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. The concept of *tabbed windows*, introduced in CPN2000 [1], raises the use of tabs to the level of document windows. In addition, users can move 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 a window deletes the window. Our experience with this technique in CPN2000 shows that tabbed windows dramatically reduce the number of windows on the screen and provide quick access to a large number of pages that would otherwise be separate windows. Adobe products use a similar technique but limited to tool palettes.

A limitation of tabbed dialogs is the number of tabs that can be displayed without overlapping. Existing solutions include arranging the tabs in several rows, adding a horizontal scrollbar or adding a pull-down menu for hidden tabs. This adds significant overhead to the selection of a

Figure 1: A page in a tabbed window being dragged

Figure 2: Tabbed window with a popped-up tab

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 popped-up tab can then be clicked or dragged as in the normal, non-overlapping, case.

This technique 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 displayed on top. Leafing is activated 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 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 leafing, the user's spatial memory, and cues such as tab color, make it easy to find the target page quickly. Because of the tight coupling between action and perception, this technique is similar to browsing a physical Rolodex or file folder.

ROTATING AND PEELING BACK WINDOWS
Windows are often stacked on top of each other to save space on the desktop. Accessing stacked windows is cumbersome because windows are rectangular, with the sides parallel to the sides of the screen. When many windows of similar sizes are stacked, most of them become invisible. By contrast, although physical books and sheets of paper on a desk are 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.

Mander et al. have applied the stacking metaphor to icons [6]. We have applied it to windows on a desktop by supporting *rotated windows* (Fig. 3): when a window is dragged, it rotates as it moves, like a sheet of paper being pulled by a corner. The motion is computed as follows (Fig. 4): when the cursor moves from P to P', the center of gravity G moves to G' such that G, G' and P' are aligned and the distance $d(P',G') = d(P,G)$. In addition, the rotation of the window is constrained so that it does not end up overly rotated, e.g., sideways or upside down.

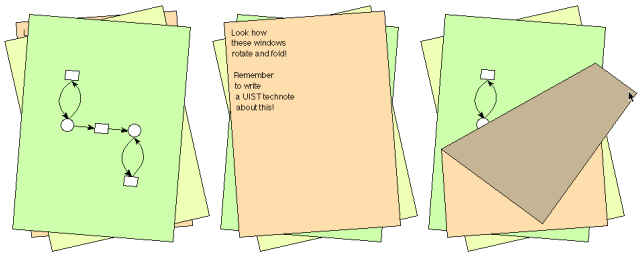

Figure 3: Rotated windows and a peeled-back window

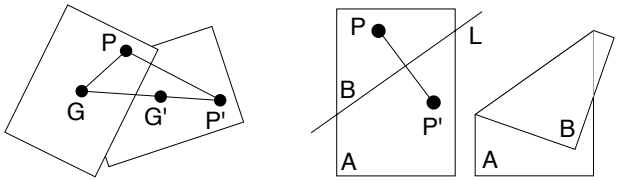

Figure 4: Computing rotation (left) and peeling (right)

Rotating the contents of the window can make it difficult to read, especially if it contains small text. When the window is rotated by less than 10º, our solution is to automatically rotate the window back to its upright position as it is selected. Rotation should occur around the cursor position so the user does not select the wrong object. When the window is rotated by less than 5º, another solution is to leave the contents of the window upright.

We have also extended the metaphor of sheets of paper 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 (Fig. 3). The window springs back to its original position when the mouse button is released, with a one-second animation. 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 several 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 (Fig. 4): the polygon being peeled back (B) and the rest of the window (A). 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 back is the reflection of polygon *B* about line *L*. Polygon *A* 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, polygon *A* is empty and 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 splitting windows into several panes, e.g. two views of a document in Microsoft Word or the folders, messages and current message in Netscape Mail. But users have little flexibility in organizing these panes.

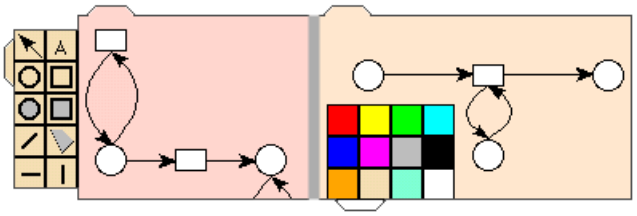

Figure 5: Two palettes snapped onto two zipped windows

Snapping windows allows users 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. The interaction is similar to snap dragging [3] or magnetic guidelines [1]. Fig. 5 shows that windows can be snapped outside (tool palette, left) or inside (color swatches palette, right).

A snapped window is slaved to its master window: moving the master moves its slave, but moving the slave unsnaps it from the master window. The slave can be collapsed and reopened by clicking its tab. MacOS 8.x/9.x has similar pop-up windows except that they can only be at the bottom of the screen. Snapping works well when the snapped window is small compared with the main one, e.g., 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 (Fig. 5). The windows are then moved and resized together, and they are unzipped by double-clicking the divider.

CONCLUSION

We have presented several techniques that improve window management by extending the metaphor of overlapping windows. All these techniques have been implemented. Future work includes creating a full window manager based on these techniques and conducting formal usability studies.

REFERENCES

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. Kandogan, 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.
6. Mander, R., Salomon, G. Y Wong, Y.Y., A 'Pile' Metaphor for Supporting Casual Organization of Information. in *CHI'92, ACM Conference on Human Factors in Computing Systems*, p.627-634, 1992.