Fluid DTMouse: Better Mouse Support for Touch-Based Interactions

Alan Esenther, Kathy Ryall
Mitsubishi Electric Research Laboratories
201 Broadway
Cambridge, MA 02139
+1-617-621-7564

{esenther, ryall}@merl.com

ABSTRACT

Although computer mice have evolved physically (i.e., new form factors, multiple buttons, scroll-wheels), their basic metaphor remains the same: a single-point of interaction, with modifiers used to control the interaction. Many of today's novel input devices, however, do not directly (or easily) map to mouse interactions. For example, when using one's finger(s) or hand directly on a touchable display surface, a simple touch movement could be interpreted as either a mouse-over or a drag, depending on whether the left mouse button is intended to be depressed at the time. But how does one convey the state of the left mouse button with a single touch? And how does one fluidly switch between states? The problem is confounded by the lack of precision input when using a single finger as the mouse cursor, since a finger has a much larger "footprint" than a single pixel cursor hotspot. In this paper we introduce our solution, *Fluid DTMouse*, which has been used to improve the usability of touch tables with legacy (mouse-based) applications. Our technique is applicable to any direct-touch input device that can detect multiple points of contact. Our solution solves problems of smoothly specifying and switching between modes, addressing issues with the stability of the cursor, and facilitating precision input.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Interaction styles *(e.g., commands, menus, forms, direct manipulation)*.

General Terms

Design, Human Factors.

Keywords

visual interaction, tabletop interfaces, mouse emulation, multi-touch.

AVI '06, May 23-26, 2006, Venezia, Italy.

1. INTRODUCTION

Since its introduction almost forty years ago, the mouse concept has been and continues to be a critical part of computer interfaces. It is arguably the most ubiquitous part of any GUI. Mice have evolved, now having multiple buttons, scroll-wheels, and other sensing devices. However, even as today's computing moves off the desktop and into the environment, the concept of a mouse remains largely the same: a single-point of interaction, with modifiers used to control the interaction.

Newer touch-sensitive surfaces such as horizontal tabletops or vertical whiteboards can allow users to interact with displayed content more naturally and directly than is possible with a computer mouse. Although some can detect simultaneous points of input, most applications, new and existing, are generally written assuming that a mouse is being used. While we note that new interaction paradigms are evolving and that we are actively researching in this very area, we must recognize the importance of providing support for legacy applications in both today's and future systems. Such support requires mouse-based input.

One of the challenges of supporting such touch-sensitive surfaces is to provide a natural mechanism for emulating mouse buttons on a surface that only detects touches. This is particularly problematic with "direct-touch" surfaces where the input and output are the same. For direct-input devices, the user is touching the actual image of the content, rather than manipulating an indirect input device such as a mouse or a touchpad off to the side. In these touch-based systems, there is a difficulty in interpreting a single-touch when used as a replacement for mouse-input; a simple touch movement could be interpreted as either a mouse-over or a drag, depending on whether the left mouse button is intended to be depressed at the time. But how does one convey the state of the left mouse button with a single touch? And how does one fluidly switch between states? For example, users may have to right-click to open a context menu, drag with the left mouse button to re-position content, double-click to engage a button, or may simply wish to reposition the mouse without any additional input being generated.

In addition to the mouse button problem, there is a second, confounding problem – the lack of precision input when using a single finger as the mouse cursor. A finger has a much larger "footprint" than a single pixel cursor hotspot, making it awkward, difficult, and at times impossible to interact with GUI elements designed for mouse interaction.

In this paper we introduce Fluid DTMouse, our solution to the finger-based, direct-touch mouse problem; it has significantly enabled the use of touch tables with legacy (mouse-based) applications. While we have developed our techniques using the DiamondTouch platform, they are applicable for any direct-touch input device that can detect multiple points of contact (e.g., [2] [11]) or future technologies with this capability. Our solution solves problems of smoothly specifying and switching between modes, addressing issues with the stability of the cursor, and facilitating precision input. Section 2 introduces the mouse emulation problems and discusses related work. Section 3 describes the Fluid DTMouse work and explains how it solves key problems. Section 4 wraps up and discusses future and ongoing work.

2. BACKGROUND & RELATED WORK

One of the more fundamental challenges for direct-touch input is that users may wish to move a mouse cursor across a touch-sensitive display without engaging any mouse buttons (e.g., simply move the mouse over a tooltip). But when a user touches a touch-sensitive surface, how is the system to know whether the touch was intended to simply move the mouse, or to "drag" with the mouse by holding down the left mouse button during the move? Thus direct-touch systems suffer from a different variant of the well-known Midas Touch problem [9]; every touch counts! It is instructive to consider how other touch surfaces deal with this problem, even though most are not designed for larger surface areas (the focus of our work).

The ubiquitous TouchPad [5] on laptop PCs has mouse buttons, too, but there is a mechanism to switch between modes without using the buttons. A user can switch between moving the mouse and dragging with the mouse by tapping once and then quickly pressing the finger down again and holding it down (a "click and a half"). This sequence is recognized as a cue to hold down the left mouse button. However, on a flat surface it can be difficult to precisely position a cursor with a "large" fingertip, particularly with direct input surfaces because the user's finger is obscuring the very content with which they wish to interact. The problem of obscuring content has been addressed by offsetting the cursor from the touch point, but this forfeits one of the big advantages of a direct input surface – that you can directly touch the content that you wish to interact with.

Fingerworks [3], a pad-like device, will use the average of two adjacent finger touches as the location of the mouse cursor. However, this is an indirect input device, and it does not address problems with obscuring content or fluidly moving between mouse-over and mouse-dragging modes.

The Touchscreen [4] is a direct input screen that also uses the average of two touch locations as the location of the mouse cursor. However it is not actually possible to detect whether one or multiple points is being touched, which limits the dimensions of its usefulness. For example, it requires a dedicated onscreen "right-click mode" button to specify whether touches should be interpreted as left-clicks or right-clicks. This solution does not support mouse-overs at all, avoiding the issue of how to move the mouse without holding down a mouse button.

Tablet PCs [6] are also direct-input devices. They rely on a special pen. These devices have the unusual ability to detect "hovering" when the pen is near the surface but not actually in contact with the surface. So if the pen is hovering, then the mouse is simply moved, and if the pen is in contact with the Tablet PC surface then dragging (left-mouse button down) is engaged. Right-clicking is supported by holding a button on the Tablet PC pen, or holding the stylus down briefly, or by selecting a "right-click" menu item to indicate that the next tap should be interpreted as a right-click. It is the lack of this third state, hovering (the first two states being touching or not touching), which makes mouse-over support so difficult on most touch surfaces. For our research we were interested in solutions that would work with fingers, since requiring the use of a special pen or other hardware may not be as convenient – particularly for a large touch surface. It should be noted, however, that another advantage of using a fine-tipped pen such as the Tablet PC pen is that the fine tip does not obscure content as much as a finger does. At any rate, currently Tablet PCs are only available with very small surface areas, compared to tabletop and whiteboard surfaces.

The DiamondTouch [1] "DiamondTouch Mouse" [7] software supports a right-click by tapping with a second finger. It also supports a TouchPad emulation mode. As with the TouchPad, normal finger drags are interpreted as mouse moves, and the tap then press ("click and a half") gesture engages the left mouse button. This provides a mechanism for switching between mouse-over and dragging modes, but it is not a very smooth mechanism. Drawing a series of lines requires extra tapping, and it is difficult to precisely aim the mouse-down since it is part of a gesture where the finger is tapped twice (but kept down the second time). There is another mode in which all touches occur with the mouse down, but this mode has two problems. All finger drags result in mouse drags (no mouse-overs), and right-clicking (again, by tapping with a second finger) causes first a left-click and then a right-click to be sent to the first finger location. While this earlier mouse-emulation provides most of the necessary mouse functionality, its moded nature made it awkward to use. Our current work was inspired by the shortcomings of the earlier system; there has been a demonstrated need to support better mouse-emulation.

The SmartBoard [8] supports mouse-overs via a dedicated Hover button. When this button is pressed from a special toolbar, a mouse-over mode is entered. A subsequent finger movement on the surface will move the mouse without dragging. SmartBoard allows right-clicking by pressing and holding until you see a right-click menu. There is also a dedicated physical button on an external pen tray that can be pressed to tell the system to interpret the next touch as a right click. SmartBoard also supports right-clicking by tapping with a second finger, which will send a left-click and then a right-click to the first touch location. The primary drawback is the dependency on dedicated modes for mouse-over and mouse-dragging, without a smooth mechanism for transitioning between the modes.

3. FLUID DTMOUSE SOLUTION

In order for mouse emulation to be smooth and natural on a touch-based surface, a number of things are desired. First, it

must be easy to precisely position the mouse cursor. This is particularly challenging when fine positioning is attempted with a finger since the user's finger typically obscures the mouse cursor. Second, there must be a simple mechanism to toggle between mouse-over mode (just moving the mouse) and mouse-dragging mode (dragging with the mouse). Third, it is undesirable for this toggling mechanism to require movement of the cursor itself. For example, once you have moved the mouse cursor over a small target that is to be dragged, you do not want the act of switching to mouse-dragging mode to move the cursor off of the target. Fourth, and perhaps most importantly, any such solution should "feel" very easy and natural – what we call "fluid."

The Fluid DTMouse solution for better mouse support was the culmination of several earlier attempts to support existing mouse-based software for users of a DiamondTouch [1] table and software [10]. The DiamondTouch table can detect multiple concurrent users as well as multiple touch points by each user. However, it is only the latter capability that was important for supporting existing software on the table, because common existing programs (and operating systems) are not designed to support concurrent mouse input from multiple users. Our focus in this work is to provide traditional single-user mouse emulation.

With Fluid DTMouse, whenever a user touches the table with one finger, the system behaves as though the left mouse button is being held down at the time. This facilitates a simple and intuitive behavior when the user is performing common operations such as scrolling, dragging, drawing, clicking buttons, and double-clicking. However, this makes it awkward to perform mouse-over operations such as activating tooltips and image rollovers in web pages (wherein moving the mouse over an image changes the image). If the left mouse button is held down during what would normally be a mouse-over operation, then text, for example, will typically become unexpectedly selected.

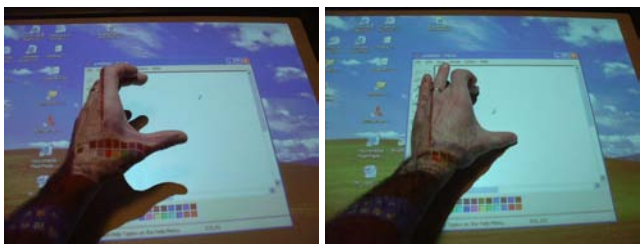

Figure 1. Normally a touch would cause the left mouse button to be held down. However when two fingers (the thumb and middle finger in this case) are placed on the table at the same time, the mouse is moved, not dragged. The mouse cursor immediately jumps to the point in between the two fingers, providing a view of the mouse cursor that is not obscured by either finger.

As shown in Figure 1, the solution was to leverage the multi-touch capability of the surface. If the user touches with two fingers, rather than one finger, *at the same time* (or nearly so) then the mouse cursor is just moved without engaging the left mouse button. Furthermore, the mouse cursor is moved to the point directly in between the two fingers. This insures that neither finger is obscuring the mouse cursor, so the mouse cursor can be precisely positioned by moving either or both fingers.

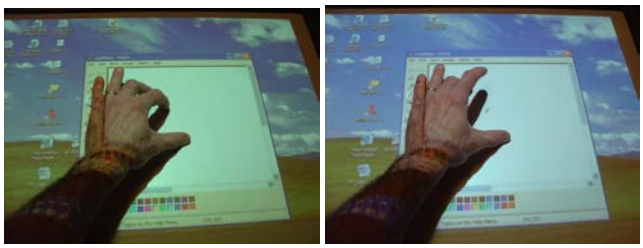

Figure 2. While in this mouse-move mode, tapping with a third finger (in this case the index finger) in between the other two toggles the left mouse button.

As shown in Figure 2, once this mouse-move mode has been entered by placing two fingers on the surface, repeated tapping in between the other two fingers with a third finger will toggle the state of the left mouse button. (It doesn't matter exactly where the third finger taps, so long as it is within the rectangle created by the first two fingers.) This allows for smoothly transitioning between mouse-moving and mouse-dragging modes, without causing the mouse cursor to move during the transition. This addresses problems with the "stability" of the mouse cursor position that are introduced with the "click and a half" technique described earlier. The new technique allows you to keep your fingers in contact with the table to save effort that might be spent re-acquiring a cursor location for subsequent precision input. For example, after drawing a line, as shown with the painting program in Figure 3, the user can tap to disengage the left mouse button, then move both fingers (still in contact) to reposition the mouse cursor, and then tap again to smoothly start drawing a new line. When the user stops touching the table, the mouse-over mode is exited and the left mouse button is disengaged if it had been engaged. Therefore, for simple dragging or for drawing just one line, the third finger tap is only required once, not twice.

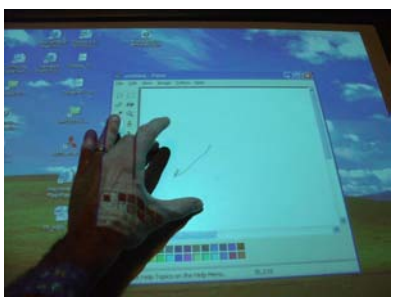

Figure 3. Now that the left mouse button has been engaged, dragging the thumb and index finger causes a drag operation centered between the fingers. The left mouse button is released when either a subsequent third finger tap is performed, or when the hand is removed from the table.

In practice it seems most natural to use the thumb and middle finger of one hand to enter mouse-over mode. This allows the index finger to be in position for tapping in between the other two. However, if the user finds that her fingers or hand is obscuring the cursor or other content from the view of herself or other users, then she can instead use, for example, her index finger from each hand for the first two touches. By keeping her

hands widely separated she can keep them from blocking anyone's view. The "widely-separated two-hand" technique is also sometimes easier to use for very precisely positioning the mouse cursor location.

The right mouse button is activated by touching with first one finger and then tapping briefly with a second finger. At that point the first finger can be moved around to perform dragging with the right mouse button depressed. When the first finger is removed from the surface, the right mouse button is disengaged. In an earlier version, the right click would be generated as soon as the second finger came up, but some applications required the capability to drag with the right mouse button, so the behavior was modified. This solution for right-clicking does not address the need for very precisely specifying the location at which to activate the right mouse button. But it was found to be so intuitive that it was desirable as implemented. Alternative techniques can supplement this default behavior.

Having two fingers on the table seems to provide a more stable touch (less muscle vibration over time) than just one finger. If the thumb and middle finger of one hand are used to specify the two points then it seems to be a very natural and stable posture for a human hand. The two fingers nicely "anchor" the touch, which is particularly important when trying to precisely position the mouse pointer.

It should be noted that it is due to nuances of the DiamondTouch hardware that the third tap should be in between the first two fingers. With some other system, such as a system based on cameras, it is conceivable that the third tap could be anywhere rather than only in between the first two fingers. Furthermore, with the DiamondTouch system, the third tap should be within the rectangular "bounding box" created by the other two fingers. Otherwise the third tap will modify the cursor location as it toggles the left mouse button. The main implication of this is that the first two fingers should not be placed along the same horizontal or vertical antennas used to detect the touch in a DiamondTouch system. Doing so would create a bounding box "rectangle" with no area in which to tap with the third finger. In practice this doesn't tend to be a problem since it is somewhat less natural (more uncomfortable) to place the hand in this problematic way.

Another issue is that the two-finger technique could make it difficult to "reach" into corners of the surface. With a DiamondTouch system, this is addressed by simply under-projecting the display onto the table a bit. Other systems may require the technique to be "intelligently magnetic" near edges.

4. CONCLUSION AND FUTURE WORK

Fluid DTMouse relies on multi-touch detection capabilities to provide improved mouse emulation on a touch-sensitive surface. It addresses issues including mouse-over support, smoothly toggling the left mouse button, ergonomics, and precision input.

With the current implementation, one issue that remains is that even with two fingers widely separated, the cursor might still be obscured by the hand of the toucher for some observers. Although two widely-separated hands could be used rather than two fingers on the same hand, in some cases it may be better to apply simple techniques for dynamically offsetting the cursor, or duplicating the cursor area elsewhere, as needed. While additional experimentation and evaluation is needed, Fluid DTMouse is a notable improvement over existing solutions.

Although not discussed in this paper, another issue is support for other mouse buttons, or for the mouse-wheel. Such support can be (and has been) added through other natural extensions. In some cases, however, the behavior is not enabled by default so as to simplify the tool. These solutions rely on variations of the basic techniques presented in this paper, and are still being assessed.

Finally, as might be expected, Fluid DTMouse has quite a few subtleties such as timing issues and error states that were not addressed in this paper due to space considerations. Hands-on exploration is the best way to understand and evaluate our techniques. It is encouraging that this latest generation of mouse emulation techniques finally allows sorely missed access to particular aspects of many legacy applications – particularly those aspects that were reliant on hovering and/or precise input via a mechanism that "feels" natural and intuitive.

5. REFERENCES

[1] Dietz, P. and Leigh, D. *DiamondTouch: A Multi-User Touch Technology*. In Proc. of UIST '01, ACM, NY, 2001, pp. 219–226.

[2] Rekimoto, J. *SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces* In ACM CHI Conference on Human Factors in Computing Systems (CHI), ACM Press. 2002, pp. 113-120.

[3] FingerWorks. www.fingerworks.com

[4] Mass Multimedia, Inc. http://www.touchscreens.com

[5] Synaptics Inc. http://www.synaptics.com/products/touchpad.cfm.

[6] Microsoft Corporation. http://www.microsoft.com/windowsxp/tabletpc/

[7] Mitsubishi Electric Research Laboratories. http://www.merl.com/projects/dtmouse/

[8] Smart Technologies SMART Board. http://www.smarttech.com/SmartBoard

[9] Hansen, J., Andersen, A., and Roed, P. *Eye-gaze control of multimedia systems.* ACM Symposium on Eye Tracking Research & Applications, 1995.

[10] Esenther, A., Forlines, C., Ryall, K., Shipman, S., *DiamondTouch SDK: Support for Multi-User, Multi-Touch Applications*, ACM Conference on Computer Supported Cooperative Work (CSCW 2002 Demonstration). Available as, Mitsubishi Electric Research Labs Technical Report TR2002-048, 2002.

[11] Han, J. Low-cost multi-touch sensing through frustrated total internal reflection. In Proc. of UIST 2005.