2007-08-07

The Spillable Environment: Expanding a Handheld Device's Screen Real Estate and Interactive Capabilities

Jeffrey S. Clement
Brigham Young University - Provo

THE SPILLABLE ENVIRONMENT: EXPANDING A HANDHELD DEVICE'S
SCREEN REAL ESTATE AND INTERACTIVE CAPABILITIES

by

Jeffrey Clement

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
Brigham Young University
December 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Jeffrey Clement

This thesis has been read by each member of the following graduate committee and by unanimous vote has been found to be satisfactory.

_____   _____

Date   Dan R. Olsen

_____   _____

Date   Eric K. Ringger

_____   _____

Date   Christophe Giraud-Carrier

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Jeffrey Clement in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

_____   _____
Date   Dan R. Olsen
                                                             Chair, Graduate Committee

Accepted for the Department

_____   _____
Date   Parris K. Egbert
                                                             Graduate Coordinator

Accepted for the College

_____   _____
Date   Thomas W. Sederberg
                                                             Associate Dean
                                                             College of Physical and Mathematical Sciences

ABSTRACT

THE SPILLABLE ENVIRONMENT: EXPANDING A HANDHELD DEVICE'S
SCREEN REAL ESTATE AND INTERACTIVE CAPABILITIES

Jeffrey Clement

Department of Computer Science

Master of Science

Handheld devices have a limited amount of screen real estate. If a handheld device could take advantage of larger screens, it would create a more powerful user interface and environment. As time progresses, Moore's law predicts that the computational power of handheld devices will increase dramatically in the future, promoting the interaction with a larger screen. Users can then use their peripheral vision to recognize spatial relationships between objects and solve problems more easily with this integrated system. In the spillable environment, the handheld device uses a DiamondTouch Table, a large, touch-sensitive horizontal table, to enhance the viewing environment. When the user moves the handheld device on the DiamondTouch, the orientation of the application changes accordingly. A user can let another person see the application by rotating the handheld device in that person's direction. A user could conveniently use this system in a public area. In a business meeting, a user can easily show documents and presentations to other users around the DiamondTouch table. In an academic setting, a tutor could easily explain a concept to a student. A user could effortlessly do all of this while having all of his/her information on the handheld device. A wide range of applications could be used in these types of settings.

Table of Contents

List of Algorithms

List of Equations

LIST OF FIGURES

Chapter 1   Introduction

Handheld devices such as cell phones and personal digital assistants (PDAs) are becoming increasingly common in everyday life.   They are also becoming more computationally powerful and cheaper.   In the future, handheld devices will be as computationally powerful as current desktops allowing us to do as much on handheld devices but inhibited by screen real estate and power supply limitations.   Because handheld devices have become so prevalent, people are able to do more on the run than ever before.

Figure 1 Collage of handheld devices [AMA07]

People use their handheld devices in a number of ways.   Cell phones, Blackberrys, and PDAs provide applications for people to surf the Internet, play games, and text message. New features are continually being added that enable users to do more

with their handheld devices. With more features, people are able to be more effective in places where they normally would not be.

1.1 The problem

With the limitation of a small screen, people can not do everything solely on their handheld devices. There is also the problem of eye strain from constantly looking at a small screen. Users want to see their applications on a much larger scale. With tiny screens, mobile users are not able to use their peripheral vision to see spatial relationships in their applications. Currently, these users have to put up with small screens since these devices need to be small enough to fit in their pockets. Creative ways of handling screen real estate have been implemented in order for users to accomplish tasks. Baudisch [BAU03] created "a visualization technique that supports spatial cognition by showing users the location of off-screen objects." Yee [YEE03] used a position-tracked handheld device to see into a larger virtual space. Because of these limitations, users just deal with these inconveniences and limit themselves to certain activities on handheld devices.

1.1.1 Our Solution

Figure 2 Spillable User Interface

We create an environment that will allow users to "spill" the user interface from their handheld device onto a much larger horizontal screen (Figures 2 and 3). Consequently, the user now has an enhanced environment enabling the use of his/her peripheral vision. One of the unique ideas in this environment is that the handheld device is on top of a large horizontal display connected to another computer. The displays of the two computers are synchronized, and the location of the handheld device is made known to the device controlling the horizontal screen. Because the handheld device is connected wirelessly with another computer, the amount of bandwidth has to be limited so that handheld devices can display interactively.
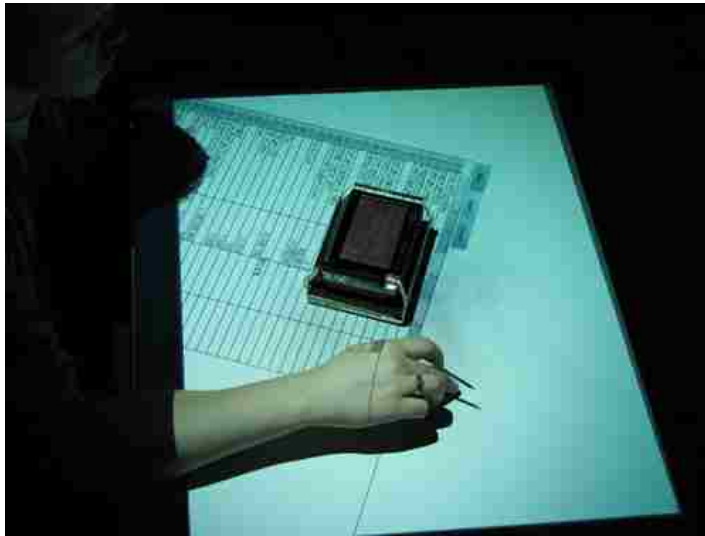
Figure 3 Spreadsheet Application on Spillable User Interface

Joe is a businessman who travels frequently. He has a mobile device that carries all of his information. When he is on the go, he uses the device by itself with minimal security risk. When Joe wants to connect to another computer, a number of issues arise. He is worried that the remote machine will have too much access to his mobile device and infect it with a virus or spyware. Since the remote machine to which the mobile device connects might be a public computer, there is even less trust. Therefore, the remote machine runs a display server that does not accept input from the remote keyboard and mouse, but does accept input from the DiamondTouch table for moving the application around. A display server holds a copy of the data model and displays it on the screen without the ability to modify the information in the data model. Input is received only through the handheld device and the large horizontal surface.

4

Figure 4 Sony VAIO Micro PC [AMA07]

Some of the handheld devices that were considered for the spillable environment were the Samsung Ultra Mobile PC and the Sony VAIO Micro PC (Figure 4). These devices are slightly bigger than a normal handheld device but allow us to create a prototype of the future environment more easily. The Sony VAIO Micro PC runs Windows XP Professional. It was chosen, although any device that runs the normal full edition of Java would work. This allowed us to program with flexibility rather than worry about the capabilities of the mobile editions. This helped us to quickly prototype the ideas of this thesis under the assumption that it would not pose a problem in the future.

Figure 5 DiamondTouch Table

Mitsubishi Electric's DiamondTouch table (Figure 5) was chosen as a large horizontal surface on which to show the expanded environment. The DiamondTouch

table is special in its ability to sense touch using capacitance [DIE01]. This ability is what allows an enhanced handheld device to be used on the DiamondTouch table. The computer connected to the DiamondTouch table uses a projector to display applications onto the DiamondTouch table (Figure 6). The DiamondTouch table was not the only choice to sense location. Other devices, like the WACOM pen with the WACOM tablet (Figure 7) [WAC07], could have been used. One could embed the sensing technology from the WACOM pen into the handheld device. However, using body capacitance is a simpler solution.

Figure 6 Projector onto DiamondTouch table

By using all of these devices together, a new unique environment is created for mobile users. A user is able to spill his/her application onto the DiamondTouch table. When the user rotates the handheld device on the table, the application has the same orientation as the handheld device. The user also has the ability to scroll the application around on the large screen to see and bring into focus certain parts of the application on

the handheld device where direct input is possible. These controls allow the user to comfortably position applications.

Figure 7 WACOM Tablet with WACOM pen [WAC07]

1.1.2 Examples

With computer stations publicly available, Joe could easily use his handheld device to connect with a larger screen to accomplish more complex tasks. Tasks usually done on handheld devices have been limited in their capabilities due to the small form factor. Handheld devices are generally slow computationally and have a short battery life. Most of these limitations will be overcome in the future except for the limited screen size, which is inherent in handheld devices.

Figure 8 DiamondTouch table in a business setting [MIT06]

The branch of the company that Joe works for has a new product which the company is excited about, so Joe sets up a meeting with some of his co-workers to demonstrate the product and get their input (Figure 8). During the meeting, Joe uses his handheld device to show the most current version of the new product using the spillable environment. In the spillable environment, Joe is able to modify documents pertinent to the new product by using the touch-sensitive screen on the handheld device. The applications used in the meeting can be shown in many different positions with the input from the DiamondTouch table. Throughout the meeting, minor changes are made to the drawings and stored on his handheld device before Joe presents it to his superiors in a different city. His co-workers are all able to sit around the DiamondTouch table and interact with the environment while not being uncomfortably close to each other (Figure 9). After the meeting, Joe heads to the airport.

Figure 9 Spillable User Interface in Group Setting

While Joe is waiting for his plane, he thinks of a new idea to add to the product, and so he decides to go to an internet café that has a spillable environment and uses it so he can see his designs easily. On the plane, Joe receives a phone call from one of his co-workers about ways to improve his presentation. During the flight, Joe makes modifications to his presentation so that it will be up-to-date with all of the changes that have been made to the designs of the product. After landing and claiming his luggage, Joe heads to the hotel in a cab. He checks with the hotel front desk to confirm his reservation of the conference room with the computer and the spillable environment. Joe continues his business trip by showing the executives the new product and using the spillable environment in the conference room.

1.1.3 Measuring Success

To show that the concept of this spillable environment works, applications must be built to show the proof of concept. These applications were initially written for the general case of the eXtended Interactive Computing Everywhere (XICE) Architecture. The applications were then modified slightly to fit our spillable environment. A number of applications can be run using our architecture without major modifications; therefore, we have demonstrated that the system has potential to allow users to enhance their handheld devices with a larger screen. Examples of applications that were built are: note-taking application, spreadsheet application, and drawing application. Part of using these applications in this novel environment is to show that they are interactive. When the handheld device is connected to the DiamondTouch table, the latency of the refresh or

paint updates on the handheld device needs to be interactive and therefore needs to be

timed and evaluated. We present such measurements and evaluations.

Chapter 2   Prior work

Many papers focus on helping the user use mobile devices in a more convenient way. The prior work has been separated into three different categories: Focus Plus Context, Mobile Devices, and Presentation Trees. Each of these has contributed to the creation of a new way to compute in the spillable environment.

2.1   Focus Plus Context

Focus Plus Context Screens [BAU01] is one of the main ideas that have influenced our novel environment. Focus Plus Context Screens combine a high-resolution display and a larger low-resolution display. Focus Plus Context Screens allow users to see the big picture while still seeing the detail in the high-resolution part of the screen (Figure 10). In our environment, the handheld device acts as the focus, with the DiamondTouch table as the context. The high-resolution part of the Focus Plus Context screen was an LCD monitor in the work by Baudisch. The focus screen allows the user to see high detail that is necessary for some types of actions. With a large screen, the size allows users to be able to see more using their peripheral vision. This gives the user a lower resolution context of the environment of the focus screen, thereby enhancing the experience of the user. In the spillable environment, the context screen uses a projector to display onto the DiamondTouch table. Baudisch's use of a projector as the larger screen allowed a relatively cheap solution (Figure 11).

Figure 10 Focus Plus Context Screen

Focus Plus Context Screens solve the problem of creating a large screen for users to see the detail and the context of the detailed portion while keeping the screen cost-effective. Baudisch's solution applies to a desktop computer. The mobile user has a similar problem with the limitations of his/her handheld device. Our solution allows a mobile device to connect with another computer to expand the handheld device's small screen. Both the solutions require two screens and integrate them into one system.
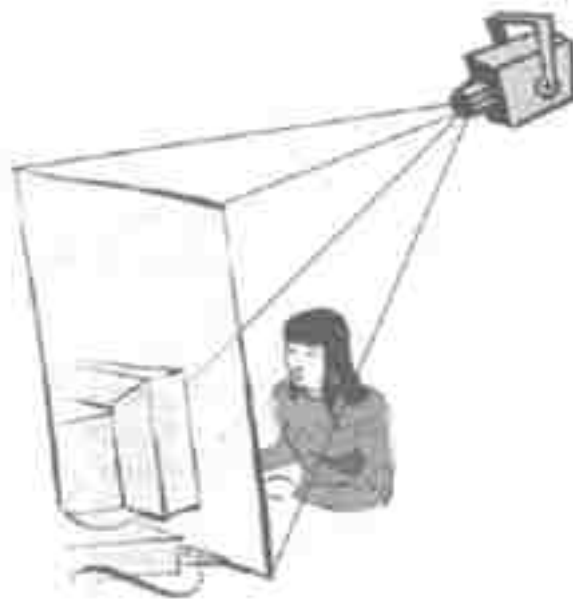
Figure 11 Focus Plus Context Setup

2.2   Mobile Devices

Research has been done to enhance handheld devices in a number of special cases. The Missing Link [MAC02] shows a PDA used to augment notebooks for laboratory biologists (Figure 12). The PDA is attached to a WACOM 4d mouse sensor, thus allowing the WACOM table behind the notebook to sense its location and orientation. In essence, the "large screen" is the laboratory notebook in the real world. Mackay augments paper; she wanted to keep the flexibility of paper while being able to enhance it.

Figure 12 The Missing Link

Although the Missing Link is interesting, it only solves the problem of enhancing a laboratory notebook, so that the biologist can take notes more effectively. The biologist is able to make annotations in the digital world. This environment is quite restrictive and narrow to a certain task. Users are not able to see the big picture of the annotations in their environment.

In Ubiquitous Graphics [SAN06], a handheld device and a wall-size display, are integrated to create a unique system (Figure 13). Sanneblad uses the Mimio technology [MIM07] to find the location of the mobile device with respect to the large screen. By doing so, he limits his environment in not being able to rotate the mobile device, although he can use multiple mobile devices. To find the orientation, multiple Mimio XI ultrasonic pens, an accelerometer, or some other device would be necessary. The mobile device has to be in the correct orientation while moving it around on the wall to display the image properly.

Figure 13 Ubiquitous Graphics

Ubiquitous Graphics "addresses the problem of interacting with very large computer graphics images." Sanneblad presents a custom solution to interacting with large screens. He creates a wireless network using the OpenTrek platform [SAN03]. Mobile devices send their display configuration (i.e., location) to the other devices to update their masking areas. Each device runs the whole application and updates itself depending on the location of all the devices. Ubiquitous Graphics uses a wireless ad hoc network that would not work for our problem, since the computation needs to be on the handheld for security reasons.

Also, it appears that the architecture is only for the map application, which limits its use. It would not be generally useful for mobile users, since they would need a variety of applications. Both devices (mobile device and computer running the large screen) are preloaded with the complete application and therefore probably need the same version of application software. The computer running the large screen may not be a computer that is accessible to or usable by the user. In our environment, we want the remote machine to be a display server rather than a machine that requires application-specific logic. The handheld device alone therefore must have the application loaded onto it. This also allows the mobile device to save battery on computation for displaying purposes and to be protected from viruses. The display server allows the mobile user to have flexibility

rather than creating a controlled environment. Stations with the spillable environment situated in a variety of locations create flexibility for mobile users.

Ubiquitous Graphics is able to integrate a mobile device into a large display environment but ignores many of the mobility issues. Although the user is able to use the mobile device freely, nothing is stated with regards to connecting or disconnecting to/from the large screen. Ubiquitous Graphics does not deal with any trust issues with the mobile device. Because of the problem Sanneblad is trying to solve, he ignores many issues that pertain to a mobile user. Ubiquitous Graphics simply creates a cool way to interact with a large screen.

Peephole Displays [YEE03] use a handheld device to provide a peephole to a larger virtual workspace, which is not completely visible to the user (Figure 14). The system uses a handheld device to see a portion of the virtual workspace but does not enable users to use their peripheral vision with a larger screen.
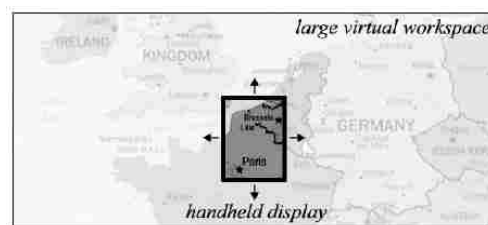
Figure 14 Peepholes

The goal of Peephole Displays was to keep the portability of the handheld device without using another screen. As shown in figure 15, only a portion of the whole application is shown on the mobile device at a time. By moving the handheld device around, a different portion of the application shows. In order to scroll around an

application, the user physically has to move the handheld device. This feature presents the problem of tracking the handheld device accurately.
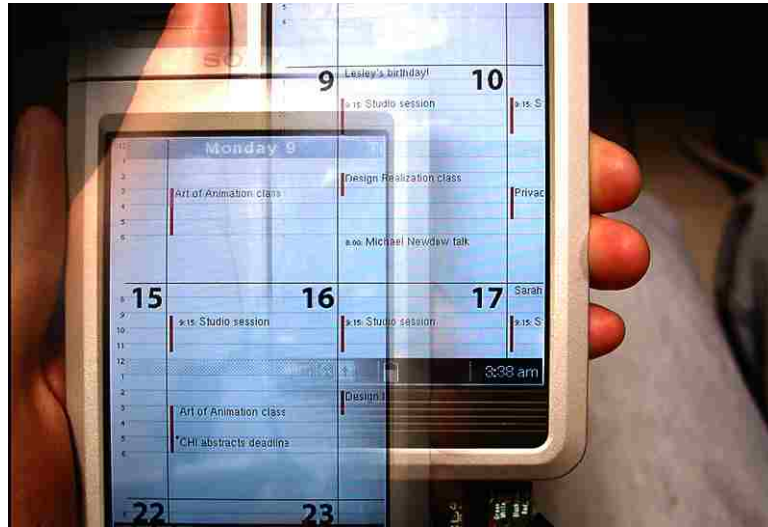
Figure 15 Calendar application in Peepholes

In order to track the mobile device, Peephole Displays used many different techniques, such as ultrasonic tracking with the Mimio [MIM07], to try to accomplish their goal, but their two-tethered tracking method was the best (Figure 16). This method used two fishing lines that ran from the handheld device to reference points and through a mechanical mouse. Weights were used to maintain tension on the line. Although this method was accurate, it does not solve the mobility issue. Yee found through his research that the tracking hardware still has a long way to go to be truly portable.

The peepholes environment creates a unique way to use a handheld device to see a larger picture without seeing everything at once. Although an attractive idea, it does not allow users to take advantage of their peripheral vision. This would be helpful to mobile users if they are not able to find a larger display screen.

17
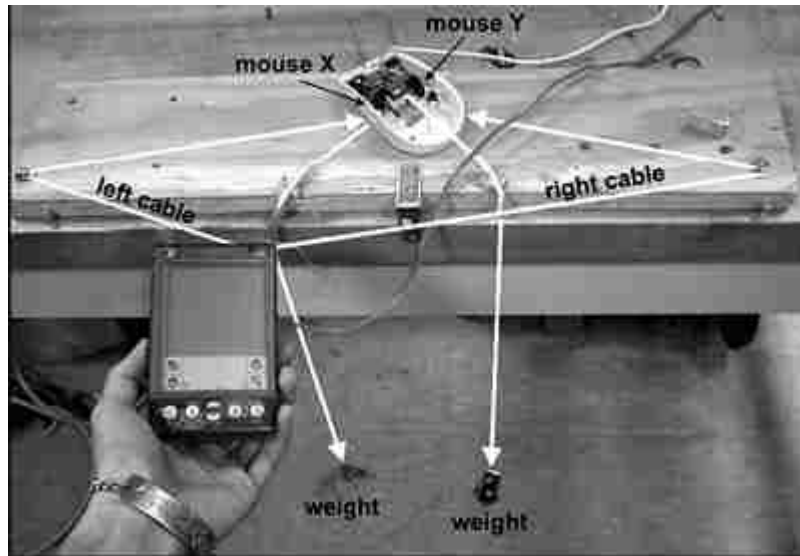
Figure 16 Two-tethered tracking method

Although much of the previous work features handheld devices, none of them enlarge the screen real estate of the PDA for the purpose of the normal mobile user while keeping the computation power of the application on the PDA.

2.3 Toolkit Designs

At the University of Maryland [BED04], Bederson analyzed toolkit designs (i.e., .NET Windows Forms, Java Swing) to support custom components like zoomable user interfaces. He used a structure called *scene graphs* which hold nodes like rectangles and custom components. This structure allows the developer to manage the placement of objects easily and make changes quickly while allowing the graphics system takes care of displaying. The toolkits Bederson built update the portion of the screen that has changed automatically (called region management) so no repaint calls are needed by the application programmer. The developer only has to maintain the tree of nodes.

Bederson looked at the design of toolkits and showed an example of a Zoomable User Interface called PhotoMesa. Although the XICE architecture does not specifically support a zooming architecture, it has the ability to do affine transforms (i.e., translate, rotate, and scale). In the XICE Architecture, presentation trees are used instead of scene graphs. They encompass the same idea but, we call them different names. Presentation trees prove to be useful in distributing the display information to remote computers. This allows presentation trees to be useful for mobile devices. Bederson built his toolkits to compare them in terms of performance, memory requirements, and programmability.

2.4   Conclusion

We found in our research that many of the concepts and principles from these research papers were integrated into the spillable architecture. Many of these other projects solved related problems, but they have not been used together to attack the problem of the limitations of handheld devices.

Chapter 3　　Overall Approach
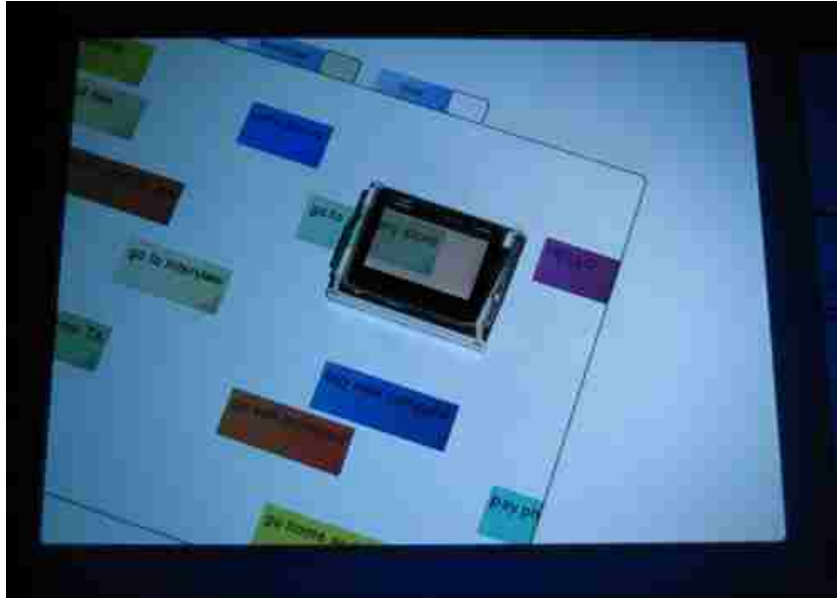
Figure 17 Sticky Notes Application in Spillable Environment

Because of the limitation of screen real estate on handheld devices, we wanted to create a system that would expand the screen of the handheld device. By using Mitsubishi Electric Research Laboratories' (MERL) DiamondTouch Table, we want to be able to expand the size of the screen to the size of the DiamondTouch and be able to interact with the application in a different way (Figure 17).

MERL created the DiamondTouch table [DIE01], a touch-sensitive horizontal surface that senses projections using capacitance. In order to sense the user touching the DiamondTouch, antennas in the DiamondTouch are aligned in a 2-dimensional grid. These antennas send unique signals through the user's body and to the user's receiver, usually located on their chair (Figure 18) [MIT06]. This creates a loop as shown in

Figure 19. With this loop, the DiamondTouch is able to sense where the user is touching the surface.

Figure 18 DiamondTouch receiver

Figure 19 Signal Loop

When the loop is connected, the antennas in the DiamondTouch produce a strength value of the circuit created depending on the area of the contact point. The antennas give two histograms of strength values: one in the x direction and one in the y direction. Therefore, the information that is received by the host computer from the

DiamondTouch table is a projection on each of the two perpendicular directions. Figure 20 is a screenshot from a diagnostic tool that shows the strength of the signals at discrete points in the x and y direction. By recording the histograms of the strengths of the signals, an algorithm is able to figure out from the x and y projections what the user is actually doing with the DiamondTouch table. If the histograms show a single, narrow, spiked peak in each direction, the user has touched the DiamondTouch with one finger. When the user touches it with two fingers, more peaks appear in the projections and therefore complicate the issue. Figuring out where the user is touching the table will be explained in Chapter 5.
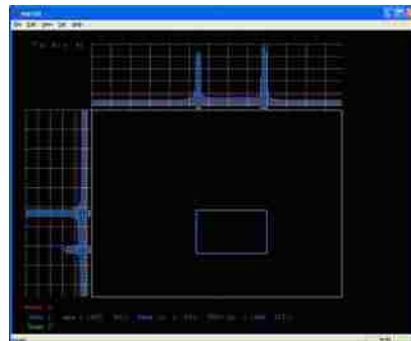
Figure 20 Diagnostic Tool [MIT03]

3.1 Environment

The spillable environment combines the handheld device with a normal desktop computer connected to a projector (with VGA) and a DiamondTouch table (with USB). The system uses a projector that projects down onto the DiamondTouch table, creating a large display. The handheld device wirelessly connects to the desktop computer through the local network. The handheld device is placed on the DiamondTouch table. Using the

points of contact with the DiamondTouch table, the desktop computer identifies the position and orientation of the handheld device.

3.1.1   User Interface

Mobile users always on the run like the ability to interact with their mobile device by itself, or with another computer (Figure 21). When they are in situations where they need to be mobile, they can use applications solely on the handheld device.   All interaction is done on the handheld. When users need to connect to a remote machine using the spillable environment for a business meeting, they have limited their mobility in a way, but are able to interact with a much larger area.   The handheld device isn't attached physically, but users will want to be near the DiamondTouch in order to interact with it.
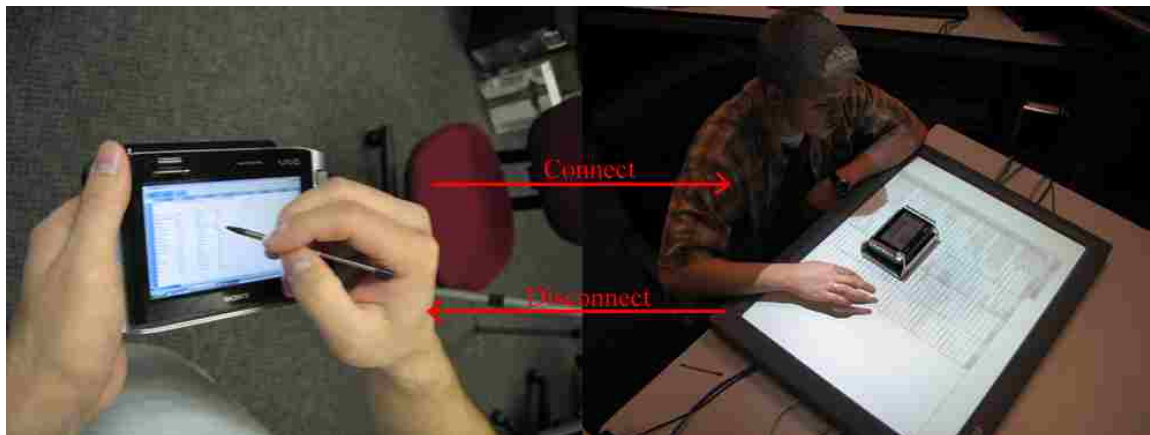
Figure 21 Changing Environment

In order for users to connect to the remote machine, they launch an application (like the spreadsheet application) and click on a button in the application to bring up a connection dialog. This dialog was integrated into the applications as a new feature.

24

Although each of the applications had to be modified slightly, it was not hard to add due to the presentation tree architecture in XICE.

After the user connects to the remote machine using the connection dialog, the displays of the handheld device and the projector from the DiamondTouch will be synchronized. Screen synchronization of the spillable environment is handled through the event system in XICE. The size of each of the widgets has increased, since the application has adapted itself to the size of the DiamondTouch table.

In this connected environment, users can interact in three different ways. They can directly interact with the handheld device as if it was not connected. Second, when connected, the user can touch the aluminum frame and move it around on the DiamondTouch table to translate and rotate the application window. The projected application moves while the handheld screen remains fixed. Third, the user can interact with the DiamondTouch by directly touching it using a single finger to scroll the application around. With these controls, the user is able to maneuver in the spillable environment conveniently.

When placing the handheld device onto the DiamondTouch table, the projector should not show information onto the handheld device. If it did, it would create a double image and confuse the user. Therefore, a black box is shown on the DiamondTouch table in the location of the handheld device. If the black box is not shown directly on top of the handheld device, it becomes obvious that there is a problem.

3.1.2   DiamondTouch Table

Figure 22 Aluminum Frame – for holding Sony VAIO Micro PC

Figure 23 Bottom of the aluminum frame

In order for the handheld device to specify its location on the DiamondTouch table, an aluminum frame (Figure 22) was specifically built to hold the Sony VAIO Micro PC. On the bottom, it has three conductive feet (Figure 23) that are connected to the aluminum frame with metallic tape. This allows a circuit from a user's hand to the DiamondTouch table, creating three special points. Since the user touches the aluminum frame when moving it around the table, the signal from the antenna goes through the metal contacts on the aluminum frame and through the user's body to the DiamondTouch's receiver on the user's chair (Figure 24). These three special points are used to find the position and orientation of the handheld device. The remaining foot is insulated and does not have tape connected to the aluminum frame.

26

Figure 24 DiamondTouch Loop Signal with Aluminum Frame

Rubber bands are used to keep the Sony VAIO Micro PC connected to the aluminum frame (Figure 25). Obviously, this is not ideal for a real user. The conductive feet would be manufactured onto the device allowing it to have the same functionality as this aluminum frame.

Figure 25 Aluminum Frame with Sony VAIO in it

By placing the handheld device on the DiamondTouch table, we create a way for mobile users to take advantage of the portability of their small device, but allow them to use a larger screen for viewing. This is done by using the position and orientation of the

handheld device to position the application correctly. The application and the handheld device are both in the same orientation, as can be seen in Figure 17.

Due to the way that the DiamondTouch table senses, it was necessary to create an algorithm that distinguishes the position and orientation using the three conductive feet on the aluminum frame. The DiamondTouch has a coordinate space that is dependent upon the number of antennas in each x and y direction. These coordinates need to be transformed into the computer's pixel space which the projector is using to project down onto the DiamondTouch table.

3.1.3   XICE Architecture

The XICE Architecture was designed and implemented to display presentation trees. Also, handheld devices are able to connect and display a presentation tree onto another computer. This architecture was designed with handheld devices in mind. It also was designed to keep the computation on the handheld, to keep the information on the device secure, and to keep the communications protocol's bandwidth to a reasonable amount.

3.1.3.1   Presentation Tree

Presentation trees are the model used to display a tree of widgets. Figure 26 shows a simplistic view of an application's presentation tree. Presentation trees are similar to scene graphs [BED04]. Both use a tree-like structure to display themselves and automatically repaint the parts that have changed. All of the objects or nodes in a presentation tree inherit from a main class, PresentObj. Objects in presentation trees have properties. For example, a line object has two properties: a start point and an end

28

point. When a property is changed, a change flag is set to true, and then the object notifies its parent. The change moves its way up the tree and modifies each node's flag up to the root node. By doing this, it is known whether the tree has changed by checking the root node, and the paths to the changed nodes are also known so that extra time is not wasted with the nodes that have not changed. This also helps serialize the changes to a remote machine.

Figure 26 Picture of presentation tree at runtime

This architecture allows user interfaces to be displayed quickly and to be more intuitive to create. A node's children are painted to the screen in order (or from left to right). For example, a simple button, as in figure 26, is an object that might consist of a rounded rectangle and a text object. The rounded rectangle would be added first and then the text, since the text should appear on top of the rounded rectangle. Creating this structure in presentation trees allows an easy way for the developer to add objects from closest to the background to the foreground in much the same way that a painter paints a picture. Another node shown in the figure is a container. This container node allows the ability to contain multiple objects. Since the button has children, it is also a container

node, along with many other nodes in the tree. The placement of nodes in the tree is critical (in most cases), and it indicates how they will show up. That is why the dialog nodes are shown last so that they are in front of everything.

The XICE architecture monitors events so that the display is repainted without explicitly calling repaint as normal toolkits do. Presentation trees can be easily modified. They allow us to transmit changes to the remote machine easily using less bandwidth. By using presentation trees, pixel rendering is shifted to a remote machine, thereby reducing the battery limitations of the mobile device. These features allow mobile devices to work effectively in this nomadic environment.

3.1.3.2 Spillable Environment

The spillable environment features some unique aspects. The handheld device and remote machine show different portions of the whole world. Therefore, the two devices have to transform (i.e., rotate, scale, and translate) their view of the world to correctly show the environment in a synchronous manner. The two devices need to be able to access the same model or presentation tree. Information from the DiamondTouch table has to make its way to the handheld device's presentation tree.

When a handheld device connects to the remote machine, the remote machine will show only the application in the portion where the DiamondTouch is located. The remote machine needs to know where it can show applications in advance in order to set this environment up properly. Therefore, if a user wants to connect to a station, he/she does not need to know where the application should show up on the remote machine. This is for the ease and convenience of the user. Therefore, a number of modifications

30

and additions had to be made in order for this environment to work within the base part of the XICE architecture.

In order to create the spillable environment, many challenges had to be overcome. With regards to the DiamondTouch table, the aluminum frame needed to be tracked appropriately and interactively. The handheld device needed to connect easily with the remote machine and handle all of the user interface problems when connected. Modifications to the XICE software architecture were made to allow an easy way for handheld devices to interact with computers connected to large screens.

Chapter 4   User Interface

Users are able to use their handheld device in two different ways: by itself and also by using the spillable environment. It will be shown how easy the process is to switch between these different ways of interacting. When the handheld device is in the spillable environment, the user needs a way to comfortably maneuver around the application. How a user is able to do that will be explained. The two different devices need to be synchronized and have a minimal amount of lag.

4.1   Handheld Input Interaction

Due to the mobility of handheld devices, applications need to be able to run solely on the handheld device, to allow normal mobile usage. In this mode, all of the interactions are done directly through the handheld device (Figure 27). Interacting with the handheld device by itself can strain the eyes due to the smallness of the screen size. Also, a user's peripheral vision is not used. The screen would be even smaller on a handheld device that fits into a pocket. This also makes it hard for two people to see the same tiny screen at the same time.

Figure 27 Handheld Input Interaction

4.2 Connection process

When users want to expand their screen real estate, they connect to a remote computer using the spillable environment. This allows them to see their applications on the DiamondTouch table and to show others information on the handheld device easily, without putting it in harm's way.
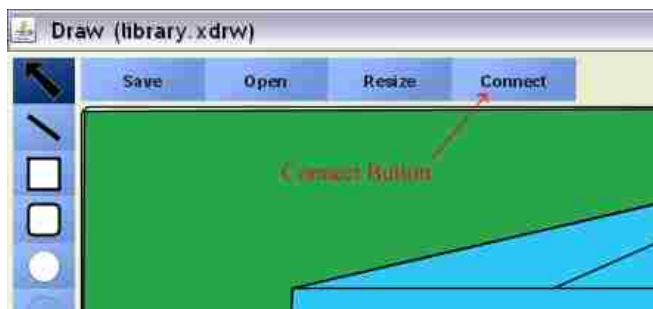
Figure 28 Connect button on application

In order to expand an application on to the DiamondTouch table, a button was added to a XICE application to facilitate the connection process (Figure 28). This connection button pops open a dialog for the connection process (Figure 29). The user is

34

able to pick the appropriate machine to connect to and press the connect button. The

connection process happens, and the environment is created.

Figure 29 Picture of connection dialog

All of this assumes that the user has already set the details for connecting to that

machine. An add/edit screen (Figure 30) gives Joe the ability to add or edit a computer to

which to connect. Each computer includes details that need to be filled out. If users

wanted to add a computer to their list, they would have to choose a name, the computer

that the user wants to connect to (or connection string), and the port number. The "Trust"

and "Default Input" fields are for later research. Also, users would have to check the

spill checkbox if they wanted to connect to that computer using the spillable

environment. If they did not check it, it would display the current application on the

respective machine's display. This connection dialog has become a generic way of

connecting with another machine rather than specifically for the spillable interface. All

of this information is saved in a file on the handheld device, so the user does not have to

enter this information again.
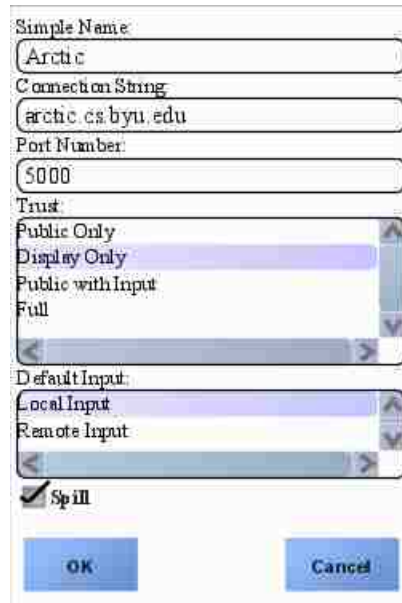
Figure 30 Picture of edit screen

Once the user has connected to the remote computer, there is much more room to display the application. It would be unwise to keep the widgets the same size, so the size of the widgets increases according to the configurations set on the remote machine. The size is determined by the viewing distance and pixels per inch of the display. By doing this, the sizes of all the widgets are in view independent coordinates and therefore are different sizes on different screens depending on the configuration. The resolution is quite better on the handheld device than on the DiamondTouch table. This creates an environment similar to Focus Plus Context [BAU01].

4.3   Interaction Methods in the Spillable Environment

In this unique environment, users can interact with the system in a number of different ways. They can make modifications to widgets that are shown on the handheld device (Figure 31). Users can interact with the DiamondTouch table by moving the

36

handheld around (Figure 32) or touching the DiamondTouch itself to scroll or move the application around (Figure 34).

4.3.1 Handheld Input When Connected

Handheld input is received like normal, whether connected or not (Figure 31). Input is received by the event source (which will be explained in Chapter 5) and passed to the presentation tree like normal events. When connected, the widgets are bigger allowing the user to see more detail. The inner workings are a little different since different classes are used but the same pattern is used. All this is transparent to the user.

Figure 31 Handheld Input Interaction when connected

4.3.2 Handheld Movement on DiamondTouch table

When the handheld device is connected to the remote machine, a new type of interaction can be used. Using the aluminum frame, we are able to create a way to recognize the position and orientation of the handheld device on the DiamondTouch table. The conductive feet on the bottom of the aluminum frame make it seem as if the user is touching the DiamondTouch in three different places. When the user moves the aluminum frame with the handheld device, the conductive feet move, and therefore the

remote machine is able to track the location on the DiamondTouch (Figure 32). When moving the handheld device on the DiamondTouch table, the application moves with the device. In other words, when the handheld device is rotated 30 degrees clockwise, the orientation of the screen of the DiamondTouch table will also rotate 30 degrees clockwise. Therefore, the display on the handheld device does not change, since it is being physically moved and therefore does not need to be updated, as opposed to the display on the DiamondTouch table, which needs to change.

Figure 32 Handheld Movement on DiamondTouch table

When the user places the handheld device on the DiamondTouch, the projector will not display the portion of the application where the handheld device is located. Therefore, a black box will be drawn over the application (Figure 33). In this way there is no confusion about the location of the handheld device on the DiamondTouch table. Also, the black box avoids a double image and the reflections on the handheld's screen.
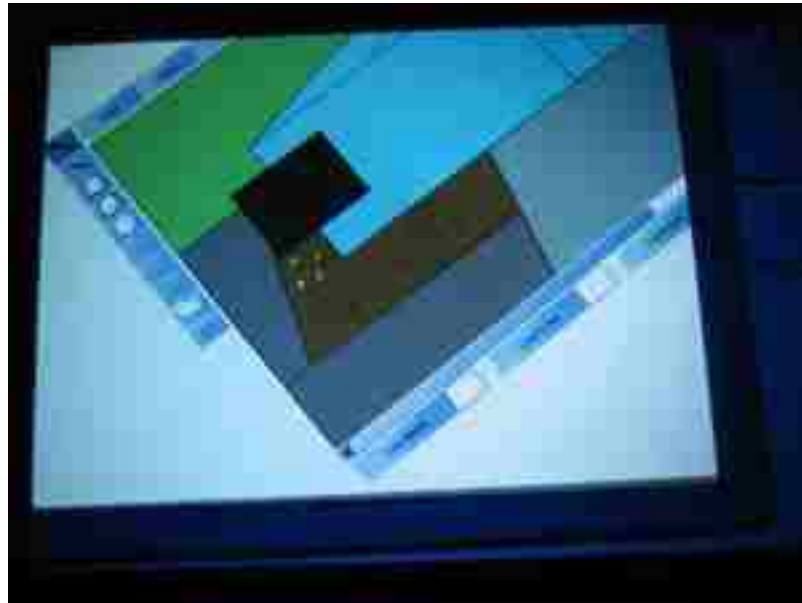
Figure 33 Black Box on DiamondTouch table

Thus, this black box will be shown only on the remote machine with the DiamondTouch. When the handheld is moved, the black box will be updated appropriately. The details about how this happens will be explained later. If the user is not touching the handheld device, the system just remembers the last position and orientation of the handheld device and maintains the current state. Therefore, if the user removes the aluminum frame from the DiamondTouch, the black box will become even more evident.

There is only one model, although it is propagated over to the remote machine through serialization. In order for the handheld and the display on the DiamondTouch table to show up correctly, each of them has to be transformed in a different way. How this is done will be explained in detail in Chapter 6.

4.3.3 Scrolling on DiamondTouch table

The other way to interact with the DiamondTouch table is to touch it with one finger (Figure 34). This allows the user to scroll the application around the large environment by moving a finger around the surface of the DiamondTouch table. It is important to use only one finger; otherwise it will appear as multiple points, which will make it think the aluminum frame is touching the DiamondTouch. This will cause the black box to show up in that location if one accidentally does that. One can easily reset the aluminum frame and begin interacting with the environment again.

Figure 34 Scrolling on DiamondTouch table

4.4 Disconnection process

In order to disconnect, the user presses the special button to bring up the dialog. The user is able to disconnect by pressing the disconnect button. During the disconnection process, the application is moved back to the normal space on the handheld device and is displayed with the same size window when the user connected.

4.5   Synchronization
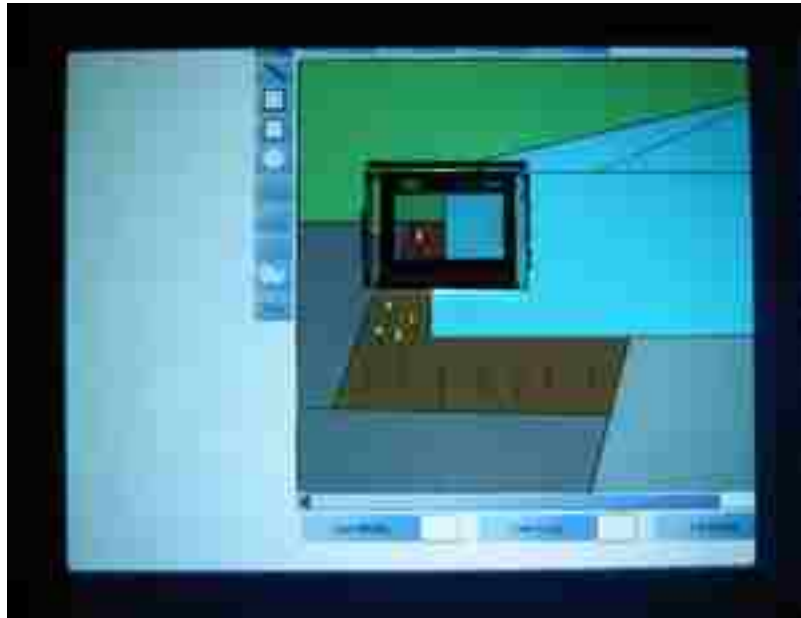
Figure 35 Draw Application on handheld device and DiamondTouch lined up

When the handheld is connected to another device, the displays need to be in synchronization. All of the synchronization features have been built into the XICE Architecture. Due to the architecture of the spillable environment, no modifications had to be made. Optimizations had to be made to make the whole XICE Architecture work more quickly in general. The displays on the handheld device and the DiamondTouch line up (Figure 35). It may not appear in the picture that they are lined up exactly. As seen in figure 36, the handheld device has been lifted and moved slightly to show that the handheld device and display on the DiamondTouch are synchronized. The top left corner of the black box matches with the top left corner of the screen of the handheld device. One thing to notice is that the handheld screen is smaller than the black box. Thus, everything missing from the black box is not shown to the user, but this is not very noticeable nor does this inhibit the user's ability to use this environment. The size of the

widgets in the application is the same on the handheld device as on the DiamondTouch table display.

Figure 36 Synchronization between handheld and DiamondTouch table

Chapter 5   DiamondTouch Table
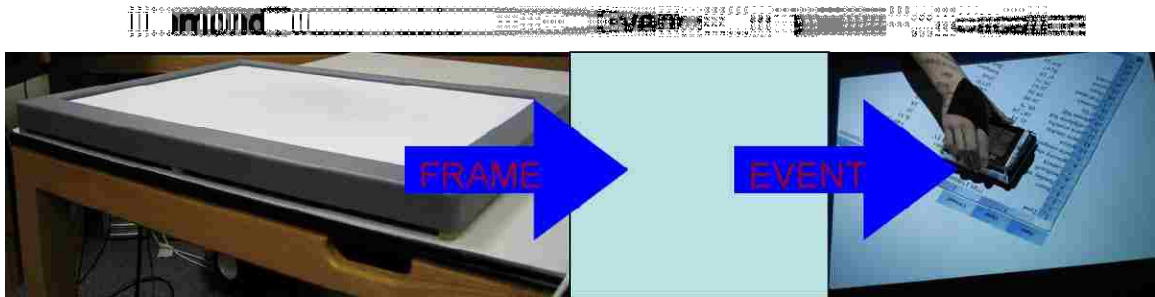
Figure 37 Overview

This chapter explains the conversion of frames from the DiamondTouch into events dispatched to the application (Figure 37). A DiamondTouch frame consists of two histograms of signal strength projections. A visual representation of these histograms in a frame is shown in the diagnostic tool (Figure 38). The DiamondTouch sends these frames to the event source many times per second. In the XICE architecture, input is received through an event source. A special event source class was made in order to accommodate the DiamondTouch table. Frames from the DiamondTouch need to be interpreted correctly so that events can be generated. In order for this to happen, a number of steps must happen:

- Do the calibration step of the DiamondTouch

  o transform from DiamondTouch's coordinate space to pixels

- Find projections from DiamondTouch frame

- Distinguish between touching with finger or touching the aluminum frame

  o For a finger touch, the change in position is put into the event

  o For the aluminum frame, the positions of the three conductive feet are inserted into the event
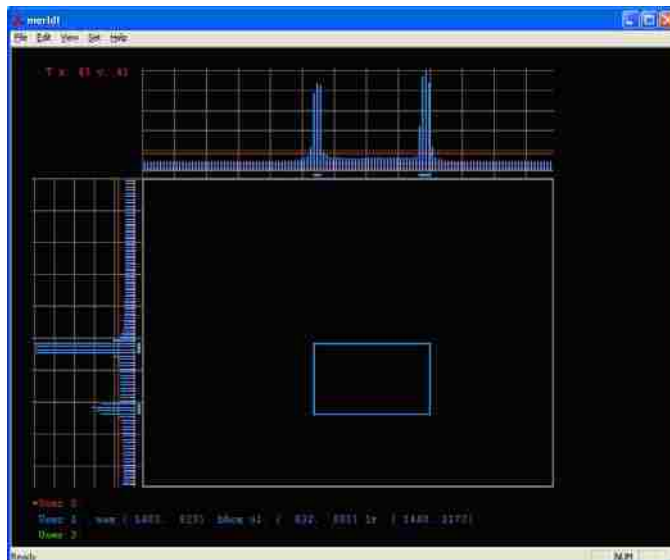
Figure 38 Diagnostic Tool

5.1 Calibration Step

A calibration step was used to synchronize the input from the DiamondTouch table with the pixel coordinate space of the projector's display. Without this calibration step, interaction with the DiamondTouch would not match with the application. The owner of the computer station has to calibrate this transformation from the DiamondTouch's space to the pixel space. The DiamondTouch table uses a coordinate

system that correlates with the antennas in the DiamondTouch. The antennas are aligned in rows and columns, as shown in the diagnostic tool (Figure 38). The DiamondTouch used here has 128 columns and 96 rows.

Figure 39 Calibration Step

A short program was created to transform the coordinates from the DiamondTouch to the pixel coordinates that the projector is using. The owner of the computer station is prompted visually to touch a red point on the DiamondTouch table (Figure 39). The red points show up on the DiamondTouch in random locations. The owner will have to touch four of these red points in order to calibrate.

The program then calculates the transformation using a linear least squares regression method. Only the DiamondTouch points were used as linear coefficients. No non-linear terms were necessary to perform this transformation. In Equations 1 and 2, $\bar{x}$ and $\bar{y}$ represent the four DiamondTouch points. Matrix z is the matrix of values of unknowns that perform the transformation. In Equation 1, the i variables are the x coordinates in pixel space. In Equation 2, the j variables are the y coordinates in pixel

space. Therefore, two equations, one for each direction in pixel space, are made to perform a transformation of one point. Matrix e is the error matrix.

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix}$$

Equation 1 Linear Least squares Equation for x pixel coordinate

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix} \begin{bmatrix} z_4 \\ z_5 \\ z_6 \end{bmatrix} = \begin{bmatrix} j_1 \\ j_2 \\ j_3 \\ j_4 \end{bmatrix} + \begin{bmatrix} e_5 \\ e_6 \\ e_7 \\ e_8 \end{bmatrix}$$

Equation 2 Linear Least squares Equation for y pixel coordinate

Equations 1 and 2 are in the form of $A \times z = b + e$. Using Equation 3, the unknowns are calculated, and now data can be transformed correctly. With four touches, the user is able to easily calibrate the DiamondTouch in a minimal amount of time.

$$z^* = (A^T \times A)^{-1} \times A^T \times b$$

Equation 3 Formula for L^2-Regression [VAN01]

5.2 DiamondTouch Frame to Event
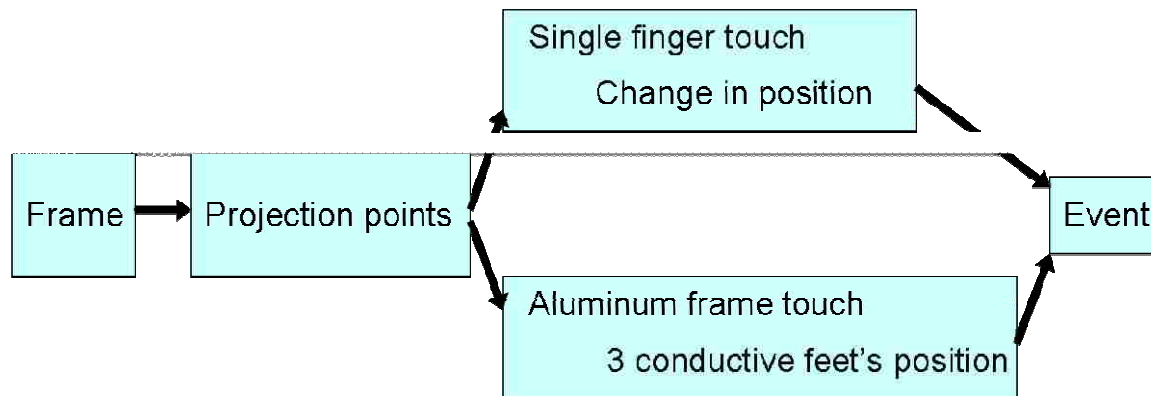
Figure 40 Process from DiamondTouch Frame to Event

Frames from the DiamondTouch table need to be converted into events to be sent to the application. The application then can be updated and displayed properly. Figure 40 shows the basic process from a frame to an event. The frame is used to create the touch points, or projection points, which are stored in two arrays (one for each axis). Using the projection points, it can be distinguished whether the user was touching the DiamondTouch with a single finger or with the aluminum frame. If the user touched the DiamondTouch with a single finger, the change in position from the last touch is calculated and put into an event. If the user touched the aluminum frame, the position of the three conductive feet are determined and put into an event. Algorithm 1 explains in pseudo code how this is implemented.

```
int[] xVals = createHorizontalProfile(frame);
int[] yVals = createVerticalProfile(frame);
if ( xVals.length == 1 && yVals.length == 1)
// single finger touch
     delta = findChangeInPosition(xVals, yVals);
else
// aluminum frame touch
     conductivePoints = findPositionOfAluminumFrame(xVals,
          yVals);
create event and notify application
```
Algorithm 1 Basic Overview

5.2.1   Find Projection Points
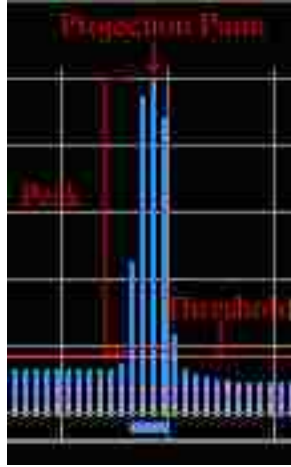
Figure 41 Peak and projection point from Diagnostic Tool

The first step in figure 40 is to find the projection points using the frame. A frame contains two histograms of the strength of the signals from each antenna. Figure 41 shows a close-up of one of the histograms from the diagnostic tool (Figure 38). Notice the horizontal red line in the figure, which represents a threshold value. Antenna signal values that are greater than this threshold value are considered part of a peak. Therefore, the peaks on the histograms are more than one column, and the problem is determining which antenna should be the projection point. This can be solved by finding the mean of the values in a peak.

In order to find the projection points, we iterate through the values in the histogram (Algorithm 2). By looping through the histogram, the first value in a peak and first value after a peak can be determined. After finding those values, the average or middle value of a peak can be calculated, called the *projection point*. The projection point is added to a list of all the projection points in a histogram. Notice in figure 41 that the middle of the five values in the peak is the chosen projection point. By following

48

Algorithm 2, the projection points are found in both the x and y directions. This method

was found to be effective, and the use of multiple threshold detection algorithms was not

necessary.

```
private int[] createHorizontalProfile(DtlibInputTframe frame)
{
        ArrayList<Integer> vals = new ArrayList<Integer>();
        double begin = -1;
        for(int i = 0; i < frame.getColLimit(); i++)
        {
                if(frame.getColSignal(i) > ythresh)
                {
                        if(begin < 0) // first value in peak
                                begin = i;
                }
                else if(begin >= 0) // first value after peak
                {
                        int cx = (int)((begin + i)/2); // average
                        vals.add(cx); // add projection point to list
                        begin = -1;
                }
        }
        int[] val = new int[vals.size()];
        for (int ind = 0; ind < vals.size(); ind++)
                val[ind] = vals.get(ind);
        return val;
}
```
Algorithm 2 createHorizontalProfile() Function

5.2.2   Single Finger Touch

If there was only one projection point in each direction, then it was determined

that the user was scrolling. The findChangeInPosition() function that was called in

algorithm 1 is shown in algorithm 3. The current point is transformed into pixel space, as

explained in section 5.1. Then the difference between the current point and the *previous*

scrolling point is found. The *previous scrolling point* is a global variable and can be

accessed from any function. It is one of three global variables used. The other two

global variables are *previous point* and *previous orientation* (these will be explained

later). If the *previous scrolling point* was not set yet, then there would be no change in position. Then the *previous scrolling point* is set to the current point and is saved to be used on the following frame. The delta is then put into the event. When the user lifts his/her finger from the DiamondTouch table, the *previous scrolling point* is set to null.

```
private findChangeInPosition(int[] xVals, int[] yVals)
{
    Point currentPoint = converter.transformPoint(new
        Point(xVals[0], yVals[0]))
    delta = findDifference(currentPoint,
        previousScrollingPoint)
    previousScrollingPoint = currentPoint
}
```

Algorithm 3 findChangeInPosition() Function

5.2.3   Aluminum Frame Touch

When a user touches the aluminum frame on the DiamondTouch table, three points need to be extracted from the projection points. In order to detect the location of the three conductive feet, three techniques were used: 1) a known starting position, 2) a rigid configuration of the three conductive feet, and 3) continuity of motion. The three points extracted from the profiles need to be the three points from the aluminum frame (Figure 42) starting with the top left conductive foot going clockwise (labeled foot 1, 2, and 3). Since the size of the aluminum frame is not known, the user must place the aluminum frame in a certain position when starting. Because of the rigidity of the aluminum frame, finding those three points becomes easier.

50
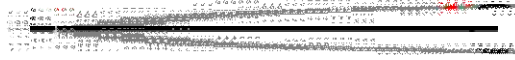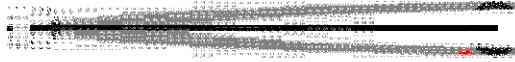
Figure 42 Layout of Conductive Feet on aluminum frame

After each time that the threePointDetection() function (Algorithm 4) figures out where the three points are located, it will record foot 1 as the *previous point* and save the *previous orientation* of the three points. By knowing this previous information, the threePointDetection() function can more accurately figure out where the three points are from the aluminum frame. The previous information allows the function to check the continuity between the current location and its previous location to make sure that it is within reason.

```
private Point[] findPositionOfAluminumFrame(int[] xVals, int[]
    yVals)
{
    Point[] points = threePointDetection(xVals, yVals);
    previousPoint = points[0]; //point of foot 1
    previousOrientation = getOrientation(points);
    return points;
}
```
Algorithm 4 findPositionOfAluminumFrame() Function

5.2.3.1 Initialization

Since the remote machine has no idea of the size of the aluminum frame, the user must place the aluminum frame down, as in figure 43, in the initialization position. The horizontal and vertical lines represent the projection points, which are labeled with numbers for their index in the projection point arrays. This initialization position has two

51

projection points in each direction. In the design of the initialization part of the algorithm, two ideas were in mind. First, it needed to be easy to reset. Second, it should not reset when the user was not trying to.
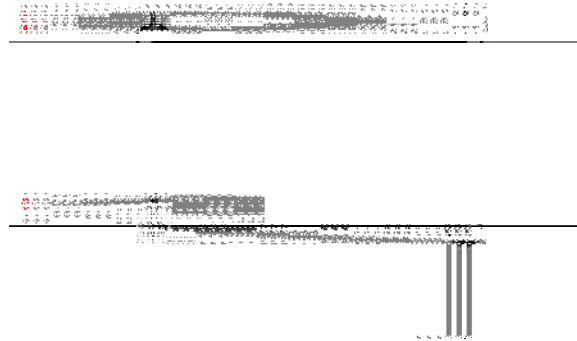
Figure 43 Initialization Position

If the user has just connected to the remote machine, the *previous point* would initially be set to null. Therefore, the size (i.e., width and height) of the aluminum frame could be found from the projection points and saved for later (Algorithm 5).

```
// Just starting out or resetting
if ( (prevPoint is null)  ||
      ( ( touchDown on DiamondTouch table ) &&
          (orientation has changed from previous) ) )
{
      width = xVals[1] - xVals[0];
      height = yVals[1] - yVals[0];
}
```
Algorithm 5 Initialization Code

If the user has not just connected to the remote machine, it is a little more complicated. There are some positions where the algorithm should not reset the width and height (Figure 44). MERL has provided a way to determine when the user begins to touch the DiamondTouch in much the same way as a mouse down event. Using touchdown event from the DiamondTouch is a good way to determine whether the user

wants to reset (Algorithm 5). One case where this does not work is if a user rotates the aluminum frame to the initialization position rotated exactly 180 degrees. This case might happen when a user is trying to show another person something so that that person can see it right side up. If the user releases the frame to explain something and then touches the frame again, the algorithm will think the user is resetting. This is not the intuitive action wanted. To eliminate this case, extra logic had to be added. If the *previous orientation* is the same as one of the positions in figure 43 or 44, the user was not intentionally trying to reset. Therefore, the size of the aluminum frame will be reset if there is a touchdown event and the orientation has changed from its *previous orientation* (Algorithm 5).
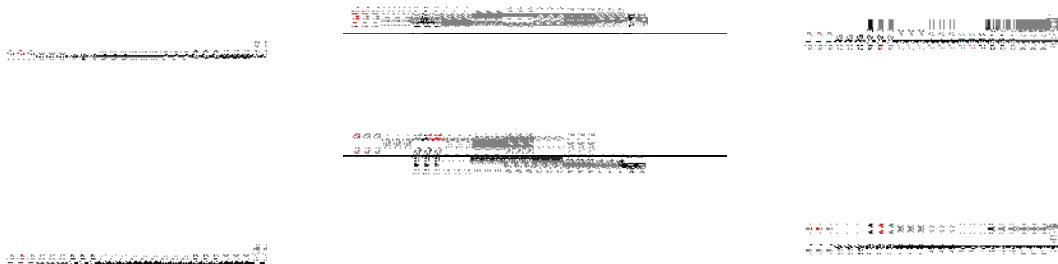
Figure 44 Incorrect positions of initialization positions

5.2.3.2   Finding the Three Conductive Feet of the Handheld Device

If the user is not trying to initialize, then the three-point detection algorithm will find the location of the three conductive points from the aluminum frame. Due to the many types of positions possible, the algorithm separates them into four situations, as shown in Algorithm 6. The four situations correspond with the following positions shown in figures 45-48.

Figure 45 Situation 1 Position

Figure 46 Siutation 2 Position

Figure 48 Siutation 4 Position

Figure 47 Siutation 3 Position

The difference between these positions is that they each have a different number of projection points in the x and y directions. By using that, the algorithm is able to find more quickly the correct location of the three conductive points.

54

```
private Point[] threePointDetection(int[] xVals, int[] yVals)
{
      if (xVals.length == 2 && yVals.length == 2)
            return situation1(xVals, yVals)
      else if 2 projection points in the x direction
            return situation2(xVals, yVals)
      else if 2 projection points in the y direction
            return situation3(xVals, yVals)
      else if 3 projection points in each direction
            return situation4(xVals, yVals)
}
```
Algorithm 6 threePointDetection() Function

5.2.3.3 Situation 1

The situation1() cases show two projection points in the x and y directions. All the cases for situation1() are shown in figure 49. These cases are case 1 and the three combinations that can be found by rotating in 90 degree increments. Notice the three conductive points for each case. These cases are the same as figure 43 and 44 from the initialization position section except that the user is not trying to initialize or reset the size of the aluminum frame. By using the initialization logic from algorithm 5, it is also known when the user is not resetting.
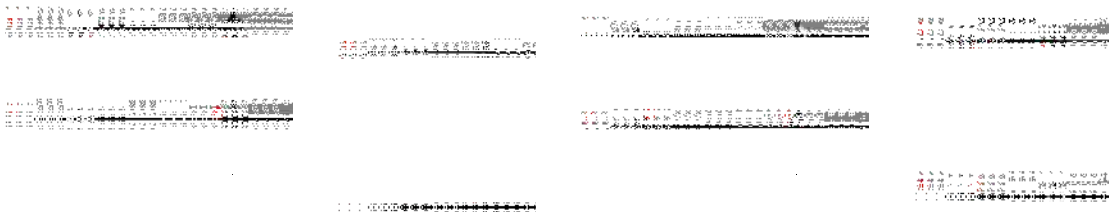
Figure 49 Cases for situation1()

In order to find the correct case that matches the projection points, the situation1() function loops through all of the cases, checking two criteria (Algorithm 7). First, it checks the distance between the foot 1 of each case and the *previous point*. The distance

will be minimized as long as the other criterion is met. The second criterion is that the

orientation of each case be within a certain threshold of the *previous orientation*

(Algorithm 8). This helps with the continuity of motion. The situation1() function then

returns the three conductive points of the best case.

```java
private Point[] situation1(int[] xValues, int[] yValues)
{
        int minIdx = 0;
        Point[] points = new Point[] {
                new Point(xValues[0],yValues[0]),
                new Point(xValues[1],yValues[0]),
                new Point(xValues[1],yValues[1]),
                new Point(xValues[0],yValues[1])
        };
        double minDist = Double.MAX_VALUE;
        for( int i=0; i < points.length; i++ ) //for each case
        {
                double dist = distance( previousPoint, points[i] );
                // distance between foot 1 and previous point
                if ( ( dist < minDist ) &&
                        (changeInOrientation(points, i)
                                < ORIENTATION_THRESHOLD) )
                {
                        minDist = dist;
                        bestCase = i;
                }
        }
        Point[] newPoints = new Point[3];
        for( int i=0; i < 3; i++ )
                newPoints[i] = points[(i+bestCase)%4];
        return newPoints;
        // Return 3 conductive points of the best case
}
```
Algorithm 7 situation1() Function

```
private double changeInOrientation(PointD[] pts, int i)
{
    Double orientation = getOrientation(pts, i);
    if (orientation == null)
        orientation = 0.0;
    if (prevOrientation == null)
        prevOrientation = 1.0;
    double diff1 = prevOrientation - orientation;
    double diff2 = orientation - prevOrientation;
    if (diff1 < 0)
        diff1 += 360;
    else if (diff2 < 0)
        diff2 += 360;
    return Math.min( diff1, diff2 );
}
```
Algorithm 8 changeInOrientation() Function

5.2.3.4   Situation 2

The situation2() cases have only two projection points in the x direction, while having three in the y direction. It is unique because two of its conductive feet are in the same x projection point (Figure 50). The first row of cases in figure 50 (Cases 1 and 2) is the same as figure 46 and figure 46 rotated 180 degrees. The second row in figure 50 (Cases 3-6) have feet 2 and 3 sharing the same projection point in the x direction. Because the DiamondTouch gives strength values in the histograms and the function that finds the projection points is not perfect, cases 3-6 present themselves periodically. These cases improve the detection of the three conductive points a little.
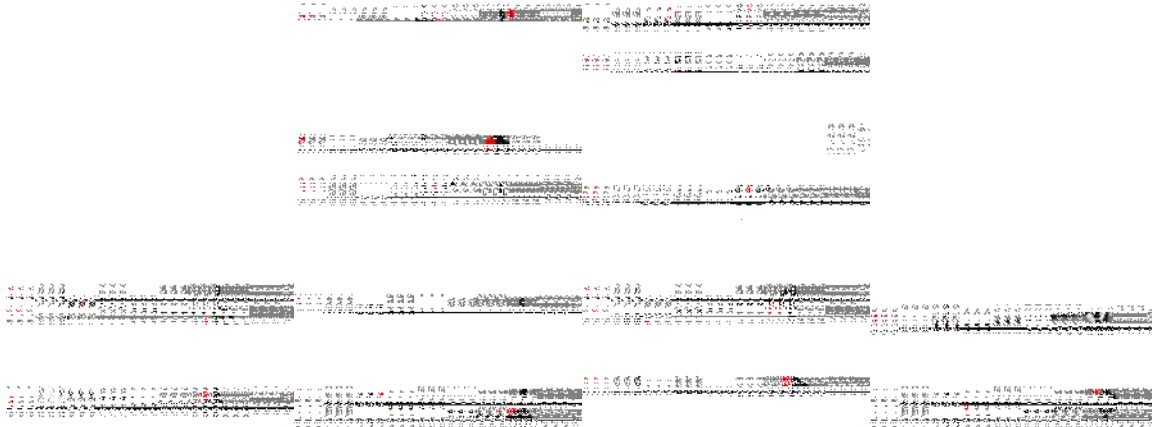
Figure 50 Cases for situation2()

The situation2() function shows the process to find the case that is associated with the given projection points (Algorithm 9). Each case has a different set of the projection points for their three conductive feet (Figure 50). This allows each distinctive case to be checked to determine if that case fits the saved width and height of the aluminum frame within a certain threshold (Algorithm 10). This is the way it is determined whether the projection points are at all similar with each case. The check() function proves to be useful, and therefore it will be used for the all of the cases in situation 2, 3, and 4. All the cases will follow the same pattern as the code for case 1 in algorithm 9. If it returns true from the check() function, it is added to a list of possible combinations.

```
private Point[] situation2(int[] xValues, int[] yValues)
{
      ArrayList<Point[]> possible = new ArrayList<Point[]>();
      Point p1, p2, p3;

      // Case 1
      p1 = new Point( xValues[0], yValues[0] );
      p2 = new Point( xValues[1], yValues[1] );
      p3 = new Point( xValues[0], yValues[2] );
      if( check( p1, p2, p3, width, height ) )
            possible.add( new Point[] { p1, p2, p3 } );

      Cases 2-6 should be checked in a similar fashion

      return findBestCase( possible );
}
```

Algorithm 9 situation2() Function

```
private boolean check( Point p1, Point p2, Point p3, float width,
float height )
{
      return Math.abs(distance(p1,p2)-width) < W_MARGIN
            && Math.abs(distance(p2,p3)-height) < H_MARGIN;
}
```

Algorithm 10 Check width and height Function

After checking cases 1-6, the best case fitting the projection points is found using

findBestCase() function in Algorithm 11. If there is only one item in the list of possible

cases, it is chosen. If there is more than one possibility in the list, the same two criteria as

situation1() allow the algorithm to find the correct case. The distance between the

previous point and the conductive foot 1 in each case is minimized as long as the change

in orientation is also within the threshold. This function (Algorithm 11) is useful in

situation3() and situation4().

```
private Point[] findBestCase( ArrayList possible )
{
      int minIdx = 0;

      if( possible.size() > 1 )
      {
            double minDist = Double.MAX_VALUE;
            for( int i=0; i < possible.size(); i++ )
            {
                  Point[] pts = (Point[])possible.get(i);
                  double dist = distance(pts[0], previous point);
                  if ( ( dist < minDist ) &&
                        (changeInOrientation(pts) <
                        ORIENTATION_THRESHOLD) )
                  {
                        minIdx = i;
                        minDist = dist;
                  }
            }
      }
      else if( possible.size() == 0 )
            return null;
      Point[] points = (Point[])possible.get(minIdx);
      return points;
}
```

Algorithm 11 findBestCase() Function

5.2.3.5   Situation 3

The cases for situation3() (Figure 51) are very similar to those of situation2().

The difference is that there are only two projection points in the y direction, while there

are three in the x direction. These cases are just rotated cases of situation2(). Therefore,

there are two conductive feet in the same y projection point rather than x. The process to

find the best case that fits the projection points is the same (Algorithm 12). The functions

check() (Algorithm 10) and findBestCase() (Algorithm 11) are used in following that

pattern.

60

Figure 51 Cases for situation3()

```
private Point[] situation3(int[] xValues, int[] yValues)
{
        ArrayList<Point[]> possible = new ArrayList<Point[]>();
        Point p1, p2, p3;

        // Case 1 handheld position 3
        p1 = new Point( xValues[0], yValues[1] );
        p2 = new Point( xValues[1], yValues[0] );
        p3 = new Point( xValues[2], yValues[1] );
        if( check( p1, p2, p3, width, height ) )
                possible.add( new Point[] { p1, p2, p3 } );

        Cases 2-6 should be checked in a similar fashion

        return findBestCase( possible );
}
```
Algorithm 12 situation3() Function

5.2.3.6 Situation 4

The cases for situation4() all have three projection points in the x and y direction (Figure 52). Again, the same pattern continues in order to find the best case that fits the projection points (Algorithm 12). In summary, a number of cases need to be checked to satisfy all the situations that might happen when using the aluminum frame.
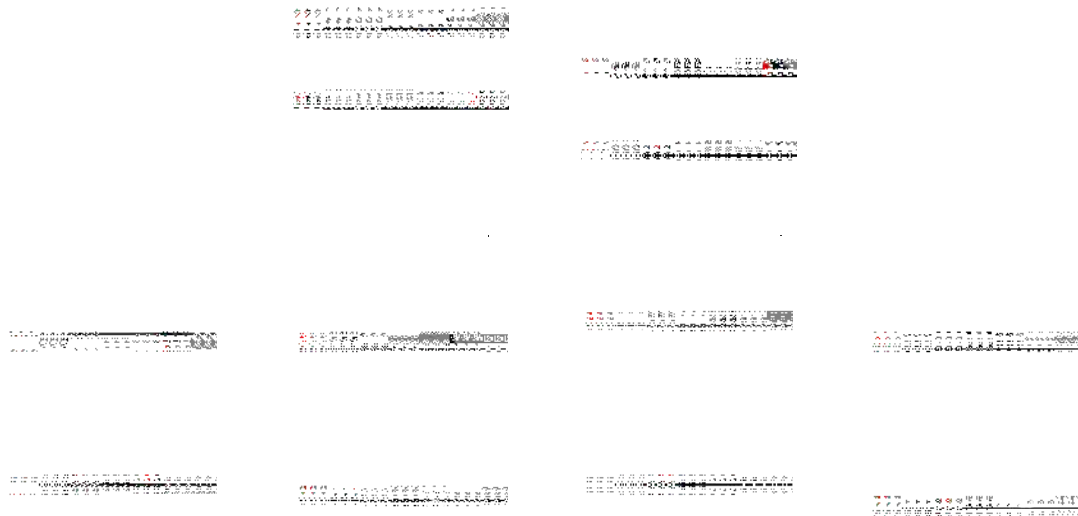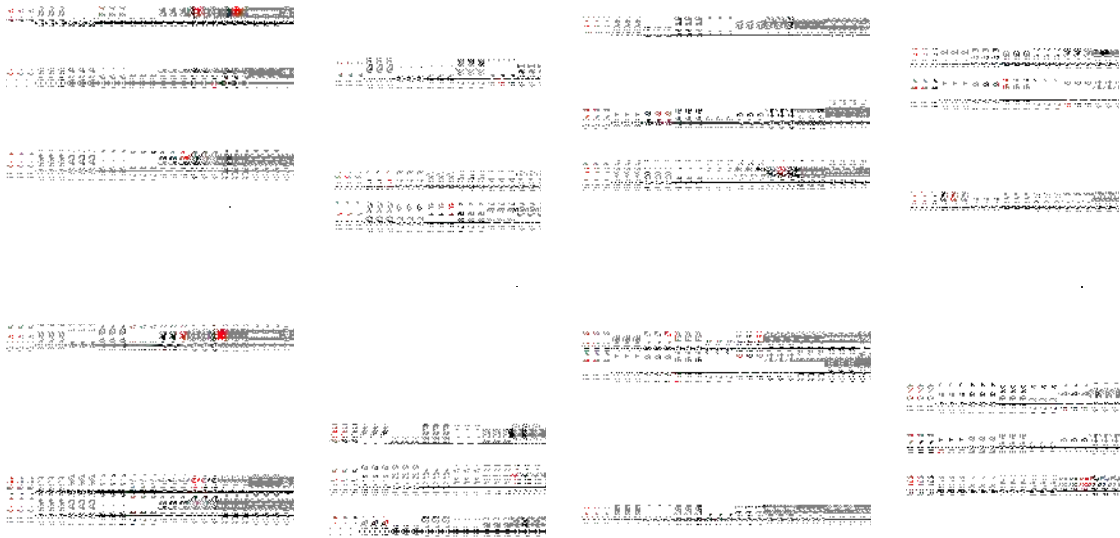
Figure 52 Cases for situation4()

```
private Point[] situation4(int[] xValues, int[] yValues)
{
    ArrayList<Point[]> possible = new ArrayList<Point[]>();
    Point p1, p2, p3;

    // Case 1 Handheld Position 4
    p1 = new Point( xValues[0], yValues[0] );
    p2 = new Point( xValues[2], yValues[1] );
    p3 = new Point( xValues[1], yValues[2] );
    if( check( p1, p2, p3, width, height ) )
        possible.add( new Point[] { p1, p2, p3 } );

    Cases 2-8 should be checked in a similar fashion

    return findBestCase( possible );
}
```
Algorithm 13 situation4() Function

5.2.3.7 Discussion

There are some minor problems with this algorithm. The DiamondTouch may not detect all three points due to user error or problems with conductivity, and thus the algorithm will not work properly. User error could be touching the DiamondTouch table along with the handheld device or not allowing the three points to have good contact with

62

the DiamondTouch. Most of the time when it does not work, the width and height were mistakenly reset. If the algorithm isn't working properly, one can always restart by picking up the handheld and placing it in the initialization position. Although one can always reset, the algorithm seems to be able to recover quite well even if it chooses incorrectly for a few frames. Also, the user can pick up the aluminum frame and place it in another position, and the algorithm will still sometimes get it correct. In general, this algorithm is fairly robust for the type of data it receives.

This algorithm can be implemented a number of different ways. Nevertheless, the implementation needs to enumerate all of the possible cases and then check their geometry. The check() function in my implementation could have also checked the geometry by making sure that the angles were approximately 90 degrees using dot products. The robustness of the algorithm might be better by using more than the previous state and using a larger history. The user can be unpredictable in his/her actions, and therefore it is difficult to know what the user is doing. These additions and more can be implemented to improve robustness, but for the purpose of our problem, the current implementation suffices.

5.2.4   Review Events

After the projection points have been processed, they are made into either a single finger touch event or an aluminum frame touch event. A single finger touch event contains a vector explaining the magnitude and direction of the movement. An aluminum frame touch event contains information on the position of the three conductive feet. With

this information, the location of foot 1 and the orientation can be calculated for the application to adjust appropriately.

Chapter 6   Architecture

The conversion of DiamondTouch frames into events has already been explained. In this chapter, the event updates the application so the user sees the environment properly. To accomplish this, details of the XICE software architecture will be explained. Two displays need to be able to access the same presentation tree, and therefore a multi-parent node was created to maintain the synchronization between the two displays. Each of the displays needs to show different parts of the presentation tree. Transformation nodes are created and updated to show the appropriate sections of the view. Events need to propagate to update the transformation nodes in order to show the displays correctly. Other objects, like the DiamondTouch event grabber node and the SpillSheet, are created to help with the events that update the proper nodes. In order to create this new environment, these problems had to be overcome.

6.1   XICE Architecture

In building the architecture, we looked at aspects of the mobile user's life and how he uses technology. The architecture needs to be able to run easily on a handheld device. Mobile users spend a great deal of their time away from their desk. Applications can not be overly computationally expensive. In order to display information on a remote machine, the communications protocol needs to be fairly light in traffic. Also, mobile users want to be able to show their information to others on larger screens.

The eXtended Interactive Computing Everywhere (XICE) Architecture addresses the issues of a mobile user. Since we want to show applications on many different types

of devices, we use our own coordinate system called *view independent coordinates* (VIC) to allow for variable viewing conditions. VICs are determined by the viewing distance and the pixels per inch of a device.

The XICE architecture was built to provide a way for mobile users to connect to other machines in an efficient manner. A wireless connection is used to communicate between the two devices. Each XICE application window is shown using a *sheet*. Each sheet is associated with a presentation tree [BED04] that displays information on the screen. When presenting, it traverses the tree from left to right and top to bottom. When receiving events, it traverses from right to left.

6.1.1   How an event is processed

Events in the XICE architecture can be created locally or remotely. The basic idea is the same, but the events are processed a little bit differently. The generic case for explaining what happens to an event occurs when it is a local event and the application is not connected to a remote machine. The other cases are in essence special cases.

The event source is responsible for grabbing the input from wherever it is coming from. It then creates an event object for the input. Then it adds it to the event queue until they are dispatched to the correct applications that are listening to that event source. Each application manages the event by passing the event to the root of the presentation tree. The event traverses the nodes from right to left. There are many types of events: region, textual, and broadcast. Depending on the type of event, it is handled a little bit different. Region events are events like mouse down, mouse over, and mouse up.

66

Textual events deal with text. Broadcast events are like adding a new window or window resizing.

Here is an example of what would happen once a mouse down region event hits a button object. It checks the path from the sheet to the button to validate that the widget is associated with the sheet. It then checks to make sure that the mouse click is within the button's bounds. If it passes, then the button will change its state to being pressed. This will cause its change flag to be true, and then its parents will be notified that a change has happened.

After an event is dispatched, the window will either be repainted or changes will be sent over to the remote machine to be repainted there. The window will be repainted only if there have been any changes to the tree.

When the handheld is connected to a remote machine, local events happen in the same way. The only difference is that the changes will be sent to the remote machine to be repainted if necessary. When it is a remote event, it comes from the remote event source. The event is sent to the handheld device using sockets and TCP in Java.

6.2   Spillable Environment

In order for a user to connect to a remote computer, a display server needs to be running on it. The display server uses XICE to display a presentation tree on the remote machine. The display server starts a thread to listen for events from its local event source, which in this case is from the DiamondTouch table. The input received from the DiamondTouch table is converted into events and sent to the handheld device.

In the spillable environment, a projector displays onto the DiamondTouch. Because the display from the projector might take up more space than the DiamondTouch, the remote display will be restricted to display on only a certain portion to fit the area where the DiamondTouch is located. Normally, a computer would try to use all of the screen space available to it, but that had to be changed for these circumstances. Therefore, the owner of the computer station will input the size of the DiamondTouch in pixels into a configuration file.

The display server has to have some flexibility built into it. It needs the ability to display applications only in certain sections of the display screen. It also needs to have the option of the type of event source that the remote machine can use. Our display server has a configuration file to specify these options.

In order to connect to the display server and display in the spillable manner, the handheld device needs to the capability to show data on different screens but have them show different parts of the same data model. When the handheld device connects to a remote machine, the objects (i.e., SpillSpace and SpillSheet) used are different than when the handheld device is being used by itself (Figure 53). The process of connecting is moving the presentation tree into an object that allows the flexibility of the spillable environment. The overall architecture stays the same as connecting to another remote machine to display an application.
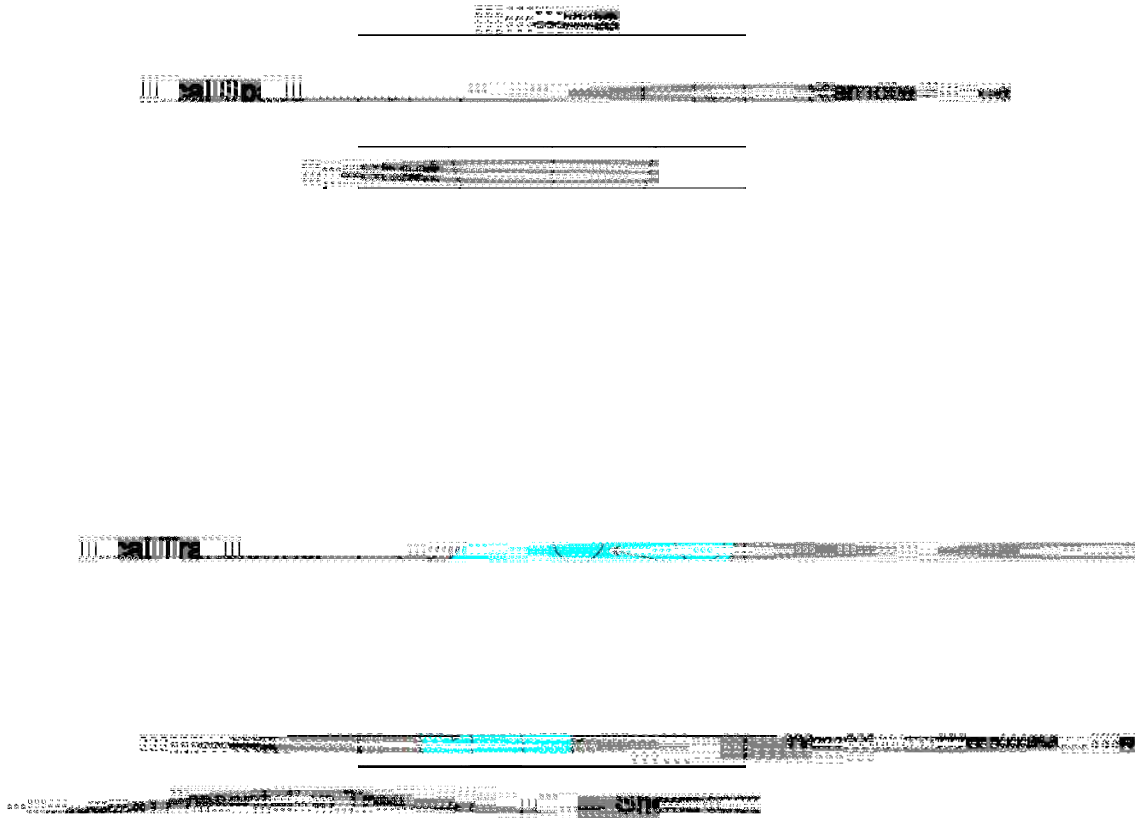
Figure 53 SpillSpace and SpillSheet Diagram

In the XICE architecture, an object called a *space* specifies the area on the screen that it will display on. Because our spillable environment requires display capabilities on two devices, the SpillSpace was used to hold a local and remote space (Figure 53). A local space is the handheld device's display area, and a remote space is the remote machine's area. Although the SpillSpace contains two spaces, it uses the remote machine's area, since that is the size of the interaction area in the spillable environment. It also uses the viewing distance and pixels per inch from the remote space to show the size of the widgets properly. With the remote space, we are able to add another event source using the DiamondTouch table to the mouse and keyboard type of interaction from the handheld device.

Each window in a space creates its own *sheet*, which is in essence an enhanced Java JFrame object. In order to simplify our architecture, the SpillSheet object was specifically built for the spillable environment (Figure 53). The SpillSheet is the sheet that shows the spillable interface and therefore creates a window the size of the remote machine's display space. It also creates the nodes necessary for the connection process and handles the events coming from the DiamondTouch table. The SpillSheet is unique in that not only is it a sheet object but also a container node. Therefore, it is in the tree, like the other nodes, to facilitate event handling and to keep the overall architecture the same. The application presentation tree, the portion of the tree that is shared by both spaces, is set to be below the SpillSheet node like normal sheet objects. The root of presentation trees are always SheetRoot objects that connect the presentation tree to the sheet. The remote presentation tree in figure 53 shows the nodes in the SpillSheet that will be serialized over to the remote machine.

Since the spillable environment shows an application in two different places with one model, there needed to be a way to facilitate it. In order to do that, a multi-parent node was created. Multi-Parent nodes allow a node to have multiple parents. For a node with two parents, the two roots of the tree can be connected to the two spaces being shown on two different devices. Therefore, events can be sourced from the DiamondTouch table and the handheld device into the same presentation tree.

On the remote side of the tree, a black box needs to be shown to take the place of where the aluminum frame will be located in the spillable environment. The black box's location will need to be updated as events are received.

70

Each display does not show the exact same part of the view. Therefore, each display needs to be able to transform the application to show their portion of the environment. Transformation nodes are also container objects meaning that they are able to have children in the tree. This contrasts with objects like rectangles that do not have children. Transformation nodes allow all their children to appear translated, rotated, and scaled. In order to do this, they have a transform object associated with them, and therefore when drawing to the screen, it changes the graphics object to draw its children with the transformation appended to it. The transformation nodes need to be placed carefully to transform the world and to be updated properly as the devices receive input from the DiamondTouch table. Three transformation nodes were added to facilitate this process: one for just the handheld, one for the remote machine, and one for both.

The local transformation node, which is just for the handheld's transformation, changes the size of the application on the handheld screen. The local space and the remote space should have different viewing distances due to the vast differences of the devices. The remote machine has to be sent a message to determine its viewing distance. The local transformation node scales the local space to be the same size as the remote space while connected.

The remote transformation node rotates and translates the application displayed on the DiamondTouch Table. Whenever an aluminum frame event is received from the DiamondTouch table, the remote transformation node is modified to fit the interaction. By changing the remote transformation node appropriately, the application will move with the handheld device and aluminum frame.

The world transformation node is the node that affects both the handheld device's and remote machine's display. It updates the scrolling done on the DiamondTouch. It receives the change in movement of the scrolling. To have the change in movement match what would be intuitive to use, the change needs to go through the inverse transform of the remote transformation node and then multiply that with the current world transformation.

Once events are sent down the tree, there must be a mechanism to update many different nodes in the tree. The DiamondTouch event grabber node was created as a special node that grabs the events as they move down in the tree and then immediately notifies the SpillSheet. With the information from the DiamondTouch event grabber node, the SpillSheet is then in charge of telling the appropriate nodes the information that they need. The DiamondTouch events are unique in that these events do not affect any other node other than the DiamondTouch event grabber node.
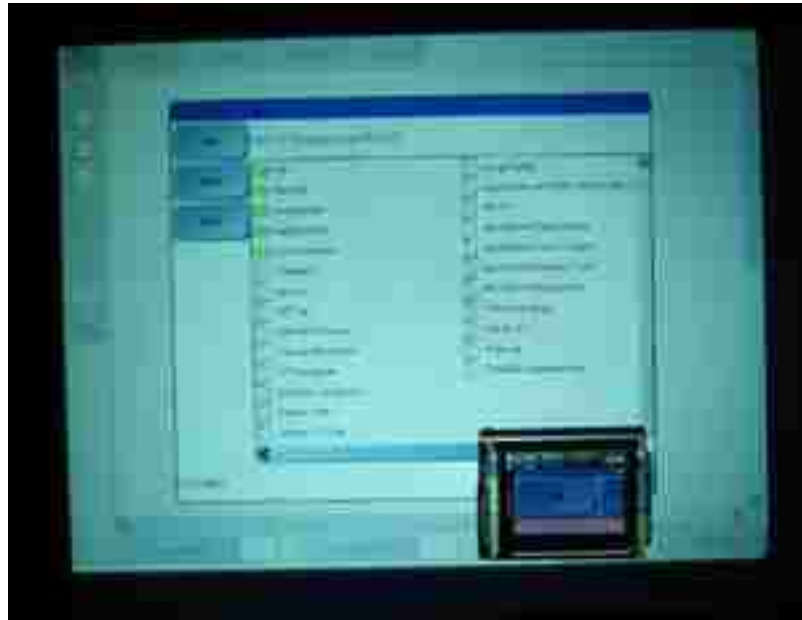
6.2.1 Dialogs

Figure 54 Dialog on DiamondTouch tableDiamondTouch Interactions

Dialogs usually are put right below the sheet root. In this way they are displayed on top. Because of the way we designed the SheetRoot object, no special modifications had to be made to allow for dialogs to show up big on the spillable interface environment as shown in figure 54.

6.3   DiamondTouch Interactions

Interactions with the DiamondTouch table are converted into the actions shown on the displays of the handheld device and remote machine (Figures 55 and 56). This process will be explained, although it is similar to normal events originating from the handheld device, but some differences make it pertinent to explain. The event source on the remote machine listens to changes from the DiamondTouch table. This event source uses the algorithm explained in Chapter 5 to be able distinguish the type of interaction.

For aluminum frame interactions, the points of the aluminum frame are put into the event. For scrolling interactions, the change of movement is sent in the event. The event is serialized and shipped to the handheld device. On the handheld device, the event is dispatched and starts traversing through the presentation tree. Due to the placement of the DiamondTouch event grabber node, the event is received (Step 1 in figures 55 and 56). The event is handled by the processBroadcastEvent() function in algorithm 14. This function sorts the event into a scrolling event or aluminum frame event by using the actionID, and the appropriate information is given to the SpillSheet (Step 2 in figures 55 and 56).

```
protected void processBroadcastEvent(Event event)
{
        String actionID = event.getActionID();
        if (actionID.equals( ALUMINUM_FRAME_TOUCH ))
        {
                Point[] points = (Point[])event.getActionContent();
                SpillSheet.aluminumFrameEvent(points);
        }
        else if (actionID.equals( SCROLLING_TOUCH ))
        {
                Point point = (Point)event.getActionContent();
                SpillSheet.scrollingEvent(point.getX(),
                        point.getY());
        }
}
```
Algorithm 14 processBroadcastEvent() Function

6.3.1 Aluminum Frame Events

In the case of the aluminum frame interaction, the SpillSheet will modify the remote transformer and black box nodes (Step 3 in figure 55). Since the remote transformer and black box nodes are inside of the SpillSheet, it can modify these nodes to update to the latest event received.
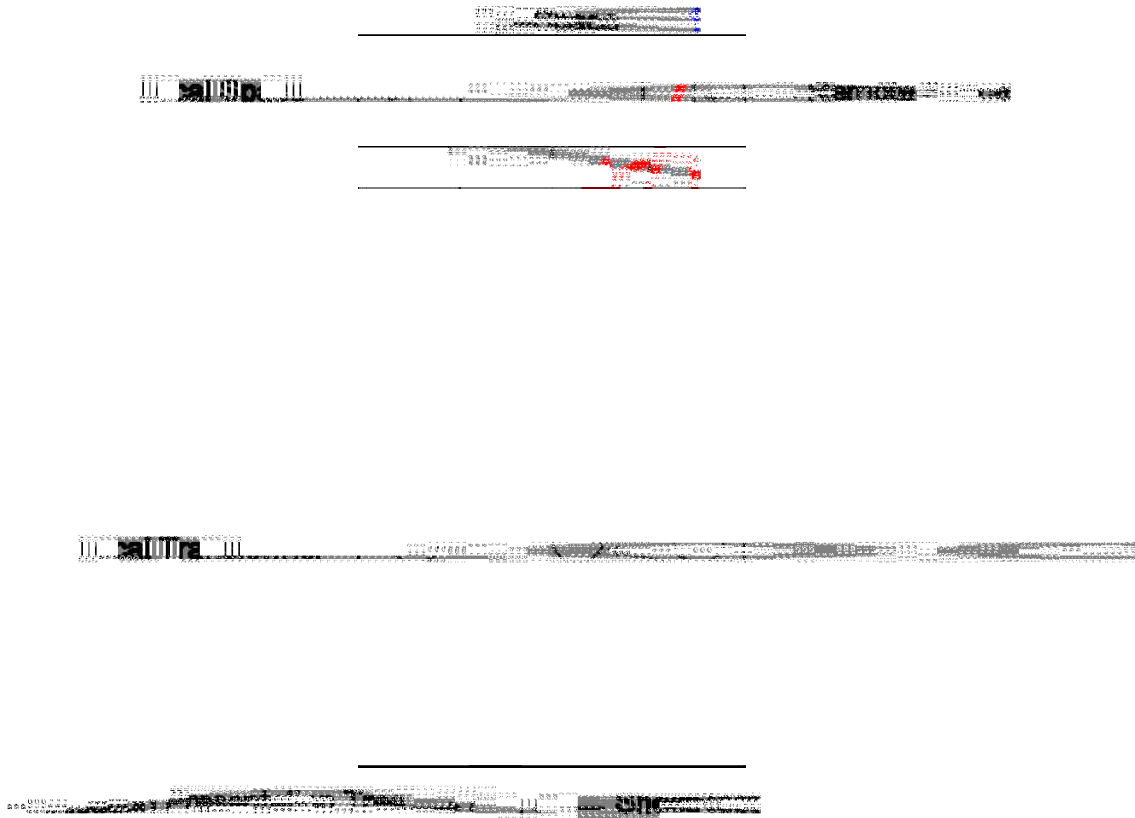
74

Figure 55 Aluminum Frame Events in SpillSpace and SpillSheet

After grabbing the position and orientation of the aluminum frame using the locations of the three conductive points, the remote transformer and black box will be updated. The remote transformer sets the transformation that rotates to the current orientation and then translates it to the foot 1 position (Algorithm 15). The black box is a polygon node that has four points. Foot 4 needs to be calculated using the three points received by the SpillSheet. The black box simply sets the four points, so it has a current update of the location of the aluminum frame (Algorithm 15). Both of these nodes' change flag become true. Because of this, its parents are notified that there was a change.

```
public void aluminumFrameEvent(Point[] pts)
{
        PointD position = getPosition(pts); // foot 1
        double orientation = getOrientation(pts);
        Transform2D rotateToOrientation =
                Transform2D.rotate(orientation));
        Transform2D translate =
                Transform2D.translate( position.getX(),
                position.getY());
        remoteTransformer.setTransform( translate.multiply(
                rotateToOrientation );
        // rotate then translate

        if (blackBox != null)
        {
                Point[] points = new Point[4];
                points[0] = pts[0]; // foot 1
                points[1] = pts[1]; // foot 2
                points[2] = pts[2]; // foot 3
                points[3] = getFourthPoint(pts); // foot 4
                blackBox.setPoints(points);
        }
}
```

Algorithm 15 aluminumFrameEvent() Function

After the event is finished being processed, it notifies the SpillSpace, which

notifies the local and remote sheets to repaint. The local sheet should ignore the repaint

call, since it should have no changes, but the remote sheet will serialize the changes and

notify the remote machine to repaint. It will then clear the changes in the tree. Because

the local sheet ignores the repaint call, this causes handheld interactions to be slightly

faster than scrolling interactions.

6.3.2   Scrolling Events

In scrolling interactions, the SpillSheet modifies only the world transformation

node (Step 3 in figure 56). The movement is transformed by the inverse transform of the

remote transformation node and then multiplied by the previous state of the world

transformation node (Algorithm 16). After the processing of the scrolling event, the

76

presentation trees on the handheld device and remote machine will have to be updated, since their change flags will be set to true. Since the local and remote machines are both modified, it will take a little longer than the aluminum frame interactions.
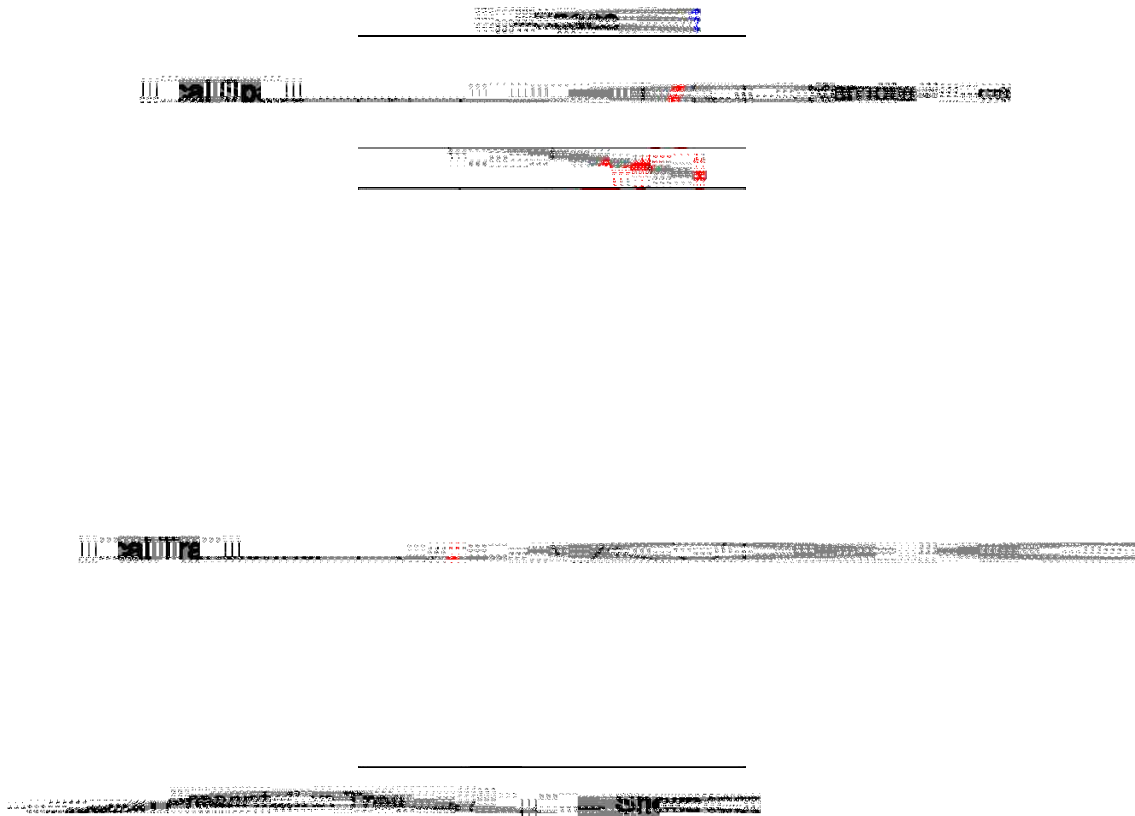

Figure 56 Scrolling Event in SpillSpace and SpillSheet

```
public void scrollingEvent(double dx, double dy)
{
        Transform2D remoteInverse =
                remoteTransformer.getTransform().getInverse();
        SizeD size = remoteInverse.multiply(new SizeD(dx, dy));
        Transform2D transform =
                Transform2D.translate(size.getWidth(),
                size.getHeight());
        worldTransformer.setTransform(
                worldTransformer.getTransform().multiply(transform));
}
```

Algorithm 16 scrollingEvent() Function

Chapter 7   Conclusion

Mobile users face the dilemma of not being able to use their mobile devices in a number of situations. Their screens are very limited. Many times mobile users are in situations where they are not moving and would rather see their applications on a larger screen.

In our unique environment, users are able to see their application on a large screen and interact with it in a novel way. No one has ever before put a mobile device on top of the DiamondTouch table. They have only been able to interact with it rather than on it. This novelty has allowed a new way for users to use their mobile devices with the DiamondTouch table. This environment was created using our enhanced XICE Architecture.

7.1   Potential Problems

The environment we have created has been for prototyping purposes. Therefore, many issues have not been resolved. If the devices become unconnected for some reason, a solution has not been implemented. Also, there has not been any extended use of the XICE architecture. These types of problems seem more for commercial purposes rather than research purposes, and therefore we have left them alone.
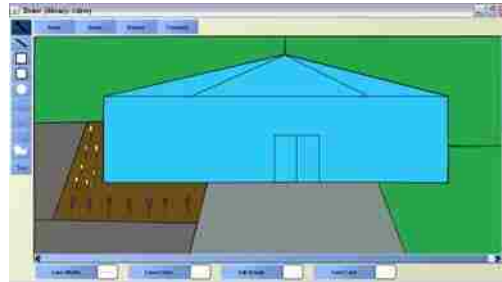
7.2 Verification

Figure 57 Draw application

We have shown that we can create an environment as proposed. We have shown the proof of concept. We have applications that can be used in our environment like a draw application, a spreadsheet application, and a note taking application (Figures 57-59). They have been shown to be interactive with the current capabilities of the XICE Architecture. Many other applications have been built that could be used in this environment.
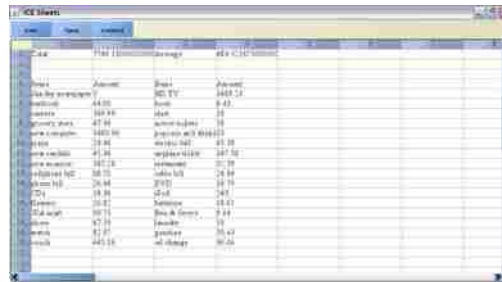

Figure 58 Spreadsheet Application

Figure 59 Sticky Notes Application

7.2.1 Latency

We have shown that it is interactive. When the user connects to the remote machine, it takes between 100 and 900 milliseconds to complete all of the setup. It usually takes between 200 and 600 milliseconds. It seems to depend mostly upon the network. To disconnect, it takes between 100 and 350 milliseconds. This is faster, since it is a simpler process.

To find approximately how long it takes an event to be propagated from the DiamondTouch and back to be repainted, an average of the first 50 DiamondTouch events were taken. When making interactions with the aluminum frame on the DiamondTouch, it takes between 10 and 70 milliseconds. When making scrolling events on the DiamondTouch, it takes between 20 and 170 milliseconds. Scrolling events take longer since they have to repaint the tree on the handheld device and the handheld events do not. The spreadsheet application is the reason that some of the numbers are high. The spreadsheet application is much more computationally expensive. The other two applications hug the low the end of the spectrum tightly.

7.3 Conclusion

These three applications have been shown to use the spillable interface. They have been shown to be interactive in this environment. We have shown that this environment can be possible with handheld devices.

Other types of environments could be created that allow a handheld device to expand itself on a large screen. One way is that moving the handheld device on the DiamondTouch would not move the application around but allow the display on the

handheld device to show the appropriate portion of the application. This would be like Ubiquitous Graphics [SAN06], but the user would not have as much flexibility. Future work might allow more than one handheld device to be used in a spillable type of environment.

Chapter 8   Bibliography

[AMA07] Amazon.com, www.amazon.com, 8 Jan. 2007.

[BAU03] Baudisch, P., and Rosenholtz, R. "Halo: A Technique for Visualizing Off-Screen Locations." *In Proceedings of CHI 2003*, pp. 481-488, 2003.

[BAU01] Baudisch, P., Good, N., and Stewart, P. "Focus Plus Context Screens: Combining Display Technology with Visualization Techniques." *In proceedings of ACM UIST 2001*, pp. 31-40, November 2001.

[BED04] Bederson, B., Grosjean J., and Meyer J., "Toolkit Design for Interactive Structured Graphics." *IEEE Transactions on Software Engineering* , Vol. 30, No. 8, pp. 535-546, August, 2004.

[DIE01] Dietz, P.H.; Leigh, D.L., "DiamondTouch: A Multi-User Touch Technology." *ACM Symposium on User Interface Software and Technology (UIST)*, pp. 219-226, November 2001

[MAC02] Mackay, W., Pothier, G., Letondal, C., "The Missing Link:Augmenting Biology Laboratory Notebooks." *In Proceedings of UIST 2002*, Paris, France, pp. 41-50, 2002.

[MIM07] Mimio XI. http://www.mimio.com.

[MIT06] Mitsubishi Electronic Research Laboratories, http://www.merl.com/projects/DiamondTouch/, 5 March, 2007.

[MIT03] Mitsubishi Electronic Research Laboratories, merldt Version 2.0, 2003.

[SAN06] Sanneblad, J. and Holmquist L. "Ubiquitous graphics: combining hand-held and wall-size displays to interact with large images." *In Proceedings of AVI 2006*, Venezia, Italy, pp. 373-7, 2006.

[SAN03] Sanneblad, J. and Holmquist, L.E. "OpenTrek: A Platform for Developing Interactive Networked Games on Mobile Devices." *In Proceedings of Mobile HCI 2003*, Udine, Italy.

[VAN01] Vanderbei, Robert J. Linear Programming: Foundations and Extensions. Springer, 2001.

[WAC07] Wacom Technology Corporation, www.wacom.com, 25 June, 2007.

[YEE03] Yee, K.-P. "Peephole Displays: Pen Interaction on Spatially Aware Handheld Computers." *In Proceedings of CHI 2003*, Ft. Lauderdale, FL, pp. 1-8, 2003.