The whole of science is nothing more than a refinement of everyday thinking.

– Albert Einstein, 1879-1955.

University of Alberta

BLURRING THE BOUNDARY BETWEEN DIRECT & INDIRECT MIXED MODE INPUT ENVIRONMENTS

by

Xing Dong Yang

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2013
Edmonton, Alberta

To my parents Jian-Jia Yang and Yun'E Chen

Abstract

The current computer input devices are either direct or indirect based on how interactive input data is interpreted by a computing system. Existing research have shown the benefits and limitations of both input modalities, and found that the limits of one input mode are the advantages of the other mode. In this thesis, we explore a new type of input device, which integrates direct and indirect input into a shared input space: mixed input mode. With a mixed mode input device, the two input modes could well complement each other in the shared input space, such that users can freely choose which mode to use to achieve optimized performance. We explore the hardware and software design space of mixed mode input device. On the hardware side, we created three novel prototype devices: 1) LucidCursor - a mixed mode input handheld device; 2) LensMouse - a mixed mode input computer mouse; and Magic Finger - a mixed mode finger-worn input device. We demonstrate how these devices can be built. We also evaluated the performance of the three proposed prototypes from various perspectives through a set of carefully designed user and system evaluations. On the software side, most of the current software user interfaces are only designed and optimized for a certain input mode, e.g. either mouse or finger input. To facilitate the mixed mode input on the existing software UIs, we proposed and evaluated two target expansion techniques - TouchCuts & TouchZoom. These techniques allow touchscreen laptop users to freely choose an input mode without impacting users' experience of the other input mode (e.g. mouse). An important finding of this thesis is that the mixed mode input devices have many add-on benefits, which can significantly improve the user experience of interacting with computer systems.

Acknowledgements

There are many people I would like to thank for giving me support, encouragement, help, and guidance in the past five years.

First, this work would not have been possible without my advisors Pourang Irani and Pierre Boulanger. Pourang is one of the most important people in my life. Pourang has been a great mentor, advisor, and friend of mine since I was a senior undergraduate student. Pourang introduced me to the world of Human-Computer Interaction and gave me a tremendous amount of help in shaping me into the researcher that I am today. Pierre has been so supportive and has been a good example of balancing between hard work and good life since the first day I joined his lab. Together, they have provided me with numerous insightful comments and inspiring discussions during my doctoral study. I can never thank them enough.

A big portion of this thesis comes from my internships at Autodesk Research, where I was extremely lucky to be mentored by Tovi Grossman, who is one of my most respected researchers in the field. Tovi had a significant influence on my standard of good research and how good research should be carried out. I have also been greatly privileged to work with George Fitzmaurice from Autodesk Research, Xiang Cao from Microsoft Research Asia, and Daniel Wigdor from University of Toronto during my internships at Autodesk and Microsoft. It was a highly rewarding experience to work with those extremely talented people and to learn from their unique perspective and ways of conducting productive research.

I would also like to thank my final examination committee members Sheelagh Carpendale, Abram Hindle, and Irene Cheng for thoroughly going over my thesis and for their excellent comments in making this thesis better. Additionally, I would like to thank my collaborators outside school, Olivier Chapuis, Emmanuel Pietriga, Michel Beaudouin-Lafon, Andrea Bunt, Shahram Izadi, Xiangshi Ren, and Huawei Tu, who hand shared with me their knowledge and passion for good work in various research projects.

School life would not have been such a fulfilling experience without my labmates and colleagues, Khalad Hasan, Barrett Ens, Hai-Ning Liang, David McCallum, and Edward Mak, who had contributed their time and hard work into my projects and had been with me days and nights before paper deadlines. Robyn Taylor and Matthew Hamilton, who had been great friends of mine, shared my joy and grief during our study at the University of Alberta.

Finally, I must thank my parents and my girlfriend Xiaozhou Zhou, who have been truly supportive during my PhD training and have always been there for me. I would not have accomplished any success without their support.

Publications from This Thesis

(Permission from ACM has been granted for these works to appear in this thesis)

X. D. Yang, T. Grossman, D. Wigdor, and G. Fitzmaurice. Magic Finger: Always-Available Input through Finger Instrumentation. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pages 147-156, 2012. (http://doi.org/10.1145/2380116.2380137)

X. D. Yang, T. Grossman, P. Irani, and G. Fitzmaurice. TouchCuts and TouchZoom: Enhanced Target Selection for Touch Displays Using Finger Proximity Sensing. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 2585-2594, 2011. (http://doi.org/10.1145/1978942.1979319)

X. D. Yang, E. Mak, D. McCallum, P. Irani, X. Cao, and S. Izadi. LensMouse: Augmenting the Mouse with an Interactive Touch Display. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 2431-2440, 2010. (http://doi.org/10.1145/1753326.1753695)
NOMINATED FOR BEST PAPER AWARD

X. D. Yang, E. Mak, P. Irani, and W. F. Bischof. Dual-surface input: Augmenting One-Handed Interaction with Coordinated Front and Behind-The-Screen Input. In *Proceedings of the ACM International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI)*, 10 pages, Article No.5., 2009. (http://doi.org/10.1145/1613858.1613865)
NOMINATED FOR BEST PAPER AWARD

X. D. Yang, P. Irani, P. Boulanger, and W. F. Bischof. One-Handed Behind-The-Display Cursor Input on Mobile Devices. In *Proceedings of the ACM SIGCHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA)*, pages 4501-4506, 2009. (http://doi.org/10.1145/1520340.1520690)

Patents from This Thesis

T. Grossman, G. Fitzmaurice, X. D. Yang, and P. Irani. TouchCuts and TouchZoom: Enhanced Target Selection for Touch Displays using Finger Proximity Sensing. (US: 20130097550; WO: 2013055997).

P. Irani, E. Mak, and X. D. Yang. Computer Mouse with Built-in Touch Screen. (US: 20120092253; WO: 2010148483).

A Note on My Contributions

I led all the projects presented in this thesis. I was involved in producing and refining the original project ideas, designing the studies, analyzing the experimental data, and writing the paper. I built the hardware prototypes for LucidCursor and Magic Finger and developed the code for LucidCursor, Magic Finger, and TouchCuts & TouchZoom. My thesis advisors and collaborators gave me tremendous amount of help during the course of the projects. I hereby use first-person plural throughout this thesis to acknowledge their contributions.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Direct and Indirect Input

Input devices can be characterized into two classes, *direct* and *indirect*, based on how interactive input data is interpreted by the computing system. With indirect input, input is carried out via an intermediate input device such as computer mice, trackpads, tablets, trackballs and joysticks. Pointing or manipulating a virtual object is carried out indirectly through the input device, where the movement of a user's hand is mapped onto the movements of an on-screen cursor. Actions such as clicking the virtual object can be performed by pressing a button on a mouse or tapping on a trackpad. On the other hand, with direct input, the input space often coincides with the display space. Thus users can point or manipulate a virtual object directly as if they are manipulating a real physical object. Clicking a virtual object can be performed simply by directly tapping on the object. Examples of direct devices include touch screens, light pens, etc.

Indirect input devices (e.g., computer mice or trackpads) are popularly used on desktop/laptop computers. With an indirect input device, a mouse cursor can be positioned using two methods: relative and absolute. With relative control, the motion of the user's hand maps onto the motion of the cursor so that the direction of the cursor's movement is consistent with the direction of the user's hand movement. Moving the user's hand moves the cursor in a similar manner. Repeating the same motion by lifting the device (or *clutching*) moves the cursor to a distant region (Figure 1.1). The computer mouse, the most widely used input device since being invented in the 1960's by Engelbart, uses relative positioning. On the other hand, with absolute control, the input device's input space has a one-to-one mapping onto the display space of a screen. For instance, tapping on a touch screen triggers a touch event at the same location on the screen.

With an indirect input device, the amount of hand movement in motor space is often unequal to the amount of cursor displacement in display space. The ratio between cursor displacement versus hand movement is called *control-display ratio* (or *CD ratio*), which is an important factor in designing cursor-based interaction techniques [2, 18, 22, 41]. A CD ratio less than 1 allows the user to move the cursor a large distance with only a small amount of hand movement. This allows

Figure 1.1: Illustration of mouse clutching. The straight, dashed arrow lines indicate the movement of the cursor every time the user clutches the mouse.

efficient cursor displacement but can be inaccurate with tasks requiring precise control [43, 81, 127]. Nevertheless, a CD ratio greater than 1 allows precise control but is considered to be less efficient [43, 81, 127]. Adjusting the CD ratio to best fit the needs of interactive tasks can largely improve the performance of indirect input [2]. In general, indirect input has several benefits:

1. pixel-size cursor tip allows indirect input to be highly accurate in pointing tasks [40];
2. adjustable CD ratio allows cursor movement to be efficient and accurate [43, 81, 127];
3. supports rich cursor enhancement techniques [1, 2, 34, 39, 77, 107];
4. the cursor can have a hover state, which allows for a rich design space for novel interaction techniques [45].

On the other hand, indirect input is also limited as it is less efficient than direct input in pointing at targets larger than 6.4 mm [117], and cursor displacement can be inefficient due to clutching [127].

In recent years, direct input devices such as touch screens have been gaining popularity. They are widely used on mobile phones, GPS devices, and laptops. With direct input users do not require an intermediate device to interact with virtual objects. For this reason, direct input is often considered to be natural and intuitive [40]. With direct input, users can interact with virtual objects directly. Most direct input devices use absolute control mechanisms, in which the direction and amount of the user's hand movement match exactly those of the cursor (if there is one) on the display. Although not very common, direct input devices can also use relative control mechanisms. Examples include the Offset-cursor [107] and DTMouse [37]. Note that direct input can also be carried out by using a stylus. However, the stylus has become less popular for handheld devices. Therefore, in this thesis, direct input is synonymous to input with the finger. We summarize the benefits of direct input as follows:

1. more efficient than indirect input for pointing at large objects [40];

2. more efficient than indirect input when accessing objects within a reachable distance because there is no clutching mechanism involved [127].

However, existing studies have demonstrated that direct input methods have several limitations:

1. they are inaccurate when pointing at small targets [128] (Figure 1.2);
2. they suffer from occlusion issues due to the 'fat finger' problem [128];
3. they suffer from reachability issues [77] (Figure 4.5 on page 24);
4. they do not support cursor enhancement techniques;
5. they do not support cursor hovering modes, thus limiting the design space of interaction techniques.

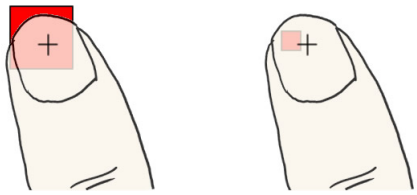

Figure 1.2: Finger input is restricted by the size of the target. The target should be bigger than the footprint of the fingertip (left). Otherwise, the target can be occluded, thus being difficult to select (right). The crosshair indicates the location of the touch.

1.2 Motivations for Mixed Input Mode

The benefits and trade-offs for each direct and indirect compensate for one another. However, ever since the introduction of the two input modes in the 1960's[1], direct and indirect input have been used in isolation from each other. This motivated the creation of new input devices that support both direct and indirect input, which we refer to as *mixed input mode* devices. In this manner, the two input modes complement each other in a shared input space, such that users can freely choose which mode to use when needed. In this manner, users' input is not limited to any given input mode. Additionally, users can use a rich set of enhancement techniques developed for both input modes. For instance, smartphone users can now use a mouse cursor to interact with small objects on the touch screen. They can also benefit from a rich set of cursor-enhancement techniques, which were originally developed for desktop applications [1, 2, 34, 39, 44, 77, 107].

Mixed input mode can also give user interface (UI) designers flexibility developing unique UI interfaces. For example, in mobile applications, icons are often made big (Figure 1.3 (left)) to mitigate the 'fat finger' challenge [128]. On the other hand, big icons are usually unnecessary in desktop

[1]The first computer mouse prototype was invented by Douglas Engelbart in 1963. In the late 1960's, IBM started to build the first touch screens.

applications because computer mice afford fine-grain control over small objects. Consequently, UI designers often use two completely different paradigms for applications on desktop and mobile platforms (Figure 1.3). Computers are now made small and portable. This brings a new challenge to traditional desktop applications, which are now expected to be used on touch-screen devices. The recently released Windows 8 utilizes a hybrid approach to support both touch and mouse input. To facilitate touch, Windows 8 uses a mobile-style application launcher, which contains big "touch-friendly" icons. On the other hand, Windows 8 still contains a traditional windows-style START menu, a counterpart of the application launcher for mouse input. While this approach provides a way to facilitate both touch and cursor input, it introduces additional development cost. *Mixed input mode* supports both input modalities, thus allowing designers to use a uniform UI paradigm across desktop and mobile platforms. Therefore, development costs can be largely reduced. More importantly, users do not need to spend extra time and effort to adapt to new UIs across desktop and mobile platforms.

Creating *mixed input mode* devices brings possibilities for new interaction techniques that have potential to improve how users interact with computing devices. In this thesis, we demonstrate a variety of new interaction techniques made available by *mixed input mode* devices we created (see Table 1.1). We demonstrate that such techniques can address a wide range of user interface challenges that users encounter in their daily tasks.

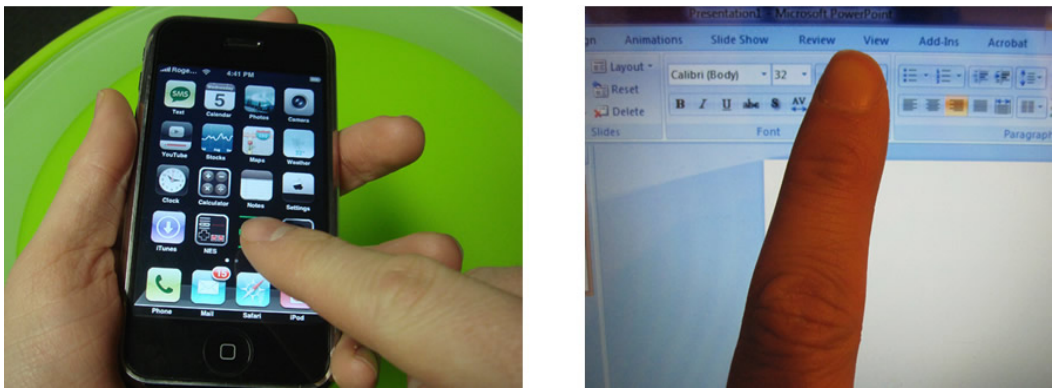

Figure 1.3: iPhone interface (left) and desktop application interface (right, MS Word 2007) has different design paradigms. The iPhone interface has big icons. The desktop interface often has small icons and cascade menus that are suitable to be used by a mouse cursor.

1.3 Challenges of Mixed Input Mode

Integrating direct and indirect input into one *mixed input mode* is not trivial for three major reasons. From a hardware perspective, most existing computing devices only support a single input mode, e.g., mouse for desktop computers and touch for mobiles. There is no well-defined guideline to inform the design of *mixed input mode* devices. Additionally, computers are becoming smaller, lighter, and more portable. This requires input devices to follow the same trend but without impacting user

experience when interacting with computers. From a software perspective, traditional software UIs are only designed to be used by a single input mode, e.g., either mouse or finger input. It is a challenge to make the general-purpose UIs friendly to both touch and mouse input without impacting users' experience of either of them. From the users' perspective, users are unfamiliar with *mixed input mode* devices. As a result, steep learning curves may be necessary to get users to learn to use these devices. Additionally, *mixed input mode* devices may involve a decision-making process from users. It is also a challenge to minimize cognitive and performance overhead introduced by this decision-making process.

1.4 Thesis Outline

In this thesis, we explore the design space of *mixed input mode*. We demonstrate *mixed input mode* is possible through proper hardware and software augmentation. We created three *mixed input mode* devices. Each device was created for a popular platform, including desktop, mobile, and always-available input (Table 1.1). These devices cover the most popular computing platforms, through which we demonstrate how *mixed input mode* can be used in any part of a user's daily life. We present our *mixed input mode* devices in order of increasing accessibility, desktop input being the least accessible and always-available input being the most accessible.

Table 1.1: Hardware prototypes and corresponding interaction techniques

User Interface / Input Platform	Hardware UI	Software UI
Mobile	LucidCursor	Dual-surface Input
Desktop	LensMouse	TouchCuts & TouchZoom
Always-available Input	Magic Finger	Magic Finger Interactions

In Chapter 2, we first review previous works in areas that are related to this thesis. In Chapter 3, we present LensMouse, a *mixed input mode* computer mouse that embeds a touch-screen display- or tangible 'lens' (Figure 1.4). Users interact with the display of the mouse using direct touch, while also performing regular cursor-based mouse interactions. We demonstrate some of the unique capabilities of such a device, in particular for interacting with auxiliary windows, such as toolbars, palettes, pop-ups, and dialog boxes. By integrating these windows onto LensMouse, challenges such as screen real-estate use and window management can be alleviated. In a controlled experiment, we evaluate the effectiveness of LensMouse in reducing cursor movements for interacting with auxiliary windows. We also consider the concerns involving view separation that results from introducing such a display-based device. Our results reveal that overall, users are more effective with LensMouse than with auxiliary application windows that are managed in either single- or dual-monitor setups.

We also present other application scenarios that LensMouse could support.

Figure 1.4: LensMouse prototype.

In Chapter 4, we present LucidCursor, a *mixed input mode* mobile device designed to augment interactions available through direct input with a behind-the-display cursor interaction. The device can be made to interact with the set of free fingers made available by holding the device with one hand. We developed a prototypical system on a PDA to which we affixed a wireless mouse. The mouse is mounted on the rear of the PDA with the optical sensor facing outwards (Figure 1.5). The system is designed to be used with one hand and can help prevent issues concerning occlusion and finger reach, concerns that are often cited when interacting with a handheld direct input device. Through several applications, we propose the benefits associated with LucidCursor. Our studies show that users welcome the novelty associated with the new technology and consider LucidCursor to be a welcome augmentation to PDA interactions.

Figure 1.5: LucidCursor prototype.

In Chapter 5, we present Magic Finger, a novel *mixed input mode* device worn on the fingertip to support always-available input (Figure 1.6). Magic Finger inverts the typical relationship between the finger and an interactive surface: with Magic Finger, we instrument the user's finger itself, rather than the surface it is touching. Magic Finger senses touch through an optical mouse sensor,

enabling any surface to act as a touch screen. Magic Finger also senses texture through a micro RGB camera, allowing contextual actions to be carried out based on the particular surface being touched. A technical evaluation shows that Magic Finger can accurately sense 22 textures with an accuracy of 98.9%. We explore the interaction design space enabled by Magic Finger and implement a number of novel interaction techniques that leverage its unique capabilities.

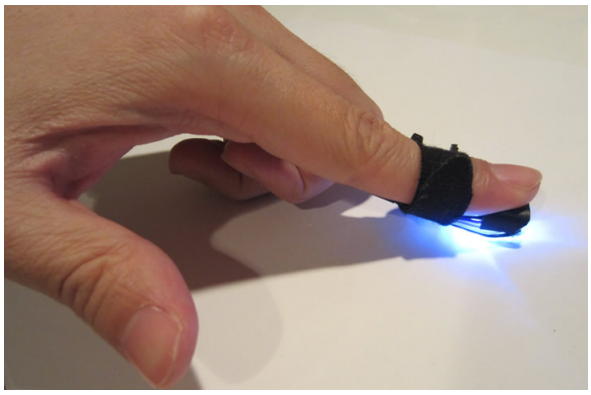

Figure 1.6: Magic Finger prototype.

Existing desktop applications were mainly designed to be used with a computer mouse. Therefore, they do not leverage the benefits of *mixed input mode*. In Chapter 6, we demonstrate the benefits of using touch as a complementary input modality along with the keyboard and mouse or touchpad in a laptop setting. To alleviate the frustration users experience with touch, we then design two techniques, TouchCuts, a single target expansion technique (Figure 1.7), and TouchZoom (Figure 6.5), a multiple target expansion technique. Both techniques facilitate the selection of small icons by detecting the finger proximity above the display surface and expanding the target as the finger approaches. In a controlled evaluation, we show that our techniques improve performance in comparison to both the computer mouse and a baseline touch-based target acquisition technique. We also present other application scenarios that our techniques support. Finally, in Chapter 7, we conclude by summarizing the contributions of this thesis and propose future research directions.

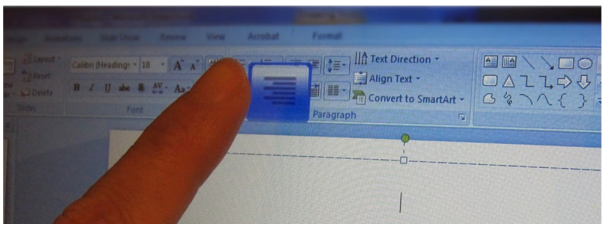

Figure 1.7: TouchCuts: expending a desired icon to facilitate finger input in MS Word 2007.

Chapter 2

Literature Review

In this chapter, we first review the related literature on direct and indirect input. We then review the previous work in the areas related to the three mixed input mode devices: LensMouse, LucidCursor, and Magic Finger. Finally, we review the previous work on target expansion techniques to inform the design of TouchCuts & TouchZoom.

2.1 Direct vs. Indirect Input

Direct input consists of direct interaction with the display device (i.e. touch screen), and indirect input is a mediated interaction using an intermediate device (i.e. touchpad or jog-dial). Indirect input requires users to translate physical distance of their hand movement to the virtual distance of a cursor movement. This spatial translation has been shown to be cognitively demanding, particularly for older users [27]. Direct input, on the other hand, does not require conscious mental translation because users' physical movement matches the display action. However, studies have shown that, when using a stylus, direct and indirect inputs were essentially equivalent for target selection tasks [39]. Direct input outperformed indirect input only when selecting a target using crossing, a selection technique not commonly used.

Although direct finger input is widely used in interactions with touch screens, studies have shown that people prefer using a mouse in some contexts [40, 92, 117]. This may be due to limitations, such as occlusion associated with direct input. Sears and Shneiderman [117] compared the performance of mouse input to finger input on a target selection task. Their results showed that finger input outperformed mouse input on targets of size ≥ 6.4mm. Their results also show that using a mouse does not necessarily lead to lower error rates. But Meyer *et al.* [92] demonstrate that a series of goal-directed tasks, that user performed better with indirect input than with direct input. They compared user performance between finger input, stylus, mouse, trackball, and mouse pen and found that a mouse interface resulted in the best performance while finger input resulted in the worst performance. Forlines *et al.* [40] stated that the mouse is the more appropriate interface for tasks like pointing and dragging. They compared mouse input to finger input on a selection and docking

task and found that selection time to be higher using the mouse than with the finger. They also found that using the mouse leads to faster docking time, which makes the overall trial time almost identical for both devices. Overall, absolute direct and relative indirect input offers numerous benefits, and there are situations that both input methods are needed by augmenting Interactive Tables with Mice & Keyboards. We propose taking advantage of both input types using *mixed input mode* interactions, and explore the design space of such input mode for typical tasks.

2.2 A Computer Mouse with Mixed Input Mode - LensMouse

The computer mouse is the established input device for manipulating desktop applications. Although arguably perfect in many ways, products and research have demonstrated the power in augmenting the mouse with new sensing capabilities [10, 26, 60, 64, 126] - perhaps the most successful being the scroll-wheel [60]. The fact that the mouse is so central in everyday computing interactions makes this a potentially rich design space to explore. Predominately these explorations have focused on expanding the input capabilities of the mouse [10, 26, 60, 64, 112, 126]. But why not looking at it output capabilities too?

Adding a touch-enabled display to a mouse follows a long-standing research trend in augmenting mice with powerful and diverse features. Many augmentations have been successful such as the scroll-wheel which is now indispensable for many tasks on today's desktops [60]. Other augmentations include extending the degrees-of-freedom of the mouse [10, 64], adding pressure input [26, 120], providing multi-touch input [96, 97, 126], supporting bi-manual interactions [10, 99], and extending the controls on a mice to a secondary device [99]. Our research prototype complements and extends this existing body of research by considering a rich set of both output and input functionalities on top of regular mouse-based input.

At one level, LensMouse can be considered as a small movable secondary display. There has been a great deal of research on the use of and interactions with multiple desktop monitors. We highlight some of this research, as it is relevant to my work.

2.2.1 Multi-display vs. Single Display Interactions

While a decade ago most computers were controlled with a single monitor, today dual-monitor setups are common, and are leading to novel multi-display systems [29, 61]. An additional monitor provides the user with more screen real estate. However, users seldom extend a window across two monitors. Instead, they distribute different tasks among monitors [46]. With more screen space available, the increased amount of mouse trips between displays becomes an issue. Therefore, users tend to put tasks involving frequent interactions on the primary monitor, while tasks that require fewer interactions, such as checking email updates, are delegated to the secondary monitor [46, 69]. A quantitative result by Bi *et al.* [19] confirmed that 71% of all mouse events in a dual-monitor setup occur on the primary display.

The primary benefits of a secondary display include allowing users to view applications, such as opening documents simultaneously or to monitor updates peripherally. For tasks involving frequent switching between applications, dual-monitor users are faster and require less workload than single-monitor users [75, 118]. The literature in multi-monitor systems also reveals that users like to separate the monitors by some distance instead of placing them in immediate proximity [46]. One may believe that this could lead to visual separation problems, where significant head movements are required to scan for content across both displays.

A study by Tan *et al.* [122] in which participants were asked to identify grammatical errors spread across two documents each on a different monitor, showed that distance between displays played a negligible role in task performance. They found only small effects of visual separation when the second display was placed far away from the main display. This result was also supported by an earlier study revealing that separating two monitors at a relatively long distance (the full width of one monitor) did not slow down users' access to information across both monitors [118]. These findings on the negligible effects of visual separation support the concept of having a separate auxiliary display, such as that on used in LensMouse, for dedicated information content. There is also the additional advantage of direct-touch interaction with the content on LensMouse.

Multi-monitor setups also present several drawbacks, the most significant being an increased amount of cursor movement across monitors resulting in workload overhead [112]. Alleviating this issue has been the focus of much research as we will highlight in the next section.

2.2.2 Minimizing Mouse Trips across Monitors

There are many ways to support cross-display cursor movement. Stitching is a technique commonly used by operation systems (e.g. Windows™ and MacOS™). It warps the cursor from the edge of one display to the edge of the other. In contrast, Mouse Ether [13] offsets the cursor's landing position to eliminate wrapping effects introduced by Stitching. Although both methods are effective [100], users still need to make a substantial amount of cursor movements to acquire remote targets. Object cursor [48] addresses this issue by ignoring empty space between objects. Delphian Desktop [6] predicts the destination of the cursor based on initial movement and rapidly 'flies' the cursor towards its target. Although these techniques were designed for single monitor conditions, they can be easily tailored for multi-monitor setups. Head and eye tracking techniques were proposed to position the cursor on the monitor of interest [7, 35]. This reduces significant mouse trips but at the cost of access to expensive tracking equipment. Benko *et al.* propose to manually issue a command (i.e. a button click) to ship the mouse pointer to a desired screen [17]. This results in bringing the mouse pointer close to the task but at the cost of mode switching. The Mudibo system [70] shows a copy of an interface such as a dialog box on every monitor, as a result it does not matter which monitor the mouse cursor resides on. Mudibo was found useful [71] but distributing redundant copies of information content across multiple monitors introduces problems such as distracting the user's attention

and wasting screen real estate. Ninja cursors [80] and its variants address this problem by presenting a mouse cursor on each monitor. This solution is elegantly simple but controlling multiple cursors with one mouse, adds an extra overhead of having to return the cursor to its original position on one monitor after using it for the other. ARC-Pad [88] is a novel touchpad, which combines absolute and relative cursor positioning, and allows seamless switch between the two control modes. Users can use absolute control to rapidly move across large distances, or clutch using relative control for fine manipulation. ARC-Pad shares a few similar functionalities with LensMouse. However, they differ largely in the sense that ARC-Pad does not support direct input.

2.3 Mixed Input Mode Mobile Device - LucidCursor

Handheld devices equipped with touch-sensitive displays are ubiquitous and considered by some to be a natural extension of our cognitive resources. Studies show that one-handed use is the preferred method for operating handheld devices [79]. With interactive tasks performed by one hand, users can free their other hand for other tasks that are commonly carried out in mobile contexts, such as carrying shopping bags or holding a bus handle. In this mode, the user grab the device and interacts with it using the thumb or other auxiliary fingers. This works well for small devices (e.g. cell-phones) equipped with physical interfaces, such as a physical keyboard. This is because most of the interactions happen in places that may be easily accessed by the thumb. For several reasons, interacting with one hand is still very difficult on many devices. Usually, the distance covered by the thumb is not sufficient to manipulate objects at extreme and opposite corners of the device (see Figure 4.5 on page 24). Furthermore, the "foot-print" of a thumb is significantly larger than that of other fingers, resulting in larger occlusions than with a stylus used with both hands [128]. Studies have also shown inaccuracies in selecting targets with a finger on a touch-display, specifically when the targets are small or in proximity to one another or small [121]. Finally, trajectory-based interactions such as scrolling a long document or highlighting a line of a sentence are difficult to operate using the thumb with one hand.

To resolve some of the problems with direct-touch input on handheld devices researchers have proposed a number of techniques. Behind-the-surface interactions [12, 121, 128] have resolved some of the complexities associated with finger occlusion. With this type of interaction, the user can interact with content on the screen that would normally be occluded. Techniques such as Shift [128] allow the use of large finger footprints for selecting small targets on a PDA (Personal Digital Assistant). It does this at the cost of an offset window that magnifies the target. However, recent studies have shown that Shift can also suffer from accessibility problems with the thumb [78]. With a few exceptions [78], most of the proposed techniques require both hands for operating the device or additional timeouts for invoking visual filters for selecting enlarged targets. Furthermore most of the developed techniques employ absolute direct input for object selection, thereby not taking advantage of what might be possible with relative input.

Relative input can assist in solving some of the intricate problems associated with one-handed interactions on a handheld device. A pixel-size cursor tip provides users with an easy and precise mechanism for pointing and selecting without occlusion. In addition, cursor-based enhancements [1, 2, 34, 39, 44, 77, 107], which are often seen on desktop PCs, can be applied to mobile devices to facilitate easier interactions. However, relative positioning introduces clutching, which can make interactions inefficient. Clutching can be alleviated by means of direct input on the front of the device. We hypothesize that one-handed interaction on mobile devices can be significantly improved by taking advantage of the best of both worlds: direct thumb input in the front to coarsely select targets, with relative cursor input for finer control. We refer to this mode of operation as Dual-Surface operations where the user can coordinate input from the front and back to perform a number of simultaneous tasks.

Several developments are related to my work. Most of them aims at addressing the issues concerning touch and single handed input.

2.3.1 Finger-based Touch Input

Finger based interactions suffer from occlusion. To address this problem, researchers have proposed cursor replacement techniques. The idea is to statically re-locate the cursor to a non-occluded position. Offset cursor [107] displays the cursor 0.5" above the finger. Fluid DTMouse [37] displays the cursor in the middle of two finger touch points so that it is not occluded by either finger. The major concern with this mode of interaction is that selection depends highly on the location of the targets. On mobile devices, edges and corners are problematic for static relocation techniques.

Shift [128] addresses some of these problems as it dynamically places a copy of the area occluded under the finger in a "shifted" callout at a non-occluded location. The callout contains a crosshair representing the contact location of the user's finger tip. By sliding the finger contact on the display, the user can fine-tune the pointer position and select the target of interest using a take-off gesture. Shift's callout was designed to be triggered only when necessary. When selecting large targets, a callout is not triggered, and selection takes place by simply tapping the screen. Similar to the static cursor relocation techniques, performance with Shift drops when targets are on an edge or at a corner of the mobile device. One drawback includes the use of a callout that can result in occlusion on small displays.

Olwal *et al.* [103] introduced Rubbing and Tapping to facilitate precise and rapid selection on touch screens. The rubbing technique activates a zoom-in action using a finger-tip gesture. The tapping technique enlarges the target by having the user touch near the target with one hand and tapping at a distance away on the screen with the other hand. Studies reveal that rubbing and tapping techniques can provide precise selection of target size ranging from 0.3mm to 4.8mm within 2 seconds [103]. Besides Tapping, other two-handed techniques, such as those presented in [18] have also been proven helpful for selecting small targets using bare fingers. However, bimanual input is

not a preferred operation mode for interacting with mobile devices [78].

To circumvent occlusion problems altogether, researchers have proposed interactions on the back of the display surface. LucidTouch [130] uses a back-mounted video camera to capture the user's finger motion. As a result, the system allows users to interact with the device using all their fingers. A circular cursor is assigned to each fingertip, allowing for precise interaction at the pixel level. The system was demonstrated with a set of tasks including object selection and dragging. The authors also explored the coordinated use of front of the display to perform basic navigation tasks. However, they did not provide any empirical support or report on any advantages of using both surfaces in a coordinated manner. Inspired by LucidTouch, NanoTouch [12] facilitated the creation of very small devices by offloading input to the back of the display. Back input was shown to make target selection in unreachable areas such as corners and edges possible and is less error prone than enhanced front techniques such as Shift. By having the users interact on the back of the device, the device can be reduced to a size as small as 0.3", without any impact on performance of selection tasks [12]. However, the authors of NanoTouch did not describe the use of coordinated front and back input. Additionally, back input was primarily absolute and not relative.

Although the discussed techniques provide solutions to occlusion from finger touch, the majority of the solutions were not devised or evaluated with one-handed input. Since one-handed input is popular and the preferred method for using a mobile device for many tasks [78], little is known about how such techniques work with one handed input.

2.3.2 One-handed Mobile Device Interactions

The study of one-handed interactions can be traced back to the 1960's [36]. Recent studies stem from formal and informal observations about usage patterns with mobile devices. Karlson and Berderson, conducted several in-situ observations which led to the conclusion that 74% of mobile users use one hand when interacting with their cellular devices [78]. The same observations and web surveys suggested that PDA users were inclined to use two hands, but expressed significant interest in using one hand if possible, suggesting the need for better tools to support one-handed interactions.

Applens and LaunchTile [79] were two of the earliest systems to support one-handed thumb use on PDAs and cell-phones. With AppLens users were provided with a simple gesture set, mimicking the up-down-left-right keys, to navigate with a grid of data values. LaunchTile allows access into the tabular data view by allowing users to press on soft buttons associated with an area of the grid. In a user study, users were found to perform correct gestures 87% of the time, suggesting that simple thumb gestures are memorable. However, they also found that users were reluctant to use gestures and preferred tapping. Their results also showed that the error rates were influenced by the direction of the gestures and the number of gestures available, suggesting a limit to the number of different gestures one should design.

Further investigations by Karlson *et al.* [78] on the biomechanical limitations of one-handed

thumb input have revealed that users do not equally interact with all areas of a device. Instead, user grip, hand size, device ergonomics, can influence dexterity and reach of the thumb. For example, right handed users have more limited thumb movement in the North-West and SouthEast direction than in the other directions. Additionally, regions of the device away from the right edge are more difficult to reach. These findings supported the development of Thumbspace [76]. Thumb-Space is based on users' natural inclination to touch the interface with their fingers when a stylus is not available. To facilitate thumb reach, users customize the workspace and shrink the entire workspace into a box. In an extensive study, users performed better at selecting targets further away using Thumbspace than other techniques. Used in conjunction with Shift [128], a technique for high-precision target selection with the finger, Thumbspace resulted in higher accuracy than using either technique alone.

Wobbrock *et al.* [135] conducted a series of experiments to study gestures of one-handed interaction with mobile devices. In their study, a USB touchpad was used to simulate a PDA. Finger position was mapped in absolute mode and was displayed via a cursor on a computer monitor. Their results suggest that, for a 1-D target selection task, input using the index finger on the back of a device had similar performance to input using the thumb on the front of the device in terms of task completion time. In their study, the target was always reachable by the fingers which may not reflect real cases when using one-hand. Furthermore, the study did not consider occlusion problems, as selection using the thumb was on a touchpad and consequently occlusion did not occur regardless of target size.

Escape [136] facilitates one-handed selection with selection using gestures. Each target on the screen is assigned a unique directional gesture for selection. A selection happens when the user presses an area near the target, and performs a motion in the direction indicated by the target. Escape improves the selection of small targets by not requiring tapping on a target. An experimental study showed that Escape could perform 30% faster than Shift on targets between 6 to 13 pixels wide while still maintaining a similar error rate. However, Escape's gesture-based selection limits its usage in applications where a large number of on screen targets are selectable. Furthermore, Escape's selection action may confound with other gesture-based interactions such as panning a map.

Overall, the results of these studies suggest that there is sufficient evidence of one-handed use of mobile devices, but we do not have a broad range of techniques to support this form of interaction. Furthermore, the design of one-handed interactions needs to be concerned with the size of the object, the place and position of control items, and the range of allowable gestures with the thumb. We utilized the recommendation offered in previous research works [77, 79, 104] to guide the design of the Dual-Surface interactions proposed in this thesis.

2.4 Finger as a Mixed Input Mode Device - Magic Finger

2.4.1 Always-available Input

Advanced sensing technologies have been used to help extend our ability to interact with computers from any-where, supporting always-available input [115].

One way to accomplish this is to extend the range of devices. SideSight [23] uses an array of proximity sensors to detect a users' finger movement on the side of a cell phone. Therefore, touch input is no longer limited to the small region of the touch screen. Similarly, Portico [8] and Bonfire [74] used cameras installed on a laptop to capture the input occurring around it. PocketTouch allows capacitive sensors to function through fabric [114]. While these devices extend the range of sensors, they also are anchored to a device.

Efforts have also been made to instrument a user's surrounding environment to enable touch. Touch [116] enables touch on everyday objects, including humans and liquids, while Scratch Input [53] recognizes gestures on larger surfaces, but each require a sensor to be connected to the touched object. Wimmer and Baudisch [134] demonstrated that touch gestures can be enabled on an arbitrary object by having the object connect to an electric pulse generator and a Time Domain Reflectometry sensor.

Camera based sensors have also been used to sense input. Wilson [132] demonstrated that a properly positioned depth camera can be used to sense touch on surfaces. In the LightSpace system [133], a small room becomes interactive by combining multiple depth cameras and projectors.

While the instrumentation of environments is promising, it is also limited. Alternatively, the human body can itself be instrumented to sense input. Interactive clothing is proposed in [66], but requires special instrumented clothing.

Many techniques enable body input of taps, postures, and whole body gestures. For example, Saponas *et al.* [115] used arm mounted electromyography sensors to sense pinch gestures made by different fingers. Cohn *et al.* [33] proposed a technique to sense whole-body gestures that utilizes existing electromagnetic noise from nearby power lines and electronics. Harrison *et al.*'s Skinput [59] used arm mounted bio-acoustic sensors to detect finger tap on the different locations of a users' forearm. Tiny sensors could also be embedded on [59], or implanted under [67] the user's skin. These techniques all open up new and interesting interaction opportunities, however they cannot sense detailed finger movements required to emulate touch input.

Several techniques have supported richer input on the human body [57]. The Imaginary Phone [49] uses a 3D camera to detect taps and swipes on a user's palm. OmniTouch [51] uses a depth-sensing camera and a micro projector to turn a user's palm into a touch-sensitive display. SixthSense uses a small projector and camera, worn in a pendant device, to enable touch interactions on and around the body [93]. While such techniques enable always-available input, the input is generally restricted to the body, and can suffer from occlusion. In contrast, we propose finger instrumentation,

enabling input on almost any surface.

2.4.2 Contextual Interactions

Magic Finger is able to recognize textures and use the in-formation to understand the environment, or object the user is interacting with, and perform associated contextual actions. While texture has been previously explored as an output channel for touch input [11, 55, 83] we are unaware of work where texture is used as an input channel.

Relevant work by Harrison and Hudson used multispectral material sensing to infer the placement of mobile devices, and discussed potential uses, such as turning off a cell phone when it is placed in a purse [52]. Alternatively a touch sensitive device can respond differently if it knows what caused the touch or who touched it [3, 58, 111, 138]. More broadly, research on contextual interactions in the ubiquitous computing literature is also relevant. We refer the readers to two comprehensive surveys [16, 28].

2.4.3 Augmenting User's Finger

Early work in augmenting the human finger lies in the virtual reality research, where data gloves are used to track the user's hand position in space [119]. Several projects also exist outside of the VR literature. Marquardt [87] augmented a user's finger with fiduciary markers to inform an inter-active tabletop what part of the finger was in contact with the surface. With Abracadabra [54], users wear a magnet on their finger to provide input above a mobile device. FingerFlux [129] also uses a small magnet on the finger, to simulate haptic feedback above an interactive tabletop. While these projects are all related in that they instrument the finger, none had the goal of supporting always-available input.

Logisys' Finger Mouse, a cylinder-shaped optical mouse that can be mounted on a finger, is a relevant commercial product [84]. It is used to control an on-screen cursor with a relative input mapping. Magic Finger extends this basic capability to also support absolute targeting, texture recognition, and fiduciary marker identification. Furthermore, with Magic Finger, we explore a smaller form factor, and a rich interaction design space beyond cursor control.

2.5 Expending Targets - TouchCuts & TouchZoom

Touch input is often considered intuitive and effective [40]. It is becoming a major input modality, especially on mobile devices. Recently, laptop manufacturers have started producing models equipped with touch-screens, allowing users to use their fingers to directly interact with their applications. However, legacy applications typically utilize a ribbon or tool palette, consisting of small tiled icons, and research has shown that targets of such size are difficult to select with the finger due to occlusion [128] and accuracy [104] problems.

While there is an abundance of work in creating new UI paradigms specifically for touch [95, 136], it is unlikely that the legacy applications used most often by users would ever go through such a transformation. Very limited work has looked at making general purpose UIs more touch friendly, without impacting their non-touch experience.

Ideally, software interfaces should seamlessly adapt to the input modality [25, 42]. With the use of proximity based sensing [31, 125, 63, 107], UI components could transition to a touch-optimized variant only when the user's hand approaches, and thus be left unaltered for mouse input.

2.5.1 Selection with Touch

Having finger input available in tandem with the mouse in a shared UI environment allows these devices to complement each other, thus providing numerous benefits. Researchers have found that the finger can be faster than a mouse when selecting targets greater than 6.4mm [117]. Nevertheless users can perform better with a mouse than with a finger in tasks requiring fine movement and selection [92] as also confirmed by Forlines *et al.* [40].

Clearly, the size of a UI component will affect touch input. Established guidelines suggest that the target size for touch should be greater than 10 mm [104]. However, smaller icons (e.g. 5mm) are still common in most of the current user interfaces. Finger based interactions also suffer from occlusion problems. Proposed solutions to alleviate this concern include the Offset Cursor [107] that relocates the cursor 0.5" above the finger; DTMouse, which presents the cursor between two fingers [37], Shift, which dynamically places a copy of the occluded area in a shifted callout [128], and Dual-Finger-Selections, which supports precise selections on multi-touch displays [18].

In Shift, users slide the finger contact on the display, to fine tune a crosshair cursor position and select the target using a take-off gesture. Shift's callout was designed to be triggered only when necessary. However, in most cases, the selection can be separated into 2 steps - invoking the callout and adjusting the cursor to make a selection. Escape [136], allows selecting small and tiled targets by assigning a unique directional gesture for selection. An experimental study showed that Escape could perform 30% faster than Shift on targets smaller than 5mm while still maintaining a similar error rate. Similarly, Sliding widgets [107] require users to slide on a target in a particular direction for selection. These two techniques require changing the traditional UIs to visualize directional clues, which may not be suitable for an interface also shared with traditional cursor input.

2.5.2 Mouse-based Target Expansion

Target expansion techniques for the mouse have also been widely researched [32, 90, 137]. These techniques have been found helpful for selecting a single target, by expanding the target as the cursor approach the target. However, in real-world situations, targets are often laid out in a tiled arrangement. The existing approaches for expanding tiled targets include expanding a single item that is predicted to be the desired target [91, 137] or expanding the predicted item as well as its immediate

neighbors [113]. Obviously, one of the limitations of these methods is the lack of guaranteeing that the desired target will be magnified due to prediction errors. Note that when using a mouse the user can still make a precise selection on an unexpanded target [113]. However with finger input, the user's intended target may be too difficult to select if it is not expanded.

Zhai *et al.* [137] suggest that if space permits it, all the targets in the view should be expanded. However, it is typical for ribbons or toolbars to span the entire display space, so there would not be room to expand all targets. An alternative would be to use some form of fisheye distortion [24], but these have been shown to be harmful to pointing and selection tasks [50, 137].

2.5.3 Touch-based Target Expansion

Despite its popularity in the target acquisition literature, target expansion has not been widely explored for touch. Olwal *et al.* [103] proposed using a rubbing gesture or a second finger tap to "zoom into" a small target before attempting to make a selection. Similarly, Benko *et al.* [18] propose two-handed techniques to enlarge the target to ease selection. These techniques have been proven helpful for selecting small targets using bare fingers. However, they have a common limitation that users need to explicitly indicate a target of interest and then expand it manually.

The more traditional, automatic target expansion has not been explored for touch. This is likely because target expansion relies on tracking the approach of the cursor towards the target, or in this case, the finger. This information is typically not available, as most touch devices today do not sense proximity information. However, advanced sensing technology has pushed the detection of finger motion to determine its proximity to the interaction surface [65, 110]. Major manufacturers (Mitsubishi, Primesense, and Cypress) have already announced such types of commercial systems or prototypes [31, 125, 85]. All of these have made target expansion feasible on small touch-screen devices, e.g. laptops.

Target expansion for touch has not been investigated thoroughly, and is promising in that it may improve touch selection without impacting the user interface for traditional cursor input. However, even in the mouse-based research, there are challenges surrounding target expansion yet to be addressed, such as how to apply expansion in tiled-target environments. Later in this thesis, we will demonstrate the added value of finger input in a keyboard/mouse setting. We will then present the designs and studies of the proposed techniques.

In summary, this chapter provides a comprehensive discussion on the topics related to our *mixed input mode* devices and their interaction techniques. In the rest of this thesis, we describe the details of the 3 novel devices: LensMouse, LucidCursor, and Magic Finger. We demonstrate the benefits of *mixed input mode* using a rich set of well-designed interaction techniques and a serious of carefully-designed studies.

Chapter 3

Mixed Input Mode in Desktop Environment - LensMouse

Table 3.1: Hardware prototypes and corresponding interaction techniques - LensMouse[3]

User Interface / Input Platform	Hardware UI	Software UI
Mobile	*LucidCursor*	*Dual-surface Input*
Desktop	**LensMouse**	*TouchCuts & TouchZoom*
Always-available Input	*Magic Finger*	*Magic Finger Interactions*

The goal of this project is to explore the *mixed input mode* in the desktop environment, where a mouse as an indirect input device has been used to interact with the virtual world for decades. We turn a mouse into a *mixed input* device, by allowing users to carry out direct touch input on its surface - via a touch screen mounted on the back of the mouse. An add-on benefit of such design is making the mouse an output device too, making it a powerful device which can support many novel interactions to facilitate users' daily work.

We start by briefly introducing the many benefits introduced by such a novel device. LensMouse allows users to view and interact with auxiliary digital content without needing to use a dedicated input device. We will demonstrate a variety of uses for such a novel device, including viewing and interacting with toolbars and palettes for an application or game, previewing web pages and folder contents, interacting with magnified primary screen con-tent, pop-up dialog boxes, and performing touch gestures.

Perhaps one of the main strengths of LensMouse is in dealing with auxiliary windows, such as instant notifications, color palettes, or navigation tools that occupy regions of the primary screen. While necessary for the user's task, they can consume precious real-estate and occlude parts of the user's primary workspace. This can result in additional window management overhead to move

[3]The video of LensMouse can be found at: `http://www.youtube.com/watch?v=og6_mQKCHTA`

or close the auxiliary window. Additionally, users have to divert their mouse cursor from their workspace over to the auxiliary window to interact with it. This task can be time consuming particularly when the user's display is large. Pop-up auxiliary windows can occasionally distract users, particularly when they are not immediately of use, such as with notifications from other applications on the desktop.

LensMouse can be used to "free up" these auxiliary windows from the primary screen, allowing them to be viewed and interacted with readily on a dedicated screen that is always within easy reach of the user. In this chapter, we present a controlled experiment that demonstrates the utility of LensMouse in dealing with the issues of auxiliary windows. The study demonstrates that users can interact and view the auxiliary display without extra cognitive or motor load; and can readily interact with on-screen content without significant mouse movements.

3.1 LensMouse Prototype

We created the LensMouse prototype by attaching a touch-enabled Smartphone (HTC touch) to the base of a USB mouse (Figure 3.1). As miniaturized touch displays become increasingly common and cheap, we can imagine such a display being integrated at a lower cost.

Figure 3.1: Illustration of LensMouse prototype. (a) base of a USB mouse; (b) HTC Smart phone; (c) Tilted base.

The LensMouse display is tilted toward the user to improve viewing angle. The display hosts a number of soft buttons, including left and right buttons and a soft scroll-wheel. The remaining space on the display allows application designers to place auxiliary windows that would normally consume screen real-estate on a desktop monitor.

Different types of auxiliary windows, such as toolbars, palettes, pop-ups and other notification windows, can be migrated to the LensMouse display. Another class of auxiliary windows that can be supported on LensMouse is overview + detail (or focus + context) views [106, 128]. Here the small overview window can be displayed on the LensMouse to provide contextual information. For example, map-based applications can dedicate an overview window or a magnifying lens on the LensMouse (Figure 3.2). Overviews have proven to be useful for a variety of tasks [68, 105].

Auxiliary windows displayed on the LensMouse are referred to as "lenses". LensMouse can

Figure 3.2: LensMouse prototype showing an overview map from a real-time strategy game.

incorporate several "lenses" simultaneously. To switch between different lenses, a "lens-bar" is displayed at the bottom of the display (Figure 3.2). Tapping on the lens-bar iterates through all the lenses. Finally, since the LensMouse display is separated from the user's view of the primary monitor, users are notified of important updates on LensMouse by subtle audio cues.

3.2 Key Benefits of LensMouse

In this section, we discuss how LensMouse addresses some of the challenges present with auxiliary windows, multiple monitors and other related work.

Reducing Mouse Trips

Auxiliary windows in particular palettes and toolbars often float in front of or to the side of the main application window. This maximizes the display area for the main application window, but also leads to mouse travel back-and-forth between windows. Operations with the LensMouse display can be performed by directly touching the mouse screen, eliminating the need to move the cursor away from the user's main working area to these auxiliary windows. For example, selecting a new color in a paint palette or changing the brush width can be done by directly touching the LensMouse screen without needing to move the mouse away from the main canvas (Figure 3.3).

Figure 3.3: LensMouse shows the floating color panel window of a drawing application. This reduces window management and minimizes on-screen occlusion.

Minimizing Window Management

Many applications such as graphics editors are often crowded with small windows hosting various widgets such as palettes, toolbars, and other dialog boxes. When the windows occlude important content, they need to be closed or relocated. The extra overhead in managing these windows can be minimized with LensMouse. LensMouse shows one window at a time. To switch to another, users simply tap on the lens-bar. As a result, window management does not incur mouse trips and only the current window of interest is presented to the user at any given time.

Reducing Occlusion

Certain applications rely heavily on auxiliary windows to relay feedback or other information content to their users. Designers typically resort to various strategies when displaying these windows, since these are known to occlude the main workspace and thus distract users from their main tasks [14, 73]. In many cases, such as with notifications, these windows will pop-up unexpectedly, thus taking the users attention away from their tasks. With LensMouse, such pop-ups can be displayed on the display of the mouse and users could be alerted of their appearance through a notification. By separating the unexpected pop-up windows from the main display, unnecessary distractions and occlusions are reduced.

Minimizing Workspace "distortions"

To improve task performance researchers have proposed a number of techniques that "distort" the user's visual workspace [90, 109]. For example, distorting target size, i.e. making it larger, can improve targeting performance [90]. This can be detrimental to the selection task if nearby targets are densely laid out [90]. Instead of "distorting" the visual workspace, with LensMouse the targets can be enlarged on the mouse display for easier selection with the finger, leaving the primary workspace unaffected. Other similar effects, such as fisheye distortions, can be avoided by leaving the workspace intact and simply producing the required effect on LensMouse. In a broad sense "distortions" could also include operations such as web browsing that involves following candidate links before finding the required item of interest. Instead, previews of web links can be displayed as thumbnail images on LensMouse, thus leaving the original web page as is. Other such examples include panning around maps to find items of interest, scrolling a document or zooming out of a workspace. Such display alterations can take place on LensMouse and thus leave the user's primary workspace intact. This has the benefit that users do not have to spend extra effort to revert the workspace to its original view.

3.3 Experiment

We have postulated that a primary benefit of LensMouse consists of reducing mouse trips by allowing the user to access contextual information with their fingertips. However, we also acknowledge

the potential drawbacks of having a display on LensMouse, such as the visual separation from the primary display, smaller screen size, and occlusion by the users' hand. Prior studies on multiple monitor setups do not show a significant impact of visual separation on task performance [71, 122]. Such studies were carried out with regular-sized monitors in vertical setups, e.g. monitors stood in front of users. This leaves it unclear whether visual separation has a significant impact on performance with LensMouse, where a smaller display, which is more susceptible to hand occlusion, is used almost in a horizontal setup.

To unpack these issues, we conducted a user study of the LensMouse. A specific goal of this study was to evaluate whether users are more effective at carrying out tasks when part of the interface is relegated to LensMouse. We evaluated LensMouse against single-monitor and dual-monitor conditions. In all conditions, the monitor(s) were placed at a comfortable distance from participants. In the single-monitor condition, the entire task was carried out on a single monitor. In the dual-monitor condition, the task was visually distributed across two monitors with each monitor slightly angled and facing the participants. The LensMouse condition was similar to the dual-monitor setup, except that the task was visually distributed across the main monitor and LensMouse display.

3.3.1 System Setup

The display on LensMouse had a size of 1.6×2.2 inch, and ran at a resolution of 480×640. We used 22" Samsung LCD monitors for both single and dual-monitor setups. Both monitors ran at a resolution of 1680×1050, and were adjusted to be roughly equivalent in brightness to the display on LensMouse. The study was implemented in Trolltech QT, and was run on a computer with 1.8 GHz processor and 3GB memory.

A pilot study showed no difference between the mousing capabilities of LensMouse and a regular mouse in performing target selection tasks. Therefore, we used LensMouse for all conditions in place of a regular mouse to remove any potential confounds caused by the mouse parameters. In the non-LensMouse conditions, participants clicked on LensMouse soft buttons to perform selection.

3.3.2 Task

To evaluate the various influencing factors, we designed a *cross-window* pointing task for this experiment. This task is analogous to that employed by users of text or graphics editing programs and is comprised of two immediate steps. The first step requires participants to click a button on the main screen to invoke a text instruction (Figure 3.4 (left)). Following the instruction, participants performed the second step by clicking one of the tool buttons in a tool palette window, corresponding to that instruction (Figure 3.4 (right)). Cross-window pointing is representative of common object-attribute editing tasks in which users must first select an object (by either highlighting or clicking it) and then visually searching for the desired action in an auxiliary window that hosts the available options. Examples of such a task include changing the font or color of selected text in Microsoft

Word, or interacting with the color palettes in Adobe Photoshop.

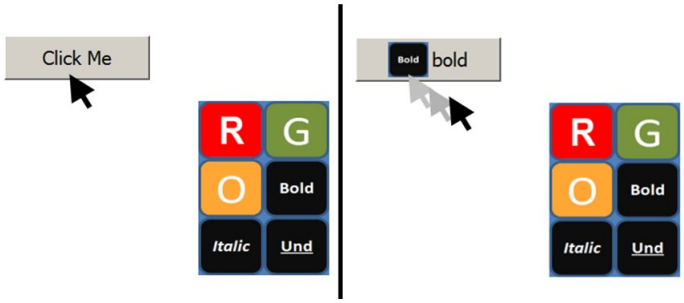

Figure 3.4: The instruction (right) *bold* showed only after participants successfully clicked the instruction button (left). It instructs to click on the bold icon in the tool palette window.

At the beginning of the task, an instruction button was placed in a random location on the main screen. Participants move the mouse cursor to click the button to reveal a text instruction. The text instruction is chosen randomly from an instruction pool, such as **Bold**, *Italic*, Und, etc. Following the instruction, participants picked the matching tool icon by clicking or tapping directly (in the LensMouse condition) on the tool palette. Upon selection, the next instruction button showed up in a different location. This was repeated over multiple trials and conditions.

3.3.3 Design

The experiment employed a 4×2 within-subject factorial design. The independent variables were *Display Type*: *Toolbox (TB)*, *Dual-monitor (DM)*, *Context Window (CW)* and *LensMouse (LM)*); and *Number of Icons*: *6 icons* and *12 icons*.

Toolbox (TB) - The Toolbox condition simulated the most frequent case in which auxiliary windows are docked in a region on the main display. In most applications the user has control of placing the window but by default these appear toward the edges of the display. In the Toolbox condition, we placed the tool palette at the bottom-right corner of the screen, such that instruction buttons would always be visible.

Dual-monitor (DM) - In the dual-monitor condition, the tool palette was shown on a second monitor that was placed to the right of the main screen showing the instruction buttons. To determine the location of the tool palette, we observed five dual-monitor users in a research lab at a local university, and found that most of them placed small application windows, such as instant messaging or media player windows, at the center of the second monitor for easy and rapid access. Tan *et al.*'s [122] study found no significant effect of document location on the second monitor. Based on these two factors, we thus placed the tool palette at the center of the second screen.

Context Window (CW) - Certain modern applications, such as Microsoft Word 2007, invoke a contextual pop-up palette or toolbar near the cursor when an item is selected. For example, in Word when text is highlighted, a semi-transparent "text toolbar" appears next to the text. Moving the

mouse over the toolbar makes it fully opaque and interactive. Moving the cursor away from the toolbar causes it to fade out gradually until it disappears and is no longer available. We created a Context Window condition to simulate such an interaction. Once the user clicked on an instruction the tool palette appeared below the mouse cursor and disappeared when the selection was completed. We did not use fade-in/fade-out transitions as this would impact performance times. We also maintained the physical size of the palette to be the same as in all other conditions.

LensMouse (LM) - In the LensMouse condition, the tool palette was shown using the full display area of the mouse. Unlike the other three conditions, participants made selections on the palette using a direct-touch finger tap gesture. On the LensMouse we can create palettes of different sizes. The literature suggests that for touch input icons less than 9mm can degrade performance [104, 128]. Based on the size of the display, we can have palettes containing up to 18 icons on the LensMouse. We however restricted the study to palettes of 6 and 12 icons, as these numbers would be the practical limits on what users could expect to have on a toolbar. The physical size of tool palette remained constant across all displays conditions (monitors and LensMouse).

In the cross-window pointing task, after the user's first click on the instruction button using the soft button on LensMouse, the rest of the display is partially occluded by the palm. We deliberately wanted this to happen as it resembles many real world scenarios in which the LensMouse display could indeed be occluded by the palm.

The Number of Icons was selected in a grid arrangement consisting of 2×3 or 3×4 icons (6 and 12 icons respectively). With 6 icons the targets were 20.5×18.7 mm and with 12 icons the targets were 13.7×14 mm.

In each trial, participants performed tasks in one of each $Display\ Type \times Number\ of\ Icons$ combination. The experiment consisted of 8 blocks, each consisting of 18 trials. The *Display Type* factor was partially counter balanced among participants. The experimental design can be summarized as: $4\ Display\ Types \times 2\ Number\ of\ Icons \times 8\ Blocks \times 18\ Repetitions \times 14\ Participants = 16128$ data points in total.

Dependent measures included the number of errors and the average task completion time. Task completion time was recorded as the time elapsed from a click on the instruction button to a click on the corresponding icon on the tool palette. An incorrect selection occurred when the participant clicked on the wrong icon in the palette.

3.3.4 Participants

Fourteen participants (10 males and 4 females) between the ages of 21 and 40 were recruited from a local university to participate in this study. Participants were daily computer users. All of the participants were right-handed users.

3.3.5 Procedure

At the start of each trial, an instruction button was placed randomly in one of three predefined regions identified by the distance to the bottom-right corner of the display where the Toolbox is placed, and also near where LensMouse is likely to be placed, as shown in Figure 3.5. The three distances were selected such that the instruction item could be either close or far away from LensMouse. The items in the Near region were between $168 \sim 728$ pixels away from the bottom-right corner; the Middle region $728 \sim 1288$ pixels; and the Far region $1288 \sim 1848$ pixels. This would allow us to test the impact of visual separation if it was present.

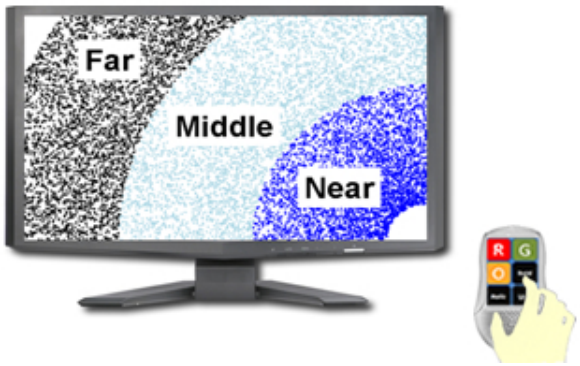

Figure 3.5: Instruction buttons were placed in 3 regions. The regions demarcated areas based on the distance to the bottom-right corner of the screen.

Prior to starting the experiment, participants were shown the LensMouse prototype and its features, and were also allowed several practice trials in each condition. Participants were asked to finish the task as fast and as accurately as possible. A break of 15 seconds was enforced at the end of each block of trials. The entire experiment lasted slightly under 60 minutes. Participants filled out a post-experiment questionnaire upon completion.

3.3.6 Results and Discussion

The collected data was analyzed using a repeated measure ANOVA test and Bonferroni corrections for post-hoc pair-wise tests.

Task Completion Time

Task completion time was defined as the time taken to make a selection in the tool palette after an instruction button was clicked. The analysis of completion time does not include trials in which an error was made during the tool selection. The overall average completion time was 1245ms. ANOVA yielded a significant effect of *Display Type* ($F_{3,39} = 50.87$, $p < 0.001$) and *Number of Icons* ($F_{1,13} = 10.572$, $p < 0.01$). Figure 3.6 shows average completion time for each *Display Type* by *Number of Icons*. We found no interaction effects on $DisplayType \times Numberof Icons$ ($F_{3,39} = 0.262$, $p = 0.852$).

Performance with LensMouse (1132ms, s.e. 6.5 ms) was significantly faster than with the Dual-monitor (1403ms, s.e. 6.4ms) and the Toolbox (1307 ms, s.e. 6.5ms) conditions. Interestingly, post-hoc pair-wise comparisons showed no significant difference between the Context Window (1141ms, s.e. 6.4ms) and LensMouse ($p = 0.917$). As expected, it took participants longer to select from the palette of size 12 (1292ms, s.e. 4.6ms) than from the palette of size 6 (1200ms, s.e. 4.6ms). This is not surprising considering that icons were smaller on the palette of 12 items.

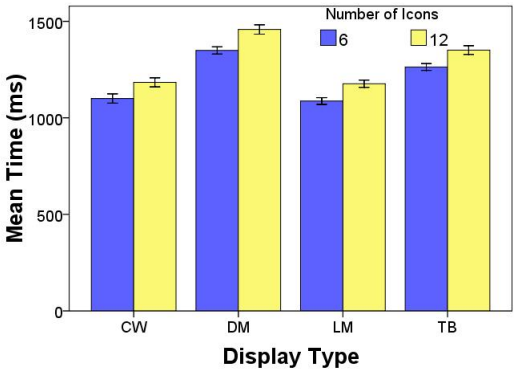

Figure 3.6: Task completion time vs. Display Type and Number of Icons. Error bars represent $+/-2$ standard error.

As expected, techniques requiring significant mouse trips, such as the Dual-monitor and Toolbox, took users longer to finish the task. This is consistent with our understanding of targeting performance based on Fitts' Law [86]. LensMouse performed as fast as the Context Window. This clearly shows that even though LensMouse may be affected by visual separation, this was compensated by the advantage of minimizing mouse trips, and resulted in a net gain compared to the Toolbox and Dual-monitor setups.

Number of Errors

Errors were recorded when participants made a wrong selection in the palette. The overall average error rate was 1.4%. Analysis showed no main effect of *Display Type* ($F_{3,39} = 1.708$, $p = 0.181$) or *Number of Icons* ($F_{1,13} = 0.069$, $p = 0.797$) on error rate. Neither did we find any interaction effects for $Display\ Type \times Number\ of\ Icons$ ($F_{3,39} = 1.466$, $p = 0.239$). All techniques scored error rates that were lower than 2%. Even though not statistically significant, LensMouse exhibited more errors (1.7%, s.e. 0.02) than the other conditions. This was followed by the Dual-monitor (1.5%, s.e. 0.02), Context Window (1.2%, s.e. 0.02), and Toolbox (1.2% s.e. 0.02). The error rate on LensMouse was largely a result of imprecise selection with fingertips, a known problem for touch displays [128], which could be alleviated in both hardware and software.

Learning Effects

We analyzed the learning effects captured by participant performance for each of the display types.

There was a significant main effect on task completion for Block ($F_{7,91} = 6.006$, $p < 0.01$) but there was no significant interaction effect for $Block \times Display\ Type$ ($F_{21,273} = 1.258$, $p = 0.204$) or for $Block \times Number\ of\ Icons$ ($F_{7,91} = 0.411$, $p = 0.893$). As can be seen in Figure 3.7 there is a steeper learning curve for LensMouse and Context Window techniques. Post-hoc analyses showed that with LensMouse, significant skill improvement happened between the first and the third block ($p < 0.001$). But there was no significant learning after the third block (all $p > 0.31$). Interestingly, a similar learning pattern was found with the Context Window technique. On the other hand, task completion time decreased almost linearly with the Dual-monitor and Toolbox techniques. But we observed no significant skill improvements (all $p > 0.75$).

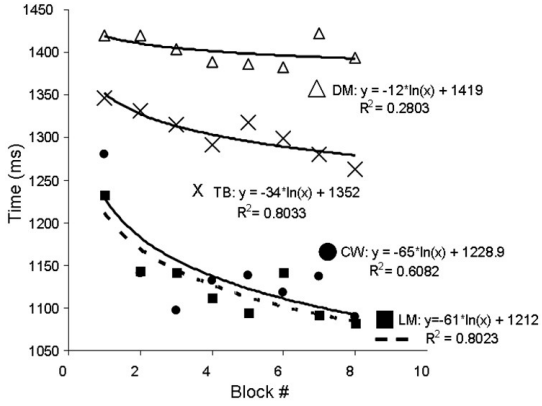

Figure 3.7: Learning effects: Task completion time vs. block number.

Although the results of LensMouse and Context Window are similar, the learning effects were markedly different. The learning effects taking place in the Context Window condition were in part due to getting familiar with different button/icon locations to reduce visual search time. However, learning effects with LensMouse were mainly due to the process of developing motor memory skills through finger selection. This was apparent in our qualitative observations - participants were no longer looking at LensMouse display after the 4^{th} or 5^{th} block of trials.

Effects of Visual Separation

The experimental design accounted for visual separation effects that would possibly be present with LensMouse. We performed the analysis by looking at targeting performance when targets were in one of the three regions Near, Middle, and Far. There was no main effect of performance time for visual separation with LensMouse ($F_{2,26} = 1.883$, $p = 0.172$). Nor did we find any main effect of visual separation on number of errors ($F_{2,26} = 3.322$, $p = 0.052$). Even though not statistically significant, LensMouse exhibited more errors in the Middle (2.2%, s.e. 0.04) and Far (1.9%, s.e. 0.04) region than in the Near region (1%, s.e. 0.04).

Subjective Preference

The post-experiment questionnaire filled out by all the participants show that the users welcomed

the unique features provided by LensMouse. They also indicated a high level of interest in using such a device if it were made commercially available. All scores reported below are based on a 5-point Likert scale, with 5 indicating highest preference.

The participants gave an average of 4 (5 is most preferred) to LensMouse and Context Window as the two most preferred display types. These ratings were significantly higher than the ratings for Toolbox (avg. 3) and Dual-monitor (avg. 2). We noticed more people rating LensMouse at 5 (50%) than the Context Window (36%). The same trend in average scores were obtained (LensMouse: 4, Context Window: 4, Toolbox: 3, and Dual-monitor: 2) in response to the question: "how do you perceive the speed of each technique?" This is consistent with the quantitative results described in the previous section. Additionally, participants found LensMouse to be easy to use (3.9, 5 is easiest). The score was just slightly lower than the Context Window (4.3) but still higher than the Toolbox (3.1) and the Dual-monitor (2.7). Finally, the participants gave LensMouse and Context Window an average of 4 (5 is most control) in response to "*rate each technique for the amount of control available with each*". The rating was significantly higher than the rating of Toolbox (3) and Dual-monitor (3).

Overall, 85% of all the participants expressed the desire to use LensMouse if it was available on the market. In addition, 92% of the participants felt that the display on LensMouse would help them with certain tasks they performed in their work, such as rapid icon selection. Finally, 70% of the participants saw themselves using LensMouse when doing tasks in applications such as MS Word, PowerPoint, or even browsing the Internet. It is worth noting that the current LensMouse is just a prototype, and could be significantly improved in terms of ergonomically. We will discuss this issue later in the paper.

Preliminary Qualitative Evaluation with a Strategy Game

In addition to the quantitative experiment, we also have begun to qualitatively test the LensMouse prototype with Warcraft 3, a real-time strategy game. While by no means a full study, we used this opportunity to distill preliminary user feedback of using LensMouse with a popular commercial software product.

Three computer science students, all with at least 50 hours of experience playing Warcraft 3, were invited to play the game using LensMouse for forty-five minutes. With LensMouse we implemented the ability to navigate around the game map using the overview (Figure 3.2). Users could simply tap on the overview with LensMouse to navigate the overall game map. This has the effect of reducing mouse movement between the main workspace and the overview window.

Users required a brief period to get familiar with LensMouse and to having a display on top of a mouse. User feedback supported the findings discussed earlier. Players were able to navigate easily around the game map and found LensMouse "exciting". The gamers immediately saw an advantage over their opponents without it. Upon completing the game, participants offered useful suggestions

such as including hotkeys to trigger in-game commands or to rapidly view the overall status of the game. Such features can be easily implemented in the prototype and would give players who have access to LensMouse a significant advantage over those without the device.

3.4 Discussion

The study shows that users can perform routine selection-operation tasks faster using LensMouse than using a typical toolbox or dual-display setup. The performance of LensMouse is similar to the performance of the context window, a popular technique for facilitating the reduction of mouse travel. However, the context window has several limitations making it less suitable in many scenarios. First, the context window is transient and needs to be implicitly triggered by the user through selection of some content, thus making it unsuitable for hosting auxiliary windows that need frequent interaction. Second, a context window may occlude surrounding objects, causing the user to lose some of the information in the main workspace. For this reason, context windows in current commercial systems such as MS Word are often designed to be small and only contain the most frequently used options. In comparison, LensMouse provides a persistent display with a reasonably large size, thus minimizing these limitations, and with the added benefit of direct-touch input and rapid access.

Prior to our study we speculated that the practical benefits of LensMouse would be severely challenged by visual separation. The results reveal that the minimal (almost negligible) effect of visual separation is compensated by the advantages of direct touch on the LensMouse, and results in a positive net gain in performance. However, we did not assess the full impact of visual separation which may depend on many factors, including the complexity of the information shown on the LensMouse, the efforts of switching between tool palettes, and the number of eye gaze and head movements required for certain tasks. Therefore, designers should strike a balance between the need for showing rich information content and minimizing the impact of visual separation.

Additionally, the benefits of direct touch also outweigh the potential cost of occluding the LensMouse display with the palm of the hand. Note that in our task users were first required to click on the instruction button using the left mouse button. This would result in partially occluding the LensMouse display and consequently the palette. Despite this concern, hand occlusion did not affect overall performance, nor did users report any frustration from this effect.

It is also worth noting that our prototype demonstrates the essential features of LensMouse, but is ergonomically far from perfect. Presently, the display could become partially covered by the palm, requiring users to occasionally move their hand to one side. However, this limitation can be easily alleviated through better ergonomic design. One possible solution is to place the display in a comfortable viewing position (using a tiltable base) with left and right mouse buttons placed on either side of the mouse (Figure 3.8a). Another solution is to place the display and the buttons on different facets of the mouse (Figure 3.8b). Such a configuration would allow users to operate

LensMouse like a normal mouse, while still keeping users' fingers close to its display. Multi-touch input [126] could be performed easily using thumb and index finger. Furthermore, a joystick-shape LensMouse (Figure 3.8c) could allow users to operate the touch screen using the thumb.

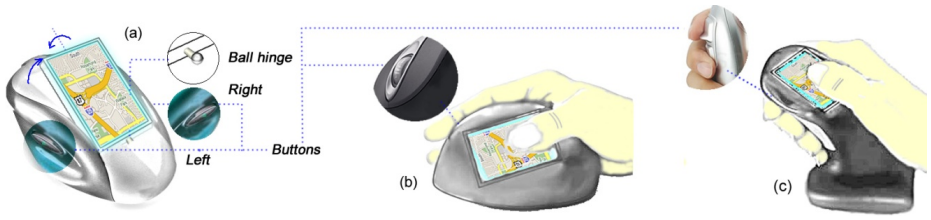

Figure 3.8: Various possibilities for an ergonomic design of LensMouse. (a) a rotatable display allowing most freedom in viewing position; (b) having the display oriented toward the user, but limited by handedness; (c) on other devices such as a joystick.

Direct touch input on the LensMouse affords a lower resolution than that with relative cursor control. However, many of the tasks on LensMouse do not require pixel-level operations. When such operation were required, techniques such as Shift [128] could be employed to alleviate the fat-finger problem.

Finally, the size of the display on LensMouse is relatively small. This could limit the number of controls we can place on this device and possibly make it difficult for applications requiring larger windows. However, we could consider supporting panning operation on LensMouse by using finger gestures to accommodate more content.

3.5 Beyond Auxiliary Windows

In addition to resolving some of the challenges with auxiliary windows, LensMouse may serve many other purposes:

Custom Screen "shortcut"

In addition to migrating predefined auxiliary windows to LensMouse, the user may take a "snapshot" of any rectangular region of the primary screen, and create a local copy of the region on LensMouse, as in WinCuts [123]. Any finger input on LensMouse is then piped back to that screen region. By doing so, the user can create a custom "shortcut" to any portion of the screen, and benefit from efficient access and direct input similar to that shown in the experiment.

Preview Lens

LensMouse can also serve as a means to preview content associated with a UI object without committing a selection. Figure 3.9a shows how such a preview lens can be used to reveal the folder's content on the LensMouse by simply hovering over the folder icon. This could aid search tasks where multiple folders have to be traversed rapidly.

See-through Lens

Figure 3.9: Special lenses:(a) Previewing folder content. (b) Seeing through overlapping windows.

Another use of the LensMouse is for seeing through screen objects [21], e.g. overlapping windows (Figure 3.9b). Overlapping windows often result in window management overhead spent in switching between them. We implemented a see-through lens to allow users to see "behind" overlapping windows. In the current implementation, users have access only to content that is directly behind the active window. However, in future implementations the user will be able to flick their finger on the display and thus iterate through the stack of overlapping screens.

Hybrid Pointing (Pointing with Mixed input Mode)

LensMouse integrates both direct-touch input and conventional mouse cursor pointing. This offers a unique style of hybrid pointing that we are keen to investigate further. In one demonstrator, we have built a prototype that shows a magnifying lens that amplifies the region around the cursor (Figure 3.10a). The user can move the LensMouse first to coarsely position the cursor near the target, and then use the finger to select the magnified target on the LensMouse display. For farther away targets that are cumbersome to reach, LensMouse shows an overview of the whole workspace (Figure 3.10b). By touching the finger on the overview, the user can directly land the cursor in the proximity of the target, and then refine the cursor position by moving LensMouse. By combining the absolute pointing of touch with the relative pointing of the mouse in different manners, there is the potential for new possibilities in selection and pointing.

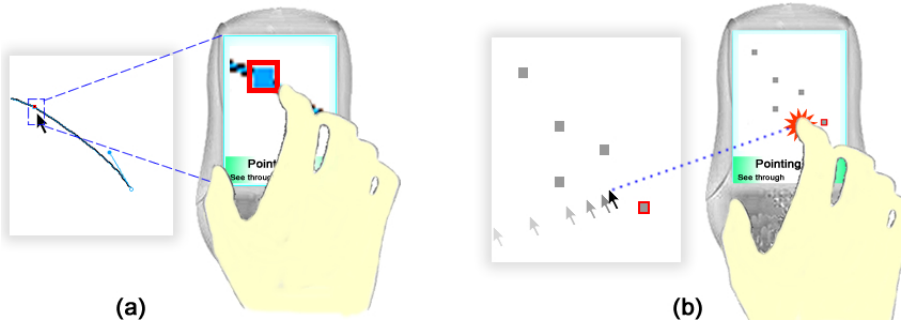

Figure 3.10: Absolute + Relative pointing. (a) For small targets, and (b) to cover long distances.

Gestural Interaction

With the addition of touch input, the user can apply various finger gestures to interact with the object under the cursor such as rotating and zooming. To rotate, the user places the cursor upon the object to be spun, then makes a circular finger motion on the mouse screen. Similarly, users can zoom into a specific location by pointing the cursor in that region and sliding the finger on a soft zoom control. The dual input capability (mouse and touch) effectively eliminates the need for mode switch between pointing and gesturing, as common in many other systems. As multi-touch becomes more broadly available, we can envisage more elaborate touch gestures being supported.

Private Notification

Notifications of incoming emails or instant messages are sometimes distracting and may potentially reveal private information to others, especially when a system is connected to a public display (i.e. during a presentation) [72]. By setting LensMouse into private mode, messages that would normally appear on-screen will be diverted to the private mouse display. While not implemented in the current prototype, users could use simple, pre-configured gestures on the mouse screen to make rapid responses, e.g. "I am busy" [10].

Custom Controller

LensMouse can support numerous types of custom controls including soft buttons, sliders, pads, etc. For example, to navigate web pages, we provide forward and back buttons, and to browse a long page we implemented a multi-speed scrollbar. As a custom controller, LensMouse can provide any number of controls that can fit on the display for a given application. In future work, the implementation will include the ability to automatically open a set of user-configured custom controls for a given application. For instance, upon opening a map-based application, the LensMouse could provide different lenses for pan + zoom controls, overviews, or other relevant controls.

Fluid Annotation

Annotating documents while reading can be cumbersome in a conventional desktop setup due to the separate actions of selecting the point of interest with the mouse and then typing on the keyboard [94]. LensMouse could support simple annotations (such as basic shapes) in a more fluid way. Users can move the mouse over the content of interest, and annotate with their fingertips.

3.6 Summary

In this Chapter, we presented LensMouse, a novel *mixed input* device that can serve as auxiliary display - or lens - for interacting with desktop computers. We demonstrated some key benefits of LensMouse (e.g. reducing mouse travel, minimizing window management, reducing occlusion, and minimizing workspace "distortions"), as well as resolving some of the challenges with auxiliary

windows on desktops. A controlled user experiment reveals a positive net gain in performance of LensMouse over certain common alternatives. Subjective user preference confirms the quantitative results showing that LensMouse is a welcome addition to the suite of techniques for augmenting the mouse.

Chapter 4

Mixed Input Mode in Mobile Environment - LucidCursor

Table 4.1: Hardware prototypes and corresponding interaction techniques - LucidCursor[4] and Dual-surface Input

User Interface / Input Platform	Hardware UI	Software UI
Mobile	LucidCursor	Dual-surface Input
Desktop	*LensMouse*	*TouchCuts & TouchZoom*
Always-available Input	*Magic Finger*	*Magic Finger Interactions*

The prototype system consists of a Pocket PC (PPC) and an optical sensor, which is mounted on the back of the PPC. Direct finger or stylus input can be made through the touch screen. Indirect mouse input is made through the optical sensor, which senses the movements of user's fingertip (see Figure 4.1). In the indirect input mode, an arrow-style cursor is displayed and the cursor movements can be controlled by moving the index fingertip over the optical sensor or by gliding the device on a flat surface.

4.1 Physical Components and Configuration

The system consists of a Dell AXIMTM Pocket PC (PPC) augmented by a core mechanics of a Stowaway™ Bluetooth mouse. The Stowaway mouse can be used on PPCs using blue-tooth connection. A regular arrow-style cursor is available on the handheld device. We remove the casing of the mouse so that the resulting surface is flat. We taped the rest of the mouse, which includes an optical sensor, a main board, and two batteries, on a base plastic card (see Figure 4.2) and mounted the entire set on the back of the PPC using Velcro hooks. The optic sensors are facing in the opposite

[4]The video of LucidCursor can be found at: `http://www.youtube.com/watch?v=2f7mUZTzcRw&feature=player_profilepage`

Figure 4.1: With LucidCursor, users can move the index fingertip over the optical sensor to control the movement of a cursor.

direction of the handheld screen. We placed the mouse as close as possible to the top of the PPC to make it conformable for the users to manipulate the cursor with the index finger.

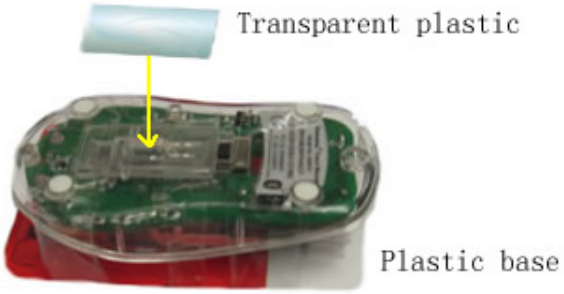

Figure 4.2: The architecture of the sensing system.

4.2 Improve Tracking Performance

The optical sensor of a mouse is designed to detect hand movements on a planar surface. Therefore, simply moving the bare fingertip over the optical sensor leads to unreliable sensor tracking. This is due to the fact that the optical sensor is sensing the uneven surface of the fingertip. Furthermore, by sliding the finger over the optical sensor can cause the fingertip to fall into the hole, leading to jitter and unexpected cursor movements. As was done in Soap [15], we improved the tracking performance by covering the sensor hole with a transparent plastic sheet (see Figure 4.2). This prevents the cutaneous fingertip tissue from "falling" into the hole. Moreover, a relative planar surface of the fingertip is formed when the user presses his/her fingertip against the covering sheet. The sensing mechanism thus works in a similar manner to a touchpad. This simple improvement significantly enhanced the reliability and fluidity of the tracking system. An informal user study showed that users are able to use the system to perform basic pointing and dragging tasks with less error than without the sensor covering sheet.

4.3 Implementing Mouse Button Clicks

With an indirect input mechanism, it is necessary to develop a method to facilitate button clicks and selection. LucidCursor is implemented to allow the user flexibility with three options, to trigger the left-mouse button click. The first option provides access to selection if the user presses one of the hardware buttons located on the left side of the PDA (see Figure 4.3a). We expect right-handed users to click the button with the index finger index finger and left-handed users to click with the thumb (see Figure 4.3b).

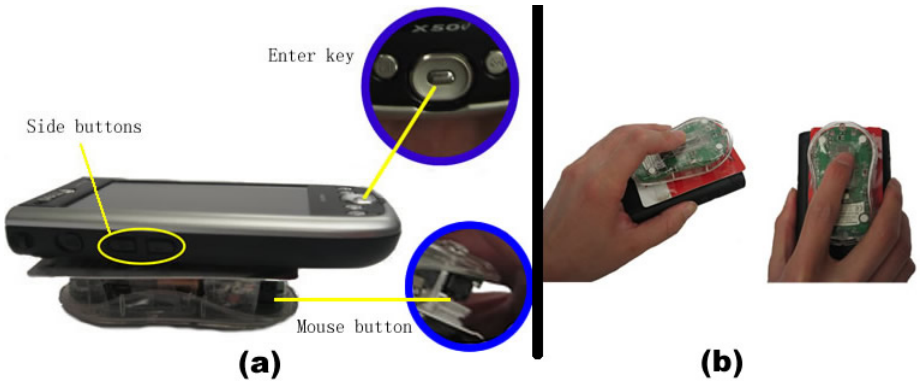

Figure 4.3: (a) Left mouse click can be triggered from three locations. (b) The system in use (see from the bottom).

The second option gives access to the "Enter" key, which is located at the center of the jog-dial on the PDA (see Figure 4.3a). Since the "Enter" key is located on the front of the device we expect users to click the button with the thumb. We believe this mechanism is not easily accessible and may not be used very frequently. The third access to selection is to take advantage of the original left-button mouse click. Since the mouse is inverted on the rear of the PDA, the buttons of the mouse are directly touching the casing of the PDA. To make the button clickable we filled the space between the button and the PDA case using a piece of plastic. The plastic is flexible so that it is easy for right handed users to click a button by pressing on the corner of the mouse, using the base of the index finger (see Figure 4.4). In contrast to the first two options, the user can perform pointing and selecting using one finger, which makes the interaction smoother and faster. Left-handed users can benefit from this by simply making the right button trigger a left click. This can be done in LucidCursor via software.

4.4 Test Applications

We developed three test applications to assess the usability of LucidCursor. All of which are commonly requiring the use of both hands; the non-dominant hand to hold the device and the dominant hand for interaction. We wanted to assess the benefit of using cursor input with respect to one-handed interaction, and therefore asked our test participants to only use one hand for the evaluation.

Figure 4.4: The user navigates the cursor to the target location (left). Releases the index fingertip, and clicks the mouse button by pressing on the corner of the mouse, using the base of the index finger (right).

We performed an evaluation and report our results after describing each of the three tasks. Five participants, all right-handed users volunteered for our evaluation.

4.4.1 Pointing and Selection

Pointing and selection are common tasks that require access to the entire screen. The task is very difficult to accomplish with one hand since the thumb cannot reach the entire screen area (see Figure 4.5). Furthermore such a task typically requires precise selection. We designed a simple Fitts' Law task which required participants to point and select at targets either 4.5mm or 2mm wide that were randomly located on the screen.

Figure 4.5: It is common that targets to be selected are located in places that are difficult to reach by the thumb (right).

4.4.2 Map Browsing

The map browsing task required users to perform a panning operation. To start panning, the users can click anywhere on the screen. It is possible to pan by using one hand as long as the user keeps the operation within the region that is reachable by the thumb. Users may have difficulties to perform

diagonal thumb movements in the North-West/South-East directions (for right handed users, and the reverse directions for left-handed users) [78]. As a consequence, the performance of thumb panning can be poor. We asked the participants to use the prototype to pan a map freely in all directions.

The back mounted optical sensor allows the user to operate the device by gliding it onto a flat surface, like a mouse (Figure 4.6). We asked the participants to repeat the previous tasks by sliding the device over a table and a wall, just as they would with a mouse. Various mappings are possible for cursor placement. We selected the following: the cursor moved in the direction opposite of the device movement. For example, if the user glided the device to the left the cursor moved to the right. This task was particularly different from the other two, as it allows users to operate the PPC as mouse, thus facilitating a number of different types of interaction including those similar to a peephole displays.

Figure 4.6: Panning a map by moving the device against a table. The device was moved in the NE direction, and the map was panned in the SW direction.

4.4.3 Group Selection

Selecting a group of targets is a common task bearing characteristics of pointing and/or dragging. Like pointing and selection, group selection is challenging with single-handed operation due to far reaching corners and occlusions. The test application displayed four target objects in pre-defined locations on the reachable and unreachable portions of the display. When operating on the front of the display, the participants were asked to select the group by tapping on each target. The size of the targets was small enough to be occluded by the thumb so that the participants needed to use the Shift [128] technique for precise selection. When operating on the back of the display, the participants were asked to select the group using the common rubber banding technique.

4.4.4 Preliminary Results

All participants were able to successfully complete the tasks. This suggests that cursor input behind-the display can become a suitable augmentation for one handed interactions. Interestingly, we observed that for different tasks, participants chose the most suitable way to trigger clicking for that task. In the pointing and selection task, the participants mainly used the original mouse button for clicking. In the map browsing and group selection task, the participants mainly used the side buttons

for clicking. The Enter key was rarely used in all of the three tasks. This suggests that future work needs to determine the most appropriate selection mechanisms for cursor input behind the screen.

For the group selection task, it took the participants an average of 5425ms to select a group of four targets by using tapping with Shift. This is not significantly different from the selection time (5437ms) of using the elastic band behind the display. The participants perform the task with tapping slightly faster than with behind-the-display cursor interaction. One possible explanation for this is the number of targets being tested was too small. We expect that with an increase in the size of the group, requiring larger movements, behind-the-display cursor interaction could outperform tapping.

Additionally, for symmetric groups of objects, cursor input with relative position mapping would be beneficial particularly for regions that are not reachable with the thumb. Overall, the results of the preliminary studies show that for certain tasks, there could be a significant advantage for behind the-display cursor input. This interaction form may prove to be particularly useful for situations where one-handed interaction is preferred. We believe a formal study will help us gain a better understanding of this form of augmentation.

4.5 Dual-surface Input

Motivated by the promising results of the preliminary results, we built a better designed prototype. The new prototype embeds a touchpad onto the rear surface of a PDA and allows users to interact with the PDA with the index finger that is free from gripping the device with one hand (Figure 4.7). The device supports Front (via thumb), Back (via cursor), and Dual-Surface (via both thumb and cursor) interaction. We run two studies to measure the performance of these 3 types of input. The results of the first experiment reveal a performance benefit for targeting with Dual-Surface input. A second experiment shows that Dual-Surface input can take advantage of virtual enhancements that are possible with relative input to perform more complex tasks such as tunneling, which requires user to navigate, or steer, the mouse cursor through a 2-dimensional tunnel, a common task in selecting an item in a cascade menu.

4.5.1 Dual-surface Interaction

Dual-surface interaction involves sequentially coordinating input from the front (typically with the thumb) with an input channel attached to the back of the device. The design of Dual-Surface input is motivated by some of the common problems with one-handed interactions, such as occlusion and out-of-thumb-reach. Unlike prior techniques for back-of-the-screen input, our design considers the use of relative input on the back. This allows users to take advantage of the best of both worlds: direct absolute input and indirect relative input.

Direct absolute input facilitates access to large targets very quickly and allows users to interact with a significant portion of the device. The drawback with absolute input with one-handed interaction is the lack of reachability based on the form factor of the device, occlusion of smaller targets,

and difficulties selecting items on the edges. We hypothesize that augmentation of a device with relative input in the back (as in [121]), will reduce or eliminate problems with occlusion. Relative input is generally unaffected by the location of targets and can itself be augmented with virtual enhancements [1, 2, 34, 39, 44, 77, 107], thereby facilitating a number of tasks. However a drawback of this input is its dependence on clutching.

We considered the pros and cons of direct input and relative input in the design of our Dual-Surface interaction. We propose that, with a sequentially coordinated interaction, users can leverage off the benefits from both input types to perform a variety of tasks. Our proposed coordination is very similar to, and inspired by, the kinematic chain model proposed for bi-manual interaction [47]. While users are not constrained to Dual-Surface interaction alone, we propose the following coordination to take advantage of this type of input. The front input leads the back input, sets the frame of reference for the back cursor to operate in, and is used for performing coarse movements. The back relative input follows the front, operates within the frame-of-reference established by the front, and is able to perform finer movements.

4.5.2 Apparatus

We developed a prototype similar to that in [121]. We used a Dell Axim X30 PDA with a 624MHz processor and 64MB memory. We attached an Ergonomic USB touchpad on the rear side of the PDA. The touchpad was oriented along the long side of the PDA (see Figure 4.7. We placed the touchpad as close as possible to the top of the PDA to make it conformable for users to manipulate the cursor with the index finger when holding the device with one hand. All gestures moved the cursor using a relative mapping. The software was implemented in C#.NET.

Figure 4.7: In the experiment, we asked participants to sit in a chair, and perform the target selection task using one hand. The touchpad on the back is operated using the index finger.

4.5.3 Experiment 1 - Selection

The goal of this experiment was to measure the performance of target selection with Dual-Surface (thumb + cursor) input against the Front (thumb only) and Back (cursor only). Participants were required to complete a series of target selection tasks using the three techniques with their dominant

hand. In this interaction, the index and thumb fingers were not used for gripping onto the device.

We implemented the Shift technique [128] for the Front in order to address the fat-finger problem introduced by interacting with a touch screen using bare finger. When selecting a target via the Front, participants tapped near the target to invoke a callout. Since our targets were always either placed on the top right or bottom left, the callout was placed in a position that it was not occluded by any part of the hand. To perform a selection, participants moved the Shift crosshair cursor onto a target and lifted the finger. The Back technique allowed the user to control an arrow cursor with their index finger using the touchpad mounted on the back of the PDA. When moving the cursor with the touchpad, the cursor was always read as a mouse-hover until the user quickly tapped the touchpad. Tapping of the touchpad registered a full mouse click. Initially Dual-Surface technique forced the user to tap the screen in the general location of the target with thumb to give an absolute cursor position near the target. After front input was performed, we disabled it for the purposes of experiment (to ensure that no more front inputs were performed) and the back touchpad was enabled, where the input was exactly the same as the Back technique. All feedback was provided visually on the PDA. A target was highlighted if the cursor was inside it.

In real world applications, selections often take place at the corners of the screen (see Figure 4.8). This makes target acquisition with one hand difficult. In order to measure the performance of the three techniques in situations close to real-world applications the targets were placed at varying distances away from the corner of the device. In our study, the targets were located at the top-right and bottom-left corners since, with right-handed users (all the participants were right-handed) movement of the thumb is difficult in the top-left to bottom-left direction [78]. We used an offset distance to place the target at varying spots away from the corner. The smaller the offset, the closer the target was to the corner, and vice versa (see Figure 4.9).

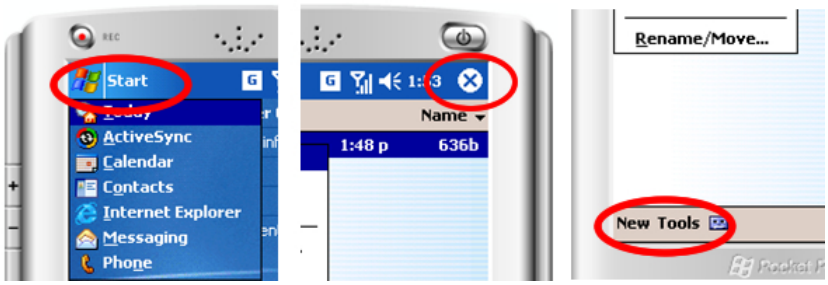

Figure 4.8: Mobile applications are mostly replicas of their desktop versions in which targetable items can occur in unreachable or difficult to reach areas of the device when using one hand.

In order to remove bias against the *Shift* technique in selecting targets being close to the corners, a small pilot study was conducted to measure the reasonable offset. The results showed that 9 pixels (3.6mm) away from the edges of the corner was the closest reasonable offset that would allow the user to select the target with the *Shift* technique. Furthermore, we found that a square target of size 5 pixels (2mm) was the smallest reasonable size to acquire target using any of the three techniques.

Procedure

To start a trial, participants clicked the "Start" button (see Figure 4.9) with either their thumb (when interacting via the Front and Dual-Surface) or the cursor (when interacting via the Back). Then, participants had to use the techniques described above to acquire the target in view. A trial finished either after participants successfully selected the target or when an error occurred. A trial was counted as an error if participants missed a target or failed to make a selection in a timeout of 25 seconds. Participants were instructed to complete the task as fast and as accurate as possible. To promote better performance of all techniques, we showed the time of the last attempt and the overall best time of the current input technique. Users were encouraged to beat their best time.

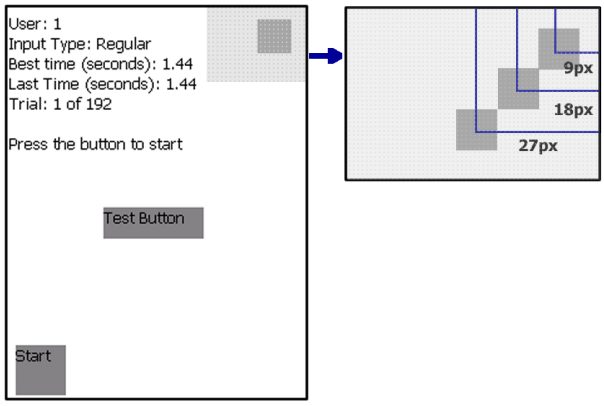

Figure 4.9: "Start" button is at the opposite corner of the target.

A warm-up session was given to participants at the start of each new input technique. Participants were instructed on how to control the cursor in a test trial. Once participants felt comfortable with the technique they were given two practice trials with the input technique before starting the real experiment. The entire experiment lasted 30 minutes. Participants were encouraged to take breaks during the experiment. Participants filled out a post-experiment questionnaire upon completing the experiment.

Experimental Design

The experiment employed a $3 \times 2 \times 3 \times 3$ within-subject factorial design. The independent variables were input *Technique* (*Back*, *Front*, and *Dual-Surface*), *Location* (*Top-Right* and *Bottom-Left*), *Offset* (9px, 18px, and 27px away from corner), and target *Size* (5px, 10px, and 15px). Each trial of the experiment represented a $Technique \times Location \times Offset \times Size$ combination, and was repeated 3 times by each participant. The order of presentation of the trials was randomly chosen. Input techniques were counter balanced among participants.

The experimental design can be summarized as: $3\ Techniques(Front, Back, and\ Dual - Surface) \times 3\ Offsets(9px, 18px, and\ 27px) \times 3\ Sizes(5px, 10px, and\ 15px) \times 2\ Locations(Top-Right\ and\ Bottom - Left) \times 3\ Repetitions \times 8\ Participants = 1296$ data points in total.

Participants

Eight participants (6 males and 2 females) between the ages of 20 and 35 were recruited from a local university participated in this study. We screened participants so that they were all right-handed, and had previous experience with graphical interfaces.

Results

For all our analyses, we used the univariate ANOVA test. Levene's test of equality indicated unequal variances between all the tested groups, thus Tamhane's T2 test was used for post-hoc pair-wise analysis.

Completion Time

A total of 51 trials out of 1296 incurred a timeout (3.9%) and the average trial completion time over all trials without timeouts was 4378.7 ms (s.e. = 91.3 ms). We excluded errors and timeouts from our analysis. We found no significant effect for *Location* and therefore we collapsed our data across this variable and our analysis was only performed on the other three variables.

There was a significant effect of *Technique* ($F_{2,14} = 16.268$, $p < 0.001$), of *Size* ($F_{2,14} = 84.578$, $p < 0.001$) and of *Offset* ($F_{2,14} = 15.94$, $p < 0.001$) on completion time. Figure 4.10 (left) shows average completion time for each *Technique*, by target *Size* and *Offset*. We found interaction effects for $Technique \times Size$ ($F_{4,28} = 29.395$, $p < 0.001$), for $Technique \times Offset$ ($F_{4,28} = 18.630$, $p < 0.001$) and for $Size \times Offset$ ($F_{4,28} = 12.817$, $p < 0.001$).

Post-hoc pair-wise comparisons (unequal variance assumed) show significant differences across each of the three pairs for all three techniques ($p < 0.001$). The performance with Dual-Surface (3604ms, s.e. 168) was significantly faster than either Shift (7107ms, s.e. 169) or just the Back alone (5106ms, s.e. 168).

Similarly, post-hoc pair-wise comparisons revealed significant differences across all three pairs of target size ($p < 0.001$). Performance was fastest with targets of 15 pixels or 6mm (3096ms), then with 10 pixel or 4mm (4680ms) and slowest with 5 pixel or 2mm (8041ms).

Post-hoc comparisons also show significant difference for the following pairs of offsets of 9 and 18 pixels ($p < 0.001$), and 9 and 27 pixels ($p < 0.001$) but not for the 18 and 27 pixel pair, with users taking approximately 1.5 times longer with the 9 pixel offset as with the 18 or 27 pixel offsets.

Attempts

In addition to recording the completion time, we also recorded the number of attempts it took users to accurately select the target. An attempt with Shift consists of releasing the thumb and pressing again. With the Back and the Dual-Surface techniques, an attempt consists of clicking outside the target. In general the number of attempts captures the number of misses that have occurred prior to properly selecting the target.

There was a significant effect of *Technique* ($F_{2,14} = 28.575$, $p < 0.001$), of *Size* ($F_{2,14} = 34.044$, $p < 0.001$) and of *Offset* ($F_{2,14} = 13.038$, $p < 0.001$) on attempts. We found no significant interaction between these two factors ($F_{6,5} = 2.2$, $p = 0.203$). Figure 4.10 (right) shows number of attempts for each *Technique*, by target *Size* and *Offset*. We found interaction effects for $Technique \times Size$ ($F_{4,28} = 27.501$, $p < 0.001$), for $Technique \times Offset$ ($F_{4,28} = 12.97$, $p < 0.001$) and for $Size \times Offset$ ($F_{4,28} = 12.504$, $p < 0.001$).

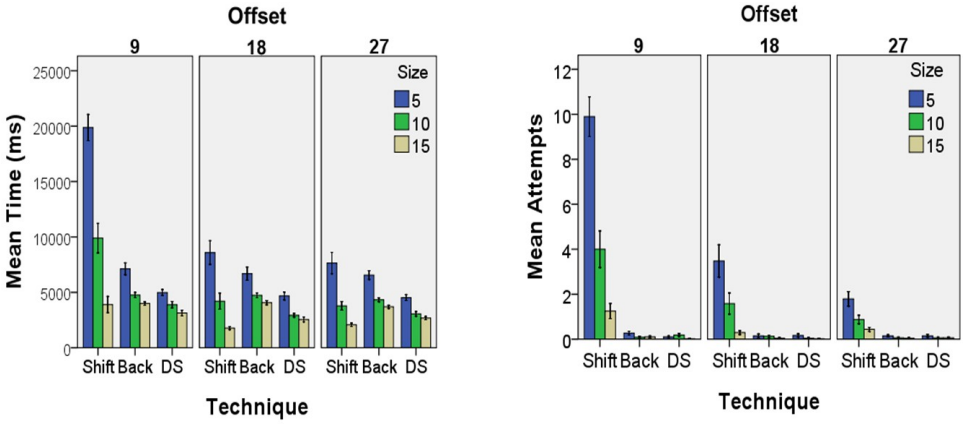

Figure 4.10: (left) Average completion time. (right) Average number of attempts or misses, for each technique, by offset and target size (bars represent $+/-1$ standard error).

Post pair-wise comparisons (unequal variance assumed) show significant differences across the pairs Dual-Surface and Shift, and Back and Shift ($p < 0.001$). The number of attempts with Dual-Surface (0.09) and Back (0.113) were significantly smaller than the number of attempts with Shift (2.62). There was no significant difference in number of attempts between Dual-Surface and Back.

Post pair-wise comparisons revealed significant differences across all three pairs of target size ($p < 0.001$). There were significantly fewer attempts with 15 pixel targets (0.252) than with 10 pixel (0.78) or 5 pixel (1.794) targets.

Finally, post comparisons also show significant differences for the following pairs of offsets: 9 vs. 18 pixels ($p < 0.001$), and 9 vs. 27 pixels ($p < 0.001$), but not for the 18 vs. 27 pixel pair. Number of attempts with 27 pixel offset was 0.403, with 18 pixel offset was 0.655, and with 9 pixel offset was 1.769.

Failure rate

Of the total 51 timeouts that occurred in the experiment, 50 timeouts resulted when using Shift, one timeout with the Back and no timeouts occurred with Dual-Surface. Of the total 50 timeouts with Shift, 88% resulted from selected targets that were offset by 9 pixels from the corner of the display.

Subjective Preference

Of the eight participants, all showed a high preference for Dual-Surface followed by Shift. Participants reported frustration with Dual-Surface as they were required to readjust their grip to perform

the task properly. Frustration with Shift resulted from small target sizes but particularly when the targets were in the corner. Participants also commented on the difficulty in selecting targets with the Back alone as users were required to clutch frequently to acquire the target.

Discussion

As expected, target size had a significant effect on task completion time. Shift outperformed Back and Dual-Surface on the larger targets when these were placed away from the edges and closer to the center of the screen. This reveals the advantage of direct input over relative cursor input in one-handed target selection when occlusion is not an issue and when targets are reachable. In contrast, both Back and Dual-Surface outperformed Shift in selection time for small targets (< 10 pixels or 4mm), and when the target was placed closer to the center of the screen.

Back and Dual-Surface have relatively consistent performance across targets of different sizes and at different locations. Participants finished the task faster with Dual-Surface compared to Back. Note that, with Dual-Surface, participants had to select the target using the Back even if the target was large enough for the thumb. It is possible that without this restriction the performance of Dual-Surface can improve further. Overall, the results support our hypothesis that one-handed interaction could benefit from the effective coordination of absolute thumb input in the front and relative cursor input in the back.

4.5.4 Experiment 2 - Tunneling

The results of the first experiment revealed that target selection with Dual-Surface was more efficient than input via the Front or the Back alone. We also wanted to evaluate the possibility of using Dual-Surface with more complex tasks. Note that none of the prior work provided any empirical evidence of the effectiveness of behind-the-screen input for complex tasks. Other tasks for one-handed input are also common. For example, scrolling through a list of contacts with one-hand is common [78]. This type of task is commonly categorized as a steering task. Steering is also routinely carried out within a variety of contexts, such as highlighting a piece of text or navigating through file menus [1]. The purpose of this experiment was to evaluate the effectiveness of the Dual-Surface interaction in steering tasks.

Virtual Enhancement

Note that one of the obvious benefits of having a relative cursor is that it can be augmented with virtual enhancements [1, 2, 34, 39, 44, 77, 107] to facilitate a variety of interactions. For instance, menu navigation is more effective with the inclusion of a pseudo-haptic enhancement that allows the cursor to stay within the menu "tunnel" [1]. In this study, we leverage upon the ability to attach virtual enhancements to the relative cursor in the Dual-Surface interaction. In our task, the virtual enhancement consisted of a pseudo-haptic effect that aids in the tunneling. While this is

specific to tunneling, we propose the use of Dual-Surface widgets (see Discussion section) for which the enhancements dynamically vary based on the control the user is operating on the front. For example, a menu or scrollbar could employ an enhancement similar to the one we describe here. However toolbar buttons or other controls may adopt other enhancements such as those proposed for expanding targets [90] or for dynamically varying the cursor size [44]. Such enhancements are not possible with absolute input and therefore the attachment is associated only with the relative input behind-the-display and not with the front.

Similar to [1], the pseudo-haptic enhancement was simulated by software, and is described as follows:

1. The cursor initially exists within a rectangle (square in this case and the dimensions were defined as the tunnel width);
2. When the cursor touches a side of the square, the cursor's position is centered on the center of that side (see Figure 4.11);
3. A new square is then centered on the new cursor position and we go back to step 1.

It is possible to fail the task by exiting from the middle of the tunnel.

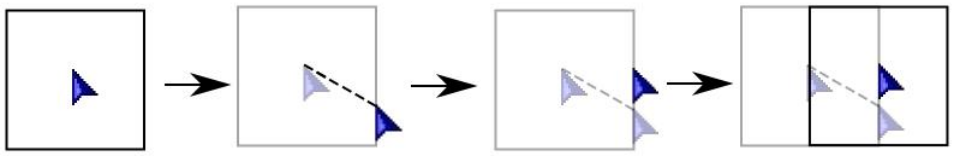

Figure 4.11: Once the cursor hits an edge, it is brought back to the center of the edge. The square itself was not visible to the user.

Task Analysis

The task was carried out by first clicking a "Start" button, and then quickly moving to the start of the tunnel to carry out the tunneling. The task simulated a canonical situation, in which a user would to move the cursor on the display to a widget before steering in the tunnel (i.e. to acquire the thumb on the scrollbar and then to move it). In the steering component, the participants click the start of the tunnel and then steer within the tunnel until they exits from the end. The tunnel was placed vertically and to the right side, for easy access with the thumb. The start and end of the tunnel were rendered in green and red. The "Start" button was placed on the left-hand side of the screen (see Figure 4.12 (left)).

Conditions

We included visual feedback for all techniques. Visual feedback displayed the amount of deviation of the thumb from the center of the tunnel. The deviation was displayed using a yellow band. The

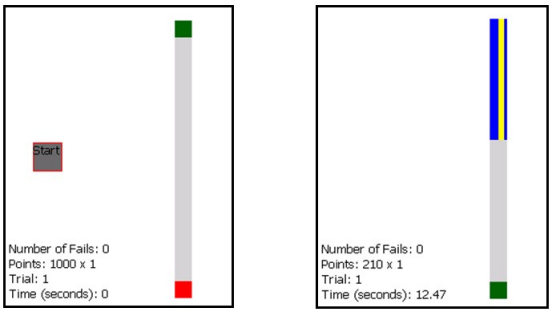

Figure 4.12: (left) "Start" button and the tunnel. The start and the end of the tunnel were rendered in green and red. (right). The "Start" button disappeared after a trial started. The yellow band indicates the offset of the user's thumb from the center of the tunnel. (better seen in color).

width of the yellow band gave feedback as to how much the user deviated from the center of the tunnel (see Figure 4.12 (right)). As with the first experiment, we evaluated three one-handed techniques: Front (via thumb), Back with virtual enhancement (via cursor), and Dual-Surface (via both). In the Front technique, participants completed the entire task using only their thumb. When steering in the tunnel, they slid their thumb on the screen from the top of the tunnel to the bottom. For the enhanced Back technique, participants completed the entire task using the cursor, whose movement was controlled by the index finger. When steering within the tunnel, pseudo-haptic enhancement was provided to restrict the cursor movement within the tunnel. For the Dual-Surface technique, participants were asked to click the "Start" button and the start of the tunnel using their thumb, but to use the enhanced Back technique for steering. As with the Back technique, the Dual-Surface technique also used a virtual enhancement of the cursor.

Experimental Procedures

A trial started after participants clicked the "Start" button, and ended either after participants successfully completed the trial or after the trial failed. A trial was marked a failure if participants failed to complete the tunneling task. Participants were given 5 attempts to complete the tunneling task once the green start button of the tunnel was clicked. The tunneling task was marked as successful when participants exited the tunnel from its red end. Failure to do so marked a failed attempt. A trial failed after 5 failed attempts. Participants were instructed to complete the task as fast and as accurately as possible. To promote better performance with the various input techniques, we implemented a rewarding system such that when the user performed better, more points were awarded.

A warm-up session was provided to the participants at the start of each technique. Participants were given 5 practice trials per condition before starting the experiment. The entire experiment lasted 40 minutes. Participants were encouraged to take breaks during the experiment. Participants completed a post-experiment questionnaire to rank the techniques.

Experimental Design

The experiment used a $3 \times 2 \times 3$ within subject factorial design. The independent variables were input *Technique* (*Front*, *Back with virtual enhancement*, and *Dual-Surface*), tunnel *Length* (160px and 260px), and tunnel *Width* (12px, 15px, and 18px). The width of the 15px tunnel (6mm) was selected as the intermediate level as it represents the standard width of a scroll bar on the PDA. Furthermore, our pilot study showed that 12px (4.8mm) was the smallest width of the tunnel that allowed participants to perform the tunneling task using the thumb.

Each trial represented a $Technique \times Length \times Width$ combination, and was repeated 10 times by each participant. The order of presentation of the trials was randomly chosen. Input techniques were counter balanced among participants.

The experimental design can be summarized as: $3\, Techniques(Front, Back\ with\ virtual\ enhancement, and\ Dual-Surface\ with\ virtual\ enhancement) \times 2\ Length(160px\ and\ 260px) \times 3\ Width(12px, 15px, and\ 18px) \times 10\ Repetitions \times 11\ Participants = 1980$ data points in total.

Participants

Eleven participants (10 males, 1 female) between the ages of 20 and 35 were recruited. Three of the 11 participants completed the first experiment. All participants were right-handed.

Results

For all our analyses we used the univariate ANOVA test. Levene's test of equality indicated unequal variances between all the tested groups, thus Tamhane's T2 test was used for post-hoc pair-wise analysis.For the tunneling task we performed the analysis on several dependent variables separately. The last attempt time, was the time taken to steer within the tunnel for the last successful steering time. This time represents the performance of the participant after attempting 1 or more times to steering within the tunnel. Total time consists of the total time taken by the participants to complete the tasks, after repeated trials or attempts. This time include also the time it took the participants to clutch to return to the top of the tunnel to begin the task. The number of attempts represents the number of times it took the participants to complete the trial.

Last Attempt Time

There was a significant effect of *Technique* ($F_{2,20} = 30.019$, $p < 0.001$) and of tunnel *Length* ($F_{1,10} = 48.425$, $p < 0.001$) on completion time. There was no main effect of tunnel *Width* ($F_{2,20} = 2.639$, $p = 0.096$) on completion time. Figure 4.13 (left) shows average completion time for each technique, by tunnel width and length. We found interaction effects for $Technique \times Length$ ($F_{2,20} = 14.664$, $p < 0.001$). There was no interaction effect for $Technique \times Width$ or for $Width \times Length$.

Post-hoc pair-wise comparisons (unequal variance assumed) show significant differences across

each of the three pairs of the three techniques ($p < 0.001$). Performance with Back (877.4ms, s.e. 29.5) alone was significantly faster than either Dual-Surface (1144.6ms, s.e. 29.5) or just the Front alone (2492.5ms, 29.5).

Post-hoc pair-wise comparisons did not reveal significant differences across all three pairs of tunnel width, suggesting that users performed equally well with the 12 pixel tunnel width as with the 18 pixel tunnel.

Total Time

There was a significant effect of *Technique* ($F_{2,20} = 40.511$, $p < 0.001$), of tunnel *Width* ($F_{2,20} = 77.022$, $p < 0.001$), and of tunnel *Length* ($F_{1,10} = 43.443$, $p < 0.001$) on completion time. Figure 4.13 (right) shows average total time for each technique, by tunnel width and length. We found interaction effects for $Technique \times Width$ ($F_{4,40} = 58.107$, $p < 0.001$), for $Technique \times Length$ ($F_{2,20} = 17.801$, $p < 0.001$), and for $Width \times Length$ ($F_{2,20} = 7.336$, $p < 0.001$).

Post-hoc pair-wise comparisons (unequal variance assumed) show significant differences across each of the three pairs of the three techniques ($p < 0.001$). Performance with Dual-Surface (1974.78 ms, s.e. 89) was significantly faster than either Front (6224.6 ms, s.e. 89) or Back alone (3147.1, s.e. 89). Note that it took users approximately 1.5 times longer to complete all attempts with Dual-Surface, approximately 4 times longer with Back alone, and 3 times longer with Front, compared to the last attempt time. In the discussion in Section 3.2.5, we explain some of the reasons for the differences in these times.

Post-hoc pair-wise comparisons did not reveal significant differences across all three pairs of tunnel width, suggesting that users performed equally well with the 12 pixel tunnel width as with the 18 pixel tunnel.

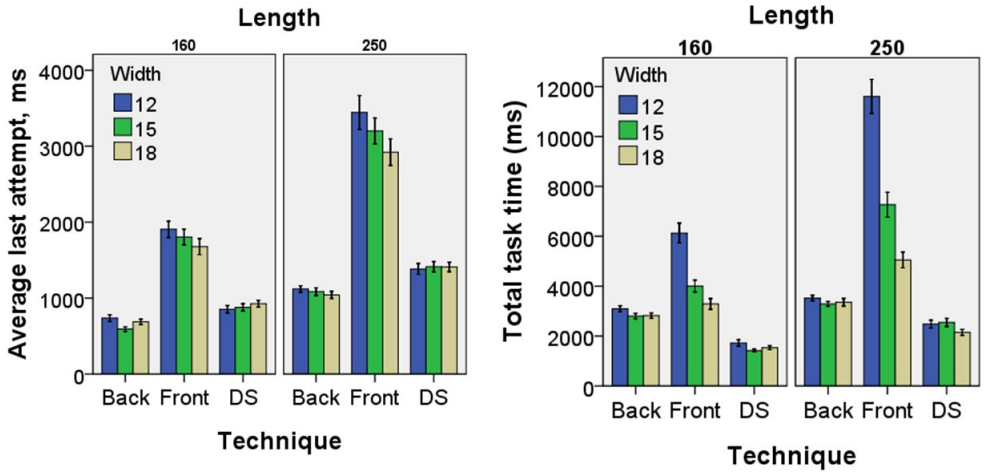

Figure 4.13: (left) Average last attempt time; (right) Average task time, for each technique, by tunnel length and width (bars represent $+/-1$ standard error).

Number of Attempts

There was a significant effect of *Technique* ($F_{2,20} = 53.695$, $p < 0.001$) and of tunnel *Width* ($F_{2,20} = 53.136$, $p < 0.001$) but no main effect of tunnel *Length* ($F_{1,10} = 1.997$, $p = .188$) on number of attempts. Figure 4.14 (left) shows average number of attempts for each technique, by tunnel width and length. We found interaction effects for $Technique \times Width$ ($F_{4,40} = 53.975$, $p < 0.001$) but none for $Technique \times Length$ ($F_{2,20} = 1.359$, $p = 0.28$) or for $Width \times Length$ ($F_{2,20} = 1.377$, $p = 0.275$).

Post-hoc pair-wise comparisons (unequal variance assumed) show significant differences across each of the three pairs of the three techniques ($p < 0.001$). The fewest number of attempts were found with Back alone (1.02 attempts, s.e. 0.03), then with Dual-Surface (1.14 attempts, s.e. 0.03), and most with Front alone (2.22 attempts, s.e. 0.03).

Post-hoc pair-wise comparisons also show significant differences across each of the three pairs of tunnel widths ($p < 0.001$). the fewest number of attempts were incurred with width 18 (1.02 attempts, s.e. 0.03), then with width 15 (1.14 attempts, s.e. 0.03), and most with width 12 (2.22 attempts, s.e. 0.03).

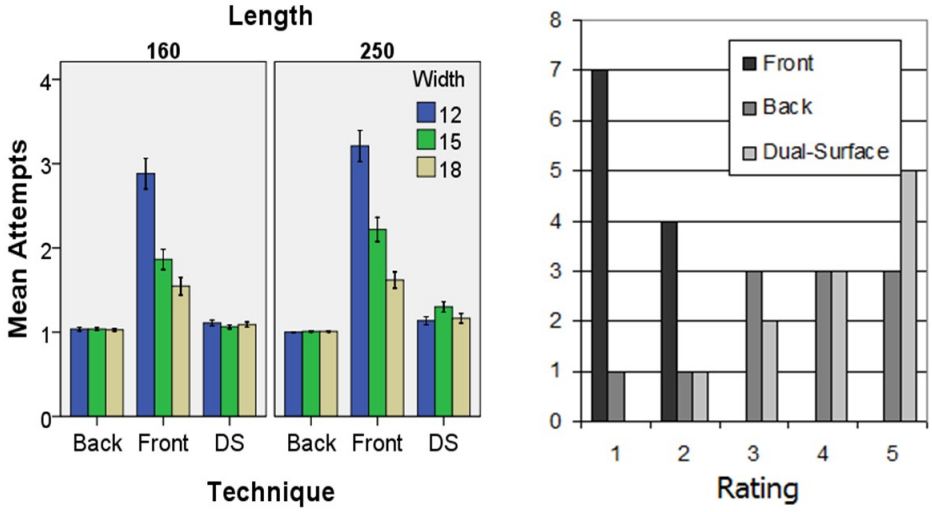

Figure 4.14: (left) Average number of attempts for each technique, by tunnel length and width (bars represent $+/-1$ standard error). (right) Frequency of ratings from 1 (least preferred) to 5 (most preferred) for each of the three techniques.

Subjective Preference

Participants rated each of the three techniques, from 1 (least preferred) to 5 (most preferred). Half of the participants rated Dual-Surface a 5, and one third rated it a 4. None of the participants gave Dual-Surface a rating of 1. On the other hand 70%of the participants rated Front a 1 and none rated it above a 2. Back was marginal and rated average across all participants. The rankings for each technique are provided in Figure 4.14 (right).

4.6 Discussion

We first discuss the results of the second experiment and then based on the results of both experiments we present some recommendations to designers. We also propose the use of Dual-Surface input in a number of applications and finally present some of the limitations of this form of interaction.

4.6.1 Experiment 2 Results

The results of the second experiment showed the benefits of using a virtual enhancement with the behind-the-surface cursor. This is only possible with a relative cursor which we made available in Dual-Surface. Even with visual feedback, participants still performed the tunneling task significantly slower with the thumb than with cursor. One interesting finding is that the average time to complete the last steering attempt for the Back was shorter than with the Dual-Surface input. We expected these two techniques to perform similarly as they both uses the Back for the tunneling task. This could be due to several reasons. We noticed that, participants used different grips in favor of different techniques. For the Front, they held the device at about a 45° angle to the index finger. By holding the device in this orientation, they maximized the region reachable by the thumb. For the Back, the device was held in the same orientation as the extension of the index finger. This position helped to better perform vertical tunneling using the index finger. However, to perform this task comfortably with the Dual-Surface input, the participants needed to switch frequently from one grip to another. This tired their hands, and may have led to the unexpected performance.

Even though Back has better performance in the last-attempt time, overall Dual-Surface was the most efficient technique. This resulted primarily as it took longer to clutch with the relative input on the Back, where, with the Dual-Surface technique users could simply move the cursor to the top of the tunnel with their thumb and then attempt to scroll again. As mentioned above, it took participants approximately 1.5 times longer to complete all attempts with Dual-Surface, approximately 4 times longer with Back alone, compared to the last attempt time. This shows that with Back, participants spent a large proportion of their time clutching the cursor towards the goal. This finding supports the hypothesis that one-handed interaction could benefit from the effective use of absolute thumb input and relative cursor input.

Analysis on the number of attempts showed that participants made significantly more mistakes when using the thumb, which lead to the highest number of attempts with this technique. One interesting finding is that participants did not make more mistakes on the 250px compared to the 160px long tunnel. As shown in Figure 4.12, the tunnel of length 250px is almost the full length of the screen. Given that it is more difficult to steer through a vertical tunnel than a horizontal tunnel [34, 135], we suspect the highest average attempts shown in Figure 4.14 (left) are close to the upper bound of the number of failures in scrolling a scrollbar on the tested PDA. This suggests that designers may want to reconsider the design of scrollbars on smaller devices, either in software or

with a solution such as the one proposed in this paper.

Tunneling tasks on mobile devices usually require both hands, one to hold the device and the other to perform the steering (sometimes with a stylus). We demonstrated that attaching a virtual enhancement to a cursor can assist in tunneling with one-hand. This provides support for the use of Dual-Surface interaction, but also further supports the need for easily accessible relative input on mobile device. While we only showed the advantage of relative input with the steering task, many other tasks including pointing and selection can benefit from the use of easily accessible relative input on mobile devices.

4.6.2 Applications

Numerous applications can benefit from one-handed interaction. We have demonstrated that small and typically unreachable targets can benefit from Dual-Surface interaction. The results also reveal that complex tasks, such as steering, can benefit, if the Dual-Surface input is augmented with virtual enhancements.

Map navigation: Panning is a common operation in map browsing applications. Usually, information that would normally be available to a user would reside off-screen. In this case, panning is used for bring an off-screen target into view, and extra cursor movements towards the target are often required as the panning operation will land the cursor a distance away from the desired target. For instance, to select the off-screen target in Figure 4.15, users need to make a ←(right-to-left) gesture to bring the target into the display, and then make a → (left-to-right) gesture to move the cursor onto the target for selection. With Dual-Surface interaction, off-screen target selection can be performed with less effort. The user can simply leave the cursor in its original position. Instead of manipulating the cursor, the user pans the map to the left to move the target towards the cursor. Once the target is underneath the cursor, the user simply taps on the back to make a selection.

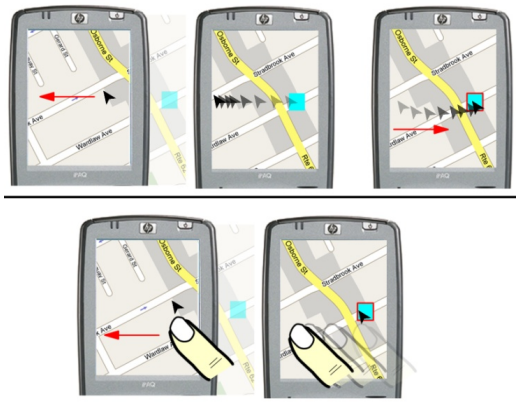

Figure 4.15: Off-screen target selection. Top-left: pan the map using cursor requires a left gesture. Top-middle: after the panning, cursor is on the left edge. Top-right: cursor moves back to the right to make selection. In contrast, bottom-left: pan the map to left using thumb. Bottom-right: the target is moved underneath the cursor.

3D Object Rotation: 3D manipulations such as rotation can take advantage of Dual-Surface interaction. Users can select a rotational axis anywhere on the screen with the back and then can pan their thumb to rotate the object along the selected axis. Intuitively, it would require more effort to perform the task with either the back or the front alone. Note that the coordination suggested in this application does not necessarily follow that proposed earlier, i.e. the back can also initiate the movement and the front can follow it.

Simultaneous Inputs: Although Dual-Surface is performed sequentially, one can also take advantage of naturally occurring simultaneous actions. One example of such an interaction is zooming. Zooming can be triggered by having the thumb and index fingers make simultaneous opposite gestures. For example, a user can trigger a zoom-in action by having the thumb make a $\leftarrow$ gesture and the index finger make a $\rightarrow$ gesture.

Dual-Surface Widgets: We can design a new class of widgets that support Dual-Surface input, which we refer to as DS-widgets. For instance, we described the use of a virtual enhancement with the scrollbar. This could eventually become a DS-scrollbar which would behave just like a normal scrollbar with the front input, but would also use any possible enhancement when coordinating the front with the back input. Other similar widgets, such as toolbar buttons could be made to work with Dual-Surface. Items on a DS-toolbar could expand and shrink under the influence of the position of the relative cursor as in [90].

Hardware Alternatives: Many hardware design options are available. In our study, the manipulation of cursor movement was through a touchpad. Alternately, this could be replaced with a mini joystick. The joystick could be similar to a trackpoint. Note that, previous research has reported that, due to the kinematic limitations of the index finger, complex gestures involving vertical motions are difficult with the index finger [135]. A joystick, however, requires minimal finger motion to control the cursor movement and also allows for rate-based control of the cursor. Furthermore, with a joystick, it is possible to provide real force feedback to support richer interactions. Another option to replace the touchpad is to use an optical sensor. Cursor movement can be controlled by the movement of the finger tip against the optical sensor.

4.6.3 Recommendations

Based on the findings of our experiments, we make the follow recommendations for enhancing one-handed use:

1. Complement mobile devices with relative cursor input that is accessible such as a touchpad behind the display;

2. Relative cursor input could be associated with virtual enhancements for making complex tasks easier;

3. Large targets that are accessible with the thumb should be placed in accessible areas for the thumb;

4. Smaller targets which would be difficult to access with the thumb but relatively easy to access with the cursor, could be placed in areas that are best suited for cursors, such as in corners and edges (the width factor in the Fitts equation increases significantly when targets are on the edges, reducing the index of difficulty);

5. To facilitate steering tasks widgets requiring tunneling should be made larger for easy access in the front.

4.6.4 Limitations of Dual-Surface Interactions

While the results show the advantages of Dual-Surface input, this interaction needs to be further developed to overcome some limitations. For instance, the results of the studies showed that, Dual-Surface input was faster than Back alone. This may not remain true in situations where the goal is too far to be reached by the thumb. In this case, most of the input will take place on the back. Moreover, based on different tasks, users may have to use different grips to facilitate input on the Back or on the Front. Frequently changing grips may discourage users from using Dual-Surface input. Furthermore, the experiments evaluated Dual-Surface input in only two, albeit common, tasks. It is left for future work to determine how Dual-Surface input would operate on other routine tasks, and whether users would employ it in real-world mobile settings, such as when walking or driving a car.

4.7 Summary

In this chapter, we present LucidCursor, a prototype mixed input mode device. We also demonstrate the results of two experimental studies designed to measure the performance of Dual-Surface input in facilitating one-handed interaction on mobile devices. We found Dual-Surface input outperformed Front input and Back input in both tunneling and target selection tasks. This clearly shows the promise of Dual-Surface interaction. The benefit of input via both sides of a mobile device is that it takes advantage of the best of both relative and absolute input. Based on the findings, we recommend Dual-Surface interaction to be widely applied on handheld devices.

Chapter 5

Mixed Input Mode in Always-available Input - Magic Finger

Table 5.1: Hardware prototypes and corresponding interaction techniques - Magic Finger[5]

User Interface / Input Platform	Hardware UI	Software UI
Mobile	*LucidCursor*	*Dual-surface Input*
Desktop	*LensMouse*	*TouchCuts & TouchZoom*
Always-available Input	**Magic Finger**	**Magic Finger Interactions**

We have demonstrated how existing devices can be turned into *mixed input* devices. In this section, we introduce a new device, called Magic Finger. The device was created to turn user's finger into a *mixed input* device. Equally important, we show that the most significant contribution of Magic Finger is the support of always-available input [115].

Recent years have seen the introduction of a significant number of new devices capable of touch input. While this modality has succeeded in bringing input to new niches and devices, its utility faces the fundamental limitation that the input area is confined to the range of the touch sensor. A variety of technologies have been proposed to allow touch input to be carried out on surfaces which are not themselves capable of sensing touch, such as walls [33, 53], tables [53], an arbitrary piece of paper [51] or even on a user's own body [51, 59, 115].

Mounting cameras on the body has enabled these regions to become portable [49, 51, 93, 101]. However, like other vision-based implementations, the range of the sensor is limited to the viewing area of the camera, thus the capabilities of these sensors are in some ways more limited than are non-vision based techniques.

[5]The video of Magic Finger can be found at: `http://www.youtube.com/watch?v=WgR1358xZaY`

These approaches to instrumentation have all focused on enabling touch capability for surfaces the user will touch with their finger. To overcome their inherent limitations, we propose finger instrumentation, where we invert the relationship between finger and sensing surface: with Magic Finger, we instrument the user's finger itself, rather than the surface it is touching. By making this simple change, users of Magic Finger can have virtually unlimited touch interactions with any surface, without the need for torso-worn or body-mounted cameras, or suffer problems of occluded sensors.

Our work has been inspired by earlier projects in always-available input [115]. Like those earlier projects, Magic Finger is capable of sensing the moment of touch and relative finger displacement on a surface. Further, inspired by the versatility of the human finger, Magic Finger can also sense the texture of the object a user touches, allowing for the use of a machine learning classifier to recognize the touched object. Thus, contextual actions can be carried out based on the particular surface being touched. Additionally, Magic Finger can also identify artificial textures and fiduciary markers. This extends its ability to recognize richer information, thus enabling many novel interaction techniques.

Magic Finger is a thimble-like device worn on the user's finger (Figure 1.6). It combines two optical sensors: a lower-resolution, high speed sensor for tracking movement, and a higher resolution camera for capturing detail. In addition to this device, the contributions of this work also include an exploration of associated hardware and software design spaces, an evaluation of its texture sensing accuracy levels, and a set of applications and interactions that are enabled by Magic Finger.

5.1 Hardware Design Space

In designing a device, our main goal is to enable the finger to sense touch on physical objects, without any environmental instrumentation. In order to emulate traditional touch techniques, the device must be able to sense contact and $2D$ positional movements. In this section, we present a design space of possible hardware implementations that could enable these goals. For a broader discussion of hardware for small on-body devices, we refer the reader to Ni and Baudisch' s work on disappearing mobile devices [101].

5.1.1 Scope of Recognition

We define three scopes of recognition: none, class, and instance. In its simplest form, the device could be able to sense finger movement without the capability of knowing anything about the contact object (none). However, when it is the finger itself performing the sensing, it would be helpful to know what the finger is touching, so the input could be directed to an appropriate device. The scope of recognition could be increased to recognize certain classes of objects (e.g. a table or a phone) (class), or could be further extended to distinguish between specific instances of objects (e.g. my table vs. your table) (instance).

5.1.2 Sensing X-Y Movements

To emulate traditional touch, the hardware must be able to sense $X - Y$ movements. Previous work has shown that an optical flow sensor (found in optical mice) can be used as a small touch sensitive device [15, 101]. As such, $2D$ finger movements could be detected by affixing an optical flow sensor to the finger. Such sensors can be extremely small and offer reliable 2D motion sensing on a wide variety of surfaces, but a light source is required. Alternatively, a mechanical device, such as a small trackball could be used, but may not be as robust to varying types of surfaces. A mechanical device may also require a larger form factor.

5.1.3 Sensing Material

To increase the recognition scope, the device could sense the material that the finger is in contact with. This could be achieved with optics, using texture recognition algorithms [102]. Using an optical flow sensors may be difficult, as their resolutions are typically low (e.g. 18×18). Higher quality cameras may be required to improve the robustness of material recognition or to extend the recognition scope to instance. Alternatively, a Multispectral Material Sensor [52] could be used, but would only support a recognition scope of class. To sense properties such as roughness, softness, and friction, a texture sensor [98] could be used, but may require the user to touch the surface in specific ways.

5.1.4 Sensing Contact

To sense contact, a hardware switch, which connects a circuit when being pressed by a touching force, may be most accurate. While this would enable explicit sensing of contact, it would require the device to have an additional single-purpose sensor. Contact could instead be inferred from an audio sensor [58], but accuracy may not be as high. Alternatively, contact could be inferred from an optical sensor that is already present to sense movements and material.

5.1.5 Output

In the simplest form, the device could provide no output at all to the user. However, this could be severely limiting, especially in cases where periphery devices that provide feedback are not available. Tactile feedback is possible but may come with high power consumption, and would require an additional component. One or more LEDs could be used to display bit-wise low resolution information [56, 101]. If the device used optical sensing, it may already be equipped with an LED, preventing the need for additional components. Richer forms of output could be provided by a speaker or LCD display, but would require larger form factors and greater power consumption.

5.1.6 Form Factor

The device needs to be small so that it can fit on the users' finger. Ideally, the device would be undetectable when dormant [101]. There are a number of ways which the device could be affixed to the finger. It could be embedded on a ring or thimble like structure worn on the tip of the finger (Figure 5.1a). This would allow users to remove the device when desired, or twist it to deactivate sensing. Alternatively, if small enough, the device could potentially be embedded under the finger nail (Figure 5.1b), on the surface of the fingertip skin, or implanted under the skin with exposed components for sensing (Figure 5.1c) [101]. A larger form factor could be worn above the finger. However this may cause offset problems in the sensing, and could be susceptible to occlusion problems.

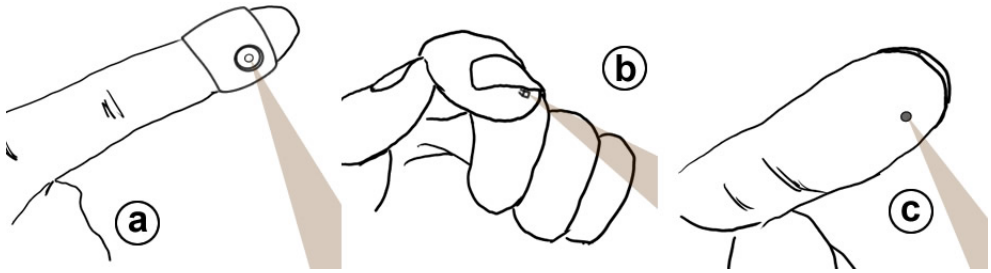

Figure 5.1: Possible form factors for the hardware.

5.2 Magic Finger

After carefully considering the aspects of the hardware design space, we developed Magic Finger, a proof-of-concept prototype that enables always-available input through finger instrumentation. In this section, we describe our implementation, and the basic capabilities of the device.

5.2.1 Hardware Implementation

To achieve sensing of positional movements and sensing of material, our prototype uses two optical sensors. A low resolution high frame-rate optical flow sensor is used to sense 2D finger movements, and a higher resolution, lower frame rate camera is used to sense material (Figure 5.2).

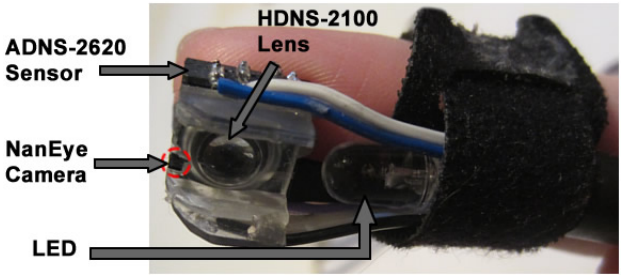

Figure 5.2: The Magic Finger device.

Optical Flow Sensor

We used an ADNS 2620 optical flow sensor with a modified HDNS-2100 lens. This sensor is often used in optical mice, and is reliable on a large variety of surfaces. The sensor detects motion by examining the optical flow of the surface it is moving on. The ADNS 2620 sensor is a low-resolution black and white camera (18×18 63 μm^2 pixels). Our initial tests showed that the sensor could be used to recognize textures, but only at the class scope of recognition, and only with a very small number of items (4).

Micro RGB Camera

To provide an enhanced scope of recognition, we used an AWAIBA NanEye micro RGB camera [9]. In comparison to the optical flow sensor, NanEye has a higher resolution and smaller pixels, and thus captures more subtle details of textures. The micro camera is $1mm \times 1mm \times 1.5mm$ (Figure 5.3). It captures color images at a resolution of 248×248, $3\mu m^2$ pixels, at 44 fps. For image processing, we converted the signal to greyscale. While it would be desirable to also use the NanEye for sensing optical flow, its low frame rate makes this difficult. We expect future solutions could be achieved with a single miniature sensor.

We embedded the NanEye on the edge of the optical flow sensor. To prevent occlusions from wiring, we crop to only use 70% camera's view to 175×175 pixels (Figure 5.2).

Light Source

In order for the camera and the optical flow sensor to function, an external light source is needed to illuminate the surface underneath it. We used a 5 mm white LED, which has similar size and brightness with those used in optical mice. The LED can also be conveniently used for output.

Physical From

Magic Finger is affixed to the fingertip using an adjustable Velcro ring. A small plastic casing holds the components. The case was designed to control the distance between the RGB camera and surface during contact (Figure 1.6).

Computer Interfacing

The optical flow sensor and the LED were connected to an Arduino UNO board [5], which facilitates communication with the computer, an HP TouchSmart Tm2 running Windows 7. The NanEye is connected directly via USB. Tethering the device to a computer was an enabling technology for our prototype, and not our vision for future implementations. See Future Work for a discussion of power and communication capabilities required for an untethered implementation.

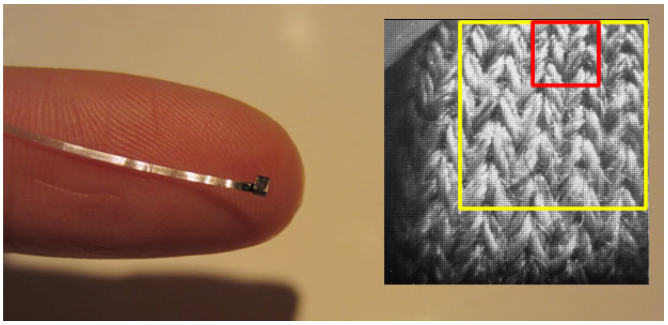

Figure 5.3: Left: the NanEye camera. Right: the field of view is a 175×175 rectangle (yellow region). The red region is used to detect changes in contrast.

5.2.2 Device Capabilities

Relative Positional Movement

Positional movement data is obtained directly from the optical flow sensor using the Arduino. Our program receives (X, Y) coordinates with a resolution of 400 cpi at 15 fps.

Sensing Contact

Magic Finger uses the micro camera to sense contact with a surface. Rapid changes in contrast (due to reflection of the illuminator) are used to signal changes in contact. To detect contact, we continuously calculate the overall contrast of a 60×60 pixel square of the sensor image, by averaging the square difference between each pixel and its neighbors (in grayscale space). When the Magic Finger is more than 5 mm from a surface, light from its built-in LED is not reflected back, and a uniformly darker image is seen. A change within a small window of time form this darker image to a brighter image is measured against threshold values of our contrast metric for changes downward (lift-up) and upward (touch-down) (see Figure 5.4).

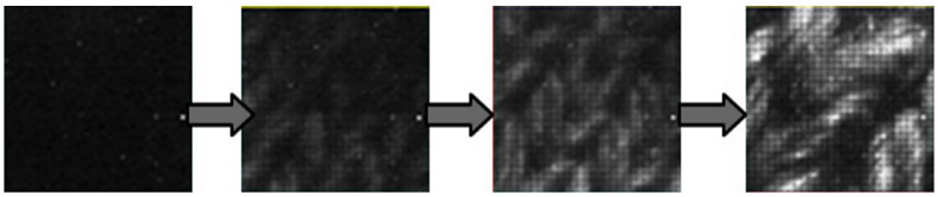

Figure 5.4: Contrast increases with the finger approaches a surface. Leftmost: the Magic Finger is more than 5mm from a surface. Rightmost: the Magic Finger is in contact with the surface.

Identifying Material

When Magic Finger detects a contact event, the current frame of the RGB Camera is used for texture recognition. Textures are classified using a machine learning algorithm. The first step is to describe the texture using a feature vector. In our implementation, we used Local Binary Patterns (LBP)

to retrieve the unique features [102]. The algorithm detects 10 microstructures inside a texture (e.g. edges, lines, spots, flat areas). The histogram of the 10 microstructures uniquely describes a texture. This algorithm is orientation invariant, allowing recognition independent of the angle which the finger touches a surface. The feature vectors are also gray-scale invariant, making it robust in different lighting conditions. To train the classifier, we used Chang and Lin's LIBSVM algorithm using a Support Vector Machine (SVM) [38]. Before we trained the model, we tuned the required SVM parameters that gave high cross-validation scores.

Environmental vs. Artificial Textures

We classify the types of textures Magic Finger can sense as environmental and artificial. Environmental textures are naturally occurring in the user's environment, such as a table or shirt. In contrast, artificial textures are explicitly created for the purpose of being used by Magic Finger. We found that a simple way to create textures was to print a grid of small ASCII characters (Figure 5.5).

Fiduciary Markers

The above classification of textures allows Magic Finger to increase its scope to class recognition. If we are to further increase the scope to instance recognition, further capabilities are required. A common approach to tag specific objects is to use bar codes or fiduciary markers [4, 30]. We used the Data Matrix code, a two-dimensional barcode consisting of a grid of black and white cells. The Data Matrix codes we used were 10×10 cells, representing numerical values between 0 and 9999.

To be properly recognized, the entire Data Matrix must be in the field of view. To ease targeting, we print a cluster of identical Data Matrix codes, each exactly $1/4$ the size of the RGB camera's field of view to ensure at least one code is fully visible. A library is used to decode the tags [124]. When Magic Finger detects a contact event, an API call is made to decode the frame. This call takes approximately 240ms, and is performed before texture classification is performed.

A future implementation of Magic Finger could utilize grids of fiduciary markers, such as is done by Anoto, in place of the ASCII-based artificial textures we designed. Nevertheless, we found that the ASCII-based technique was an effective way to easily generate artificial-textures.

5.3 System Evaluation

We conducted a study in order to evaluate the accuracy at which Magic Finger can recognize environmental textures, artificial textures, and Data Matrix codes. This study was meant to serve as a technical evaluation of Magic Finger's capabilities, not a user study of the device.

For the environmental textures, we collected 22 different textures from a large variety of everyday objects, including public items, personal items, and parts of the user's body. The textures we sampled are shown in Figure 5.5.

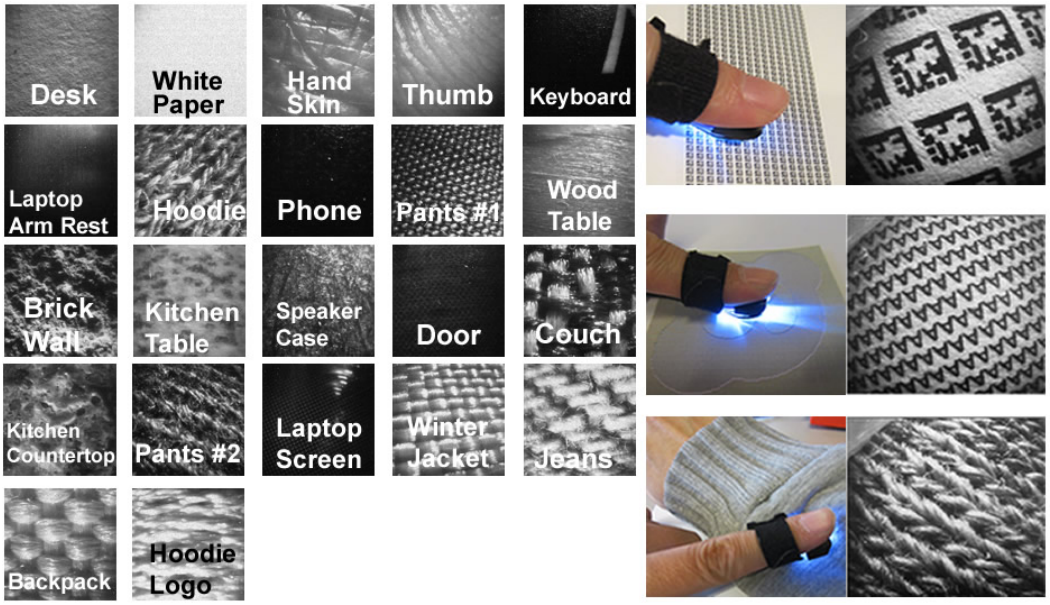

Figure 5.5: Left: 22 environmental textures used in the study. Right: Data Matrix, artificial texture, and environmental texture, as seen by a human and by the NanEye.

To create artificial textures, we printed a grid of ASCII characters in black Calibri font, with a font size of 2pt and a line space of 1.6pt. The textures were printed on a Ricoh Aficio MP $C5000$ laser printer at 1200 DPI. We chose 39 characters found in a standard US English keyboard, including all 26 English capital letters (A - Z) and 13 other characters [˜ ! @ # $ % ˆ & (- + = ."]. For the data matrix, we randomly selected 10 out of $10,000$ possible codes: 1257, 3568, 9702, 3287, 4068, 5239, 8381, 6570, 0128, and 2633, with each printed clusters of 72×15 identical codes.

For each environmental texture, artificial texture, and Data Matrix, 10 samples were collected twice a day for 3 days (as in [52]). Samples in the same class were obtained from the same object but from different locations. In total, we collected 60 samples for each material. The accuracy of recognition was tested using a 5-fold cross validation procedure. Because training and testing data may include points that were sampled in an adjacent time (e.g. points that tend to be similar), we randomized the order of the points prior to the test [52]. The Data Matrix recognition was evaluated by calculating the accuracy of decoding.

5.3.1 Results

Cross validation achieved an accuracy of 99.1% on the 22 tested environmental textures. This is a promising result given that we were only using 175×175 pixels (e.g. 70% of the camera's view). To better understand how the classification accuracy is affected by the resolution of the camera, we cropped the sample images from their center into smaller images of 18 pix^2, 70 pix^2, and 122 pix^2. The results of the 5-fold cross validation are shown in Figure 5.6 (left). Samples of 70 pix^2 achieved 95% accuracy across the 22 textures. This gives future engineers some flexibility when choosing

sensors for finger instrumentation.

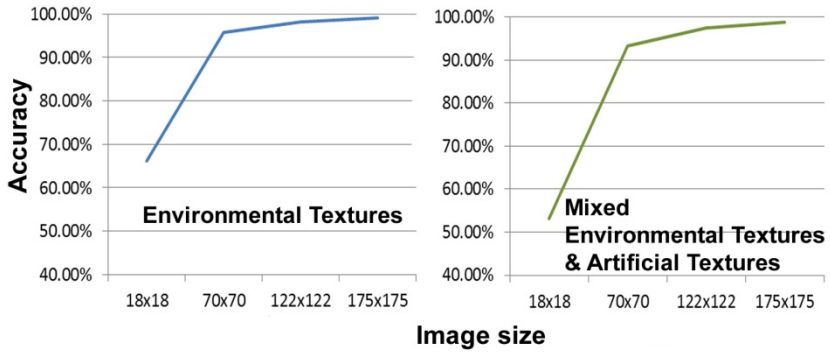

Figure 5.6: Cross validation accuracy. Left: On environmental textures. Right: On mixed environmental and artificial textures.

For the ASCII artificial textures, cross validation yielded 83.8% accuracy with all the 39 tested characters. Given the large number of textures, this result is also promising. The lower accuracy is due to the larger number of textures, and also because the different textures are less visually distinguishable than the environmental textures we selected.

To see how the number of textures impact accuracy levels and find an optimal set of characters to use, we performed another 4 iterations of tests by progressively removing textures that resulted in lower accuracy rates. The accuracy levels through each iteration were 86.4% for 36 textures, 89.9% for 31 textures, 97.9% for 13 textures, and 99.5% for 10 characters [B G I ! # % ^ (+ .].

The above results indicate that Magic Finger can recognize 22 environmental textures or 10 artificial textures with an accuracy of above 99%. We also tested accuracy levels when combining these two sets, for a total of 32 textures. The Cross validation accuracy was 98.9% using the 175×175 pixel image. We also tested the classification accuracy with the smaller cropped images. The results of the 5-fold cross validation are shown in Figure 5.6 (right).

The Data Matrix codes were correctly decoded for 598 of the 600 samples we collected, which yields an accuracy of 99.7%. In both failure cases (one for 3568, one for 9702) the API reported that no Data Matrix was recognized.

The results of our evaluation are very promising. Accuracy levels indicate that Magic Finger would be able to distinguish a large number of both environmental and artificial textures, and consistently recognize Data Matrix codes. Compared to Harrison and Hudson's multispectral material sensing, which was able to classify 27 materials with an accuracy of 86.9% [52], we have obtained an accuracy of 98.9% across 32 textures. Overall, this demonstrates the feasibility of using a Magic Finger device for interactive tasks. In the following sections, we explore the interaction techniques and usage scenarios that Magic Finger supports.

5.4 Interaction Design Space

Having developed a robust device, we sought to develop interaction techniques to facilitate its use. Here, we define the interaction design space for Magic Finger using 5 dimensions, described below.

5.4.1 Input Texture

The input textures vary according to the degree of specificity and information bandwidth they allow. We differentiate our interaction design space by 3 types of textures the user may encounter: environmental textures are textures of real objects, unmodified for Magic Finger; artificial textures refer to textures engineered for the Magic Finger, as we have discussed previously; and dynamic textures are those that change over time. Dynamic textures, for example, can be used to communicate between 2 Magic Finger devices.

5.4.2 Input Type

The possible input type depends on the specific capabilities of an implementation of Magic Finger. That is, our implementation is limited to having the Magic Finger act as a 2-state input device. Thus, input actions can either be tap input (sensing contact), or gesture input (positional movement).

5.4.3 Texture Mapping

The ability to recognize different textures allows contextual actions to be carried out by the Magic Finger. The mappings of texture to result vary by the degree with respect to their artificiality: Intrinsic mappings rely on the semantics of the touched object to define the mapping. For example, tapping on the empty space of a phone always triggers a function of the phone, e.g. fast dial a number. Intrinsic mappings may suffer from a lack of flexibility, and conflicts in mappings. In contrast, extrinsic mappings associate commands to unrelated physical objects. Such mappings are flexible but may be less discoverable or memorable.

5.4.4 Possible Interaction Types

Many types of interactions could result from user input. We classify three main types: perform a command execution, provide continuous control, or serve as a mode delimiter.

5.4.5 Feedback

Based on needs, feedback can be internal or external to Magic Finger. For example, internal feedback could provide information using an LED while richer external feedback could be provided by an attached or secondary device.

5.5 Authoring Environment

We created an application to help users manage their registered and recognized textures (environmental and artificial) and their associated functional mappings. A window displays a list of the defined texture classes (Figure 5.7 (left)). Each texture class in the list contains a sample image of the texture and a user defined title for that texture class. Users can double-click the sample images to view all the samples of that object that are used in training the SVM model. A button in the user interface can be used to create a new texture class. Users can also add samples to a new or existing texture class as input to the recognizer.

Each texture class also has an associated dropdown menu, which shows a list of functions which are preset resulting actions that could be mapped to any texture class (Figure 5.7 (middle)). For the purposes of demonstration, we have predefined a variety of resulting actions which can be mapped to Magic Finger input. In some cases, the authoring environment allows the user to adjust parameters of an action. For example, if the user selects the "Send SMS" action (which automatically sends an SMS message) from the drop down list, a pop up dialog is displayed where the user can enter a recipient number and message. To automatically launch external applications, the user can select "external application" from the action drop down list. The user can then drag-and-drop an icon of a desired application onto the menu item (Figure 5.7 (right)). This allows the user to link Magic Finger actions to existing applications, as well as to define their own scripts in their preferred languages and link them to Magic Finger.

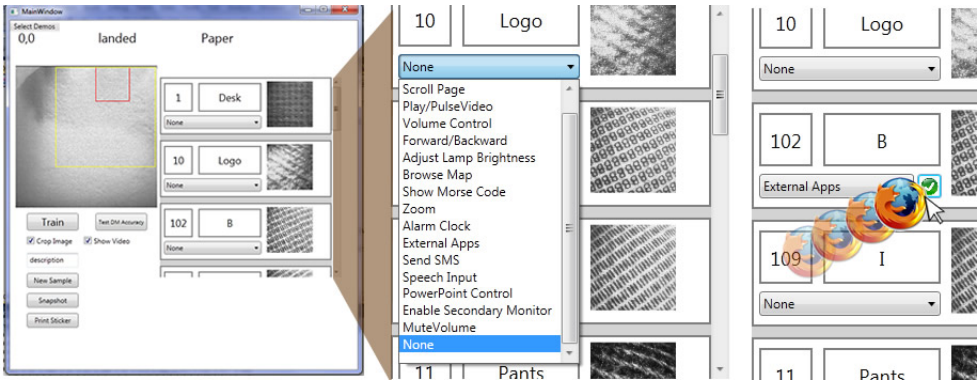

Figure 5.7: Left: list of texture classes. Middle: the drop down list of functions; Right: external applications.

5.5.1 Authoring Artificial Textures

The authoring application also allows users to generate ASCII-based artificial textures which can be used, for example, as stickers to be placed on objects. In these textures, the individual ASCII characters are so small that they are indiscernible to the human eye, and can be used as pixels to create two color graphical images. While each "pixel" is the same character, the pixels can take-on

a foreground or background color. We chose two colors that are distinguishable by the human eye, but are indistinguishable in greyscale during recognition (Figure 5.8 (right)).

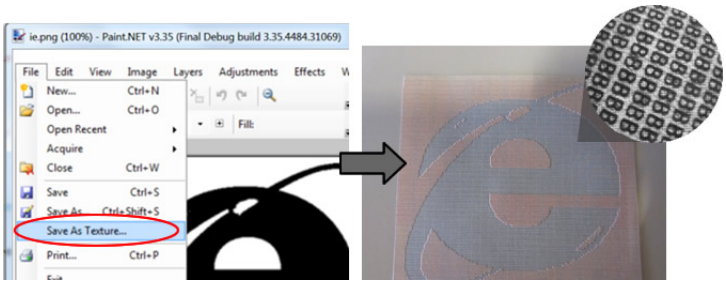

Figure 5.8: Left: the paint.net application; Right: the resulting texture.

To author these textures, we modified Paint.NET, an image editing application, to include a "save as texture" option (Figure 5.8 (left)). This option prompts the user to enter a character, and then converts the current image into an HTML document consisting entirely of that character rendered in either the foreground or background color. Such images can be used to give the artificial textures user friendly appearances, such as icons representing their functions.

5.6 Interaction Techniques

In order to explore the interaction design space we have described, we implemented a number of interaction techniques. Each of the following techniques serves as an exemplar of a point in the design space, as well as a portion of a coherent set of possible uses for Magic Finger. Some of the interactions are novel while others show how Magic Finger can be used to implement previously published techniques that required numerous different hardware configurations. In the example interaction techniques below, we explicitly mention the corresponding design dimension.

Some of the below interactions involve multiple Magic Fingers. To implement these scenarios, we constructed a second Magic Finger without the RGB camera. This device was unable to sense textures, but was sufficient for demonstrating the desired multiple-device interactions.

5.6.1 Mixed Input Mode

Magic Finger supports both direct and indirect input on touch-screens. The current prototype supports the *mixed input mode* on resistive touchscreens. Future product with smaller form factor (as shown in Figure 5.1b and Figure 5.1c) allows the *mixed input mode* to be carried out on capacitive touchscreens.

Supporting the *mixed input mode* on environmental and artificial texture can be tricky. However, Magic Finger supports absolute positioning for artificial textures if location is encoded into the textual pattern (e.g. Anoto [4]). For natural textures, absolute positioning may never be possible, although with improved sensing, location could be sensed at higher granularity (e.g. recognize

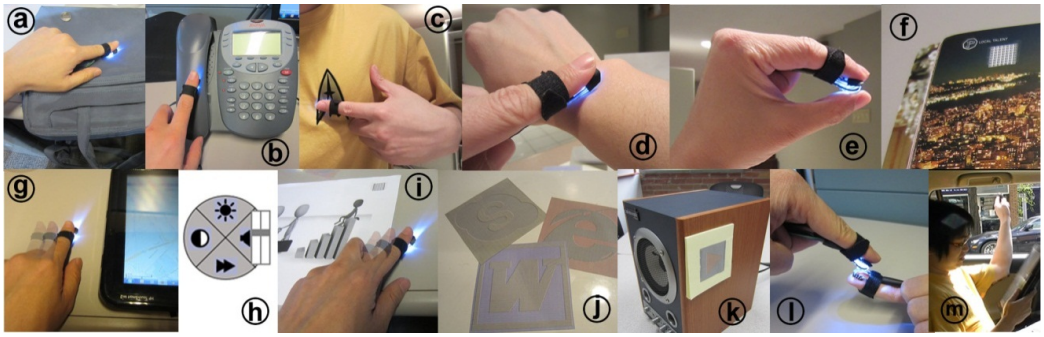

Figure 5.9: Tap and Gestural-based Interactions. (a) Tap on the bag to mute a Skype call; (b) Tap on the phone to send a frequent SMS message; (c) Using the logo of a t-shirt as a mode delimiter; (d) Tap the wrist to check an upcoming appointment; (e) Pinch gesture (f) Data Matrix Buttons; (g) Occlusion Free Input; (h) FaST slider widget; (i) Multi-surface crossing gesture; (j) Application launch pad; (k) Sticker as disposable remote controls; (l) Passing texture to another Magic Finger; (m) Use Magic Finger as a periscope to check out traffic.

locations of skin rather than simply "skin").

5.6.2 Tap Input

PocketTouch

Magic Finger can be used to remotely access or control mobile devices. We implemented a resulting action of muting an incoming Skype call (command execution), and mapped this to touching a handbag. If the user receives an unwanted call when their Smartphone is in the handbag, the user can simply tap the handbag to mute the notification. This interaction is much like PocketTouch [114], but does not require a phone's capacitive sensor to be placed in a special mode, or require the phone to be placed at a specific location (Figure 5.9a).

Tap-to-Talk

Because Magic Finger can identify the object that users touch, tapping different surfaces can trigger different command executions. In our implementation, the empty surface of a phone is used as a shortcut 'button' to send a frequent SMS message to a predefined recipient, e.g. "I'm on my way home." (Figure 5.9b). This is an example of an intrinsic mapping. Users may also take advantage of inherent features of physical artifacts to define input textures. We created an extrinsic mapping of tapping on the logo of a t-shirt, to launching the Windows Voice command app. Thus, tapping on this special area of the shirt serves as a mode delimiter for entering speech input (Figure 5.9c).

Skinput

By recognizing skin as a texture, Magic Finger provides a method for using skin as an input surface, without requiring additional sensors. We utilized this to create an "am I late?" gesture - when the user taps their wrist (where a watch might have been - intrinsic mapping), Magic Finger checks for

an upcoming appointment and either displays it onscreen (external feedback), or blinks its built-in LED (internal feedback) to indicate "time to head out!" (Figure 5.9d).

Pinching Finger to Thumb

The finger to thumb pinch action has been shown to be a useful gesture for freehand input [131]. By recognizing the thumb as an environmental texture, Magic Finger can easily detect a pinch when the thumb is tapped (Figure 5.9e). We mapped pinch to a command execution of advancing slides in a presentation, as a replacement for holding a wireless presentation mouse.

Data Matrix Buttons

In addition to tapping environmental textures, users can also tap artificial ones. A grid of printed Data Matrix codes can be used as physical buttons. The decoded numeric data can be used to trigger a command or to retrieve contextual information. In our implementation, we assigned specific numbers to PowerPoint documents. A user can print the PowerPoint slides with the associated Data Matrix code at the bottom of each page. Tapping the code opens the presentation on the user's computer. We also associated codes on magazine articles with webpages (Figure 5.9f). Tapping the Data Matrix opens a URL associated with the article.

5.6.3 Gesture Input

Occlusion Free Input

Occlusion Free Input allows users to interact with a touch screen device by gesturing on a nearby surface. This eliminates the occlusion of the screen by a users' hand. We implemented a navigation application, where users can pan or zoom a map displayed on a tablet device (continuous control). By mapping this action to the back of a tablet, users can carry out back-of-device input [130], without needing a specially instrumented tablet device. Additionally, SideSight [23] can be emulated by mapping the action to the table that the device is resting on (Figure 5.9g).

FaST Sliders

We implemented a FaST slider widget to control volume and playback of multi-media [89] (Figure 5.9h). FaST sliders allow manipulation of multiple, discrete or continuous parameters from a single menu. Once activated, the user can select a parameter to manipulate with a directional mark. While the finger remains down, the user can then adjust the value of the selected parameter by dragging. If the user is close to a display device, visual feedback for the menu can be displayed onscreen (external feedback). When a display is not available, we blink the Magic Finger's LED to indicate when a parameter has been selected, so the user knows to begin parameter adjustment (internal feedback).

Multi-Surface Crossing Gestures

Magic Finger can detect a crossing gesture from one surface to another (Figure 5.9i). This is accomplished by performing texture recognition any time the finger dwells while still in contact with a surface. We mapped this gesture to activating a slide deck on a projector in a meeting room. The user can tap a Data Matrix on their slide printout to open the PowerPoint, and tap the table to connect the laptop to a secondary monitor (i.e. projector). If a crossing gesture is detected from the Data Matrix to the table, the PowerPoint window is moved from the laptop display to the projector. In this case, the crossing gesture has an intrinsic mapping, as it represents a relationship between the two mapped surfaces. These multi-surface gestures are similar to stitching [62], but do not require inter-device communication between the surfaces.

5.6.4 Input Stickers

Artificial textures can be used to increase the number of distinguishable surfaces in an area. By printing the artificial textures on adhesive strips, we create Input Stickers. We describe the associated interaction techniques below.

Application Launch Pad

We printed a set of Input Stickers that take-on the appearance of desktop icons (Figure 5.9j). The user can place these stickers in a convenient location in their work area, and tap the stickers to launch the associated desktop application. This emulates functionality shown in MagicDesk without requiring the entire desk surface to be a touch sensitive surface [20].

Disposable Remote Controls

Input stickers can also be used as remote controls. We printed a stack of stickers and put them beside the physical on-off switch for the speakers in a public room (Figure 5.9k). When entering a room, a user can pick up a sticker before sitting down. Tapping the sticker plays and pauses music on the room's sound system. When leaving the room, the user can replace the sticker on the stack, or dispose of it.

5.6.5 Additional Features

Identify Awareness

An interesting property of Magic Finger is that it is inherently user-identity aware. We configured the music remote control, described above, to play a user's favorite type of music, based on the identity of the user tapping the sticker. Traditional touch devices are not able to recognize users, without specialized hardware and making potentially error prone inferences [111, 138].

Passing Textures

In some cases, a user may want to virtually pass a texture that has been read by the Magic Finger to a remote receiver or to another user. We use the pinch gesture as a mode delimiter for "picking up" a texture. If a pinch is detected immediately after a Data Matrix is tapped, then the Data Matrix code is temporarily stored. When the pinch is re-leased, the LED emits a Morse code pattern, representing the 4 digit number associated with the Data Matrix. We implemented a simple product scanner application that uses a webcam to read the Morse code and display information about the associated item, such as its price. We also implemented a second Magic Finger that can read the Morse code as a dynamic texture and carry out the associated functions of the Data Matrix code after it is passed to the user from the source user (Figure 5.9l).

The Periscope

People often use their fingers to sense areas that they cannot see. For example, we may use our hand to search for a pen dropped between the cushions of a couch. Magic Finger allows users to enhance the capabilities of such probing. By viewing its real-time video feed on a handheld device, the Magic Finger becomes an extension of the user's eyes. For example, a user can look behind them, around a corner or out the window of a car by simply pointing their finger in the associated direction (Figure 5.9m).

5.7 Discussion

Having used Magic Fingers, we have made several observations which may be of benefit to those seeking to implement one. In this section, we discuss some of the insights we have gleaned from our experiences.

Device Size and From Factor

While our form factor is small enough to fit on a fingertip, it is still larger than what we envision for the future.

The hardware does have room to be made much smaller. Our prototype is limited by the size of the ADNS 2620 chip. The chip was designed to be used in a mouse, and so its form factor is optimized to fit with the other components of the mouse. The core sensor of the chip, however, is only 2×3 mm wide. Therefore, reengineering the chip to fit the sensor will significantly reduce the size of Magic Finger.

The NanEye camera has an ideal form factor. However, it is limited by its frame rate. Ideally, Magic Finger would need only one optical sensor to handle all of its functionality. For optical flow to be detected reliably, the sensor would need to have a frame rate of approximately 1500fps. Based on our evaluation, the resolution should be at least 70×70 pix to enable both optical flow and

texture recognition. Given existing technologies, we are optimistic that a sensor matching these specifications will soon be available.

The other component that would need to be miniaturized is the LED. Micro LED's are available today; testing would be needed to determine minimum brightness requirements.

Power and Communication

A limitation of our hardware is that it is tethered to a near-by host PC. This simplified our implementations, but for Magic Finger to become a completely standalone device, power and communication needs to be considered. Holz *et al.* provide a thorough review of potential technologies that can be used for power and communication in a micro form factor [67]. Powering can be accomplished through rechargeable batteries, or harvesting power from the body or the environment [67]. Communication can be provided through a Bluetooth protocol, which consumes little power. Processing responsibilities could be performed by a Blue-tooth-tethered mobile phone.

5.8 Summary

This chapter introduces the concept of finger instrumentation and our prototype Magic Finger, a novel device that supports *mixed input mode*, and extends users' touch capability to everyday objects. Instead of requiring instrumentation of the user's environment, or external cameras which may be prone to occlusion, Magic Finger is unique in that the finger itself is instrumented to detect touch. Our system evaluation showed that Magic Finger can recognize 32 different textures with an accuracy of 98.9%, allowing for contextual input. We presented a design space of interactions enabled by Magic Finger, and implemented a number of interaction techniques to explore this design space. Future work will focus on investigating topics such as "Midas touch" and false activations, and on understanding how users will use the Magic Finger. We will also explore hardware solutions for miniaturizing the form factor of the Magic Finger.

Chapter 6

Reconfiguring Existing Software for Mixed Input Mode - TouchCuts & TouchZoom

Table 6.1: Hardware prototypes and corresponding interaction techniques - TouchCuts & TouchZoom[6]

User Interface / Input Platform	Hardware UI	Software UI
Mobile	*LucidCursor*	*Dual-surface Input*
Desktop	*LensMouse*	**TouchCuts & TouchZoom**
Always-available Input	*Magic Finger*	*Magic Finger Interactions*

In the previous chapters, we presented our design of hardware solutions for the *mixed input mode*. Note that the success of *mixed input mode* also relies on the design of software user interfaces. In the rest of the thesis, we describe the challenges we face with the existing software UIs. We also propose our solutions to enhance the UIs to better facilitate *mixed input mode*.

Recently, computer manufacturers have started producing laptops equipped with touch-screens, allowing users to use both mouse and fingers to interact with their applications. Therefore, input to touch-screen laptops is by default using *mixed input mode*. However, current desktop applications often use ribbon or tool palette to host small and tiled icons that are difficult to select by finger due to occlusion [128] and accuracy [40] problems. We explore the design space of novel interaction techniques to facilitate the use of *mixed input mode* input in popular legacy applications, e.g. MS Word. We propose the following design guidelines.

1. *Global Optimization*: the new UI components should be optimized for both touch and mouse input. For instance, the widgets should be large enough to be 'touched' by finger, while still

[6]The video of TouchCuts & TouchZoom can be found at: `http://www.youtube.com/watch?v=Vj3d1hiPvuM`

take minimum screen real estate during mouse input.

2. *Effective*: the new interaction techniques should guarantee a good performance (e.g. efficiency and accuracy) for both input modes.

3. *Minimal transformation cost*: it should not require designers to create a different UI paradigm for each input mode. This not only eliminates extra implementation cost from application designers, but also minimizes the learning efforts from users when switching between the two input modes.

4. *Easy to use*: The new interaction techniques should be intuitive and easy to use.

5. *Easy to learn*: The newly designed UI components should have a similar interface paradigm as those from the existing applications. This can ensure minimal learning from users.

In this chapter, we first describe a study investigating the benefits of touch input in a laptop configuration. We then introduce our new techniques, TouchCuts, a single target expansion technique, and TouchZoom, a multiple target expansion technique, both designed to reduce high error rates found for small targets. Finally, we describe a controlled study, showing that our techniques improve performance in comparison to both the computer mouse and a baseline touch-based target acquisition technique, Shift [128].

6.1 Experiment 1: Evaluation of Touch Input

We believe touch input could be particular useful in a laptop configuration, since a mouse may not be available, and the hands, when in a resting state on the keyboard are already quite close to the display. However, despite the prevalence of touch-based laptops, the efficiency of using touch on such devices has not been investigated and is thus not fully understood. To better understand if and when target expansion techniques would be useful, we first study traditional target acquisition in a laptop configuration. While previous work has compared touch to other forms of input [40, 92, 117], on screen target acquisition is unique, since the hand would be moving from a horizontal plane of the keyboard to a vertical plane of the display. In addition, strictly following the KLM GOMS model, we would predict that such a task would require homing time (switching to the touch input device) and pointing time (acquiring the target). We postulate that one of the primary benefits of using touch for target selection tasks is allowing pointing and homing to take place in parallel. We are unaware of any investigation dealing with this issue.

Motivated to answer these open questions, the goal of this experiment is to evaluate the performance of touch versus a mouse and touchpad in a task involving the use of a pointing device in conjunction with a keyboard.

6.1.1 Apparatus

Our study was conducted on a Dell SX2210 21.5" touch-screen monitor, as it provided more accurate touch input then current touch-enabled laptops. The display was used in conjunction with a standard desktop keyboard. To simulate a laptop configuration, we lowered the height of the touch-screen so that the button of the display is 4 cm above the keyboard, which is about the same distance that can be found on a Dell TouchSmart TX2 tablet PC. The display was tilted to a comfortable viewing angle (13° to the vertical plane). A USB touchpad (97 × 78 mm) from Ergonomic was placed below the space bar of the keyboard, and was raised to the same height as the keyboard (Figure 6.1).

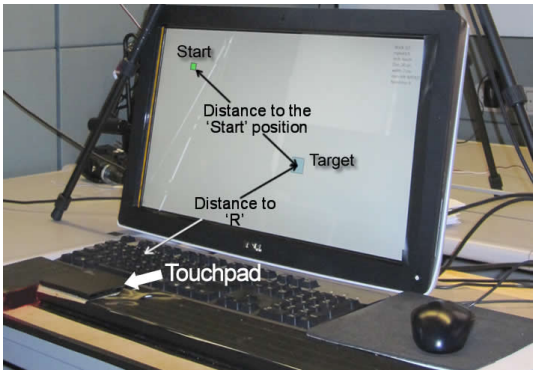

Figure 6.1: Hardware setup for Experiment 1.

6.1.2 Task and Procedure

The task required participants to click a key on the keyboard and then using the same hand select a square target of various sizes in a random location on the screen. This task is analogous to that frequently employed by users of text editing programs, where the main task is typing on a keyboard but requires users to switch to a mouse or a touchpad to click an icon or other on-screen widget. The right hand was used for the mouse and touchpad. The left hand (non-dominant for all participants) was used for touch, since we felt testing touch with the dominant hand may be biased, since in some cases users may want to use the non-dominant hand (for example, if a target is on the left side, or if their right hand is on the mouse). Participants were told to only use their index finger during the touch conditions.

In the mouse and touchpad conditions, participants were asked to position a cursor inside a 0.5 × 0.5 cm "Start" square prior to the start of a trial. A trial started after participants pressed a keyboard key and finished after a target was successfully selected. Ideally, we would have like to use the "F" key when using touch and the "J" key when using the mouse and touchpad, as these are the typical resting keys for the left and right hands respectively. However, we used the "R" key for touch and the "\" key for mouse, and the "J" key for touchpad since the distance between these keys and their respective input devices more closely matched the distances on a Dell TouchSmart TX2

tablet PC.

Participants were asked to finish the task as fast and as accurately as possible. They were encouraged to take breaks during the experiment, which lasted about 40 minutes.

6.1.3 Design

The experiment employed a $3 \times 4 \times 3$ within-subject factorial design. The independent variables were *Pointing Device* (*Finger*, *Mouse*, and *Touchpad*), *Target Distance* (18, 24, 30, and 36cm), and *Target Size* (0.5, 1, and 2cm).

The size of the target was chosen to be close to the size of the icons in real-world applications. For instance, 0.5cm is approximately the same size as the ***Bold*** button in Microsoft Word. Similarly, 1cm is approximately the same size as the *Paste* button. Target distance was measured from the center of the goal target to the center of the 'Start' position in the mouse and touchpad conditions. Distance was measured from the center of the "R" key to the center of the goal target in the touch condition. Target locations were the same for all conditions. The "Start" square was repositioned (in the mouse and touchpad conditions) according to the target position to ensure it satisfied the distance condition.

Windows cursor acceleration was turned on to facilitate pointing by using the cursor. If the user missed the target, they had to click again until successful. In each trial, participants performed tasks in one of each $Pointing\ Device \times Target\ Distance \times Target\ Size$ combination. The experiment consisted of 5 blocks, each consisting of 3 repetitions of trials. The first block was used as practice trails, thus the data was not used in analysis. The *Pointing Device* was counter balanced among participants. The *Target Distance* and *Target Size* were randomized among trials.

6.1.4 Participants

Twelve paid participants (7 males and 5 females) between the ages of 21 and 56 participated in this study. All participants were right-handed. They were all familiar with a computer mouse and touchpad, and had previous experience with mobile touch-screen devices.

6.1.5 Results and Discussion

Dependent measures included the number of errors and the average task completion time. This data was analyzed using Repeated-measures ANOVA and Bonferroni corrections for pair-wise comparisons.

Task Completion Time

ANOVA yielded a significant effect of *Pointing Device* ($F_{2,22} = 14.43$, $p < 0.001$), *Target Distance* ($F_{3,33} = 3.61$, $p < 0.05$), and *Target Size* ($F_{2,22} = 101.72$, $p < 0.001$). There is also significant interaction effects on $Input\ Device \times Target\ Distance$ ($F_{6,66} = 8.68$, $p < 0.001$), $Input\ Device \times$

$Target\ Width$ ($F_{4,44} = 29.07$, $p < 0.001$), and $TargetDistance \times TargetWidth$ ($F_{6,66} = 5.53$, $p < 0.001$). The interaction effects were mainly caused by the poor performance of finger touch on the target of 0.5cm (see Figure 6.2).

Overall (including trials with errors), the performance of finger (1528ms) was significantly faster than mouse (1639ms) ($p < 0.05$), which was significantly faster than touchpad (2242ms) ($p < 0.001$). As expected target size has more impact on finger than mouse or touchpad (Figure 6.2 (left)). Participants spent more time selecting the smallest target using finger than using mouse or touchpad.

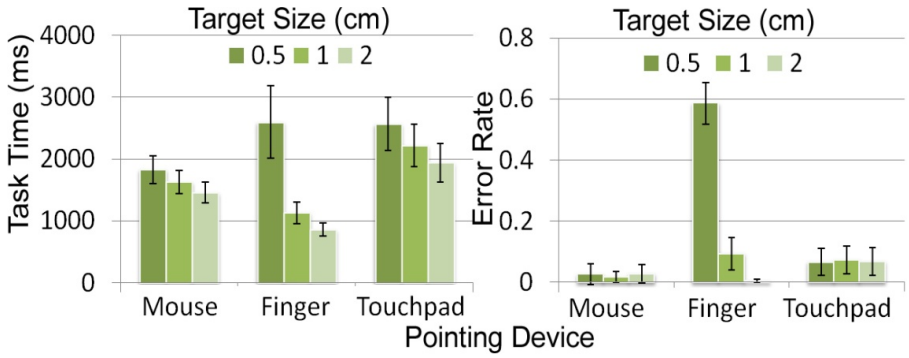

Figure 6.2: Task time and error rate shown by Technique and Target Size. (Error Bars show 95% CI in all figures).

Number of Errors

ANOVA yielded a significant effect of *Input Device* ($F_{2,22} = 57.46$, $p < 0.001$), *Target Distance* ($F_{3,33} = 7.68$, $p < 0.05$), and *Target Size* ($F_{2,22} = 202.36$, $p < 0.001$). We also found significant interaction effects on $Input\ Device \times Target\ Distance$ ($F_{6,66} = 4.18$, $p = 0.001$) and $Input\ Device \times Target\ Width$ ($F_{4,44} = 225.43$, $p < 0.001$).

Overall, touch made significantly more errors (23%) than touchpad (7%) ($p < 0.001$), which made significantly more errors than mouse (2%) ($p < 0.001$). Figure 6.2 (right) shows that participants made closed to 60% errors using touch on target size 0.5 cm but made less errors than mouse on target of size 2 cm.

Fitts' Law Analysis

To perform a Fitts' Law analysis, we removed all trials in which errors occurred. Linear regression tests indicated that the task highly conformed to Fitts' Law, for all three conditions, showing that touch is constantly faster than mouse or touchpad across all index of difficulties (Figure 6.3). It can be seen that the main difference is due to the 'a' constant, which is typically reaction time, but for this study, encompasses the homing time as well. This is an interesting results, as it shows that touch does allow homing and pointing to be carried out concurrently. To repeat the analysis without homing time, we subtracted the elapsed time until the cursor began to move in the mouse

and touchpad conditions. After doing so, we still see a 16% advantage of touch over the mouse ($p < 0.001$), and a 35% advantage over the touchpad ($p < 0.001$).

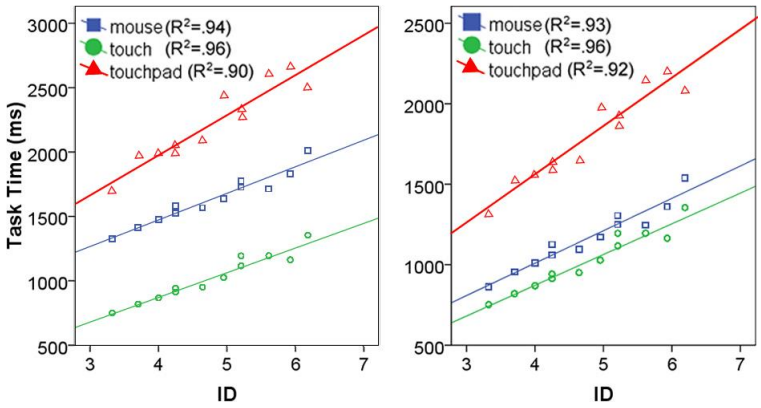

Figure 6.3: Task completion time with homing time (left) and without homing time (right) by the index of difficulty.

6.1.6 Summary

The study demonstrates certain benefits of using touch in a routine task, which requires users to switch from a keyboard to a pointing device prior to start acquiring a target. For targets that are at least 1cm large, touch was 36% faster than the mouse and 52% faster than the touchpad. Even without the homing time, touch was 8% faster than the mouse and 35% faster than the touchpad. However, as expected, the performance of touch decreased significantly with small targets. In particular, our study shows that touch completely fails for target sizes of 0.5×0.5cm. Unfortunately, many graphical user interfaces contain targets of this size, so further considerations must be made for touch to be practical on desktop applications.

6.2 Expanding Target Techniques for Touch

The results of Experiment 1 provide an important lesson: if targets are big enough, using touch to acquire them can have significant advantages. However, in desktop applications, increasing the size of the targets is not practical, as it would diminish the experience for users who never intend to use touch. It is also impractical to expect traditional legacy applications to be rewritten specifically for touch. Instead, we propose that UI components transition to be optimized for touch only when the user intends to use touch. Already, numerous technologies exist to detect the proximity of fingers [31, 125, 85, 65, 110]. Expanding the targets as a finger approaches the screen could be an efficient mechanism to overcome the challenges encountered in Experiment 1. In this section, we initiate the investigation of expanding targets for touch, through the design of two techniques: TouchCuts and TouchZoom.

6.2.1 TouchCuts

TouchCuts are a basic implementation of expanding targets for touch, where only certain targets within a UI palette, such as a toolbar or ribbon, expand when the finger approaches it. Only this subset of targets is accessible through touch, as surrounding targets may become occluded by them. As such, TouchCuts are akin to keyboard shortcuts, providing efficient access to some of an application's commands. However, they are not exhaustive or do not provide a replacement for other command access methods. TouchCuts use a visual gloss overlay, to indicate to the users which of the targets are accessible through touch. We also implemented a simple method to allow users to customize which targets are TouchCuts. The user taps close to a desired target using their finger, and drags the mouse to specify the size of a TouchCut (Figure 6.4). Similarly, tapping on a TouchCut, followed by a mouse click on the enlarged button can remove it. Using a transparent overlay window, we were able to prototype an application independent implementation of TouchCuts, which could be used to customize and access TouchCuts on any windows program.

Figure 6.4: Left: define a TouchCuts. Right: TouchCuts expands when a finger approaches.

The advantage of this technique is that it is a single target expansion, so no target prediction is required, and no targets are displaced. The limitation is that it cannot provide access to every target within a tool palette. Therefore, TouchCuts are most suitable for functions that are used frequently. In the case when a function is not available through TouchCuts, the user will have to use a mouse or re-customize the TouchCuts.

6.2.2 TouchZoom

We also wanted to develop a technique that could provide full access to a palette's icons. We focus our design on the ribbon, because it has become a common UI component for desktop applications, but the technique would work for any horizontal tool palette. Our approach is to magnify the entire ribbon, with a center of expansion at a predicted endpoint, similar to what Zhai *et al.* previously proposed [137]. This ensures expansion of the desired target, even if there is an error in the endpoint prediction. Thus, unlike TouchCuts, the TouchZoom makes every target within the ribbon accessible through touch. One concern, which will require investigation, is that the goal target may become offset from its original location, and in the worst case, the target may be displaced off-screen.

For prediction, a 2D motion vector is generated based on the projection of the finger movement on the screen. We use only the last 2 points of the sample to estimate the intended vector, as it tended to perform better than other regression algorithms we implemented. The intersection of the

motion vector and the top of the ribbon determines the center of expansion (*CE*), and the ribbon expands once the index finger crosses an expansion point in a sufficient speed. If *CE* falls outside of the ribbon, it will be placed on the corresponding ribbon endpoint. The *CE* was fixed at the top of the ribbon in all cases. When the index finger crosses a certain distance threshold, the *CE* is calculated and the resulting ribbon expansion occurs (Figure 6.5). When the prediction is wrong, the goal target will be offset from its original location at a magnitude proportional to the prediction error. To recover from an off-screen error, users can either re-launch the expansion by moving below the distance threshold, or scrub the ribbon with a touch gesture to pan the target back into view.

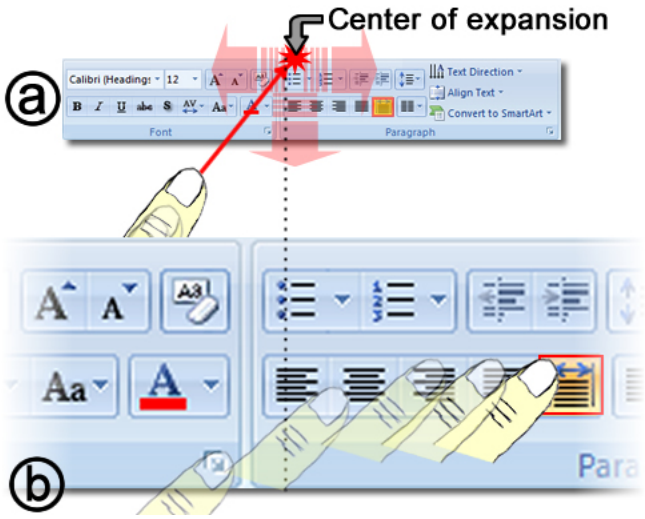

Figure 6.5: TouchZoom. (a) The ribbon expands at the intersection of a finger's motion vector and the top edge of the ribbon. (b) After the expansion, finger movement is adjusted to acquire the new position of the highlighted goal target icon.

We anticipated that the accuracy of the prediction would be dependent on the time when the ribbon expansion is triggered. Expanding late might lead to better prediction since the finger would be closer to its goal. However, considering that no prediction algorithm works perfectly [82, 90, 91], a user will need a certain amount of time to perceive target displacement, and to adjust his/her finger motion accordingly. Thus, it may be preferable to expand the ribbon early.

Another design option we considered is to group sets of ribbon icons together, and to set the *CE* to the center of the predicted group (For groups adjacent to the screen edge, the *CE* would be set to the edge) (Figure 6.6). This would allow a user to aim his/her finger movement at the group containing the desired icon, instead of aiming at a desired icon itself. Having this larger initial target would minimize prediction errors, and, once expanded, the user could adjust his/her finger movement to make the final selection. However, the target offset would be proportional to the distance between the target and the center of its group.

In the following section, we study the relevant parameters for our TouchZoom design. In Experiment 3 we will compare TouchZoom to TouchCuts and two baseline techniques.

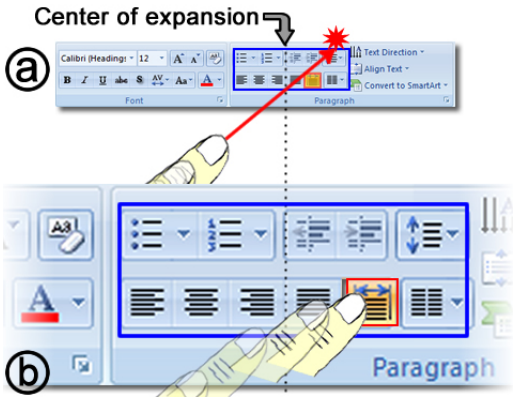

Figure 6.6: (a) The ribbon expands at the center of a group of icons shown in the blue outline. (b) User adjusts finger movement to the new position of the highlighted target icon.

6.3 Experiment 2: Evaluation of TouchZoom

In this study, we were interested in measuring the impact of the pertinent design parameters for TouchZoom, to optimize the techniques efficiency. The parameters we investigated were group size, target position, and expansion point.

6.3.1 Apparatus

The hardware setup was the same as in Experiment 1. In addition, we used four OptiTrack motion capture cameras to capture the off-screen movement of the user's index finger. The cameras have 100Hz frame rates and millimeter accuracy. We put 3 reflective markers on a user's hand (Figure 6.7). The marker on the user's index finger was tilted to the right size to avoid occlusion. This should be considered enabling technology only, simulating the more practical technologies discussed earlier.

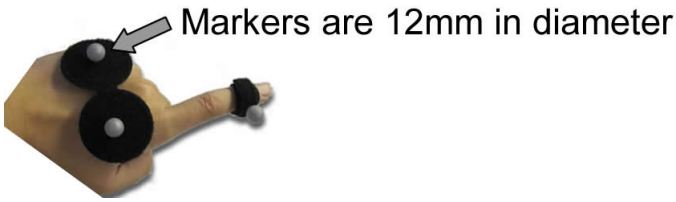

Figure 6.7: Reflective marker placement.

6.3.2 Task and Procedure

The task required participants to use their left index finger to press the "O" key on a keyboard and then select a target button in our abstracted ribbon using the same finger. Upon pressing the "O", the participants were instructed to select the target as fast and as accurately as possible. A trial started after the "O" was pressed and finished after a target was successfully selected. Participants were

encouraged to take breaks during the experiment. The entire experiment lasted about 45 minutes.

6.3.3 Design

The ribbon was 1.5cm high and 45cm width, and was rendered near the top of the screen in a window which has the same width as the ribbon and the same height as the screen. A 100ms animation was used for the Ribbon expansion. In each trial, a 0.5×0.5cm goal target was shown in the ribbon. An expansion magnification level of 3x, which was based on Experiment 1, was used to reduce the chance of errors. The ribbon had 90 icons across, and 3 icons in each column. The "O" key was centered with the ribbon.

The experiment employed a $5 \times 2 \times 2 \times 9$ within-subject factorial design. The independent variables are *Group Size* (1, 3, 9, 15, and 30), *Expansion Point* (*Late* and *Early*), *Target Y Position* (*Up* and *Down*), and *Target X Position* (1 to 9).

Group Size (GS) - Group size indicates the number of icons in a row that a group has. We chose to explore a range of groups size, that evenly divided into the 90 targets across: 1, 3, 9, 15, and 30.

Groups were visualized using vertical bars (Figure 6.8). For targets close to the screen edges, the participants were shown that by biasing their acquisition movement towards the edge of the screen, off-screen errors could be minimized.

Expansion Point (EP) - The distance to the goal target was measured as the distance from the "O" key to the target in 3D space. Expansion Point took on the values 90% and 40%, representing the amount of distance travelled to the targets, before the expansion occurred. We chose 40% as a minimum value because our informal test showed that the distances below 40% could significantly impair the prediction. The value of 90% was chosen as it has been suggested by previous expanding target techniques [90].

Target Y Position (TY) -In the Up condition, the target was placed on the top row. In the Down condition, the target was placed in the bottom row.

Target X Position (TX) - In each trial, the target was placed in one of 9 different horizontal positions across the ribbon (see Figure 6.8). In addition to the 9 absolute positions, we were also interested in investigating the effects of 3 relative positions (left, middle, and right) of a target within a group of the ribbon. To ensure each group size had exactly 3 targets in each of these relative positions, we slightly shifted some of the X positions by 1 icon.

The experiment consisted of 3 blocks, each consisting of 1 trial for each combination of $Group-Size\ (GS) \times Expansion\ Point\ (EP) \times Target\ Y\ Position\ (TY) \times Target\ X\ Position\ (TX)$. The order of *Group Size* was randomized between subjects. Within each group size, *Expansion Point* was randomized. Finally, target position was randomized in each $GroupSize \times ExpansionPoint$ pair.

Dependent measures included the number of off-screen errors, the number of selection errors, and the average task completion time. An off-screen error was recorded when a target was pushed

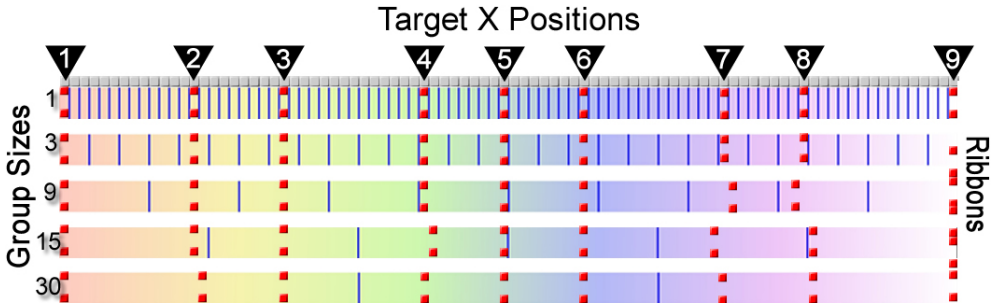

Figure 6.8: Illustration of target positions (red dots) within each of the 5 group sizes. The gradient effect was added to provide spatial grounding.

into off-screen space. A selection error was recorded when a participant missed a target. Task completion time was recorded as the time elapsed from the "O" being pressed to a successful selection on the target.

6.3.4 Participants

Ten paid participants (5 males and 5 females) between the ages of 20 and 34 participated in this study. All participants were right-handed. They were all familiar with graphical user interfaces, and had previous experience with mobile touch-screen devices.

6.3.5 Results and Discussion

The results were analyzed using Repeated-measures ANOVA and Bonferroni corrections for pairwise comparisons. Before the analysis, we checked the ordering effects of group size on all the dependent measures, and found no significant effects.

Task completion time

ANOVA yielded a significant effect of *TX* ($F_{8,72} = 13.65$, $p < 0.001$). Interestingly, we found no significant effect of *GS* ($F_{4,36} = 1.16$, $p = 0.35$), *EP* ($F_{1,9} = 0.9$, $p = 0.37$), and *TY* ($F_{1,9} = 0.67$, $p = 0.44$). There were significant interaction effects on $GS \times TX$ ($F_{32,288} = 7.17$, $p < 0.001$), and $TY \times EP$ ($F_{1,9} = 6.1$, $p < 0.05$).

Post-hoc analysis showed that task time decreased significantly towards the center of the ribbon (Figure 6.9 (left)). In particular, task time at TX 1, 7, 8, and 9 were significantly longer than at 3, 4, 5, 6 (all $p < 0.05$).

Task completion time without off-screen error

There was a significant difference between trials when off-screen errors occurred (2700ms s.e. 119.74) and when off-screen errors did not occur (1094ms s.e. 20.82) ($F_{1,9} = 172.89$, $p < 0.001$).

After removing these trials (7%), we found significant effects of *TX* ($F_{8,72} = 12.35$, $p < 0.001$), *EP* ($F_{1,9} = 47.13$, $p < 0.001$), and *TY* ($F_{1,9} = 14.22$, $p < 0.005$). There was a weak effect of *GS*

($F_{4,36} = 2.88, = 0.04$), but pair-wise comparison showed no significant difference between group sizes. There was a significant interaction effect on $GS \times TY$ ($F_{4,36} = 0.67, p < 0.05$). Figure 6.9 (left) shows the task time with and without off-screen errors.

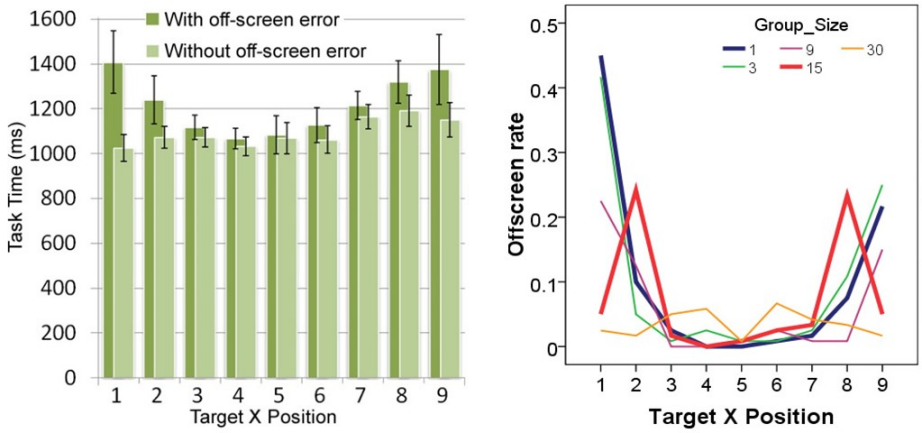

Figure 6.9: Left: Task time shown by target position. Right: Off-screen rate shown by target position and group size.

The participants performed the task faster when the ribbon was expanded at 40% distance (1038 ms s.e. 25.5 ms) than when it was expanded at 90% distance (1147 ms, s.e. 18.8 ms). We believe this is because the early expansion allowed users to adjust their initial movement path earlier, instead of following two separate acquisition paths (one for the group, and then one for the goal target position resulting from the expansion).

We also analyzed the effect of the relative position of the targets within their groups, excluding group size 1. We found a significant effect of target position ($F_{1,18} = 14.13, p < 0.001$). Targets on the left (1116 ms s.e. 24.23 ms) and right side (1133 ms s.e. 28.62 ms) of a group took significantly longer than those on the center of a group (1046 ms s.e. 23.68 ms). This explains why we found only weak effect of group size. Although big groups (e.g. 15 and 30) have better prediction, their larger target displacements increase acquisition times.

Off-screen Errors

ANOVA yielded a significant effect of *TX* ($F_{8,72} = 22.72, p < 0.001$), *GS* ($F_{4,36} = 10.68, p < 0.001$), and *EP* ($F_{1,9} = 2.12, p < 0.05$). There was no significant effect of *TY* ($F_{1,9} = 0.67, p = 0.44$). There were also a significant interaction effect on $EP \times TX$ ($F_{8,72} = 2.58, p < 0.05$).

The participants made more off-screen errors when the ribbon was expanded at 40% distance (0.09 s.e. 0.02) than when it was expanded at 90% distance (0.05 s.e. 0.01).

Figure 6.9 (right) shows that most off-screen errors were made on the targets on the left and right edge of the ribbon (TX 1 and 9). Post-hoc analysis showed that big groups (e.g. 15 and 30) introduced significantly less off-screen errors than the smaller groups ($p < 0.05$). It also shows that large group sizes can also cause off-screen errors. For instance, the participants made significantly

more off-screen errors on the targets at position TX 2 and 8 with group size 15 than with other group sizes ($p < 0.05$). Going back to Figure 6.8 we see that these two target positions are at the edge of their respective groups. If a participant aimed at the target instead of the group, there was a chance the wrong group would be predicted, pushing the desired group off-screen.

Selection Error

Overall, the average selection error rate was 6.2%. No main effects or interaction effects were found on error rate.

6.4 Hybrid Interpolation for TouchZoom

Experiment 2 shows that reducing prediction error by increasing the size of the group does not improve the efficiency of the task, because of the larger target offsets. We also found that off-screen errors had an overwhelming effect on overall completion time. In this section we discuss a redesign of TouchZoom to prevent off-screen errors.

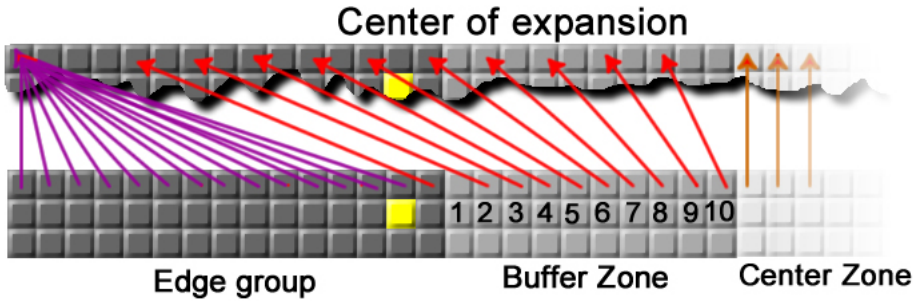

Figure 6.10: Illustration of the left side of a ribbon with buffer zone. Red arrows associate the center of expansion for each of the icons in the 3 zones.

As suggested by Experiment 2, larger group sizes are effective in preventing off-screen errors on targets close to the left and right edges of the screen but can be error prone with targets closer to the center. The opposite was true for small group sizes. To leverage the benefits of both designs, we implemented a hybrid technique which uses a group size of 15 on the two edges, and individual ungrouped targets in between. To further reduce the chance of off-screen errors, we use a buffer zone between the group of 15 targets, and the center zone. The buffer zone consists of 10 individual targets, but their *CE* is based on a linear interpolation of the screen edge and the *CE* of the first target in the center zone (Figure 6.10). Having this buffer zone minimizes the impact of a prediction error when aiming at a target in the edge group. The exact sizes of the edge groups and buffer zones were chosen based on prediction error rates from Experiment 2, in an effort to minimize off-screen targets as much as possible.

6.5 Experiment 3

We have described two expanding target techniques for touch, TouchCuts, and TouchZoom. In study 2 we performed an evaluation of TouchZoom, which resulted in a resigned hybrid interpolation, as discussed above. In this study, we measured the performance of the redesigned TouchZoom and TouchCuts, in comparison to a baseline input device - the Mouse and a baseline touch technique - Shift.

While numerous techniques exist for aiding touch-based target acquisition, we used Shift as a baseline, since it does not have any visual impact on the user interface, unless the technique is used. Following previous guidelines, our implementation placed the callout window (16mm in diameter) 22mm on the right side of the initial touch point to facilitate the selection on a ribbon by using the left hand. The callout was placed on the opposite side if the touch point was within 30mm from the right end of the ribbon. We set the escalation time to zero so that the callout popped up as soon as a user touches the screen.

6.5.1 Apparatus

We used the same apparatus as in Experiment 2. Reflective markers were used in the target expansion techniques.

6.5.2 Task and Procedure

The participants were asked to press a keyboard key ("\" for the cursor and "O" for the others), and to select a target (0.5×0.5cm) in a ribbon by using one of the 4 techniques. The task was required to be carried out by using the left hand for all the techniques except for the mouse. In the mouse condition, prior to pressing "\", the participants were asked to place the cursor in a start square ($0.5 \times 0.5\ cm^2$) rendered in the center of the workspace. In the TouchZoom condition, we only showed the border between the edge groups and the buffer zone. The buffer zone and the groups of size 1 were invisible to the users. As in Experiment 2, for targets close to the screen edges, the participants were shown that by biasing their acquisition movement towards the edge of the screen, off-screen errors could be minimized. For both target expansion techniques, the expansion point was set to 60%.

Prior to the study, the participants were given a 3 minute training section for each technique. They were encouraged to take breaks during the experiment. The entire experiment lasted about 40 minutes. Participants filled out a post experiment questionnaire upon completion.

6.5.3 Design

The experiment employed a 4×3 within-subject factorial design. The independent variables are *Technique* (*TouchZoom*, *TouchCuts*, *Shift*, and *Mouse Cursor*) and *Target Zone* (*Edge Group*, *Buffer Zone* and *Center Zone*).

In each trial, participants performed tasks in one of each $Technique \times Target\ Zone$ combination. The experiment consisted of 3 blocks, each consisting of 30 trials, 10 for each target zone. In each trial, the position of the target was randomized for each target zone. The target was evenly distributed to the left and right side as well as the 3 rows of the ribbon. The order of the presentation of the techniques was counter balanced between participants.

6.5.4 Participants

Twelve paid participants (6 males and 6 females) between the ages of 18 and 34 participated in this study. None of them had participated in the Experiment 1 and 2. All participants were right-handed. They were all familiar with graphical user interfaces. All but one had previous experience with mobile touch-screen devices.

6.5.5 Results

The data was analyzed using Repeated-measures ANOVA and Bonferroni corrections for pair-wise comparisons.

Task completion time

ANOVA yielded a significant effect of *Technique* ($F_{3,33} = 245.36$, $p < 0.001$) and *Target Zone* ($F_{2,22} = 15.82$, $p < 0.001$). There was also a significant interaction effect on $InputTechnique \times TargetZone$ ($F_{6,66} = 4.53$, $p = 0.001$). Figure 6.11 (left) shows average completion time for each *Target Zone* by *Technique*.

Performance with TouchCuts (768ms) was faster than TouchZoom (1130ms), which was faster than Mouse cursor (1280ms) and Shift (1883ms). Post-hoc analysis showed significant differences between all pairs of techniques. It is not surprising that TouchCuts performed the best overall, since it provides target expansion without any target offset. The difference between TouchCuts and TouchZoom (362 ms) is the added cost of the target offsets that TouchZoom introduces. It is worth reiterating that TouchCuts is slightly different from the other techniques, in that it would only provide access to a predetermined subset of the ribbon icons, likely those most frequently used by the user.

What was more surprising was the difference between TouchZoom and Shift. Both techniques require initial and adjustment pointing phases. We believe that TouchZoom performed better because with Shift the two phases are explicitly sequential, while with the TouchZoom technique, the adjustment phase can be predicted and integrated into the end of the initial phase. In addition, Shift does not increase the motor activation size of the target.

Performance in the Edge group (1306ms) was slightly slower than Center zone (1249ms) and Buffer zone (1241ms) ($p = 0.001$), while no significant difference was found between Buffer zone and Center zone ($p = 1$). However, we found no significant effect of *Target Zone* ($F_{2,22} = 1.2$,

$p = 0.321$) in the TouchZoom condition, indicating that users can perform equally well across the ribbon.

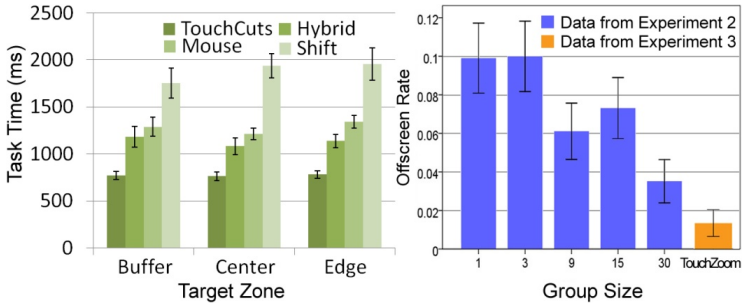

Figure 6.11: Left: average task complete time shown for each target zone by technique. Right: off-screen rate shown by group sizes.

Selection Error

For selection errors, there was a significant effect of *Technique* ($F_{3,33} = 8.18$, $p < 0.001$). But no significant effect of *Target Zone* ($F_{2,22} = 0.03$, $p = 0.97$). There was also no significant interaction effect.

TouchCuts had significantly less errors (0.02 s.e. 0.01) than the TouchZoom (0.11 s.e. 0.025), Mouse cursor (0.07 s.e. 0.01), and Shift (0.08 s.e. 0.02). We found no significant difference between TouchZoom, Mouse cursor, and Shift.

In the TouchZoom condition, we also found no significant effect of *Target Zone* ($F_{2,22} = 0.07$, $p = 0.93$). This again confirms that users can perform equally well across the ribbon. Average off-screen error rate for the TouchZoom condition was 0.014(s.e. 0.006). We also found no significant effect of *Target Zone* on off-screen error ($F_{2,22} = 0.836$, $p = 0.45$). This was an encouraging result, demonstrating that the hybrid interpolation we designed based on the results of Experiment 2 effectively minimized the chance of off-screen errors, in comparison to the static group sizes (shown in Figure 6.11 (right)).

Subjective Preference

A short questionnaire was administered after the study. All scores reported below are based on a 7-point Likert scale, with 7 indicating highest preference.

The participants gave an average of 6.7 to TouchCuts and 5.2 to TouchZoom as the two most easy to use techniques. Whereas, Shift and Mouse cursor received an average of 3.7 and 4.6 respectively. Additionally, the participants gave an average of 6.9 to TouchCuts and 5.3 to TouchZoom as the two most enjoyable techniques. Shift and Mouse cursor received an average of 3.9 and 4 respectively. The mouse was rated lower because of both the switching between the keyboard and mouse, and also the small target sizes. Shift was rated lower because of the longer acquisition times. In addition, numerous users reported Shift as being fatiguing because it required careful movements of the finger

while it was positioned on the screen.

6.6 Discussion

We have demonstrated the benefits two new techniques, TouchCuts and TouchZoom, to facilitate selection on touch-based laptops. In comparison to a mouse, TouchCuts reduced selection times by 40%, but does not provide access to every UI element. In contrast, TouchZoom gives the user access to an entire ribbon, and reduced selection times by 12% in comparison to a mouse. We also know form Experiment 1, that if the user does not have a mouse attached to their laptop, and has to use a touchpad; the levels of improvement would be even more substantial.

It is worth pointing out that we are not trying to replace the mouse. To the contrary, we made our design decisions carefully so that traditional cursor interaction would be unaffected. If the user is already performing mouse interactions close to the top of the screen, it may make more sense to acquire ribbon icons with the mouse. However, if the user is performing cursor intensive interactions, the non-dominant hand could be used to access UI elements in parallel, saving a round trip of the cursor (for example, changing colors while drawing). While our studies did show that touch can be effective with the non-dominant hand, future studies could explore this form of parallel, bimanual input.

Although the differences were not significant from the mouse or Shift, TouchZoom did exhibit a higher error rate (11%) in Experiment 3 than we expected. Our observations indicated that some of these errors were caused by users accidently touching the screen at the end of their first ballistic movements, just after the ribbon expanded. Since this error rate was higher than in Experiment 2 (6.2%), one potential explanation is that there were detrimental transfer effects from the Shift and TouchCuts techniques, where participants could touch the screen imprecisely after an initial ballistic movement.

Our work focused on a laptop configuration for three reasons: Most major manufactures have touch-enabled laptops; the hands are positioned close to the display; a mouse may not be available, and the track-pad is inefficient for pointing tasks. However, our work could generalize to desktop settings as well, but fatigue issues must be considered, since further reaching may be required. In addition, our work assumed proximity sensing was available, and for TouchZoom, the control being zoomed is horizontal. In the next section, we discuss several design variations to demonstrate how our techniques could generalize to other usage scenarios.

6.7 Further Design Alternatives

Functionality reduction controls - inspired by TouchCuts, we introduce functionality reduction controls to facilitate finger input. A functionality reduction control transitions to a preset touch-optimized component, that has the same screen footprint, but offers a subset of the functionality of

its cursor-based counterpart. For example, we implemented a functionality reduction color palette, which replaces the color picker with 12 large color icons when the finger approaches. When users know they want to select one of these main colors, they can do so quickly with the non-dominant hand, saving a cursor round-trip.

Multi-level expansion - In this technique, the ribbon has 2 expansion points: 50% and 80%. It expands half-way if the finger crosses the 50% distance, and fully expands after the finger crosses the 80% distance. No discontinuity will be seen if the finger crosses the 2 expansion point at a sufficient speed. This technique was mainly designed to help users learn to use the TouchZoom technique.

Depth-based expansion - Depth-based expansion triggers the expansion when the finger is within a threshold distance to the screen, with the center of expansion equal to the on-screen projection of the current finger position. The finger is first positioned directly above the target of interest, and then the finger moves towards the screen to trigger the expansion. This could be particularly useful for implementing TouchZoom on vertical tool palettes, since endpoint prediction would be difficult. In addition, it could be useful on systems with a small proximity sensing range.

Touch activated expansion - To demonstrate the use of TouchCuts without proximity sensing, we delay expansion until the finger actually makes contact with the screen. This could also be used for *functionality reduction controls*, if the touch-optimized layout is predictable.

6.8 Summary

In this chapter, we have presented 3 studies to motivate and evaluate the design of TouchCuts and TouchZoom. We demonstrated that finger input has the benefit of allowing homing and pointing to be carried out concurrently, but suffers from extremely high error rates on icons in existing legacy applications. To support touch in such applications, our techniques trigger a transition of the user interface only when a finger approaches. Thus, controls can be effectively shared by both a traditional cursor and touch. The study showed positive benefits of both techniques. Furthermore, the results of our studies show the hybrid interpolation used for TouchZoom effectively reduces the chance of off-screen errors. Finally, we present several alternative designs to show how the techniques could generalize to scenarios which we did not explicitly study. We believe with the continued increase in popularity of touch-based displays, the techniques may serve as important groundwork for integrating the benefits of touch into existing applications.

Chapter 7

Conclusion and Future Research Directions

7.1 Conclusion

This thesis demonstrates that *mixed input mode* is possible through hardware and software argumentation and can be an effective input alternative. We created three prototypes showcasing *mixed input mode* devices on three varied input platforms - desktop, mobile, and always-available input.

For the desktop environment, we created LensMouse, a *mixed input mode* mouse that embeds a touch-screen display-or tangible 'lens'-onto a mouse. Equally important is that the LensMouse allows a mouse to become an output device. We presented a solution to overcome challenges with auxiliary windows, a demonstration of some of the benefits of our device through a user experiment, and a set of applications and interactions that can benefit from such a device.

For the mobile environment, we created LucidCursor, a *mixed input mode* PDA, allowing users to carry out direct input on the front and indirect input on the back. We then introduced the Dual-Surface input technique. Through two controlled studies, we demonstrated that Dual-Surface input allowed users to rapidly select small targets located in regions that are hard to access when interacting using the thumb with one-handed input. We also showed that with behind-the-display relative input, complex tasks, such as steering, could be carried out easier using virtual enhancement techniques.

For always-available input, we created Magic Finger, a new input device worn on the user's finger. Unlike traditional touch input, Magic Finger allows both direct and indirect input to be carried out via the fingertip. More importantly, Magic Finger allows popular touch interactions to be carried out on any surface. It can detect the texture of the object the user touches, which allows the device to trigger contextual actions. Through a system evaluation, we showed that Magic Finger could distinguish 22 daily objects with 98.9% accuracy.

Finally, we showed that popular desktop applications were not ready for mixed input mode. This is because these applications were mainly designed to be used by a computer mouse. We demon-

strated that touch was more efficient than a mouse. This is because using touch allows homing and pointing to take place simultaneously. However, our studies also showed that touch could be extremely inaccurate when interacting with small icons in popular desktop applications. We therefore developed two target expansion techniques for touch screen input. Through a controlled user study, we showed that interacting with small targets using TouchCuts & TouchZoom was more efficient than with a mouse or Shift. We also developed several alternative designs to show how these two techniques could generalize to scenarios that we did not explicitly study in this thesis..

7.2 Future Research Directions

This thesis opens a new space for further exploratory and in-depth research. Although we proposed a rich set of interaction techniques for the LensMouse, it is still unclear whether users can use these new techniques to perform common tasks better than using existing ones. Therefore, the proposed techniques need to be evaluated. Furthermore, many of the LensMouse techniques are worthy of further exploration. For example, Hybrid Pointing allows users to select small objects using the finger through a magnifying lens on the LensMouse. This technique requires the user to visually search the enlarged target on the LensMouse before a selection can be made. This visual search process may introduce performance overhead, and the problem may become worse if multiple targets cluster together. Future work will focus on techniques that can reduce visual search. A good candidate is gesture-based input because the performance of target acquisition using gestures is independent of the size and the location of the target. Using gestures can also allow eyes-free interactions to be carried out on the LensMouse.

For the LucidCursor, we evaluated Dual-Surface interaction in a controlled lab environment. Future work could include an experiment in a more ecologically valid environment, where participants may be asked to perform the task when walking or sitting in a moving vehicle. The results would provide deeper insight into the performance of Dual-Surface input in real-life situations.

Magic Finger could benefit from a formal user study to help understand how the users use the device in various tasks. Additionally, a user survey may be helpful in finding the preferred location on the user's finger when wearing the device. It is also important to run studies to identify important topics we have not explored, such as the "Midas touch" and false activations. The interactions that we implemented and demonstrated for Magic Finger should be considered exploratory in nature and only represent a small sample of what is possible with the device. In our actual implementation, issues such as conflicts between operating modes and resulting actions would require a more careful examination. Introducing aids to help users learn or remember mappings would also be a fruitful topic for future work.

Mixed input mode interaction allows users to choose an input mode that is best suitable for a certain task, e.g., using a cursor to interact with small virtual objects. However, choosing between the two input modes requires making a decision, which may cause extra cognitive costs and negatively

impact task performance [108]. Future work should investigate the cost of this decision-making process and design a recommender system to reduce such overhead, if they exist.

7.3 A View of the Future

This thesis presents the outcomes of several projects that my thesis advisors, my colleagues from both academic (*University of Alberta and University of Manitoba*) and industry research labs (*Autodesk Research*), and I have worked on for the past five years. I started my PhD program with extreme passion and a strong belief on the value of *mixed input mode*. During the course of these projects, I have learned numerous lessons and collected extremely valuable experiences. I present them in this thesis and hope that this thesis can inspire further research in this area and provide insight into what future input devices may look like.

In the future, input devices of different modes may eventually merge into a single *mixed input mode* device. It is not hard to imagine that computer mice may be replaced by devices like the Magic Finger. For example, users may use the Magic Finger to control the mouse cursor on any flat surface and use the same finger to interact with virtual objects directly on a touch screen. An obvious benefit is that users won't have to carry a mouse for situations requiring interaction with small virtual objects. One single input device can facilitate a large number of tasks independent of computer platform. Magic Finger provides an ideal solution for input against an object's surface, either on- or off-screen. However, users may still need to use an additional input device, e.g., 3D depth-sensing camera, for popular midair gesture input. Future *mixed input mode* devices may integrate a 3D depth-sensing camera to fill the gap. Users will thus be able to interact with the virtual world in the entire continuous input space around it.

Similarly, software user interfaces on different platforms, e.g., desktop or mobile, may eventually converge to a single paradigm. TouchCuts & TouchZoom suggests that such paradigm can use zoomable widgets to facilitate the needs for both fine and coarse control. Alternatively, applications could automatically switch UI paradigms based on users' intended input mode. Therefore, sensing technologies will be the key. Many existing sensing technologies are capable of sensing the user's intention. For example, PixelSense (*http://www.microsoft.com/en-us/pixelsense/default.aspx*) can detect the user's hand motion on or above a touch screen, a good indicator for touch input. With developments on novel sensing technologies, software user interfaces can be adapted more favorably toward *mixed input modes*.

Bibliography

[1] D. Ahlström. Modeling and improving selection in cascading pull-down menus using fitts' law, the steering law and force fields. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 61–70, 2005.

[2] D. Ahlström, M. Hitz, and G. Leitner. An evaluation of sticky and force enhanced targets in multi target situations. In *Proceedings of the Nordic Conference on Human-Computer Interaction*, pages 58–67, 2006.

[3] M. Annett, T. Grossman, D. Wigdor, and G. Fitzmaurice. Medusa: A proximity-aware multi-touch tabletop. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 373–382, 2011.

[4] ANOTO. http://www.anoto.com/.

[5] Arduino. http://arduino.cc.

[6] T. Asano, E. Sharlin, Y. Kitamura, K. Takashima, and F. Kishino. Predictive interaction using the delphian desktop. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 133–141, 2005.

[7] M. Ashdown, K. Oka, and Y. Sato. Combining head tracking and mouse input for a gui on multiple monitors. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1188–1191, 2005.

[8] D. Avrahami, J. O. Wobbrock, and S. Izadi. Portico: Tangible interaction on and around a tablet. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 347–356, 2011.

[9] AWAIBA. www.awaiba.com.

[10] R. Balakrishnan and P. Patel. The padmouse: facilitating selection and spatial positioning for the non-dominant hand. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 9–16, 1998.

[11] O. Bau, I. Poupyrev, A. Israr, and C. Harrison. Teslatouch: electrovibration for touch surfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 283–292, 2010.

[12] P. Baudisch and G. Chu. Back-of-device interaction allows creating very small touch devices. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1923–1932, 2009.

[13] P. Baudisch, E. B. Cutrell, K. Hinckley, and R. Gruen. Mouse ether: accelerating the acquisition of targets across multi-monitor displays. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1379–1382, 2004.

[14] P. Baudisch and C. Gutwin. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 367–374, 2004.

[15] P. Baudisch, M. Sinclair, and A. Wilson. Soap: a pointing device that works in mid-air. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 43–46, 2006.

[16] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini. A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys*, 44(4):1–49, 2012.

[17] H. Benko and S. Feiner. Multi-monitor mouse. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1208–1211, 2005.

[18] H. Benko, A. D. Wilson, and P. Baudisch. Precise selection techniques for multi-touch screens. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1263–1272, 2006.

[19] X. Bi and R. Balakrishnan. Comparing usage of a large high-resolution display to single or dual desktop displays for daily work. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1005–1014, 2009.

[20] X. Bi, T. Grossman, J. Matejka, and G. Fitzmaurice. Magic desk: bringing multi-touch surfaces into desktop work. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 2511–2520, 2011.

[21] E. A. Bier, M. C. Stone, K. A. Pier, W. Buxton, and T. D.Derose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the Annual Conference on Computer Graphics*, pages 73–80, 1993.

[22] R. Blanch, Y. Guiard, and M. Beaudouin-Lafon. Semantic pointing: improving target acquisition with control-display ratio adaptation. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 519–526, 2004.

[23] A. Butler, S. Izadi, and S. Hodges. Sidesight: multi- touch interaction around small devices. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 201–204, 2008.

[24] S. Carpendale, J. Light, and E.Pattison. Achieving higher magnification in context. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 71–80, 2004.

[25] S. Carter, A. Hurst, J. Mankoff, and J. Li. Dynamically adapting guis to diverse input devices. In *Proceedings of the ACM Conference on Assistive Technologies*, pages 63–70, 2006.

[26] J. Cechanowicz, P. Irani, and S. Subramanian. Augmenting the mouse with pressure sensitive input. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1385–1394, 2007.

[27] N. Charness, P. Holley, J. Feddon, and T. Jastrzembski. Light pen use and practice minimize age and hand performance differences in pointing tasks. *Human Factors*, 46(3):373–384, 2005.

[28] G. Chen and D. Kota. A survey of context-aware mobile computing research. Technical report, Dartmouth College, 2000.

[29] N. Chen, F. Guimbretière, M. Dixon, C. Lewis, and M. Agrawala. Navigation techniques for dual-display e-book readers. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1779–1788, 2008.

[30] K. Y. Cheng, R. H. Liang, B. Y. Chen, R. H. Laing, and S. Y. Kuo. icon: utilizing everyday objects as additional, auxiliary and instant tabletop controllers. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1155–1164, 2010.

[31] Cypress Semiconductor Co. http://www.cypress.com/?rid=42793.

[32] A. Cockburn and P. Brock. Human on-line response to visual and motor target expansion. In *Proceedings of the Graphics Interface*, pages 81–87, 2006.

[33] G. Cohn, D. Morris, S. N. Patel, and D. S. Tan. Your noise is my command: sensing gestures using the body as an antenna. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 791–800, 2011.

[34] J. T. Dennerlein, D. B. Martin, and C. J. Hasser. Force-feedback improves performance for steering and combined steering-targeting tasks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 423–429, 2000.

[35] C. Dickie, J. Hart, R. Vertegaal, and A. Eiser. Lookpoint: an evaluation of eye input for hands-free switching of input devices between multiple computers. In *Proceedings of the Australasian Computer-Human Interaction Conference*, pages 119–126, 2006.

[36] D. C. Engelbart. Sri-arc. a technical session presentation at the fall joint computer conference in san francisco. Technical report, Engelbart Collection, Stanford University Library, Menlo Park (CA), 1968.

[37] A. Esenther and K. Ryall. Fluid dtmouse: better mouse support for touch-based interactions. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 112–115, 2006.

[38] LIBSVM-A Library for Support Vector Machines. http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[39] C. Forlines and R. Balakrishnan. Evaluating tactile feedback and direct vs. indirect stylus input in pointing and crossing selection tasks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1563–1572, 2008.

[40] C. Forlines, D. Wigdor, C. Shen, and R. Balakrishnan. Direct-touch vs. mouse input for tabletop displays. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 647–656, 2007.

[41] R. Fung, E. Lank, M. Terry, and C. Latulipe. Kinematic templates: end-user tools for content-relative cursor manipulations. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 48–56, 2008.

[42] K. Gajos and D. S. Weld. Supple: automatically generating user interfaces. In *Proceedings of the Intelligent User Interfaces*, pages 93–100, 2004.

[43] L. Gallo, M. Ciampi, and A. Minutolo. Smoothed pointing: A user-friendly technique for precision enhanced remote pointing. In *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems*, pages 712–717, 2010.

[44] T. Grossman and R. Balakrishnan. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 281–290, 2005.

[45] T. Grossman, P. Dragicevic, and R. Balakrishnan. Strategies for accelerating on-line learning of hotkeys. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1591–1600, 2007.

[46] J. Grudin. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 458–465, 2001.

[47] Y. Guiard. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Motor Behavior*, 19(4):486–517, 1994.

[48] Y. Guiard, R. Blanch, and M. Beaudouin-Lafon. Object pointing: A complement to bitmap pointing in guis. In *Proceedings of the Graphics Interface*, pages 9–16, 2004.

[49] S. Gustafson, C. Holz, and P. Baudisch. Imaginary phone: learning imaginary interfaces by transferring spatial memory from a familiar device. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 283–292, 2011.

[50] C. Gutwin. Improving focus targeting in interactive fisheye views. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 264–274, 2002.

[51] C. Harrison, H. Benko, and A. D. Wilson. Omnitouch: wearable multitouch interaction everywhere. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 441–450, 2011.

[52] C. Harrison and S. E. Hudson. Lightweight material detection for placement-aware mobile computing. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 279–282, 2008.

[53] C. Harrison and S. E. Hudson. Scratch input: creating large, inexpensive, unpowered and mobile finger input surfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 205–208, 2008.

[54] C. Harrison and S. E. Hudson. Abracadabra: wireless, high-precision, and unpowered finger input for very small mobile devices. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 121–124, 2009.

[55] C. Harrison and S. E. Hudson. Texture displays: a passive approach to tactile presentation. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 2261–2264, 2009.

[56] C. Harrison, B. Y. Lim, A. Shick, and S. E. Hudson. Where to locate wearable displays?: reaction time performance of visual alerts from tip to toe. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 941–944, 2009.

[57] C. Harrison, S. Ramamurthy, and S. E. Hudson. On-body interaction: armed and dangerous. In *Proceedings of the International Conference on Tangible, Embedded, and Embodied Interaction*, pages 19–22, 2012.

[58] C. Harrison, J. Schwarz, and S. E. Hudson. Tapsense: Enhancing finger interaction on touch surfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 627–636, 2011.

[59] C. Harrison, D. S. Tan, and D. Morris. Skinput: appropriating the body as an input surface. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 453–462, 2010.

[60] K. Hinckley, E. Cutrell, S. Bathiche, and T. Muss. Quantitative analysis of scrolling techniques. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 65–72, 2002.

[61] K. Hinckley, M. Dixon, R. Sarin, F. Guimbretire, and R. Balakrishnan. Codex: a dual screen tablet computer. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1933–1942, 2009.

[62] K. Hinckley, G. Ramos, F. Guimbretiere, P. Baudisch, and M. Smith. Stitching: pen gestures that span multiple displays. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 23–31, 2004.

[63] K. Hinckley and M. Sinclair. Touch-sensing input devices. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 223–330, 1999.

[64] K. Hinckley, M. Sinclair, E. Hanson, R. Szeliski, and M. Conway. The videomouse: a camera-based multi-degree-of-freedom input device. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 103–112, 1999.

[65] S. Hodges, S. Izadi, A. Butler, A. Rsustemi, and B. Buxton. Thinsight: versatile multi-touch sensing for thin form-factor displays. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 259–268, 2007.

[66] P. Holleis, A. Schmidt, S.Paasovaara, A. Puikkonen, and J. Häkkilä. Evaluating capacitive touch input on clothes. In *Proceedings of the Mobile HCI*, pages 81–90, 2008.

[67] C. Holz, T. Grossman, G. Fitzmaurice, and A. Agur. Implanted user interfaces. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 503–512, 2012.

[68] K. Hornbæk, B. B. Bederson, and C. Plaisant. Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Transactions on Computer-Human Interaction*, 9(4):362–389, 2002.

[69] D. R. Hutchings, G. Smith, B. Meyers, M. Czerwinski, and G. G. Robertson. Display space usage and window management operation comparisons between single monitor and multiple monitor users. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 32–39, 2004.

[70] D. R. Hutchings and J. T. Stasko. Mudibo: multiple dialog boxes for multiple monitors. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1471–1474, 2005.

[71] D. R. Hutchings and J. T. Stasko. Consistency, multiple monitors, and multiple windows. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 211–214, 2007.

[72] H. M. Hutchings and J.S. Pierce. Understanding the whethers, hows, and whys of divisible interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 274–277, 2006.

[73] E. W. Ishak and S. K. Feiner. Interacting with hidden content using content-aware free-space transparency. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 189–192, 2004.

[74] S. K. Kane, D. Avrahami, J. O. Wobbrock, B. L. Harrison, A. D. Rea, M. Philipose, and A. Lamarca. Bonfire: a nomadic system for hybrid laptop-tabletop interaction. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 129–138, 2009.

[75] Y. A. Kang and J. T. Stasko. Lightweight task/application performance using single versus multiple monitors: a comparative study. In *Proceedings of the Graphics Interface*, pages 17–24, 2008.

[76] A. K. Karlson and B. B. Bederson. Thumbspace: generalized one-handed input for touchscreen-based mobile devices. In *Proceedings of the IFIP Conference on Human-Computer Interaction*, pages 324–338, 2007.

[77] A. K. Karlson and B. B. Bederson. One-handed touchscreen input for legacy applications. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1399–1408, 2008.

[78] A. K. Karlson, B. B. Bederson, and J. L. Contreras-Vidal. Understanding single-handed mobile device interaction. Technical report, University of Maryland, College Park, 2006.

[79] A. K. Karlson, B. B. Bederson, and J. SanGiovanni. Applens and launchtile: two designs for one-handed thumb use on small devices. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 201–210, 2005.

[80] M. Kobayashi and T.Igarashi. Ninja cursors: using multiple cursors to assist target acquisition on large screens. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 949–958, 2008.

[81] W. A. König, J. Gerken, S. Dierdorf, and H. Reiterer. Adaptive pointing - design and evaluation of a precision enhancing technique for absolute pointing devices. In *Proceedings of the IFIP Conference on Human-Computer Interaction*, pages 658–671, 2007.

[82] E. Lank, Y. C. Cheng, and J. Ruiz. Endpoint prediction using motion kinematics. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 637–646, 2007.

[83] V. Lévesque, L. Oram, K. E. MacLean, A. Cockburn, N. D. Marchuk, D. Johnson, J. E. Colgate, and M. A. Peshkin. Enhancing physicality in touch interaction with programmable friction. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 2481–2490, 2011.

[84] Finger Mouse (Logisys). http://www.logisyscomputer.com.

[85] Primesense Ltd. http://www.primesense.com/?p=486.

[86] I. S. Mackenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7(1):91–139, 1992.

[87] N. Marquardt, J. Kiemer, and S. Greenberg. What caused that touch? expressive interaction with a surface through fiduciary-tagged gloves. In *Proceedings of ACM International Conference on Interactive Tabletops and Surfaces*, pages 139–142, 2010.

[88] D. C. McCallum and P. Irani. Arc-pad: absolute+relative cursor positioning for large displays with a mobile touchscreen. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 153–156, 2009.

[89] M. McGuffin, N. Burtnyk, and G. Kurtenbach. Fast sliders: integrating marking menus and the adjustment of continuous values. In *Proceedings of the Graphics Interface*, pages 35–41, 2002.

[90] M. J. McGuffin and R. Balakrishnan. Acquisition of expanding targets. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 57–64, 2002.

[91] M. J. McGuffin and R. Balakrishnan. Fitts' law and expanding targets: Experimental studies and designs for user interfaces. *ACM Transactions on Computer-Human Interaction*, 12(4):388–422, 2005.

[92] S. Meyer, O. Cohen, and E. Nilsen. Device comparisons for goal-directed drawing tasks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (Extended Abstract)*, pages 251–252, 1994.

[93] P. Mistry and P. Maes. Sixthsense: a wearable gestural interface. In *Proceedings of SIGGRAPH Asia 2009*, page Sketch, 2009.

[94] M. R. Morris, A. J. B. Brush, and B. Meyers. Reading revisited: evaluating the usability of digital display surfaces for active reading tasks. In *Proceedings of the Workshop on Horizontal Interactive Human-Computer Systems*, pages 79–86, 2007.

[95] T. Moscovich. Contact area interaction with sliding widgets. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 13–22, 2009.

[96] Apple Magic Mouse. http://www.apple.com/magicmouse/.

[97] Microsoft Touch Mouse. http://www.microsoft.com/hardware/en-ca/products/touchmouse/microsite/.

[98] Y. Mukaibo, H. Shirado, M. Konyo, and T. Maeno. Development of a texture sensor emulating the tissue structure and perceptual mechanism of human fingers. In *Proceedings of the International Conference on Robotics and Automation*, pages 2565–2570, 2005.

[99] B. A. Myers, R. C. Miller, B. Bostwick, and C. Evankovich. Extending the windows desktop interface with connected handheld computers. In *Proceedings of the USENIX Windows Systems Symposium*, pages 79–88, 2000.

[100] M. A. Nacenta, R. L. Mandryk, and C. Gutwin. Targeting across displayless space. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 777–786, 2008.

[101] T. Ni and P. Baudisch. Disappearing mobile devices. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 101–110, 2009.

[102] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(7):971–987, 2002.

[103] A. Olwal, S. Feiner, and S. Heyman. Rubbing and tapping for precise and rapid selection on touch-screen displays. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 594–304, 2008.

[104] P. Parhi, A. K. Karlson, and B. B. Bederson. Target size study for one-handed thumb use on small touchscreen devices. In *Proceedings of the Mobile HCI*, pages 203–210, 2006.

[105] E. Pietriga, C. Appert, and M. Beaudouin-lafon. Pointing and beyond: an operationalization and preliminary evaluation of multi-scale searching. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1215–1224, 2007.

[106] C. Plaisant, D. A. Carr, and B. Shneiderman. Image-browser taxonomy and guidelines for designers. *IEEE Software*, 12(2):21–32, 1995.

[107] R. L. Potter, L. J. Weldon, and B. Shneiderman. Improving the accuracy of touch screens: an experimental evaluation of three strategies. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 27–32, 1988.

[108] P. Quinn, A. Cockburn, K. J. Rih, and J. Delamarche. On the costs of multiple trajectory pointing methods. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 859–862, 2011.

[109] G. Ramos, A. Cockburn, R. Balakrishnan, and M. Beaudouin-lafon. Pointing lenses: facilitating stylus input through visual-and motor-space magnification. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 757–766, 2007.

[110] J. Rekimoto. Smartskin: an infrastructure for freehand manipulation on interactive surfaces. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 113–120, 2002.

[111] S. Richter, C. Holz, and P. Baudisch. Bootstrapper: recognizing tabletop users by their shoes. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1249–1252, 2012.

[112] M. Ringel. When one isn't enough: an analysis of virtual desktop usage strategies and their implications for design. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 762–763, 2003.

[113] J. Ruiz and E. Lank. Speeding pointing in tiled widgets: understanding the effects of target expansion and misprediction. In *Proceedings of the Intelligent User Interfaces*, pages 229–238, 2010.

[114] T. S. Saponas, C. Harrison, and H. Benko. Pockettouch: through-fabric capacitive touch input. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 303–308, 2011.

[115] T. S. Saponas, D. S. Tan, D. Morris, R. Balakrishnan, J. Turner, and J. A. Landay. Enabling always-available input with muscle-computer interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 167–176, 2009.

[116] M. Sato, I. Poupyrev, and C. Harrison. Touch: enhancing touch interaction on humans, screens, liquids, and everyday objects. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 483–492, 2012.

[117] A. Sears and B. Shneiderman. High precision touchscreens: design strategies and comparisons with a mouse. *International Journal of Human-computer Studies / International Journal of Man-machine Studies*, 34(4):593–613, 1991.

[118] M. St.John, W. Harris, and G. A. Osga. Designing for multitasking environments: Multiple monitors versus multiple windows. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, pages 1313–1317, 1997.

[119] D. J. Sturman and D. Zeltzer. A survey of glove-based input. *IEEE Computer Graphics and Applications*, 14(1):30–39, 1994.

[120] S. Subramanian and P. Irani. Pressuremove: pressure input with mouse movement. In *Proceedings of the IFIP Conference on Human-Computer Interaction*, pages 25–39, 2009.

[121] M. Sugimoto and K. Hiroki. Hybridtouch: an intuitive manipulation technique for pdas using their front and rear surfaces. In *Proceedings of the Mobile HCI*, pages 137–140, 2006.

[122] D. S. Tan and M. Czerwinski. Effects of visual separation and physical discontinuities when distributing information across multiple displays. In *Proceedings of the IFIP Conference on Human-Computer Interaction*, pages 252–255, 2003.

[123] D. S. Tan, B. Meyers, and M. Czerwinski. Wincuts: manipulating arbitrary window regions for more effective use of screen space. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1525–1528, 2004.

[124] 2D Technology. http://www.2dtg.com/.

[125] Mitsubishi 3D touch panel. http://techon.nikkeibp.co.jp/english/news_en/20090310/166952/.

[126] N. Villar, S. Izadi, D. Rosenfeld, H. Benko, J. Helmes, J. Westhues, S. J. Hodges, E. Ofek, A. Butler, X. Cao, and B. Chen. Mouse 2.0: multi-touch meets the mouse. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 33–42, 2009.

[127] D. Vogel and R. Balakrishnan. Distant freehand pointing and clicking on very large, high resolution displays. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 33–42, 2005.

[128] D. Vogel and P. Baudisch. Shift: a technique for operating pen-based interfaces using touch. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 657–666, 2007.

[129] M. Weiss, C. Wacharamanotham, S. Voelker, and J. Borchers. Fingerflux: near-surface haptic feedback on tabletops. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 615–620, 2011.

[130] D. Wigdor, C. Forlines, P. Baudisch, J. Barnwell, and C. Shen. Lucidtouch: a see-through mobile device. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 269–278, 2007.

[131] A. D. Wilson. Robust vision-based detection of pinching for one and two-handed input. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 255–268, 2006.

[132] A. D. Wilson. Using a depth camera as a touch sensor. In *Proceedings of the International Conference on Interactive Tabletops and Surfaces*, pages 69–72, 2010.

[133] A. D. Wilson and H. Benko. Combining multiple depth cameras and projectors for interactions on, above, and between surfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 273–282, 2010.

[134] R. Wimmer and P. Baudisch. Modular and deformable touch-sensitive surfaces based on time domain reflectometry. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 517–526, 2011.

[135] J. O. Wobbrock, B. A. Myers, and H. H. Aung. The performance of hand postures in front- and back-of-device interaction for mobile computing. *International Journal of Human-computer Studies / International Journal of Man-machine Studies*, 66(12):857–875, 2008.

[136] K. Yatani, K. Partridge, M. W. Bern, and M. W. Newman. Escape: a target selection technique using visually-cued gestures. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 285–294, 2008.

[137] S. Zhai, S. Conversy, M. Beaudouin-Lafon, and Y. Guiard. Human on-line response to target expansion. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 177–184, 2003.

[138] H. Zhang, X. D. Yang, B. Ens, H. N. Liang, P. Boulanger, and P. Irani. See me, see you: a lightweight method for discriminating user touches on tabletop displays. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 2327–2336, 2012.