Novel Interaction Techniques for Overlapping Windows

Michel Beaudouin-Lafon
Laboratoire de Recherche en Informatique
Bâtiment 490 - Université Paris-Sud
91405 Orsay - France
mbl@lri.fr - http://www-ihm.lri.fr

ABSTRACT
This note presents several techniques to improve window management with overlapping windows: tabbed windows, turning and peeling back windows, and snapping and zipping windows.

KEYWORDS: window management, interaction technique

INTRODUCTION
The dominant model for window management in desktop interfaces is overlapping windows. As the number of windows on the screen increases, the task of flipping between windows becomes more and more tedious and time-consuming. Previous work, e.g. [2,5], has addressed this issue with tiled windows. This note presents techniques to improve this situation with overlapping windows.

TABBED WINDOWS
Many commercial systems already use dialog boxes with several pages accessible through tabs. We introduced the concept of *tabbed windows* in the CPN2000 application [1]. It raises the tab technique to the level of document windows and gives the user the additional flexibility of moving pages from one window to another by dragging them via their tabs (Fig. 1). A page can be dragged to an existing window, which adds a tab to it, or to the background, which creates a new window with a single page and tab. Dragging the last page out of its window deletes the window.

Our experience with this technique in CPN2000 shows that tabbed windows dramatically reduce the number of windows on the screen while still providing quick access to a large number of pages that would otherwise be separate windows. Users thus control their working sets of pages and windows without sacrificing efficiency.

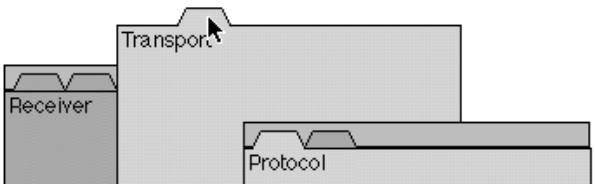

Figure 1: A page in a tabbed window being dragged.

Submitted to UIST 2001 - Do not cite

Figure 2: Tabbed window with a popped-up tab.

A well-known limitation of tabbed dialogs is the number of tabs that can be displayed in a single dialog box, or, in our case, window. Several strategies exist, including arranging the tabs in several rows, adding a horizontal scrollbar or adding a pull-down menu for hidden tabs. Unfortunately each technique adds a significant overhead to the selection of a tab. Our approach lays out all the tabs in a single row, with the currently-active tab on top. Other tabs may overlap, but pop-up when the mouse passes over them, revealing their names (Fig. 2). The salient tab can then be clicked or dragged as in the normal, non-overlapping, case.

In terms of interaction, this technique is more efficient than others, even though it requires the user to mouse over the tabs in order to find the target page. *Leafing* facilitates this look-up phase: in addition to popping up the tabs while mousing over them, the corresponding pages are also displayed on top. This is enabled after a time-out that starts when the mouse first appears over a tab in the window. While leafing, the pages being displayed are dimmed as feedback of the mode to the user. If the user leaves the tabs region without clicking or dragging a tab, the display reverts to its previous state, with the original page on top.

Our experience shows that the combination of spatial memory to locate the tabs, leafing and other cues, such as the colors of the tabs, make it easy to find the target page quickly. Because of the tight coupling between action and perception, this technique is very close to the way we leaf and search through a physical Rolodex or file folder, making it simple to learn.

ROTATING AND PEELING BACK WINDOWS
Windows in desktop environments are almost always rectangular, with the sides parallel to the sides of the screen. When two windows of the same size overlap, at most two sides of the rear window can be seen. When many windows of similar sizes overlap, many of them become invisible, making them difficult to access.

By contrast, while physical books and sheets of paper on a desk are also rectangular, they are rarely perfectly aligned. This gives the user clues for locating them and an easy way to access them even when they are stacked.

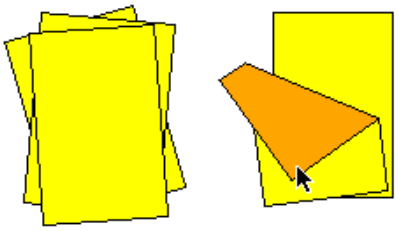

Figure 3: Rotated windows and a peeled-back window.

We have applied this metaphor to windows on a desktop by supporting *rotated windows*: when a window is moved, a simple kinematic model is applied so that it rotates as it moves, like a sheet of paper being moved by a corner. The rotation angle is computed so that when the point *P* of the page under the cursor is moved to *P'*, the center of gravity *G* of the window is moved to *G'* so that *G*, *G'* and *P'* are aligned and the distance between *G'* and *P'* is the same as the distance between *G* and *P* (otherwise the window would be distorted). In addition, the rotation angle is constrained so that the window does not end up overly rotated, e.g., sideways or even upside down.

Given the resolution of current screens, rotating the contents of the window can make it hard to read, especially if it contains small text. We have experimented with two approaches to solve this problem. The first automatically rotates a window back to its upright position when it is selected. This works well when the window does not need to be rotated too much (up to 10º) and if it is rotated around the cursor position (otherwise the user may miss the selection). The other approach does not rotate the contents of the window at all. This works well when the window is rotated by less than 5º.

We have also extended the metaphor of sheets of papers on a desk by experimenting with *peeling back windows*. Clicking on a corner of a window and dragging towards the inside of a window peels it back, revealing the window underneath it. The window springs back to its original position when the mouse button is released, with an animation that lasts approximately one second. This is enough to move the mouse pointer to a window that was hidden and select or move it. Typically, a traditional window manager would require moving windows around to get to the hidden one, destroying any layout that the user may have created.

Peeling back a window involves partitioning its rectangle into two polygons: the polygon being peeled back and the rest of the window. First we compute the perpendicular bisector *L* of segment *PP'* where *P* is the point where the mouse was clicked to start the interaction and *P'* the current position of the mouse. We then split the rectangle into two polygons using this line. The polygon being peeled up is the reflection of the polygon that contains *P* about line *L*. The other polygon is the rest of the window. This technique makes it possible to turn the window completely over: when line *L* does not intersect the rectangle, the whole window is peeled back.

SNAPPING AND ZIPPING WINDOWS

Users often work with windows that are strongly related, such as an outline and a page layout of the same document, or a document window and some tool palettes. Some applications support tiling, placing several panes together inside a window: Microsoft Word can split a window to display two views of a document, Netscape Mail can split a window into a list of folders, a list of messages and the current message. However the user has little flexibility in organizing these panes.

Snapping windows allows the user to assemble several windows into a single entity. When a window is moved so that one of its sides is close to the side of another window, the two windows are snapped together and stay snapped unless the user moves the window away before releasing the mouse button. The interaction is similar to snap dragging [3] or magnetic guidelines [1]. Note that a window can be snapped inside, overlapping the other window.

A snapped window is slaved to its master window: moving the master window moves its slave, while moving the slave unsnaps it from the master window. This works well when the snapped window is small compared to the main one, e.g. when it is a tool palette. When windows are of similar sizes along the side being snapped, snapping becomes *zipping*: the two windows are given the same size along their common side, which becomes a divider line that can be moved. The windows are then moved and resized together, and they are unzipped by double-clicking the divider. Finally, snapped windows can be collapsed and reopened by clicking their tabs. MacOS 8.x/9.x has similar pop-up windows except that they can only be at the bottom of the screen.

CONCLUSION

We have presented several techniques that improve window management by extending the metaphor of overlapping windows. All these techniques have been implemented and preliminary results are encouraging. Our future work will create a full window manager based on these techniques and we plan to conduct more formal usability studies.

1. Beaudouin-Lafon, M. & Lassen, H.M., The architecture and implementation of CPN2000, a post-WIMP graphical application. in *UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters* 2(2):181-190, 2000.
2. Bell, B. & Feiner, S., Dynamic space management for user interfaces. in *UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters* 2(2):239-248, 2000.
3. Bier, E. & Stone, M. Snap-dragging. In *Proc. ACM SIGGRAPH*, 20(4):233-240, ACM Press, 1986.
4. Bly, S. & Rosenberg, J., A comparison of tiled and overlapping windows. in *CHI '86, ACM Conference on Human Factors in Computing Systems*, p.101-106, 1986.
5. Kandgan, E. & Shneiderman, B., Elastic windows: evaluation of multi-window operations. in *CHI '97, ACM Conference on Human Factors in Computing Systems*, p.250-257, 1997.