input server rather than having the application compute the gesture on its own as the internal Java gesture manager does. As a prototyping environment, it was more advantageous to the internal Java gesture manager for a number of reasons. First the internal gesture manager allows the application to have access to all the available touch data, not just the specific set of gestures provided by the input server. This allows the user interface to have access to the raw touch data at the widget level allowing the developer more control of how a particular widget will use that information. Also having full access to the gesture manager and touch data allowed for easier testing and debugging of touch widgets and new gestures since using Processsing, debugging information can be quickly displayed on screen during runtime. Finally 172 milliseconds from incoming touch data to displaying the menu on screen is well within reasonable real-time performance for a user interface menu.

### 6.1.5 Cluster Manager

The user study described later ran using the cluster version of SAGETouchUI. With synchronization events being received at 14.23 milliseconds on average, the user interface performed well between cluster nodes. Some of the participants in the user study who were familiar with running Processing applications on a single machine configuration of the Cyber-Commons wall commented that there was barely any noticeable difference between running a Processing sketch on a single machine or a cluster.

| | Time (milliseconds) |
|---:|---:|
| Network from SAGETouchUI master node to display node | 0.11 |
| Gesture event passed to UIManager and menu displayed | 14.12 |
| Total | 14.23 |

Table IV: Menu open speed between master and display nodes

## 6.2 User Experience

In order to evaluate the usability of the interface prototyped in SAGETouchUI, 15 participants (11 male, 4 female) took part in a user study. Participants were a mix of undergraduate, graduate, and faculty at UIC. A majority of the students who participated were from Computer Science with others from Communications, Art, and Engineering. Eleven of the participants reported as right-handed, one as left-handed, and the remaining three with no preference. The focus of the study was the usability of two aspects of the Mediabrowser. The first was the reliability of the gesture manager to detect and trigger the menu. The second examined the usability of different layouts of the menu and the thumbnail window components of the media browser.

### 6.2.1 **User Study Design**

The user study used to evaluate the interface was divided into two parts. The first part evaluated the usability of the radial menu and thumbnail window. The second part focused on how the interface should handle overlapping menus in a simulated multi-user setting. Prior to beginning the study, participants were given an overview of the multi-touch wall and encouraged to try out the touch gesture and user interface before starting the task.

For the user study the thumbnail viewer was populated with colored thumbnail images. Participants were tasked with using the radial menu and thumbnail browser to search for all thumbnails of a particular color. This task was repeated using five different layouts of the menu and thumbnail window. The radial menu used three layouts: left-handed, centered, and right-handed. The thumbnail browser was also presented in two different sizes: within and outside comfortable reach.

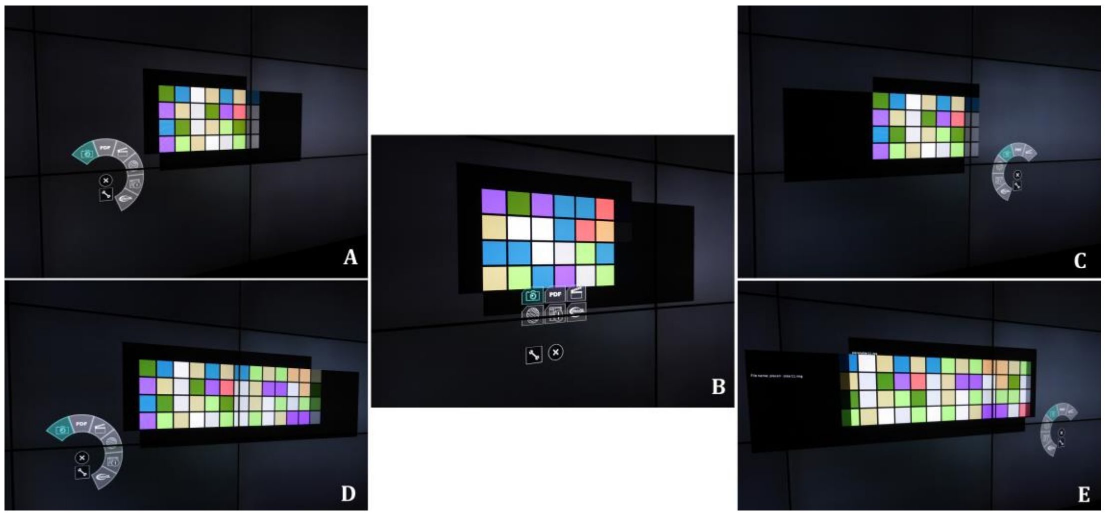

Figure 8: Menu layout types. A: Left-handed radial menu with small thumbnail window. B: Centered small window. C: Right-handed radial small window. D: Left-handed large window. E: Right-handed large window

The second part of the study focused on how overlapping menus affected the user experience and evaluated potential methods of how the interface would handle the overlap. While still performing the search task in part one, a second automated menu was programmed to appear near and overlap with the participant's menu. The interface would handle the overlap using four methods. The first method simply allows the second menu to cover up the existing one (Figure 9, left). The second and third methods (Figure 9, center) resized the menu giving one menu priority over the other. The second method would give spatial priority to the newer menu. The third method would give spatial priority to the older menu. The final method scaled both menus such that the space was divided evenly (Figure 9, right)

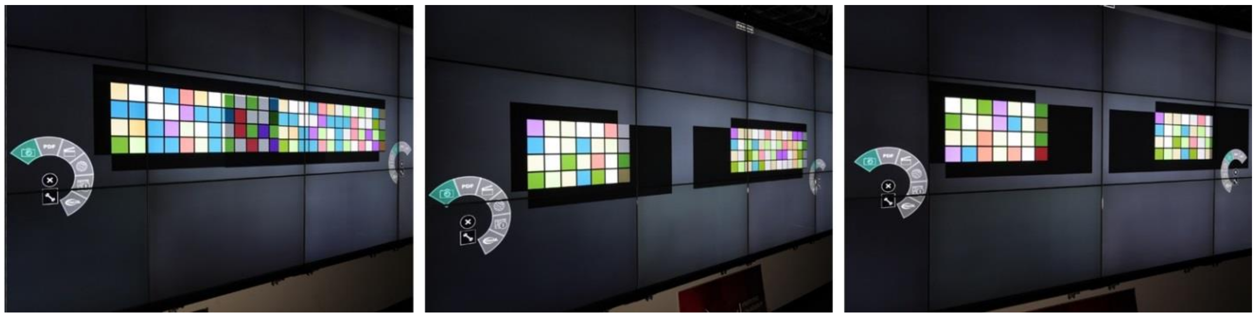

Figure 9: Menu overlap methods: Method 1: Allow overlap (left), Method 2 and 3: spatial priority given to one menu (center), Method 4: equal spacing (right)

### 6.2.2 <u>**Data Collection**</u>

For the study, all input and event data was recorded in addition to video recordings to observe where participants were looking and where they moved relative to the interface presented. Following each part of the study, participants were given a survey to gauge user preferences to the different interface layouts presented as well as general questions about the experience with the interface and any areas of improvement that could be made.

### 6.2.3 <u>**Observations on Menu Layout**</u>

For all trials, participants opened the menu using the three-finger gesture, opened the scrollable thumbnail window, and searched for and selected all thumbnails of a particular type. For part one of the study, the participants were presented three different layouts of the menu relative to the thumbnail window: left-handed menu, centered menu, and right-handed menu. For this part of the study the automatic menu layout selection based on which hand the participant uses to open the menu was disabled. The centered menu opens the thumbnail window above and centered from the position of the menu. As seen in Table 5, most participants used their right hand to open the menu and select thumbnails. This is likely due to almost all of the participants indicating that they are right-handed.

| | |
|---|---:|
| Menu opened with left hand | 23 % |
| Menu opened with right hand | 77 % |
| Select with left hand | 6 % |
| Select with right hand | 65 % |
| Select with both hands | 29 % |

Table V: Hand Usage for Centered Menu

The right-handed menu was designed such that when a user would use their right hand to open the menu, the thumbnail window would open to the left of the menu thus in front of the user's field of view. The left-handed menu would open the thumbnail window to the right of the menu.

| | |
|---|---:|
| Menu opened with left hand | 12 % |
| Menu opened with right hand | 88 % |
| Select with left hand | 16 % |
| Select with right hand | 36 % |
| Select with both hands | 47 % |

Table VI: Hand Usage for Right-Handed Menu

For one set of trials, the interface only showed the right-handed menu. As designed, 88% of participants used their right hand to open the menu. Most participants who started using their left hand to open the menu would eventually switch to their right after the first trial. As seen with the centered menu, when participants favored one hand over the other, the right-handedness of the participants is apparent.

Another set of trials had the interface only showed the left-handed menu. Many participants switched to their left hand to open the menu after previously favoring the right hand of opening the menu. According to the survey nine of the fifteen participants preferred the left-handed menu out of the three layouts. A commonly reported reason for the preference was that the left-handed menu allowed for easier selection using their right hand.

| | |
|---|---:|
| Menu opened with left hand | 58 % |
| Menu opened with right hand | 42 % |
| Select with left hand | 3 % |
| Select with right hand | 59 % |
| Select with both hands | 38 % |

Table VII: Hand Usage for Left-Handed Menu

### 6.2.4 **Observations on Window Size**

In addition to the different layouts of the menu relative to the thumbnail window, two different sizes of the thumbnail window were presented to the user. The smaller version is designed to keep the entire thumbnail window within a comfortable arm's reach of the menu's initial position. The larger version is double the length with part of the menu outside arm's reach.

| | Thumbnail Window Size | |
|---|---|---|
| | Small | Large |
| Select with left hand | 16 % | 4 % |
| Select with right hand | 45 % | 51 % |
| Select with both hands | 39 % | 46 % |
| Multiple hands used | 36 % | 38 % |
| Multiple fingers used | 28 % | 25 % |
| Sidestepping used | 15 % | 71 % |

Table VIII: Thumbnail Window Observations

Considering part of the thumbnail window is out of reach, it was not surprising that 71% of the time participants would physically move in order to select the thumbnail targets. Most participants would sidestep to the center of the thumbnail window and remain at that location until all targets were found. Three participants continually moved in a 'typewriter' fashion where they would walk to the far end of the thumbnail window, sidestep along the window clicking targets, and then walk back while scrolling the window. Four of the participants did not physically move at all, preferring to scroll the thumbnail window from their initial position.

### 6.2.5 <u>**Observations on Menu Positioning**</u>

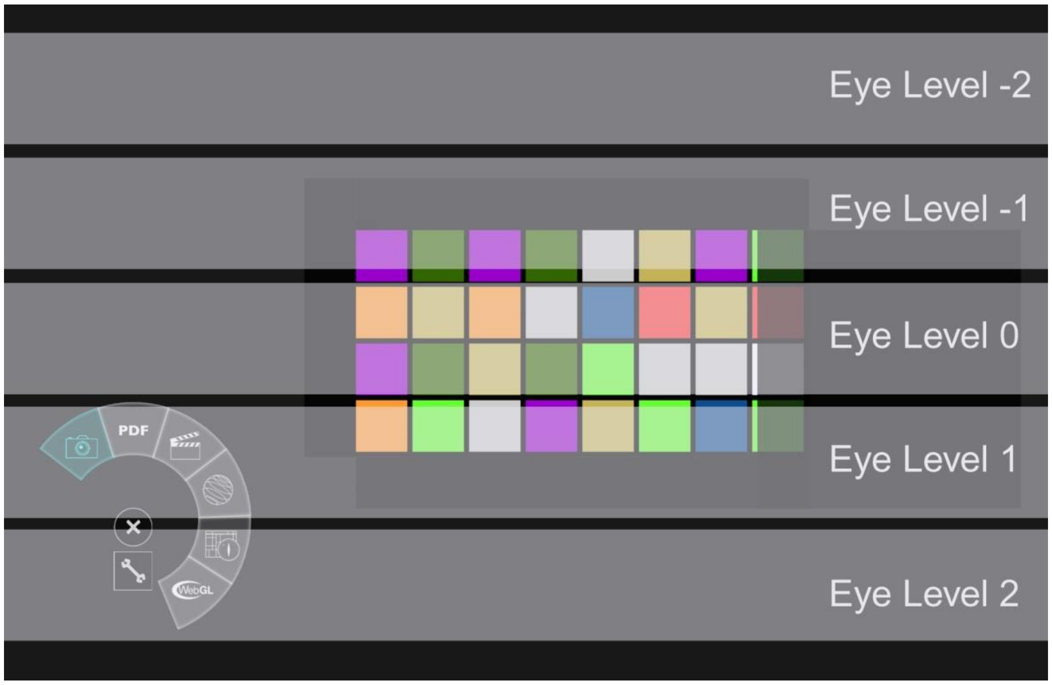

Figure 10: Menu Eye Level Coding

During the study, participants would open the menu at a certain height and typically move the menu to better view the thumbnail window. The initial menu location and the final moved location relative to the participant's eye level were recorded based on video observation. Locations were coded as shown in Figure 10. Almost all participants started the study by opening the radial menu at eye level (Level 1 or 2) followed by moving the menu down so that the thumbnail window would be at or slightly below eye level (Level 0 or -1).

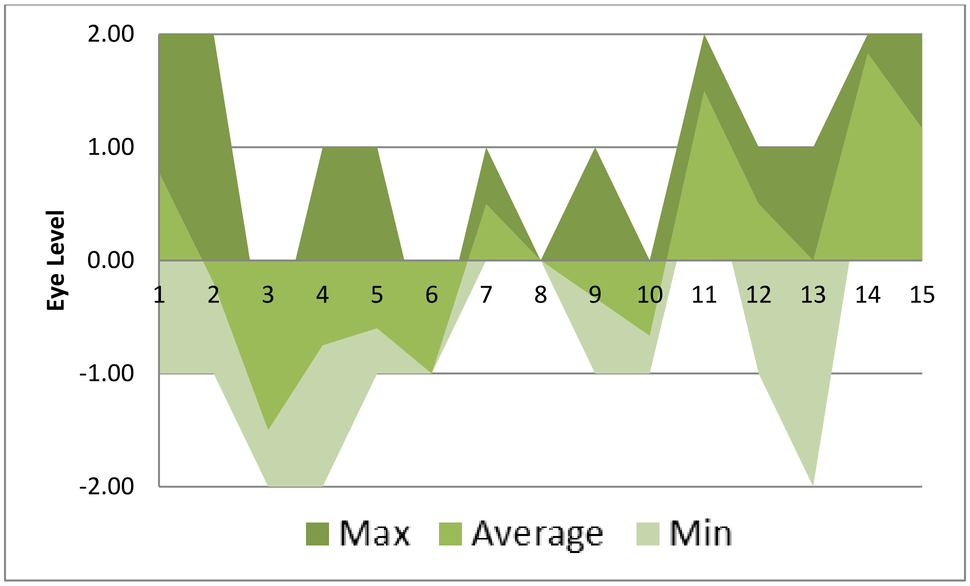

Figure 11: Initial menu height per participant

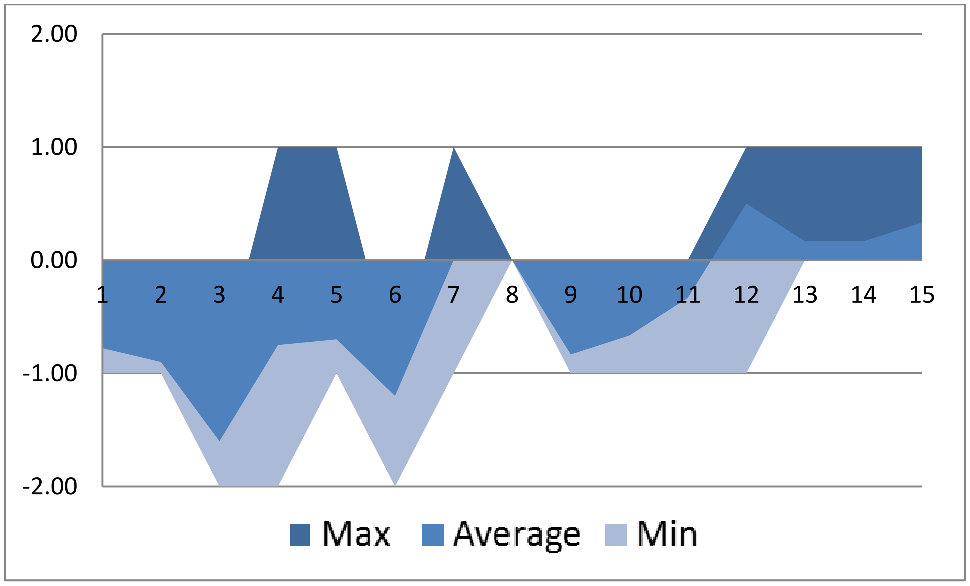

Figure 12: Final menu height per participant

Figure 11 and Figure 12 shows the initial and final menu heights for the left-handed menu, small thumbnail window trial. Almost all of the participants here opened the menu at a higher eye level initially (Level 1 or 2) and later moved the menu down to a lower eye level (Level 0 or -1). After a few trials participants would start opening the menu at a lower height so that the menu height would not have to be adjusted later. Figure 13 shows the menu height over multiple trials for one of the participants.

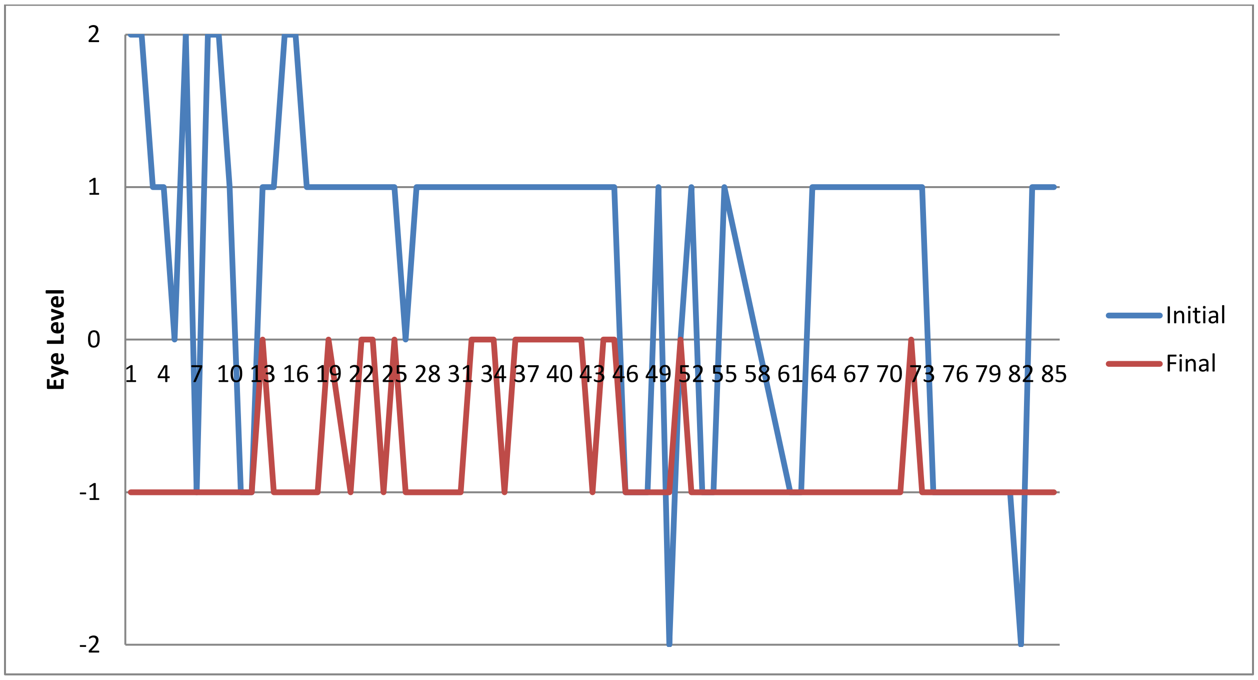

Figure 13: Menu height over multiple trials

### 6.2.6 **Survey Results**

A survey was given to all participants after completing each part of the user study. Part one of the study focused on the usability of the menu and thumbnail window as well as preferences regarding the five layouts that were presented: Centered menu with a small thumbnail window, left-handed menu with the small window, right-handed menu with the small window, left-handed menu with the large window, and right-handed menu with the large window.

| | Score out of 5 | | |
|---|---|---|---|
| | Minimum | Average | Maximum |
| Easiness of three-finger gesture | 2 | 4.07 | 5 |
| Easiness of the scrolling thumbnail window | 2 | 3.60 | 5 |
| Size of the thumbnails | 4 | 4.47 | 5 |
| Ease of moving the menu | 1 | 3.67 | 5 |
| Usefulness of moving the menu | 1 | 4.07 | 5 |
| Usability of the interface as a whole | 2 | 3.93 | 5 |

Table IX: Survey part one responses (0 – not easy/useful to 5 very easy/useful)

Part two of the survey involved how the user interface handled the overlapping of multiple thumbnail windows. For each trial the participant would be performing the thumbnail search task before the system would generate a second menu intentionally overlapping with the participant's menu. The interface handled the overlap in four ways: doing nothing and allowing the overlap, shrinking the first menu, shrinking the second menu, or equally shrinking both menus.

| | Score out of 5 | | |
|---|---|---|---|
| | Minimum | Average | Maximum |
| Usefulness of overlap manager | 1 | 3.87 | 5 |
| Shrinking the first (participant's) menu | 1 | 2.47 | 5 |
| Shrinking the second (system generated) menu | 1 | 3.07 | 5 |
| Equally shrinking both menus | 1 | 2.87 | 5 |

Table X: Overlap Manager Responses

While most of the participants found it useful to have the system handle overlapping windows, none of the three methods particularly worked out well. When the overlap occurred, the thumbnail window size was adjusted. However the layout of the internal contents of the window was not preserved in some cases. This caused confusion among many participants on what was going on.

| | |
|---|---|
| Centered Menu | 4 |
| Left-Handed Radial | 9 |
| Right-Handed Radial | 1 |
| | |
| Small Thumbnail Window | 5 |
| Large Thumbnail Window | 10 |

Table XI: Menu and thumbnail window preferences for all 15 participants

Most participants reported that the left-handed radial menu was preferred since it enabled them to use their right hand to select the thumbnails. Eleven of the participants reported being right-handed, one as left-handed, and three with no preference. A common response to preferring the centered menu was the closeness of the menu to the thumbnail window. Participants favoring the small thumbnail window cited less walking distance and keeping the window in one's field of view. Those favoring the large window cited less scrolling and easier visual scanning of information.

## 6.3 **Design Implications**

Based on the observational data on the positioning of the menu, it became clear the height of the thumbnail window should likely be at the same level or lower than the radial menu. This will allow the contents of the thumbnail window to be at a more comfortable height around eye level or below. Future designs of the thumbnail window will scroll faster as well as attempt to maximize the size of the window while remaining in comfortable arms reach. The size of the thumbnail window will also be adjusted to allow more information to be visible, but also still be within arm's reach of the user. Allowing users to expand and shrink the window size could also be beneficial to allow users to search through the window using their own preference of virtual or physical navigation.

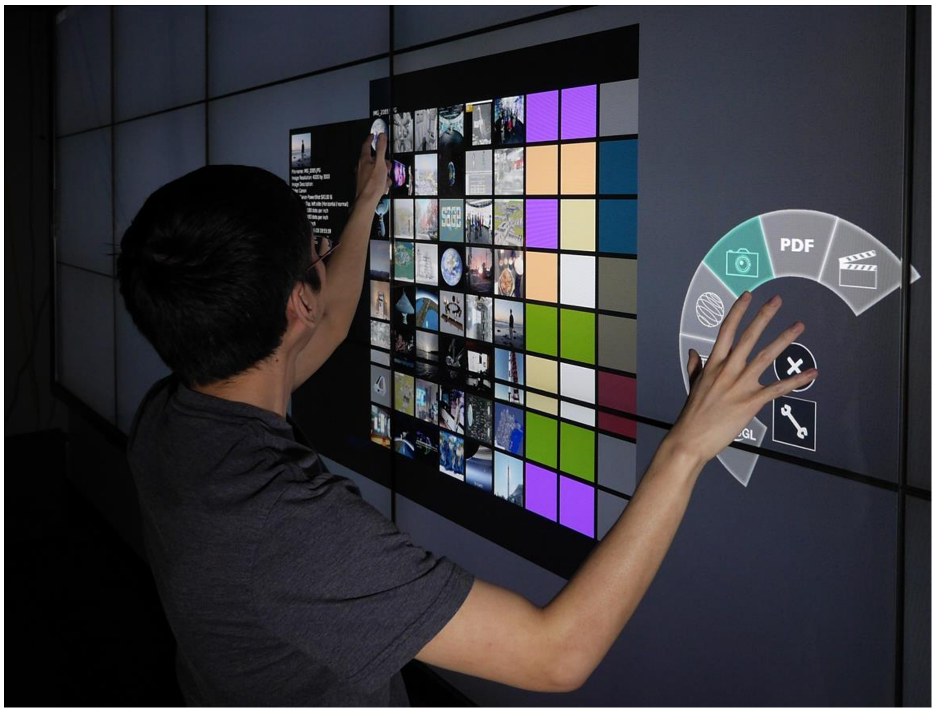

Figure 14: Revised thumbnail window using more vertical space and vertically centered radial menu

During the user study the system occasionally failed to detect the three-finger gesture. For two of the participants the gesture manager incorrectly detected the hand orientation and opened the menu contrary to expectations. However participants did report finding the option of having the system use the handedness of the user useful (Average: 3.76 out of 5, Max: 5, Min: 2). Observations of some of these events uncovered that some of the participants rotated their hand nearly 90 degrees to open a menu at arms-length to the side. The handedness calculation of the gesture will be adjusted to account for hand rotation.

# 7. CONCLUSION

Scalable resolution display walls are a growing tool for visualizing and analyzing large amounts of visual information. The larger the wall gets the more possibilities there are for multiple users to simultaneously interact with the display and make discoveries. This thesis seeks to gain a better understanding of how an increasingly popular natural user interface like multi-touch can be applied specifically to scalable resolution display walls. Building on the multi-user collaborative work for the SAGE, the SAGETouchUI application presented in this paper examines and evaluates a different, touch-centric approach to the user interface design. By decoupling fixed menu, multiple users are able to bring the menu and interaction tools to them. When the user interface can be located anywhere, the UI components should be aware of where they are relative to the screen. Where it is common for button widgets to fall off screen, SAGETouchUI demonstrates a methodology where essential widgets remain on screen and interactive with limited occlusion of important information as seen in the PDFWindow. Together the radial menu, thumbnail window, and the screen-aware widgets provide an example of a user interface tailored to allowing users have the same level of control on a scalable resolution touch wall whether working independently or collaboratively.

## 7.1 <u>Future Work</u>

Once users begin defining their own personal space on scalable resolution touch walls, how to handle crowding and overlapping of personal zones while briefly explored, could use a more comprehensive task than the simulated search task described in the evaluation chapter. Another next step would be the implementation of the user interface presented in this paper into the next version of SAGE to also allow evaluation of the interface in more rigorous use cases beyond the image and PDF viewers shown here. Other uses cases such as native applications or remote collaboration may require additional interface design considerations.

## CITED LITERATURE

1. Bailly, Gilles, Eric Lecolinet, and Yves Guiard. “Finger-count & Radial-stroke Shortcuts: 2 Techniques for Augmenting Linear Menus on Multi-touch Surfaces.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 591–594, 2010.

2. Callahan, Jack, Don Hopkins, Mark Weiser, and Ben Shneiderman. “An Empirical Comparison of Pie vs. Linear Menus.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 95–100. ACM, 1988.

3. Dam, Andries V.: “Post-WIMP User Interfaces”. *Communications of the ACM*, 2: 63–67, 1997.

4. Ecker, Ronald, Verena Broy, Andreas Butz, and Alexander De Luca. “pieTouch: A Direct Touch Gesture Interface for Interacting with in-Vehicle Information Systems.” In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 22. ACM, 2009.

5. Febretti, Alessandro and Arthur Nishimoto. Omicron. Available at: http://github.com/uic-evl/omicron.

6. Fry, Ben, and Casey Reas. Processing. Available from: http://processing.org.

7. Go, Kentaro, and Hiroki Kasuga. “Multi-tapping Shortcut: a Technique for Augmenting Linear Menus on Multi-touch Surface.” In *Proceedings of the 10th Asia Pacific Conference on Computer Human Interaction*, 209–218, 2012.

8. Hansen, Thomas E., Juan Pablo Hourcade, Mathieu Virbel, Sharath Patali, and Tiago Serra. “PyMT: A Post-WIMP Multi-Touch User Interface Toolkit.” In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, 17–24, 2009.

9. Hesselmann, Tobias, Stefan Flöring, and Marwin Schmitt. “Stacked Half-Pie Menus: Navigating Nested Menus on Interactive Tabletops.” In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, 173–80, 2009.

10. Isenberg, Petra, and Danyel Fisher. “Cambiera: Collaborative Tabletop Visual Analytics.” In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, 581–82, 2011.

11. Jagodic, Ratko, Luc Renambot, Andrew Johnson, Jason Leigh, and Sachin Deshpande. “Enabling Multi-User Interaction in Large High-Resolution Distributed Environments.” *Future Generation Computer Systems* 27, no. 7 (2011): 914–23.

12. Jakobsen, Mikkel, and Kasper Hornbæk. “Proximity and Physical Navigation in Collaborative Work with a Multi-Touch Wall-Display.” In *CHI’12 Extended Abstracts on Human Factors in Computing Systems*, 2519–24, 2012

13. Leigh, Jason, Andrew Johnson, Luc Renambot, Tom Peterka, Byungil Jeong, Daniel J. Sandin, Jonas Talandis, et al. “Scalable Resolution Display Walls.” *Proceedings of the IEEE* 101, no. 1 (January 2013): 115–29. doi:10.1109/JPROC.2012.2191609.

14. Lou, Yihua, Wenjun Wu, and Hui Zhang. “Magic Input: A Multi-User Interaction System for SAGE Based Large Tiled-Display Environment.” 157–62. IEEE, 2012. doi:10.1109/ICMEW.2012.34.

15. Ponto, Kevin, Kai Doerr, Tom Wypych, John Kooker, and Falko Kuester. “CGLXTouch: A Multi-User Multi-Touch Approach for Ultra-High-Resolution Collaborative Workspaces.” *Future Generation Computer Systems* 27, no. 6 (June 2011): 649–656. doi:10.1016/j.future.2010.11.026.

16. Rittenbruch, Markus. “CubIT: Large-scale Multi-user Presentation and Collaboration.” In *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces*, 441-444, 2013.

17. Wang, Feng, Qi Fan, Hui Deng, and Jibin Yin. “PWIG-Interactive Paradigm of Direct Touch Interaction,” 2013.

## VITA

| | |
|---|---|
| NAME: | Arthur Nishimoto |
| EDUCATION: | M.S., Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2014 |
| | B.S., Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2010 |
| PROFESSIONAL EXPERIENCE: | Research Assistant – Electronic Visualization Laboratory (EVL), University of Illinois at Chicago, 2010 – present |
| | Research Experience for Undergraduates (REU) – EVL, University of Illinois at Chicago, 2009 – 2010 |
| TEACHING EXPERIENCE: | Teaching Assistant – University of Illinois at Chicago, 2012 - 2013<br>Video Game Design and Development<br>Visualization and Visual Analytics I |
| PROFESSIONAL ACTIVITIES: | Demonstrations:<br>Global LambdaGrid Workshop, Chicago, IL, 2012.<br>Supercomputing 2011, Seattle, WA.<br>SIGGRAPH 2011, Vancouver, BC, Canada.<br>Lucasfilm, San Francisco, CA, 2011. |
| PUBLICATIONS: | Febretti, Alessandro, **Arthur Nishimoto**, Victor Mateevitsi, Luc Renambot, Andrew Johnson, and Jason Leigh. “Omegalib: A Multi-View Application Framework for Hybrid Reality Display Environments.” In *Virtual Reality (VR), 2014 iEEE*, 9–14. IEEE, 2014. |
| | Febretti, Alessandro, **Arthur Nishimoto**, Terrance Thigpen, Jonas Talandis, Lance Long, J. D. Pirtle, Tom Peterka, Alan Verlo, Maxine Brown, and Dana Plepys. “CAVE2: A Hybrid Reality Environment for Immersive Simulation and Information Analysis.” In *IS&T/SPIE Electronic Imaging*, 864903–864903. International Society for Optics and Photonics, 2013. |
| | Reda, Mhd Khairi, Dennis Chau, Yasser Mostafa, Sujatha Nagarajan, Jason Leigh, **Arthur Nishimoto**, Ed Kahler, and Jason Demeter. “Design Guidelines for Multiplayer Video Games on Multitouch Displays.” In *Computers in Entertainment, ACM (in Press).* |
| | Febretti, Alessandro, Victor A. Mateevitsi, Dennis Chau, **Arthur Nishimoto**, Brad McGinnis, Jakub Misterka, Andrew Johnson, and Jason |

Leigh. “The OmegaDesk: Towards a Hybrid 2D and 3D Work Desk.” In *Advances in Visual Computing*, 13–23. Springer, 2011.