Ninja Cursors: Using Multiple Cursors to Assist Target Acquisition on Large Screens

Masatomo Kobayashi
Department of Computer Science,
The University of Tokyo
7-3-1 Hongo, Bunkyo, Tokyo, Japan
kobayash@is.s.u-tokyo.ac.jp

Takeo Igarashi
Department of Computer Science,
The University of Tokyo / SORST, JST
7-3-1 Hongo, Bunkyo, Tokyo, Japan
takeo@acm.org

ABSTRACT
We propose the "ninja cursor" to improve the performance of target acquisition, particularly on large screens. This technique uses multiple distributed cursors to reduce the average distance to targets. Each cursor moves synchronously following mouse movement. We present the design and implementation of the proposed technique, including a method to resolve the ambiguity that results when multiple cursors indicate different targets simultaneously. We also conducted an experiment to assess the performance of the ninja cursor. The results indicate that it can generally reduce movement time. However, the performance is greatly affected by the number of cursors and target density. Based on these results, we discuss how our technique can be put into practical use. In addition to presenting a novel method to improve pointing performance, our study is the first to explore a variable number of cursors for performing pointing tasks.

Author Keywords
Pointing, Multiple Cursors, Large Screens, Fitts' Law.

ACM Classification Keywords
H5.2. Information interfaces and presentation (e.g., HCI): User Interfaces. - Graphical user interfaces.

INTRODUCTION
Pointing is the most fundamental operation in windows, icons, menus, and pointers (WIMP) interfaces. For this reason, several techniques have been proposed to improve the performance of pointing tasks in various contexts. In general, these techniques have attempted to reduce the index of difficulty (ID) based on Fitts' law [7]:

$$ID = \log_2\left(\frac{D}{W} + 1\right),$$

where D is the distance to the target and W is its width. A longer movement distance or a smaller target object reduces performance.

Several approaches have been used to reduce the ID. One is to modify the cursor's behavior to increase W and/or reduce D. For example, the bubble cursor [9] dynamically changes cursor size to increase the virtual W to exploit the void space around the target. This technique is very effective for pointing to a small target in otherwise empty space. However, it is not particularly effective for pointing to a distant target located beyond many obstructions, as the user must move the cursor all the way to the distant target. In a desktop configuration, the user is forced to repeatedly reposition the mouse to reach a distant target location, which reduces the performance of pointing and results in user frustration. In addition, the logarithmic relationship between D and movement time (MT) would likely be broken when pointing to a very distant target on a wall-sized screen [3, 10]. Instead of enlarging W, the Delphian Desktop [2] reduces D by allowing the cursor to jump toward the target location. It uses a prediction algorithm to permit this long-distance movement with minimal mouse operation. This works as long as the algorithm can successfully determine the goal location. However, it is generally difficult to predict a distant target, and prediction errors can confuse the user.

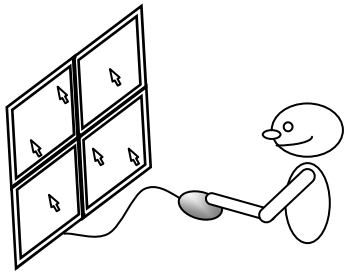

Figure 1: Multiple synchronous movement of cursors cover a large screen.

We propose another method, called the "ninja cursor," to reduce the ID by modifying cursor representation. It uses multiple cursors to reduce D without applying a prediction algorithm. The cursors move synchronously, following the physical movement of the mouse (Fig. 1). The user can point to a target with one of the cursors located nearest to it.

We use a simple waiting queue algorithm to prevent two or more cursors from pointing to multiple targets simultaneously.

In general, our technique aims to improve the performance of pointing operations by mapping the single mouse movement to the movement of multiple cursors. By allowing users to point to a target with less motion, the proposed technique should reduce the *MT* as long as the number of cursors is suitable for the target density. We conducted an experiment to determine how cursor number and target density affect performance. Cursor number is a parameter that so far has been explored only rarely, unlike cursor size, target number, and target size. The ninja cursor technique allows the investigation of this parameter.

The remainder of this paper is organized as follows. First, we summarize related work. Second, we describe the design and implementation of the ninja cursor. Third, we evaluate our technique using different numbers of cursors and targets and compare the resulting performance to that of the normal single-point cursor. Fourth, we discuss the ninja cursor in more practical contexts. Finally, we present our conclusions.

RELATED WORK

Modifying Target Representation

Two approaches have been used previously to modify target representation to reduce the ID: the actual or virtual *W* of the target is enlarged, and/or *D* is reduced by temporarily bringing the target toward the cursor.

The Dock in the Apple Mac OS X is a typical example of the first approach. It enlarges the actual size of the target dynamically, by predicting which item is the target according to the location of the mouse cursor. McGuffin and Balakrishnan [17] examined the performance of pointing operations with such a temporarily expanded target. They found that target expansion improved the performance even if the expansion occurred relatively late in the movement of the cursor toward the target. They also found that these tasks can be modeled with Fitts' law using the expanded *W*. However, it seems that the target expansion technique would not work well with closely spaced targets. Manual target expansion techniques have also been proposed. For example, pointing lenses [18] provide a magnified view of the screen so that users can interact with enlarged targets. A user action such as pen pressure or time delay activates the lens.

One of the studies that took the second approach utilized drag-and-pop [3]. This creates temporary proxies of possible targets and shows them near the pointing cursor when users start dragging an object so that they can easily drop it on the target. This technique improved performance for a large *D*. However, it can create erroneous distracting proxies because the prediction of distant targets is difficult, especially when many potential targets exist. Grossman and Balakrishnan [9] reported that this technique would work well only with a low-density target distribution.

In contrast to these attempts at changing the visual representation of targets, Blanch *et al.* [6] proposed semantic pointing, which made the control-display (C-D) gain adaptive to improve both *D* and *W* virtually in the motor space; dynamically increasing the gain while outside a target and decreasing it while inside a target will virtually shorten *D* while virtually enlarging *W*. However, this technique would be less effective with multiple targets because intervening targets would slow down the cursor in its movement toward the distant target.

Modifying Cursor Representation

As described in the Introduction, two approaches have been used to improve pointing performance by modifying the behavior of the cursor: enlarging the size of the cursor's hotspot to increase virtual *W*, and causing the cursor to jump toward the target to reduce *D* in motor space.

The area cursor [13] is one of the earliest attempts using the former approach. Instead of a single pixel hotspot, it uses a rectangular activation region to enlarge the effective target width, which is determined by the width in motor space. This approach is effective for pointing to small targets. The performance of pointing tasks with an area cursor can be modeled with Fitts' law using the effective width *W*. However, the larger activation region causes ambiguity because it is possible for there to be multiple targets inside the region at the same time. Worden *et al.* [19] proposed the combination of an area cursor and a point cursor to address this problem. This allows using the point cursor to determine the single target when multiple targets exist inside the area cursor. The bubble cursor [9] is a more sophisticated solution to the problem of ambiguity. This technique dynamically changes the cursor size so that it contains only a single target.

One example of the latter approach is object pointing [11]. This technique makes the cursor jump across the void space between selectable targets, making it easier to point at distant ones. The Delphian Desktop [2] proposed more aggressive jumping based on a prediction algorithm. Using an online algorithm, it determined the goal location based on the direction of movement and peak velocity. Lank *et al.* [15] proposed a method based on the theory of motion kinematics to improve the precision of the prediction. However, these prediction-based interfaces all share a problem of uncertainty. Even using a high-precision prediction algorithm, the resulting behavior is still nondeterministic in nature, and unexpected results can confuse the user. The behavior of our ninja cursor, however, is completely deterministic and continuous, which makes it much less distracting. Another attempt to reduce *D* was the Multi-Monitor Mouse [5], which allowed users to move the cursor quickly across displays in a multi-display environment using hot keys. The ninja cursor does not require an explicit action to switch displays, and is applicable to a single large screen.

Performance Evaluation of Pointing Operations

Many studies, including Fitts' original work [7], have shown how the target size affects the pointing performance in diverse contexts. Generally, these demonstrated a positive relationship between the target size and the performance. Recent studies [9, 13, 19] have also examined the cursor size, as described above. An enlarged cursor could improve the performance, increasing the virtual width of targets. The number of targets has also been the subject of frequent study. For example, both target expansion [17] and object pointing [11] were evaluated with a high density of targets. The results showed that these techniques were less effective when distracting targets existed around the goal target. However, Grossman and Balakrishnan [9] experimented with a range of target densities to show that their bubble cursor technique was effective in the presence of many targets. The target density was also considered for specific input devices such as the touch screen [14].

Although the target size, the cursor size, and even the number of targets have been examined, few studies have explored how the number of cursors affects the performance of target acquisition tasks. Therefore, we focused on this aspect using our ninja cursor technique.

Using Multiple Cursors

Several applications have supported the use of multiple cursors for a specific purpose. One of them is a collaborative drawing tool [8]. Using *n* cursors that can be moved individually, this tool allows *n* users to interact with different objects on the same shared screen.

Bimanual interfaces [12] might be considered multiple cursor systems because they give users an individual cursor for each hand. Exploiting two hotspots, the users could take advantage of two-handed manipulation, moving two points of action interactively on the screen.

These techniques did not attempt to improve the basic performance of target pointing. However, they did inspire the concept of the ninja cursor.

THE NINJA CURSOR TECHNIQUE

To reduce the ID in target acquisition tasks, our ninja cursor technique uses multiple pointing cursors that move synchronously following the mouse movement (Fig. 2). A user can point to an object with minimal effort using the cursor that is nearest to the target object. If mouse cursors and target objects are both uniformly distributed on the two-dimensional (2-D) screen, the average minimum distance to a target is expected to be reduced to D_n as shown in this equation:

$$D_n = \frac{D_1}{\sqrt{n}},$$

where n is the number of cursors and D_1 is the mean distance with a normal single pointing cursor (Fig. 3).

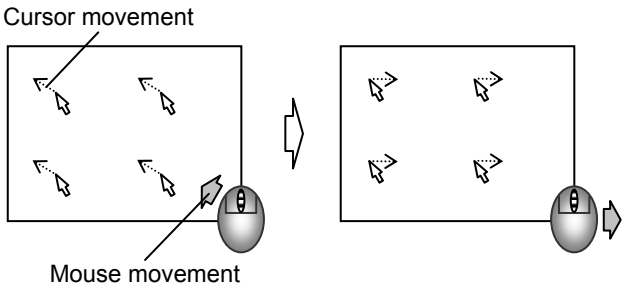

Figure 2: Each cursor follows the physical movement of the mouse.

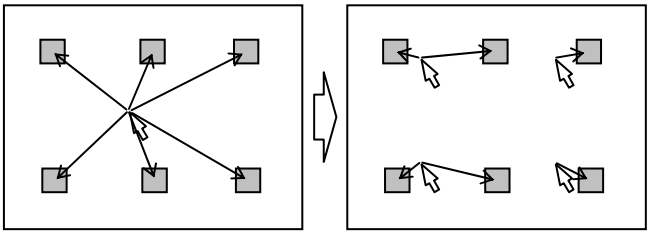

Figure 3: The arrows indicate the distances between each rectangular target and the nearest cursor. With a single cursor, the targets are far from the cursor (left). Using more cursors, on average, they become closer to at least one of the cursors (right).

We must address the possibility that two or more cursors could point to different targets at the same time. To resolve this ambiguity, the ninja cursor modifies the spatial distribution of the cursors dynamically. We use a simple waiting queue algorithm to accomplish this.

1. If multiple cursors are initially inside a target, the one that is closest to the center of the target is made active. The others are put outside the target. If only one cursor is inside a target, it is simply made active. In this context, the term "active" means currently pointing at a target.
2. If cursor C_i is not the active cursor C_{active}, and it is about to move into a target T_i, then C_i is appended to a queue, Q. As long as C_i is in Q, it never goes inside T_i even if the user moves the mouse toward T_i. The position of C_i does not change on the screen while the cursor is in this state.
3. When the user moves the mouse in another direction, that is, away from T_i , then C_i is removed from Q.
4. When C_{active} leaves the target to which it is pointing, it becomes inactive. Then, the first element of Q, C_j, is removed from the queue. As it is no longer in Q, C_j can go inside the target, T_j, and become active following mouse movement.

Using this algorithm, the ninja cursor guarantees the following four criteria:

- No more than one cursor is active at any one time.

- As long as a cursor does not attempt to enter any target, it moves freely following the movement of the mouse.
- If no active cursor is on the screen, each cursor moves freely following the movement of the mouse.
- Each cursor can point to any target as long as the user continues moving the mouse toward the target.

The order of the cursors becoming active simply depends on how long each cursor is in the waiting state because the method described above is based on a simple first-in, first-out (FIFO) strategy. The behavior of the ninja cursor will thus be easily understood. In addition, we provide several visual feedback cues to help users understand the current state of each cursor. First, the cursor color indicates one of three states for each cursor. In Fig. 4, the gray cursor is the active cursor while the black ones are currently in the waiting queue. The remaining cursors are shown as normal white arrows. Second, once a cursor tries to enter a target and goes into the waiting state, a "wall" appears around the target (Fig. 5). The length of the wall indicates the position of the cursor in the waiting queue. A shorter (longer) wall shows that the cursor is closer to (farther from) the head of the queue. The dynamic change of wall length shows a user how much movement is required to point to the intended target.

The transitions between the normal and waiting states may reduce the regularity of cursor distribution as shown in Fig. 6, as some cursors stop moving while the others continue moving following the movement of the mouse. Once the regularity is perturbed, the average distance to targets will increase. This might reduce the performance of the ninja cursor. To rectify this, we provide a "reset" feature; shaking the mouse or pressing the F5 key moves the cursors to their original uniformly distributed locations.

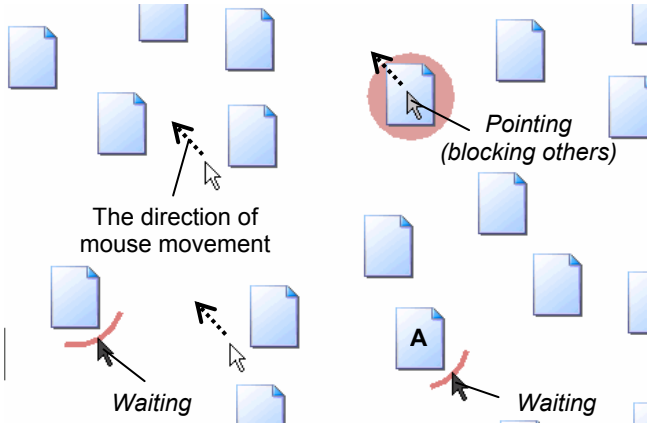

Figure 4: The user attempts to point to object A. The gray cursor currently points the highlighted object. The black cursors wait for the gray one to leave the object. Using multiple cursors would reduce the minimal distance to the target object.

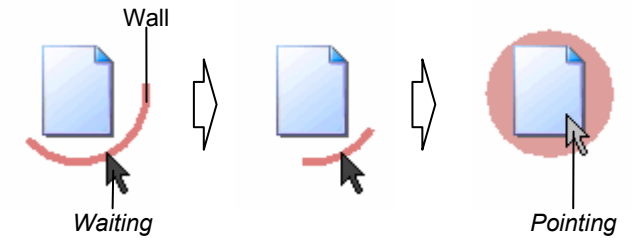

Figure 5: The length of the wall represents the position in the waiting queue. As the user continues to move toward the target, the wall length becomes shorter, indicating the amount of movement remaining to enter the target.

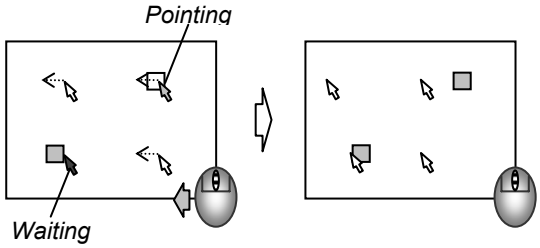

Figure 6: A transition between the normal and waiting states may reduce the regularity of the cursor distribution.

We anticipate that users of traditional pointing interfaces will quickly be able to start using the ninja cursor because its behavior is locally identical to a traditional one as long as the cursor that the user focuses on is not in the waiting queue. The behavior is completely the same as the traditional interface when the number of cursors is equal to 1. This indicates the possibility of a seamless transition from traditional interfaces to ninja cursors. In addition, it is common for a cursor to follow mouse movement imperfectly because of small errors in the optical or physical sensor in the mouse. Although such poor behavior reduces the performance of target pointing operations, it does not confuse or surprise the user. This is one of the possible advantages of the ninja cursor over previous "cursor-jumping" techniques, which might occasionally cause the user to lose the cursor location. In the ninja cursor, every cursor is always visible and moves continuously.

EXPERIMENT

Although the ninja cursor theoretically reduces the ID of pointing tasks, some uncertainties still exist that might reduce its performance in practice. For example, how often each cursor enters into the waiting state affects the performance because the cursors in the waiting queue require extra mouse movement before becoming free again. Moreover, such a waiting event makes it difficult to assess the amount of movement accurately in advance. This could reduce the efficiency of pointing operations. We needed to determine empirically how the number of cursors and the frequency of waiting events affect the performance. In this section, we describe a study we conducted to determine the relationships among the number of cursors, the target

density, and the pointing performance. As a baseline condition, we also examined the performance of a traditional single cursor using the same test configuration. One of our main concerns was whether the ninja cursor could outperform the normal cursor even with a high target density.

Participants

Eight volunteers aged 23–28 participated in the experiment. All were frequent users of traditional WIMP interfaces, and used the mouse with their right hand.

Equipment

We use a 3.0 GHz Pentium 4 PC running Windows XP, connected to dual 19" displays with a resolution of 3200 × 1200 pixels (1600 × 1200 for each) and a standard optical mouse. Figure 7 shows the experimental setup. The two displays were located side-by-side and worked as a single virtual screen. The left and right displays were the primary and secondary displays, respectively. The mouse speed and acceleration rate were set to the Windows XP default values (middle speed, no acceleration). We developed our experimental software using Java. In this experiment, distances were measured in pixels.

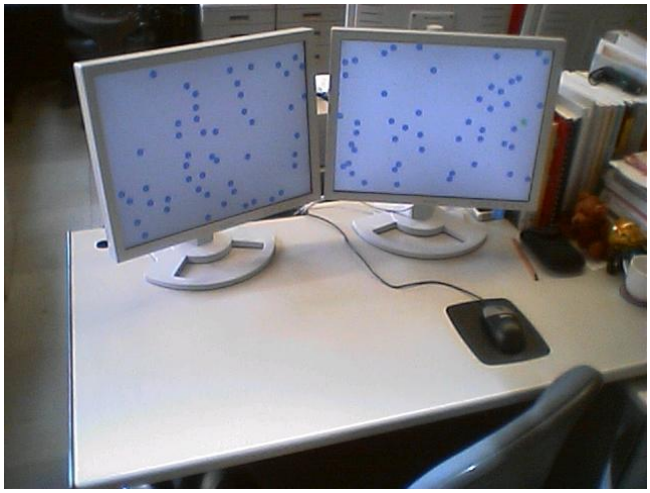

Figure 7: The experimental setup consisted of a Windows PC, dual displays forming a 3200 × 1200 virtual screen, and a standard optical mouse.

Cursor Arrangement

We tested four cursor configurations: a traditional point cursor (Point), 2 cursors (Ninja-2), 8 cursors (Ninja-8), and 18 cursors (Ninja-18). The latter might not be practical because too many cursors would be visually distracting. However, we studied all these configurations to analyze the characteristics of the ninja cursor in detail. For each ninja cursor configuration, the cursors were located in an evenly distributed regular grid pattern, as shown in Fig. 8. In Point (single-cursor configuration), the cursor was initially located in the center of the primary screen to emulate common cursor movement within and between screens.

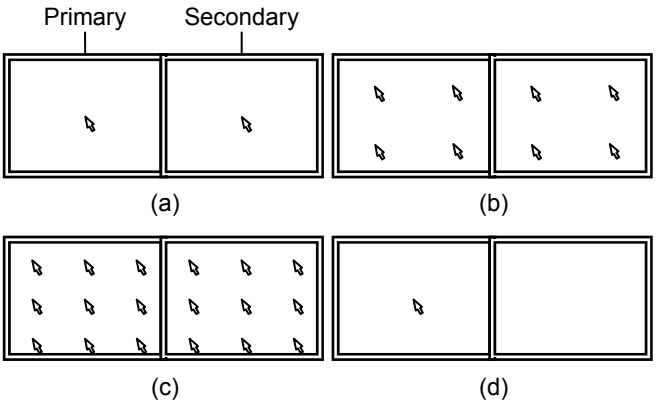

Figure 8: (a)–(d) show the initial cursor distributions in Ninja-2, Ninja-8, Ninja-18, and Point, respectively.

Design

We used a within-participant design. The independent variables were cursor type, *CT* (Point, Ninja-2, Ninja-8, and Ninja-18), the number of targets, *N* (1, 100, and 400), and the target width *W* (32, 48, and 64 pixels). We tested 36 combinations in total. A combination of *N* and *W* determined the target density, where $N = 100$ corresponds roughly to the density of a typical desktop and $N = 400$ corresponds roughly to the density of a desktop filled with icons. Thirty-six combinations of *CT*, *N*, and *W* were used. Each participant performed 10 trials of each combination presented in a pseudorandom order.

The aim of the ninja cursor is to reduce the ID on average rather than for each trial. Thus, we did not control the distance, *D*, for each cursor–target pair. Instead, we evaluated the mean ID through repeated trials, using targets distributed pseudorandomly. For each trial, the ID was calculated as follows:

$$ID = \log_2\left(\frac{D_{\min}}{W} + 1\right),$$

where $D_{\min}$ is the distance between the goal target and the nearest cursor, and *W* is the target width. Note that this definition represents the lower limit of the ID for each trial. Once the cursor falls into the waiting state, the actual value of ID would increase.

Procedure

Figure 9 shows a screenshot of the experimental software (*CT* = Ninja-8, $N = 100$, and $W = 48$). We used circular targets to control the target width in all directions. Previous research [16] showed that the acquisition of a circular target can be modeled similar to that of a rectangular target of the same width. The highlighted target is the goal target, and the others are distracting targets. All targets are distributed pseudorandomly across the screen, avoiding overlap. In each trial, each participant was required to click on the goal target with any cursor as quickly as possible.

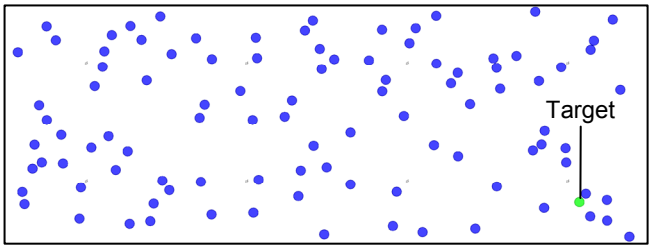

Figure 9: The highlighted target is a goal target while the others are distracting targets.

We explained the purpose of the experiment and the use of the experimental software to each participant prior to an experimental session. The participants were allowed to familiarize themselves with the system and practice each task before testing. They were also allowed to take a break between blocks. Each session took approximately 30 minutes including practice trials.

All cursors and targets were shown on the screen before each trial started. This allowed the participants to know in advance where the goal target was and how the cursors and targets were distributed. We did not test the time to decide which cursor to use because decision time could not be examined properly in this experiment. As each cursor was initially placed on a simple grid layout for each trial, the participants could easily determine which cursor was nearest regardless of whether they could see the cursors in advance. Although decision time could possibly affect the performance of the ninja cursor, we decided to address this issue in future studies.

Each trial consisted of two clicks with a mouse movement between them. The first click indicated the beginning of a trial. The participant then moved the cursors to click on the goal target with one of them. The trial ended when the second click occurred regardless of whether the goal target was successfully clicked. If no cursor pointed to the goal target when the second click occurred, the trial was counted as a failure. Distinctive sound effects indicated the success or failure of each trial. Note that participants could not move any cursor until they clicked the mouse button to start a trial. When the trial was complete, the cursors returned to their original locations shown in Fig. 8.

Results

Movement Time

Analysis of variance showed significant main effects of *CT* ($F_{3,2764} = 63.67$, $p < .001$), *N* ($F_{2,2764} = 397.2$, $p < .001$), and *W* ($F_{2,2764} = 40.97$, $p < .001$) on *MT*. The overall *MT*s were 1.48 s for Point, 1.16 s for Ninja-2, 1.14 s for Ninja-8, and 1.38 s for Ninja-18. There were interaction effects of *CT* × *N*, *CT* × *W*, and *N* × *W*. A *post hoc* analysis indicated that Ninja-2 and Ninja-8 significantly outperformed Point and Ninja-18. In addition, there were significant positive effects of target number and size. In contrast to Fitts' law, target size had a positive effect on *MT*. This indicates that the negative effect of cursor blocking was greater than the positive effect observed in Fitts' law.

As shown in Fig. 10, the ninja cursor generally outperformed the traditional point cursor except under a few high-density conditions. In addition, the different types of cursor were affected differently by the target density. In Point, the *MT* was affected less by the number of targets. This is natural as a single cursor never causes cursor blocking regardless of the target density. The target size generally had a negative effect, which is also a natural result predicted by Fitts' law. Under lower-density target conditions, increasing the number of cursors reduced *MT* monotonically as expected based on the assumption of average ID reduction. However, this effect was smaller under higher-density conditions. In particular, Ninja-8 and Ninja-18 had positive effects on *MT* under the highest-density conditions used, while Ninja-2 outperformed or was at least as efficient as the traditional point cursor under all test conditions.

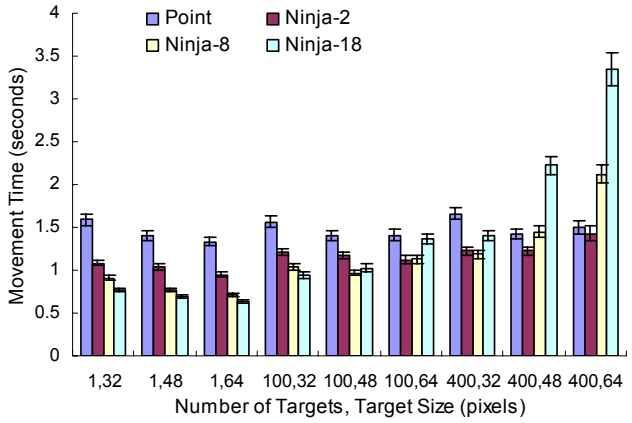

Figure 10: Movement time for each cursor type with standard errors.

To clarify the results in detail, we considered the amount of movement as well as the *MT*. This value is defined as the amount of mouse movement in the motor space. It would be larger than the amount of cursor movement on the screen if the cursor had fallen into the waiting state because the cursor required extra mouse movement to make it leave the waiting queue.

Analysis of variance showed significant main effects of *CT* ($F_{3,2764} = 760.4$, $p < .001$), *N* ($F_{2,2764} = 23.30$, $p < .001$), and *W* ($F_{2,2764} = 4.87$, $p < .01$) on the amount of movement. The overall amounts of movement were 1.21×10^3 pixels for Point, 5.80×10^2 pixels for Ninja-2, 3.66×10^2 pixels for Ninja-8, and 3.15×10^2 pixels for Ninja-18. We observed interaction effects of *CT* × *N*, *CT* × *W*, and *N* × *W*. A *post hoc* analysis indicated significant negative effects of cursor number and significant positive effects of target number and size.

Figure 11 shows that the target density did not affect the amount of movement in Ninja-2 and Point. This indicates that Ninja-2 put few cursors into the waiting state, whereas the *MT* for Ninja-2 increased for higher-density targets. A possible explanation for this paradox is that participants moved the mouse carefully under higher-density conditions, being wary of cursor blocking, regardless of whether the cursor was actually blocked. For Ninja-8 and Ninja-18, the increase in the amount of movement under higher-density conditions indicates that cursors frequently entered the waiting state, requiring extra mouse motion to leave that state. Note that the rate increase of *MT* was higher than that for the amount of movement. This is because mouse movement during the waiting phase cannot be modeled using Fitts' law, forcing users to move the mouse continuously toward the intended target. The movement in the motor space seems to be a tunnel steering task of the width *W*. Thus, we could model the operation using the steering law [1] rather than Fitts' law.

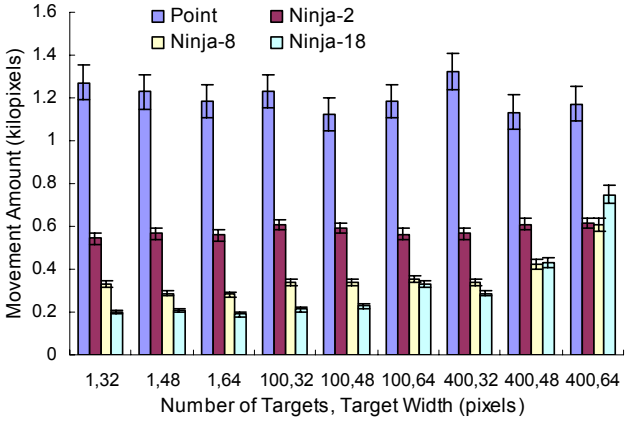

Figure 11: Movement amount for each cursor type with standard errors.

Figure 12 shows the regression lines of *MT* as a function of the estimated ID, averaged for each ID interval [*n*, *n*+1). Point and Ninja-2 showed good fits with the equation of Fitts' law (R^2 = 0.964 for Point and 0.985 for Ninja-2). Ninja-8 and Ninja-18 did not fit the equation (R^2 = 0.597 for Ninja-8 and 0.004 for Ninja-18) because of outliers caused by cursor blocking, particularly when the ID was low.

In summary, Ninja-2 outperformed the traditional pointing cursor even when the targets were as dense as a desktop filled with icons. Ninja-8 outperformed Ninja-2 only when the target density was less than or equal to a normal desktop. Ninja-18 often reduced the performance. We conclude that we can use two to eight cursors to improve the performance of pointing depending on the target density. Since the frequency of cursor blocking depends only on the number of cursors and the target density, this guideline is also expected to apply to screens larger or smaller than those used in the experiment.

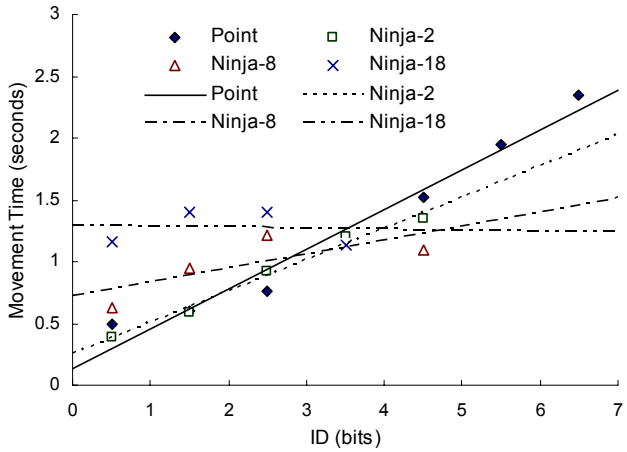

Figure 12: Movement time for each cursor type by *ID*.

Error Rate

Figure 13 shows the error rates for each cursor. The rates averaged over all *N* and *W* values were 1.39% for Point, 2.5% for Ninja-2, 3.19% for Ninja-8, 4.03% for Ninja-18, and 2.78% in total. However, as shown in the figure, the error rate distribution is almost random and the difference is not statistically significant. We require further investigation of error rates to determine the detailed characteristics of the ninja cursor.

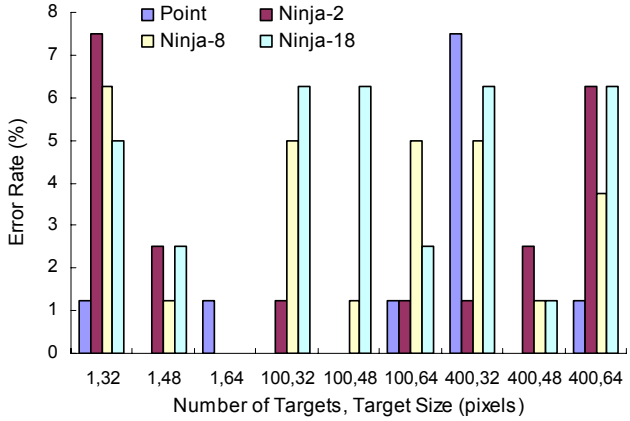

Figure 13: Error rates for each cursor type by *N* and *W*.

Feedback and Observation

All of the participants indicated that they preferred the ninja cursor as long as the cursor on which they focused rarely entered the waiting state; they disliked the cursor becoming stuck repeatedly in the waiting queue. It is interesting that the participants often used the second- or third-nearest cursor to avoid passing through a cluster of distracting targets. They preferred to move the cursor, bypassing distracting targets, to prevent the cursor from becoming stuck in the waiting state. It is also interesting that the participants sometimes used the cursor that they had used in the previous trial instead of the nearest one.

DISCUSSION

Realistic Target Distribution

In the experiment, we examined the ninja cursor with pseudorandomly distributed targets to determine the statistically average performance. However, most WIMP interfaces locate clickable targets in a regular pattern. The regularity might have a negative effect on the performance of the ninja cursor; multiple cursors likely fall into the waiting state simultaneously when both cursors and targets are distributed in a regular pattern (Fig. 14). However, this problem should be alleviated once the first transitions between the normal and waiting states occur and the regularity of the cursor distribution is slightly reduced (Fig. 15). A small reduction in regularity is enough to avoid target ambiguity, at least for equally spaced, discrete targets like desktop icons. This means we can add small amounts of randomness to the initial locations of cursors to avoid an initial ambiguous state. The expected gain in pointing performance would be mostly preserved as long as the amount randomness is small.

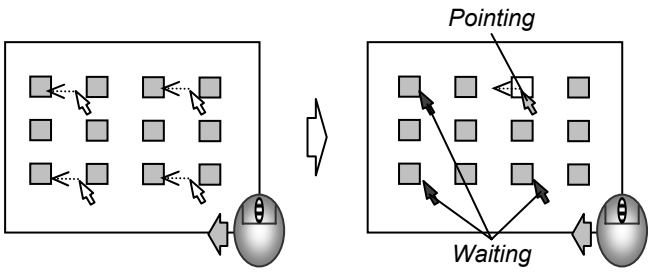

Figure 14: If both cursors and targets are located in a grid pattern, multiple cursors will likely fall into the waiting state simultaneously.

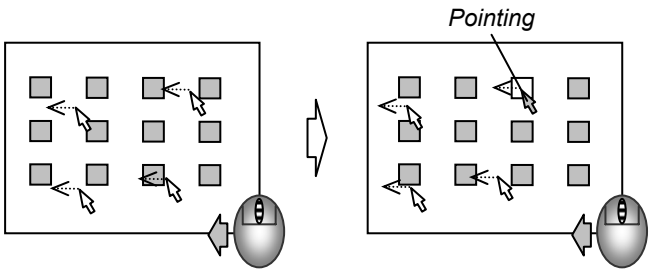

Figure 15: Once the regularity of the cursor distribution is reduced slightly, the problem described in Fig. 14 will be alleviated.

In addition to the regularity of their distribution, targets tend to form semantic clusters as shown in Fig. 16. This could decrease the cursor speed locally around the clusters and reduce the uniformity of cursor arrangement. Further investigations are required to determine how the regular and clustered distributions of targets affect the performance of the ninja cursor.

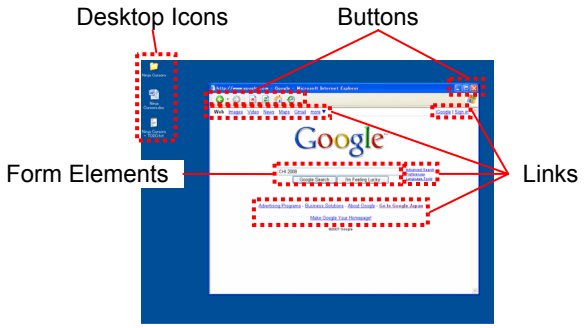

Figure 16: Several semantic clusters of clickable objects, for example, desktop icons, toolbar buttons, and navigation links exist.

Adaptive Cursor Relocation

In the experiment, we located the cursors in a regular pattern because we were interested in the general performance with pseudorandomly distributed targets. In a common WIMP interface, however, the target distribution is more specific as described in the previous subsection. Moreover, the distribution often changes in response to user input. Thus, dynamic adjustment of cursor locations could make it easy to point to the most likely target. For example, when a dialog box pops up, it is likely that the user will click on one of the buttons in the box. Thus, the system could provide an extra cursor inside the dialog box to help users point to a button with minimal effort (Fig. 17). Unlike cursor-jumping techniques such as object pointing [11], targets outside the dialog box are still easy to select with the remaining cursors.

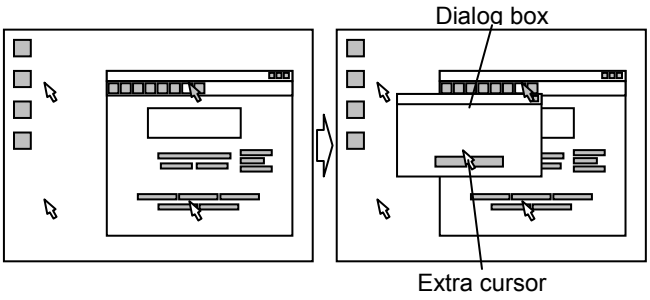

Figure 17: A dialog box invocation presents new clickable targets such as dialog buttons. An extra cursor can help users point to them.

Selecting Multiple Objects

As with most previous techniques that modified the cursor behavior to reduce the ID, the ninja cursor is less applicable to high-density targets such as characters in a text editor and pixels in a paint tool. To work well, the ninja cursor requires a certain void space around each target in which nothing is clickable.

Specifying a region to select multiple targets is another operation in which the ninja cursor has difficulty because void space surrounding the targets must be clickable to use

standard region selection methods, such as rectangular rubber band or lasso tools. However, region selection is one of the most essential features for object manipulation systems. Thus, we present two methods for region selection with the ninja cursor.

The first method allows using the single point cursor to specify a region. Pressing a shortcut key temporarily reduces the number of cursors to 1. Once the system enters single-cursor mode, the user can specify a region using a common rectangular selection tool. The appearance of the ninja cursor is identical to the traditional point cursor while it is in the single-cursor mode. Pressing the key again restores the multiple cursors. One of the drawbacks of this method is that it requires explicitly pressing a key to change modes. However, we can apply this pointing method for very dense targets.

Figure 18 describes the second method, which allows the use of a lasso tool without changing modes. Once the user presses the mouse button, each cursor in the void space starts drawing a lasso while the active cursor starts dragging the pointed target if it is draggable. Note that two or more cursors may simultaneously draw a lasso. This introduces the question of which is the intended lasso and whether the user is attempting to draw a lasso or drag an object. To reduce these ambiguities, we introduce the following three criteria:

- Any lasso stroke must be closed.
- No lasso stroke ever intersects with targets.
- Any lasso must contain at least one target.

The first criterion distinguishes a lasso gesture from a drag gesture. The other two omit less likely lassoes from the screen. If multiple lassoes still meet these criteria when a lasso gesture is completed, then multiple regions are presented. The user can resolve this final ambiguity simply by starting interaction with one of them, such as clicking with the right mouse button to open a pop-up menu.

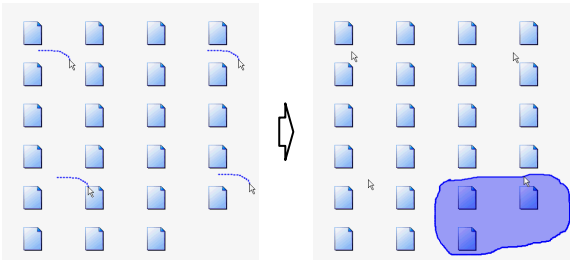

Figure 18: All the cursors start drawing a lasso (left). Unlikely lassoes are canceled dynamically and the user obtains a single lasso (right).

Integration with Other Techniques

The concept of the ninja cursor, increasing the number of cursors, is orthogonal to previous cursor-enhancing techniques, and various combinations are possible. For example, the bubble cursor [9] might be improved by increasing the number of cursors (Fig. 19). Using multiple bubble cursors, a user could point to any target with the nearest cursor just like the ninja cursor. A user could point to a small target with the expanded virtual width just like the bubble cursor. Another possible application is semantic pointing [6]. The performance of pointing to distant targets might be improved by adjusting the C-D gain for each cursor independently. One possible future direction of this research would be to examine the properties of these integrations of the ninja cursor and other cursor-enhancing techniques.

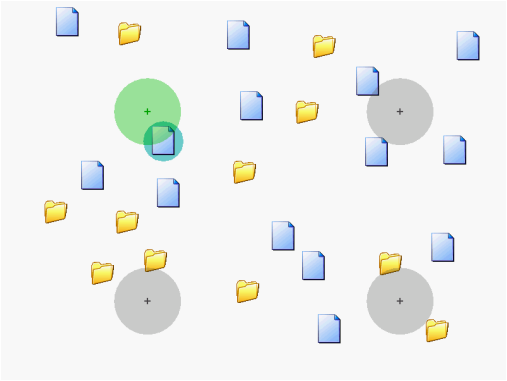

Figure 19: Each bubble cursor moves synchronously following the physical mouse movement. Users can point to a target with one of several bubble cursors.

Limitations

As with other techniques that change the mapping of physical mouse movement to cursor motion, the ninja cursor can be used only with indirect input devices such as mice, touch pads, and trackballs. It does not work well with direct input devices such as pen tablets and touch panels. This is one of the known limitations of the ninja cursor.

The performance of the ninja cursor becomes worse with large numbers of cursors and targets because the cursors frequently enter the waiting state. This problem is expected to be reduced by using a priority queue as the waiting queue, and by assigning appropriate priorities to each cursor. In the current implementation, each cursor in the waiting queue is handled equally without considering the movement direction. However, the cursors moving toward the center of targets would be more likely than those just intersecting the edge of targets (Fig. 20). The total waiting time is expected to be reduced by giving a higher priority to the former.

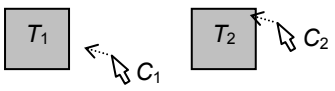

Figure 20: C_1 moves towards the center of T_1 while C_2 is about to intersect the edge of T_2.

If the user fails to click on the intended target, a very distant and incorrect target may be selected erroneously by a distant cursor. This will confuse users because this type of problem cannot occur in traditional single-cursor systems. To help users understand what happens in the case of failed target acquisition, we recommend providing visual feedback, such as animations and afterglow effects [4], to indicate which object is selected.

CONCLUSION

We presented the design, implementation, and performance evaluation of our ninja cursor technique. With multiple, synchronously moving cursors, the ninja cursor technique was designed to reduce the expected ID, especially in a large screen such as the dual displays used in this study. A simple waiting queue algorithm was used to prevent multiple cursors from pointing to different targets simultaneously. Our experimental results showed that both the number of cursors and the target density significantly affect pointing performance. The ninja cursor generally outperformed the traditional point cursor. However, it resulted in low performance when both the number of cursors and the target density were relatively high. We also discussed several issues to be addressed before putting our technique into practice, including the support of realistic target distributions and region selection methods. We hope that our study will encourage the investigation of usability issues related to the number of cursors, a subject that has received little attention to date.

ACKNOWLEDGMENTS

We thank Patrick Baudisch for helpful suggestions. We also thank all who participated in our experiment.

REFERENCES

1. Accot, J. and Zhai, S. Beyond Fitts' Law: Models for Trajectory-Based HCI Tasks. In *Proceedings of CHI'97*, 1997, pp.295–302.
2. Asano, T., Sharlin, E., Kitamura, Y., Takashima, K, and Kishino, F. Predictive Interaction Using the Delphian Desktop. In *Proceedings of UIST'05*, 2005, pp.133–141.
3. Baudisch, P., Cutrell, E., Robbins, D., Czerwinski, M., Tandler, P., Bederson, B., and Zierlinger, A. Drag-and-Pop and Drag-and-Pick: Techniques for Accessing Remote Screen Content on Touch- and Pen-Operated Systems. In *Proceedings of INTERACT'03*, 2003, pp.57-64.
4. Baudisch, P., Tan, D., Collomb, M., Robbins, D., Hinckley, K., Agrawala, M., Zhao, S., and Ramos, G. Phosphor: Explaining Transitions in the User Interface Using Afterglow Effects. In *Proceedings of UIST'06*, 2006, pp.169–178.
5. Benko, H. and Feiner, S. Pointer Warping in Heterogeneous Multi-Monitor Environments. In *Proceedings of Graphics Interface 2007*, 2007, pp.111–117.
6. Blanch, R., Guiard, Y., and Beaudouin-Lafon, M. Semantic Pointing: Improving Target Acquisition with Control-Display Ratio Adaptation. In *Proceedings of CHI'04*, 2004, pp.519–525.
7. Fitts, P.M. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, Volume 47, 1954, pp.381–391.
8. Greenberg, S., Roseman, M., Webster, D., and Bohnet, R. Issues and Experiences Designing and Implementing Two Group Drawing Tools. In *Proceedings of Hawaii International Conference on System Sciences*, 4, 1992, pp.138–150.
9. Grossman, T. and Balakrishnan, R. The Bubble Cursor: Enhancing Target Acquisition by Dynamic Resizing of the Cursor's Activation Area. In *Proceedings of CHI'05*, 2005, pp.281–290.
10. Guiard, Y., Bourgeois, F., Mottet, D., and Beaudouin-Lafon, M. Beyond the 10-bit Barrier: Fitts' Law in Multiscale Electronic Worlds. In *Proceedings of IHM-HCI 2001*, 2001, pp.573–587.
11. Guiard, Y., Blanch, R., and Beaudouin-Lafon, M. Object Pointing: A Complement to Bitmap Pointing in GUIs. In *Proceedings of Graphics Interface 2004*, 2004, pp.9–16.
12. Hinckley, K., Czerwinski, M., and Sinclair, M. Interaction and Modeling Techniques for Desktop Two-Handed Input. In *Proceedings of UIST'98*, 1998, pp.49–58.
13. Kabbash, P. and Buxton, W. The "Prince" Technique: Fitts' Law and Selection Using Area Cursors. In *Proceedings of CHI'95*, 1995, pp.273–279.
14. Karlson, A.K. and Bederson, B.B. ThumbSpace: Generalized One Handed Input for Touchscreen-Based Mobile Devices. In *Proceedings of INTERACT 2007*, 2007, pp.324–338.
15. Lank, E., Cheng, Y.N., and Ruiz, J. Endpoint Prediction Using Motion Kinematics. In *Proceedings of CHI'07*, 2007, pp.637–646.
16. MacKenzie, S. and Buxton, W. Extending Fitts' Law to Two-Dimensional Tasks. In *Proceedings of CHI'92*, 1992, pp.219–226.
17. McGuffin, M. and Balakrishnan, R. Acquisition of Expanding Targets. In *Proceedings of CHI'02*, 2002, pp.57–64.
18. Ramos, G., Cockburn, A., Balakrishnan, R., and Beaudouin-Lafon, M. Pointing Lenses: Facilitating Stylus Input through Visual- and Motor-Space Magnification. In *Proceedings of CHI'07,* 2007, pp.757–766.
19. Worden, A., Walker, N., Bharat, K., and Hudson, S. Making Computers Easier for Older Adults to Use: Area Cursors and Sticky Icons. In *Proceedings of CHI'97*, 1997, pp.266–271.