Switchback Cursor: mouse cursor operation for overlapped windowing

Shota Yamanaka[1] and Homei Miyashita[1,2]

[1] Meiji University, [2] JST CREST

stymnk@meiji.ac.jp, homei@homei.com

Abstract. When we perform a task that involves opening a number of windows, we cannot access the objects behind them. Thus, we are forced to switch the foreground window frequently or to move it temporarily. In this paper, we propose a Switchback Cursor technique where the cursor can move underneath windows when the user presses both the left and right mouse buttons. We also discuss some of the advantages of our method and effective situations that may be suited to the Switchback Cursor.
Keywords: Cursor, Graphical user interfaces (GUIs), Mouse, Pointer, WIMP.

1 Introduction

Numerous window manipulations are performed when we work on a PC running multiple windows, such as moving and resizing. Besides, we need to click background windows frequently to switch to foreground windows. These actions are performed simultaneously in our main task, such as application operation or file organizing, and we are sometimes required to perform many complicated window manipulations.

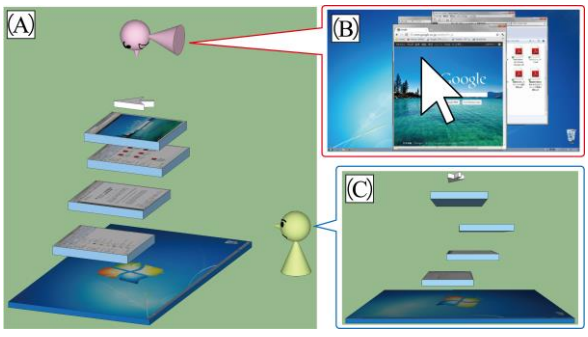

Fig. 1. (A): Observing overlapping windows from different perspectives. (B): The normal view of windows and the cursor on a PC. (C): The windows are piled up on the desktop and the cursor is always on the top.

These operations are required because there is a difference between the dimensions of the windows and the movement of the mouse cursor. In Fig. 1, (A) shows a 3D image of some overlapping windows, where (B) represents the picture that we normally view on the PC. The windows are distributed in 3D but the cursor can only move in 2D. Thus, the background windows can only come to the foreground if we click on them or use a keyboard command to control them.

In our novel approach, called Switchback Cursor, the mouse cursor can move underneath windows by hitting them (it does not move freely in 3D). When a user holds both the left and the right mouse buttons, the cursor moves from the edge of a window to the background, so that it can control the objects there. Fig. 2 shows how the cursor moves underneath windows using our proposed technique. A movie showing the behavior of the Switchback Cursor can be viewed at [1].

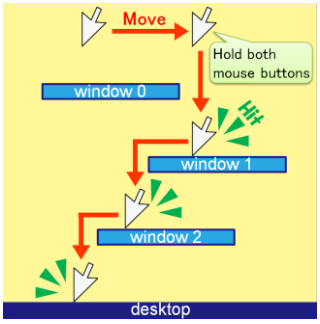

Fig. 2. The cursor moves progressively deeper after hitting successive windows.

2 Related Work

Many techniques have been developed to address the problems of overlapping windows. Free-space Transparency (FST) [2] allows the free space in a window to become transparent so users can see through the objects behind the foreground window. This also allows us to perform basic interactions such as drag-and-drop and clicking icons using the white region in a window, which is similar to our proposed system. In the stack leafing technique [3], a mouse button is pressed and the foreground windows on the same layer switch to another layer by dragging. In QuickSpace [4], a user moves a window and it pushes *friend-registered* windows so they are not overlapped. Beaudouin-Lafon proposed a technique [5] that allows some window manipulations such as tabbed windows and peeled-back windows. The tabbed windows technique has now been implemented in Google Chrome and this method is used widely to solve the overlapped windows problem. Metisse [6] is a windows management system that allows users to rotate windows in 2D and 3D, so the windows can be allocated without overlapping to use the desktop space efficiently. WindowScape [7] addresses rearranging windows problem by photograph

metaphor. Screenshots are stocked each time windows are miniaturized, and when the user selects one of the shots, WindowScape automatically allocates the windows state.

Some systems have been developed that allow the cursor to adopt irregular behaviors and their goal is mainly to make the mouse operation more efficient. Bubble Cursor [8] is a round-shaped cursor that extends the pointing range from one dot to a large circle, which reduces the movement distance. Dynaspot [9] enhances the size of the cursor area, depending on the cursor movement speed. When the cursor moves rapidly, the selection area becomes larger to increase the accuracy. However, *area cursor* methods such as Bubble Cursor and Dynaspot make it difficult to point when the targets lay side-by-side, but like our Switchback Cursor, these two techniques also allow movement into inaccessible areas using traditional techniques and clip manipulation. Delphian Desktop [10] moves the cursor to the target object immediately, while Drag-and-Pop [11] is a technique that allows the target object to travel to the cursor. These methods reduce the mouse manipulation time in a direct manner by allowing the targets or the cursor to warp. Ninja Cursors [12] uses multiple pointers and the user operates them all, i.e., the user only moves the cursor nearest the target object. The double mouse system [13] operates two cursors using two mice with one in each hand, which has the same effect as Ninja Cursors in reducing the manipulation time by operating a convenient cursor. Semantic Pointing [14] allows the cursor to speed up when it is further from a target by estimating the object that the user wants to select. MAGIC [15] makes the cursor jump to near the gaze-point using an eye tracker. This technique exploits the tendency to look at a target first before moving the cursor there. Fold-and-drop [16] allows items to be drag-and-dropped into the back window by turning over the windows like papers. This limits the manipulations using drag-and-drop but the user can search folders in the windows and run an icon on the desktop with the Switchback Cursor.

3 The Problem

When we perform our main task on a PC, it is often necessary to operate a number of windows, which are associated with the main task. However, a problem arises from the difference between the dimensionality of the windows and that of the cursor. Windows are set in depth layers from the front to back where the layers are structured in 3D. However, the cursor can only move in 2D. Thus, we cannot control objects hidden by windows and we have to click on a background window to perform our intended manipulation. This is a typical problem of the overlapping window system. A tiling window system does not have this problem, but it has bad visibility due to the small size of the windows so it is used less widely than the overlapping system.

There is also the problem with window focus, which refers to a window that receives inputs from the keyboard and mouse. A focused state is referred to as “the active window.” In Windows7, a clicked window is focused (focus follows click; FFC). However, the active window comes forward and the foreground window becomes hidden. This does not cause a problem when we begin another task but if we only want to access a background object briefly it is necessary to switch the

foreground window a number of times. Another approach to window focusing depends on the position of the mouse cursor (focus follows mouse; FFM). However, the active window can be changed accidentally if the mouse is moved carelessly, so this approach has not been adopted by the Windows OS or the Mac OS. Thus, the general window viewing method and mouse cursor manipulation cause problems. This is a major problem because unnecessary window operations are required when controlling background objects using the overlapping window approach. Therefore, we can make mouse operation more comfortable by solving these two problems, i.e., the cursor cannot reach objects behind windows so we are forced to switch the foreground window many times in an FFC environment.

4 Switchback Cursor

We developed our system for Windows7. The system obtains windows information such as the handler, position, size, window style, and z-order and monitors these parameters. When a user presses both mouse buttons, the cursor moves to the same layer as the window beneath it. For example, Fig. 2 shows that a user moves the cursor from its initial position to a position above *window 1* and presses both buttons, so the cursor moves to the layer of *window 1*. If the user keeps pressing both buttons and moving the cursor to the left, it moves underneath *window 0*. In the same way, the cursor can move to progressively deeper layers by hitting successive windows.

When a user presses both buttons, the windows in front of the cursor are set as the *topmost window*, which become semi-transparent so the objects can be seen underneath them, and these windows are set *through mouse actions*. Subsequently, mouse actions such as click and drag are not received by the windows in front of the cursor and the actions are received by the window beneath the cursor. If the window receives a mouse signal, the order of the windows in front of the cursor is not changed because the front windows are set as the *topmost windows*. A transparency setting of 0% (where the windows are completely transparent) is discouraged because the user cannot perceive the position and shape of the windows, but this parameter can be changed by the user.

When both buttons are not pressed, the cursor aims to move to the front side but it hits the window above itself and remains in the layer behind the window. For example, in Fig. 2, a user releases the mouse button(s) when the cursor is underneath *window 0*, it remains behind *window 0*. If the cursor is returned from *window 0* without pressing at least one button, it moves to the foreground and dissolves the topmost setting of *window 0*.

When a user presses both buttons, the cursor size becomes 15% smaller as it moves to a window. When the cursor is around a window while pressing both buttons, the angle of the cursor changes so the user can see the direction where it is going as it moves beneath a window. When the cursor moves to a rear layer, a metallic sound rings to indicate that it has hit a window.

When the maximum-sized windows are open, they become slightly smaller (40 pixels less from each edge) while pressing both buttons so the cursor can move underneath them, as shown in Fig. 3. Further, when a window hides one or more windows, the background windows slide slightly to produce a small gap (30 pixels), as shown in Fig. 4.

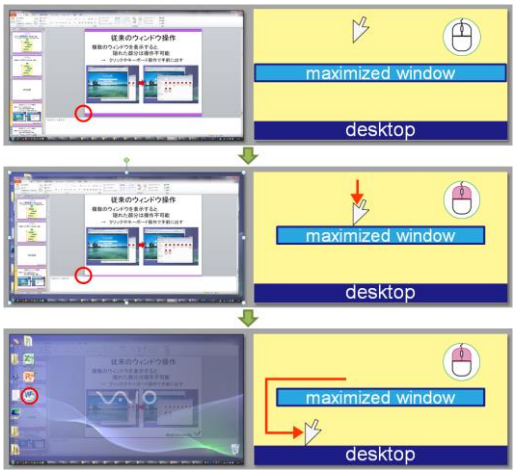

Fig. 3. A maximized window(s) becomes slightly smaller when both mouse buttons are pressed. The red circles indicate cursor positions. The left screenshot shows a maximized window while the images on the right show the left images from a location rotated to 90°, as shown in Fig. 1 (C). The cursor can reach the desktop icon while both buttons are pressed.

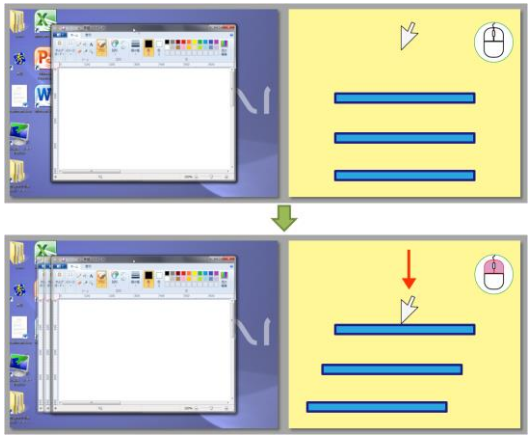

Fig. 4. The windows hidden by a front window (same size and same position, or completely covered) move to produce a gap when both buttons are pressed so that the cursor moves to their layers. The cursor cannot move under two window layers if not pressing both buttons. When pressing both buttons, the covered windows move so that the cursor can reach their layers.

5 Advantages and Effective Situations

In our proposed approach, the cursor moves from the foreground to the target window so the cursor moves in three dimensions within the display. A number of 3D mouse or 3D desktop systems have been developed in the past but in most cases the cursor moves in the Z direction freely and a special device or GUI is required for 3D input. Unlike these systems, the cursor moves backward in our approach but it hits a window and stays there. No special devices are needed and only a normal mouse with left and right buttons is used.

We do not suggest that users employ our technique alone instead of traditional window switching. If he/she uses another application for a while, it is better to switch the foreground window; if a keyboard shortcut is suitable, it is preferable that the user performs keyboard operations. However, in situations where these manipulations become cumbersome and our technique provides a better alternative, users might prefer to use our technique. Therefore, we developed this system for use in a traditional GUI environment so the user can invoke this technique anytime by pressing both buttons, depending on the situation. Our system has high compatibility with modern overlapping windows settings and it does not block existing manipulations. In the subsections below, we describe some scenarios that may be suited to our technique.

5.1 A Window Hides Other Windows when Switching the Foreground Window

Fig. 5 shows a situation where a person is browsing the web while listening to music. If we want to listen to a different piece of music, as shown in Fig. 5, we normally click on the background folder to switch to the foreground and drag-and-drop a file onto the music player (or double-click it). The browser then becomes hidden partly by the folder so we have to click the browser again to switch to the foreground to return to our browsing task. An alternative is to arrange the windows so they do not overlap one another, but the process is complicated in both cases. With the Switchback Cursor, we simply allow the cursor to reach the background of the browser and drag-and-drop the file so we do not need to click on the folder or the browser multiple times to switch to the foreground window.

Fig. 5. Web browsing while listening to music.

5.2 Double-clicking an Icon on the Desktop

In Fig. 6, the top screenshot and the illustration show a situation where several windows are open. If we want to double-click on an icon on the desktop to run an application or open a file, we would typically minimize all of the windows using the command Show Desktop shortcut, or move them out of the way and then double-click on the target icon. We would also have to reconstruct the original window layout to return to our former task, which increases the number of operation steps. By contrast, our technique requires no window manipulations because the cursor moves directly to the desktop and double-clicks on the icon, as shown at the bottom of the screenshot and in the illustration in Fig. 6.

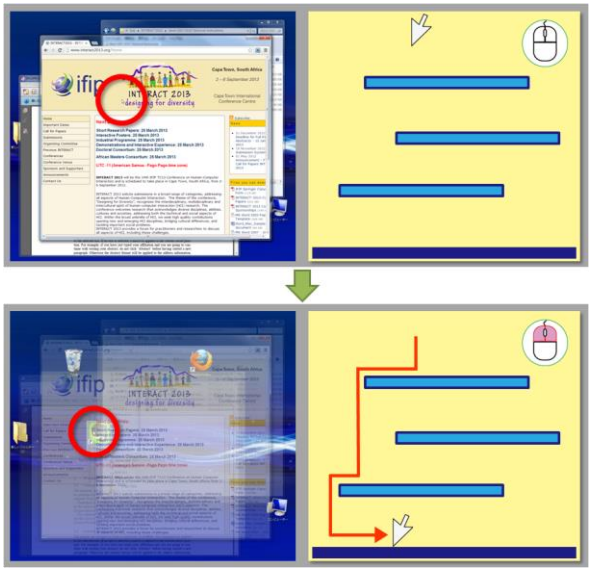

Fig. 6. The cursor moves underneath the windows and reaches the target icon on the desktop. The red circles indicate cursor positions. Some windows are pile up in the left screenshots while the images on the right show the left pictures from a location rotated by 90°. The user simply needs to press both buttons and move the cursor to the desktop before double-clicking the icon, without any window manipulations.

6 Conclusion

In this paper, we described the two problems of overlapping window systems: the difference in dimensions between the cursor and the windows, and the high number of manipulations required to switch foreground windows. To address these problems, we proposed the Switchback Cursor technique, which allows the cursor to move underneath the windows. We discussed the advantages of this methods and situations where

using our method might prove effective. In our future work, we would like to test and verify the effectiveness of our technique by some tasks, such as moving icons within folders that exist in various layers. And we also plan to evaluate the limitation of Switchback Cursor; not only performance time and error rate but also the visibility of layers beneath the overlapped windows, and the window layout that makes the cursor hard to go underneath.

References

1. Switchback Cursor movie: http://www.youtube.com/watch?v=_nLt5PurgPY
2. Ishak, E.W. and Feiner, S.K.: Interacting with Hidden Content Using Content-Aware Free-Space Transparency. In *Proc. of UIST '04*, pp. 189-192 (2004).
3. Faure, G., Chapuis, O. and Roussel, N.: Power tools for copying and moving: useful stuff for your desktop. In *Proc. of CHI '09*, pp. 1675-1678 (2009).
4. Hutchings, D.R. and Stasko, J.: QuickSpace: New operations for the desktop metaphor, In *Proc. of CHI Extended Abstracts '02*, pp. 802-803 (2002).
5. Beaudouin-Lafon, M.: Novel interaction techniques for overlapping windows, In *Proc. of UIST '01*, pp. 153-154 (2001).
6. Chapuis, O. and Roussel, N.: Metisse is not a 3D desktop!, In *Proc. of UIST '05*, pp. 13-22 (2005).
7. Tashman, C. and Edwards, W.K.: WindowScape: Lessons learned from a task-centric window manager. *ACM Transactions on Computer-Human Interaction*, Vol. 19, No. 1, Article 8 (2012).
8. Grossman, T. and Balakrishnan, R.: The Bubble Cursor: Enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proc. of CHI '05*, pp. 281-290 (2005).
9. Chapuis, O, Labrune, J.-B. and Pietriga, E.: DynaSpot: Speed-Dependent Area Cursor. In *Proc. of CHI '09*, pp. 1391-1400 (2009).
10. Asano, T., Sharlin, E., Kitamura, Y., Takashima, K. and Kishino, F.: Predictive interaction using the Delphian Desktop. In *Proc. of UIST '05*, pp. 133-141 (2005).
11. 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 *Proc. of Interact '03*, pp. 57-64 (2003).
12. Kobayashi, M. and Igarashi, T.: Ninja Cursors: Using Multiple Cursors to Assist Target Acquisition on Large Screens. In *Proc. of CHI '08*, pp. 949-958 (2008).
13. Nakamura, S. and Tsukamoto.: M. Design and Implementation of the Double Mouse System for a Window Environment. In *Proc. of PACRIM '01*, pp. 204-207 (2001).
14. Blanch, R., Guiard, Y. and Beaudouin-Lafon, M.: Semantic pointing: improving target acquisition with control-display ratio adaptation, In *Proc. of CHI '04*, pp.519-526 (2004).
15. Zhai, S., Morimoto, C. and Ihde, S.: Manual And Gaze Input Cascaded (MAGIC) Pointing, In *Proc. of CHI '99*, pp. 246-253 (1999).
16. Dragicevic, P.: Combining Crossing-Based and Paper-Based Interaction Paradigms for Dragging and Dropping Between Overlapping Windows. In *Proc. of UIST '04*, pp. 193-196 (2004).