LensMouse: Augmenting the Mouse with an Interactive Touch Display

Xing-Dong Yang[1], Edward Mak[2], David McCallum[2], Pourang Irani[2], Xiang Cao[3], Shahram Izadi[3]
[1]Dept. of Computing Science University of Alberta, Canada
[2]Dept. of Computer Science University of Manitoba, Canada
[3]Microsoft Research Cambridge United Kingdom
xingdong@cs.ualberta.ca, ed.mak.is@gmail.com, {ummccal, irani}@cs.umanitoba.ca, {xiangc, shahrami}@microsoft.com

ABSTRACT
We introduce LensMouse, a novel device that embeds a touch-screen display – or tangible 'lens' – onto a mouse. Users interact with the display of the mouse using direct touch, whilst 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 migrating 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 the view separation that results from introducing such a display-based device. Our results reveal that overall users are more effective with LenseMouse than with auxiliary application windows that are managed either in single or dual-monitor setups. We conclude by presenting other application scenarios that LensMouse could support.

Author Keywords
Input devices, mouse with touch display.

ACM Classification Keywords
H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

GENERAL TERMS
Design, Experimentation, Performance.

INTRODUCTION
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 [6, 19, 39, 9, 20] – perhaps the most successful being the scroll-wheel [20]. The fact that the mouse is so central in most peoples' 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 [6, 19, 39, 20, 9, 35]. But why not output too?

In this paper we explore this very theme. We present a novel mouse prototype augmented with a direct touch display, which we call LensMouse. The display acts as a tangible and multi-purpose auxiliary window – or *lens* – through which users can view additional information without consuming screen real-estate on the user's monitor. Equally important is the provision of direct-touch input on the LensMouse display. With a touch of a finger, users can directly interact with content on the auxiliary display.

LensMouse allows users to interact with and view auxiliary digital content without needing to use a dedicated input device, or indeed change their hand posture significantly. We 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 content, pop-up dialog boxes, and performing touch gestures.

Figure 1 - LensMouse prototype showing an overview map from a real-time strategy game.

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. Whilst 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 or close the auxil-

CHI 2010, April 10-15, 2010, Atlanta, Georgia, USA.

iary 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 paper 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.

Our contributions in this paper are therefore: 1) a novel input device prototype augmented with an interactive touch display; 2) a solution to overcome challenges with auxiliary windows; 3) a demonstration of some of the benefits of our device through a user experiment; 4) a set of applications and interactions that can benefit from such a device.

RELATED WORK

Adding a touch-enabled display onto 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 [20]. Other augmentations include extending the degrees-of-freedom of the mouse [6, 19], adding pressure input [9, 35], providing multi-touch input [39], supporting bi-manual interactions [6, 25], and extending the controls on a mice to a secondary device [25]. Our research prototype complements and extends this existing literature by considering a rich set of both output and input functionalities on top of regular mouse-based input.

On 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 our work.

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 [10, 21]. 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 [12]. With more screen space available, the increased amount of mouse trips among 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 second monitor [12, 14]. A quantitative result by Bi et al. [7] 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 document windows simultaneously or to monitor updates peripherally. For tasks involving frequent switching between applications, dual-monitor users are faster and have less workload than single-monitor users [24, 36]. 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 [12]. 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. [37] 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 [36]. These findings on the negligible effects of visual separation support the concept of having a separate auxiliary display, such as that on 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 [34]. Alleviating this issue has been the focus of much research as we highlight in the next section.

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 [4] offsets the cursor's landing position to eliminate wrapping effects introduced by Stitching. Although both methods are effective [29], users still need to make a substantial amount of cursor movements to acquire remote targets. Object cursor [13] addresses this issue by ignoring empty space between objects. Delphian Desktop [2] 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 [1, 11]. 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 [3]. This re-

sults in bringing the mouse pointer close to the task but at the cost of mode switching. The Mudibo system [15] 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 [16] but distributing redundant copies of information content across multiple monitors introduces problems such as distracting the user's attention and wasting screen estate. Ninja cursors [23] 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.

LENSMOUSE PROTOTYPE

We created our LensMouse prototype by attaching a touch-enabled Smartphone (HTC touch) to the base of a USB mouse (Figure 1). As miniaturized touch displays become increasingly common and cheap, we can imagine such a display being integrated at a lower cost.

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 are overview + detail (or focus+context) views [30, 41]. 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 1). Overviews have proven to be useful for a variety of tasks [18, 31].

Auxiliary windows displayed on the LensMouse are referred to as 'lenses'. LensMouse can incorporate several 'lenses' simultaneously. To switch between different lenses, a 'lens-bar' is displayed at the bottom of the display (Figure 1). 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.

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 2).

Figure 2 - 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 [5, 22]. 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 [26, 33]. For example, distorting target size, i.e. making it larger, can improve targeting performance [26]. This can be detrimental to the selection task if nearby targets are densely laid out [26]. 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 ef-

fects, 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.

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 [37, 16]. 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.

Materials

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.

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 our participants were right-handed users.

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-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-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.

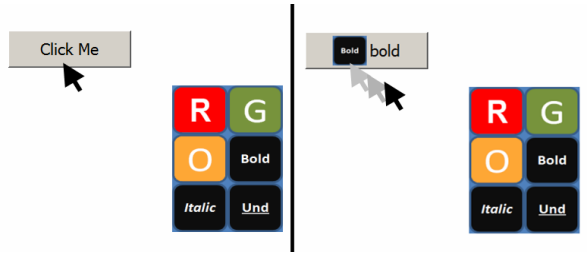

Figure 3 –The instruction (right) *bold* showed only after participants successfully clicked the instruction button (left). It instructs to click on the *bold* icon in the tool palette window.

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.

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 [37] 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 [32, 40]. Based on the size of our display, we can have palettes containing upto 18 icons on the LensMouse. We however restricted our 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 users 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* × *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* × 2 *Number of Icons* × 8 *Blocks* × 18 *Repetitions* × 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.

Procedure

At the start of each trial, an instruction button was placed randomly in one of three predefined regions indentified 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 4. 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 168~728 pixels away from the bottom-right corner; the Middle region 728~ 1288 pixels; and the Far region 1288~1848 pixels. This would allow us to test the impact of visual separation if it was present.

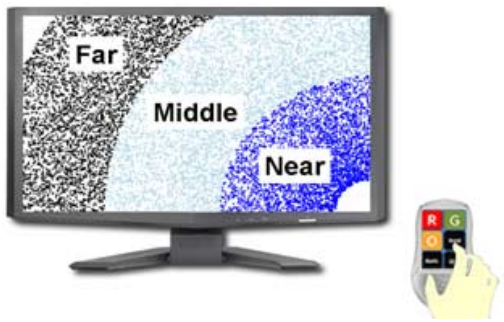

Figure 4 – Instruction buttons were placed in 3 regions. The regions demarcated areas based on the distance to the bottom-right corner of the screen.

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.

Results and discussion

The collected data was analyzed using a repeated measure ANOVA test and Tamhane 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. Our analysis of completion time does not include trials in which an error was made during the tool selection. The overall average completion time was 1245 ms. 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 5 shows average completion time for each *Display Type* by *Number of Icons*. We found no interaction effects on *Display Type* × *Number of Icons* ($F_{3,39}$ = 0.262, p = 0.852).

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

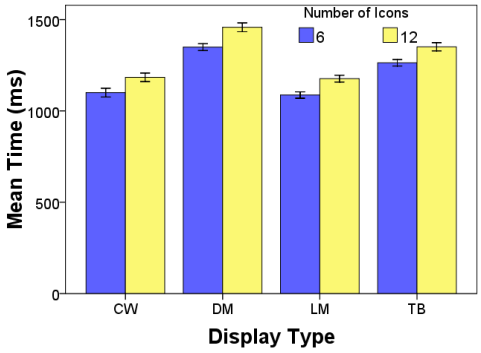

Figure 5 - Task completion time vs. Display Type and Number of Icons. Error bars represent +/- 2 standard error.

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 [28]. 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* × *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 [40], 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* × *Display Type* ($F_{21,273}$ = 1.258, p = 0.204) or for *Block* × *Number of Icons* ($F_{7,91}$ = 0.411, p = 0.893). As can be seen in Figure 6 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 $p > 0.75$).

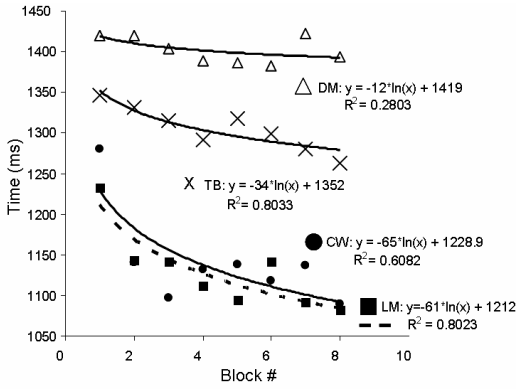

Figure 6 – Learning effects: Task completion time vs. block number.

Although the results of LensMouse and *Context Window* are similar, the learning effects between all techniques were slightly 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 4th or 5th block of trials.

Effects of visual separation

Our 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 our 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 our 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 significantlly 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. Whilst 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 1). 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 our 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 our prototype and would give players who have access to LensMouse a significant advantage over those without the device.

DISCUSSION

Our 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. Our 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.

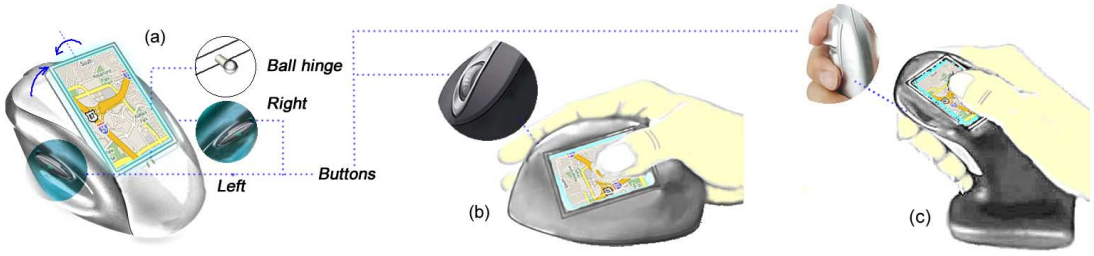

Figure 7 – Various possibilities for an ergonomic design of LensMouse. (a) a rotatable display allowing most freedom in viewing position; (b) having the display oriented toward the user, but limited by handedness; (c) on other devices such as a joystick

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 7 a). Another solution is to place the display and the buttons on different facets of the mouse (Figure 7 b). 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 [39] could be performed easily using thumb and index finger. Furthermore, a joystick-shape LensMouse (Figure 7c) could allow users to operate the touch screen using the thumb.

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 [40] 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.

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 [38]. 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 our experiment.

Preview lens. LensMouse can also serve as a means to preview content associated with a UI object without committing a selection. Figure 8a shows a 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.

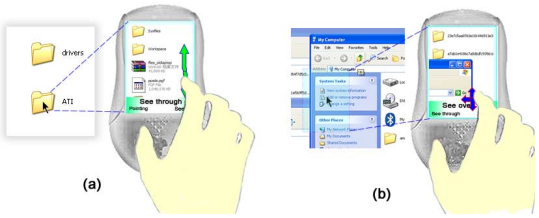

Figure 8 - Special lenses:(a) Previewing folder content. (b) Seeing through overlapping windows.

See-through lens. Another use of the LensMouse is for seeing through screen objects [8], e.g. overlapping windows (Figure 8b). 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 our 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. 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 9a). The user can move the LensMouse first to coarsely position the cursor near the target, 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 9b). 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.

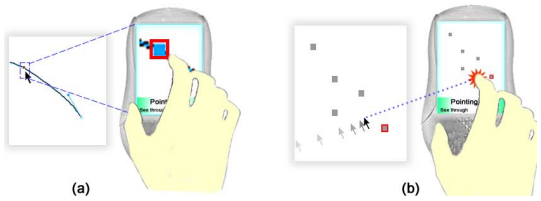

Figure 9 - Absolute + Relative pointing. (a) For small targets, and (b) to cover long distances.

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) [17]. 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 our current prototype, users could use simple, pre-configured gestures on the mouse screen to make rapid responses, such as "I am busy" [6].

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 scroll-bar. As a custom controller, LensMouse can provide any number of controls that can fit on the display for a given application. In future work, our 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 [27]. 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.

CONCLUSION

In this paper, we presented LensMouse, a novel device that serves 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 our quantitative results showing that LensMouse is a welcome addition to the suite of techniques for augmenting the mouse.

Additionally, we demonstrated the utility of LensMouse through various applications, including preview and see-through lenses, gestural interaction, and so on. Future work will focus on improving the hardware design, especially its ergonomics. We also plan to carry out several studies to evaluate the performance of LensMouse in various display environments including wall-size displays and tabletop displays. Finally, we will also explore diverse usage contexts of LensMouse, such as in multi-user environments.

REFERENCES

1. Ashdown, M., Oka, K., Sato, Y. (2005). Combining head tracking and mouse input for a GUI on multiple monitors. *CHI Extended Abstracts*, 1188-1191.
2. Asano, T., Sharlin, E., Kitamura, Y., Takashima, K., and Kishino, F. (2005). Predictive Interaction Using the Delphian desktop. *UIST*, 133-141.
3. Benko, H. and Feiner, S. (2005). Multi-Monitor Mouse. *CHI Extended Abstracts*, 1208-1211.
4. Baudisch, P., Cutrell, E., Hinckley, K., and Gruen, R. (2004). Mouse Ether: Accelerating the Acquisition of Targets Across Multi-Monitor Displays. *CHI*, 1379-1382.
5. Baudisch, P. and Gutwin, C. (2004). Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. *CHI*, 367-374.
6. Balakrishnan, R. and Patel, P. (1998). The PadMouse: Facili-tating selection and spatial positioning for the non-dominant hand. *CHI*, 9-16.
7. Bi, X. and Balakrishnan, R. (2009). Comparing Usage of a Large High-Resolution Display to Single or Dual Desktop Displays for Daily Work. *CHI*, 1005-1014.
8. Bier, E.A., Stone, C.M., Pier, M, Buxton, W., and DeRose, T.D. (1993). Toolglass and Magic Lenses: The See-Through Interface. *SIGGRAPH*, 73-80.
9. Cechanowicz, J., Irani, P., Subramanian, S. (2007). Augmenting the mouse with pressure sensitive input. *CHI*, 1385-1394

10. Chen, N., Guimbretière, F., Dixon, M., Lewis, C., and Agrawala, M. (2008). Navigation Techniques for Dual-Display E-Book Readers. *CHI*, 1779-1788.
11. Dickie, C., Hart, J., Vertegaal, R., and Eiser, A. (2006). LookPoint: an evaluation of eye input for hands-free switching of input devices between multiple computers. *OZCHI*, 119-126.
12. Grudin, J., (2001). Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. *CHI*, 458-465.
13. Guiard, Y., Blanch, R., and Beaudouin-Lafon, M. (2004). Object pointing: a complement to bitmap pointing in GUIs. *GI*, 9-16.
14. Hutchings, D. R., Smith, G., Meyers, B., Czerwinski, M., Robertson, G. (2004). Display space usage and window management operation comparisons between single monitor and multiple monitor users. *AVI*, 32-39.
15. Hutchings, D.R. and Stasko, J. (2005). mudibo: Multiple dialog boxes for multiple monitors. *CHI Extended Abstracts*, 1471-1474.
16. Hutchings, D.R. and Stasko, J. (2007). Consistency, Multiple Monitors, and Multiple Windows. *CHI Extended Abstracts*, 211-214.
17. Hutchings, H.M. and Pierce, J.S. (2006). Understanding the whethers, hows, and whys of divisible interfaces. *AVI*, 274-277.
18. Hornbæk, K., Bederson, B. B., and Plaisant, C. (2002). Navi-gation patterns and usability of zoomable user interfaces with and without an overview. *TOHCI*, 9(4), 362–389.
19. Hinckley, K., Sinclair, M., Hanson, E., Szeliski, R., and Con-way, M. (1999). The VideoMouse: a camera-based multi-degree-of-freedom input device. *UIST*, 103-112.
20. Hinckley, K., Cutrell, E., Bathiche, S., and Muss, T. (2002). Quantitative analysis of scrolling techniques. *CHI*, 65-72.
21. Hinckley, K., Dixon, M., Sarin, R., Guimbretiere, F., and Balakrishnan, R. (2009). Codex: a dual screen tablet computer. CHI, 1933-1942.
22. Ishak, E. W., Feiner, S. K. (2004). Interacting with hidden content using content-aware free-space transparency. *UIST*, 189-192.
23. Kobayashi, M. and Igarashi, T. (2008). Ninja Cursors: Using Multiple Cursors to Assist Target Acquisition on Large Screens. *CHI*, 949-958.
24. Kang, Y. and Stasko, J. (2008) Lightweight Task/Application Performance using Single versus Multiple Monitors: A Comparative Study. *GI*, 17-24.
25. Myers, B. A., Miller, R. C., Bostwick, B., and Evankovich, C. (2000). Extending the windows desktop interface with con-nected handheld computers. *4th USENIX Windows Systems Symposium*, 79-88.
26. McGuffin, M. and Balakrishnan, R. (2002). Acquisition of expanding targets. *CHI*, 57-64.
27. Morris, M.R., Brush, A.J.B., and Meyers, B. (2007). Reading Revisited: Evaluating the Usability of Digital Display Surfaces for Active Reading Tasks. *Tabletop*, 79-86.
28. MacKenzie, S. (1992). Fitts' law as a research and design tool in human-computer interaction. Human-Computer Interaction 7(1): 91–139.
29. Nacenta, M., Mandryk, R., Gutwin, C. (2008). Targeting Across Displayless Space. *CHI*, 777-786.
30. Plaisant, C., Carr, D., and Shneiderman, B. (1995). Image-browser taxonomy and guidelines for designers. *IEEE Soft-ware*, 12(2), 21–32.
31. Pietriga, E., Appert, C., and Beaudouin-Lafon, M. (2007). Pointing and Beyond: an Operationalization and Preliminary Evaluation of Multi-scale Searching. *CHI*, 1215–1224.
32. Parhi, P., Karlson, A., and Bederson, B. (2006). Target size study for one-handed thumb use on small touch screen devices. MobileHCI, 203-210.
33. Ramos, G., Cockburn, A., Beaudouin-Lafon, M. and Balakrishnan, R. (2007). Pointing Lenses: Facilitating Stylus Input throughVisual- and Motor-Space Magnification, *CHI*, 757-766.
34. Ringel, M. (2003). When One Isn't Enough: An Analysis of Virtual Desktop Usage Strategies and Their Implications for Design. *CHI Extended Abstracts*, 762-763.
35. Shi, K., Irani, P, and Subramanian, S. (2009). PressureMove: Pressure Input with Mouse Movement, *INTERACT*, 25-39.
36. St. John, M., Harris, W., and Osga, G. A. (1997). Designing for multitasking environments: Multiple monitors versus multiple windows. *HFES*, 1313-1317.
37. Tan, D.S. and Czerwinski, Mary (2003). Effects of Visual Separation and Physical Discontinuities when Distributing Information across Multiple Displays. *INTERACT*, 252-260.
38. Tan, D.S., Meyers, B., and Czerwinski, M. (2004). WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. *CHI EA*, 1525-1528.
39. Villar, N., Izadi, S., Rosenfeld, D., Benko, H., Helmes, J., Westhues, J., Hodges, S., Butler, A., Ofek, E., Cao, X., and Chen, B. (2009). Mouse 2.0: Multi-touch Meets the Mouse. *UIST*, 33-42.
40. Vogel, D. and Baudisch, P. (2007). Shift: A Technique for Operating Pen-Based Interfaces Using Touch. *CHI*, 657-666.
41. Ware, C. and Lewis, M. (1995). The DragMag image magni-fier. *CHI*, 407-408.