Combining Crossing-Based and Paper-Based Interaction Paradigms for Dragging and Dropping Between Overlapping Windows

Pierre Dragicevic
LIIHS-IRIT
Université Paul Sabatier
F-31062 Toulouse Cedex
France
dragice@irit.fr

ABSTRACT
Despite novel interaction techniques proposed for virtual desktops, common yet challenging tasks remain to be investigated. Dragging and dropping between overlapping windows is one of them. The fold-and-drop technique presented here offers a natural and efficient way of performing those tasks. We show how this technique successfully builds upon several interaction paradigms previously described, while shedding new light on them.

Categories and Subject Descriptors: H.5.2 [**User Interfaces**]: Interaction Styles, Windowing Systems; I.3.6 [**Methodologies and Techniques**]: Interaction Techniques.

Additional Keywords: Drag-and-drop, crossing-based interfaces, gestural interaction, paper-based metaphors.

INTRODUCTION

New interaction techniques are regularly being suggested for improving virtual desktops and making common operations such as drag-and-drop and window manipulation easier [2, 8, 10]. Still, not all usability issues have been addressed. Among the most significant is the task of dragging and dropping between overlapping windows.

Having to drag and drop an object towards a partially or totally hidden window is a recurring problem while using virtual desktops. One common example is moving – or copying – a file from a working directory to another one, when each directory is displayed in a separate window. Several strategies may be employed by the user, including:

• *Dropping the object on the visible part of the destination window*. This supposes that 1) the destination window shows enough clues to be properly identified by the user, 2) the actual target – a given object inside the window or the window itself – is not totally hidden.

• *Rearranging the windows so that both are visible* before performing the drag-and-drop. This requires the user to find the target window, move it to the front, and then move/resize one or both windows.

• *Using cut-and-paste* instead of drag-and-drop. This requires cutting, finding the target window, moving it to front, then pasting.

Although the last two operations require much more steps than a single drag-and-drop gesture, there is often no better alternative especially when the target window is totally obscured. Other strategies, such as using Window's *Alt-Tab* during drags, are rather intricate. Fortunately, there seems to be a trend towards making window navigation during drags easier: Windows XP's task bar can be crossed to bring a window to the front whereas a keyboard shortcut called *exposé* on Mac OS 10.3 allows temporarily tiling all windows and selecting one of them [10]. However none of these techniques is wholly satisfactory because they require switching back and forth between two different representations of the same window set. The biggest issue is that compact window visualizations do not always show sufficient information for the target window to be recognizable while they are too small to contain numerous drop targets.

This paper describes a new interaction technique for addressing this problem. The technique, called "fold-and-drop", uses a natural metaphor that makes it possible to seamlessly drag and drop objects from a window to any window underneath.

THE FOLD-AND-DROP TECHNIQUE

Using the fold-and-drop technique consists in "leafing through" windows while holding the dragged object, until the target window is found. More precisely, the object is dragged and dropped in the usual way, except that "folding interactions" can be performed with windows as long as the mouse button remains pressed. There are several types of folding interactions:

Leaving Windows. Each time the mouse leaves a window, a small fold named *transient fold* appears at the exit area during a brief period of time, then springs back. This animation is illustrated in the Figure 1.

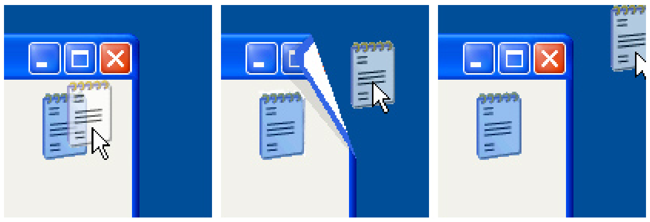
Figure 1: A small fold briefly appears as the mouse leaves the window.

Confirming and Pushing Folds. During the short time a transient fold remains visible it can be crossed back with the mouse to be *confirmed*, in which case it will remain folded. Conversely, no confirmation occurs if the mouse continues in the same direction or if it returns back after the transient fold has disappeared.

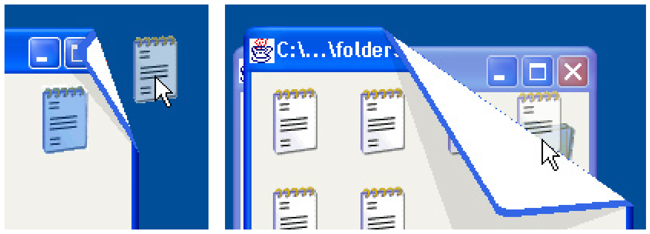
Figure 2: Pushing a fold.

Once confirmed, a fold can be pushed by the mouse in order to be enlarged and reveal windows behind. This can immediately follow the confirmation gesture, as shown on Figure 2. The way the fold is pushed also has an effect on the orientation it takes.

Discarding Windows. When a fold keeps being pushed so that only a small part of the pushed window content remains visible, the window fades away and eventually disappears. This can also be done with a single long gesture (see Figure 3).

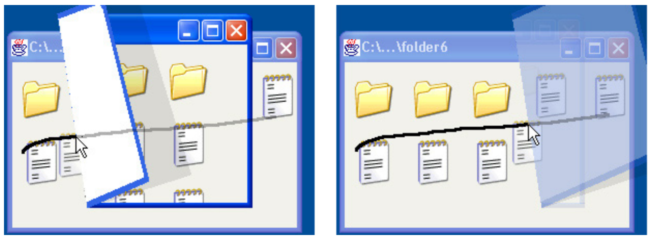
Figure 3: Discarding a window with a single gesture.

Unfolding. Moving around the fold and pushing it *from the inside to the outside* will unfold it entirely. This interaction is illustrated in the Figure 4.

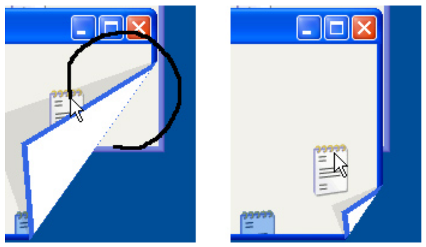
Figure 4: Going around a fold and pushing it from inside cancels it.

Manipulating Multiple Folds. Several folds can coexist on separate windows and be manipulated successively. Multiple folds can also be manipulated at the same time, as illustrated on the right side of Figure 5: folding a window will fold all the windows over it, and unfolding a window will unfold all windows underneath.

Back to the initial state. After the mouse button has been released or the drag-and-drop has been cancelled (e.g. by a right-click), all windows spring back to their normal state. In case a command such as a file copy has been issued, a short pause is made beforehand to make its results apparent.

A typical scenario of use for the fold-and-drop technique could be the following: an icon is dragged outside a window, which is folded enough so that windows below can be identified. If the target window is not among them, another window is folded, and so on. Backtracking is possible by unfolding previously folded windows.

Such manipulations can follow one another at a high pace, especially if the user has some idea on the location of the target window. For example, if the user knows it is situated two windows below, he/she can move the mouse two times back and forth without having to uncover the window in-between. Multiple lined-up windows can be leafed through in this way (Figure 5 on the next page).

DISCUSSION

The fold-and-drop technique relates to several interaction paradigms previously described in the literature, namely crossing-based tasks, gestural interaction and paper-based metaphors. In this section, we briefly recall those concepts, motivate their use and discuss each of them in the light of the fold-and-drop technique.

Crossing-based Tasks

To enrich modern interfaces that almost exclusively rely on pointing, an alternative interaction paradigm has recently been proposed based on *goal-crossing tasks* [1]. Those tasks involve moving the mouse beyond the boundary of graphical objects for triggering actions. The authors briefly suggest the possibility for crossing a boundary back and forth (double-crossing) or more, in order to increase the command vocabulary. The technique we described, e.g. folding a window by double-crossing its border while dragging, is a perfect application of double-crossing. Here are two reasons:

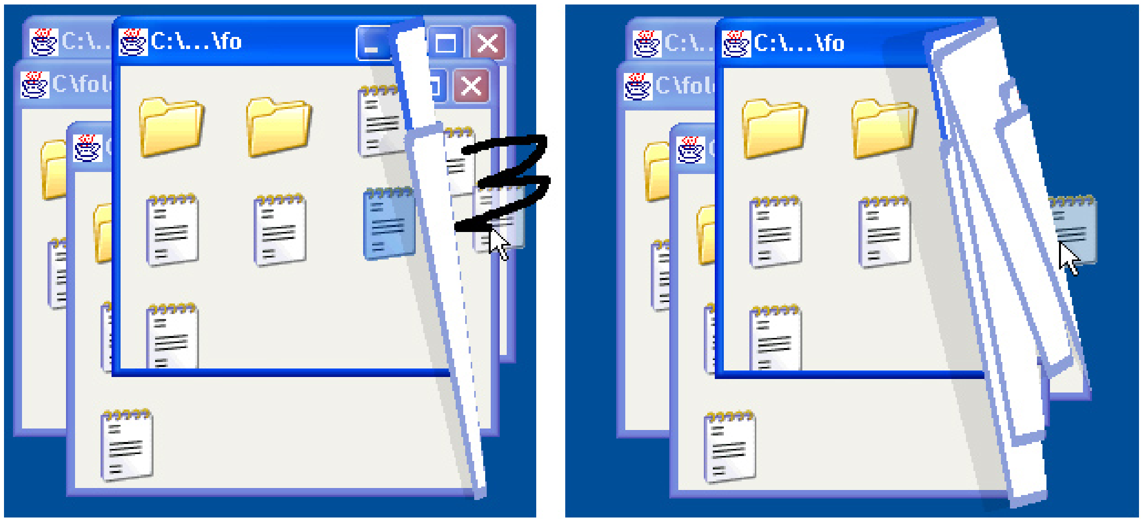

Figure 5: Leafing through windows while holding the dragged object.

• Because pointing-based tasks are not performable during the time of a drag (unless using other buttons), crossing-based tasks become quite beneficial in this context. One rudimentary example is automated scrolling in some text editors.
• There is a close mapping between the two-dimensional double-crossing task and the tri-dimensional task of sliding an object behind another one. There is also a natural relationship between multiple crossings performed on lined-up boundaries, and the everyday task of leafing through.

Fold-and-drop additionally extends the crossing-based paradigm with two interesting concepts:

• *Timed double-crossing*. Distinguishing between slow and fast double-crossing makes it possible to cross boundaries without triggering any action. Timed double-crossing can be seen as a *crossing-based equivalent to double-clicking*. As with double-click, it is likely that timing should be customizable to adapt to different user skills. Our technique also shows that with appropriate animated feedback, timing can be made visible to the user.
• *Pushing boundaries*. After being confirmed, a fold boundary continuously adjusts its location each time it is crossed, as if it were pushed. Pushing can also be seen as *crossing-based dragging*.

Gestural interaction

A parallel has already been established between double or multiple crossing and *gesturing* [1]. In fold-and-drop, fast double-crossing is obviously perceived as a gesture. Double-crossing gestures are however different from traditional ones as no classification technique is needed and ambiguity is not an important issue. Moreover, the gestures used in fold-and-drop are self-explanatory and only involve inductive learning. The animated fold feedback, which helps the novice understand the underlying metaphor, is not needed any more as faster gestures are used. This is a feature fold-and-drop shares with techniques such as Marking Menus [5, 7]. Another interesting characteristic of fold-and-drop is that it combines gestures with direct manipulation, as it interprets the mouse trajectory during drag-and-drops. From this point of view, fold-and-drop adds a new and convincing example to a family of hybrid techniques sometimes called “ecological gestures” [6, 7].

We also believe that the type of gesture we exploit is rarely made during regular drag-and-drops, and thus should seldom be initiated by mistake. In an exploratory study in which we monitored the activity of several users, we discovered that “exit window / re-enter window” gestures *do* occur during mouse drags ; however, we also observed that they happen most of the time when moving scrollbars located near the window border (e.g., in an Internet browser or a text editor). We believe that those gestures occur much less often when dragging *droppable* objects, although this still has to be confirmed by user experiments.

Paper-Based Window Metaphors

The metaphor used in drag-and-fold directly borrows from Beaudouin-Lafon’s peeling-back technique [3] (see also [4] and [9]). Using this technique the user can drag a corner of a window to fold it then activate a window underneath before the folded window springs back. We use the same graphical effect with some aesthetical enhancements such as shadows. The main differences reside in the metaphor and the interaction techniques used. For example, we push the fold instead of dragging its corner. The ability to fold multiple windows at the same time, a central feature of our technique, also required a significant extension of the original paradigm.

There are two reasons why we use folding instead of simply discarding windows, for example. First, a window does not need to be completely hidden – or minimized – when looking for a window behind. Adjusting its graphical attributes instead (e.g., location, size, shape or transparency) keeps the window and thus part of the context visible; it also facilitates backtracking in case the window needs to be displayed again. In our case, one main advantage of folding compared to other graphical effects is that it blends particularly well with the double-crossing technique and dramatically adds to the metaphor.

IMPLEMENTATION

We implemented the fold-and-drop technique using Java Swing's internal frames and a simulated file manager. We briefly describe how we display folded windows and handle user manipulation. We also raise some implementation issues.

• *Displaying folded windows*. Using Swing, non-rectangular internal frames are easily obtained by subclassing their `paint` method and clipping their graphics. The `contains` method also has to be redefined for mouse events to be dispatched through holes. Our frames additionally install a shared layered pane that handles fold display. For details on how to compute fold shapes see [3]. We add a scale transform in the direction perpendicular to the fold line in order to distinguish the folded corner from its shadow. Shadow shapes are ANDed into a unique area before being displayed with a translucent color. The painting order is the following: the clipped frames, the shadow area then the folded corners in reverse order with respect to frames.

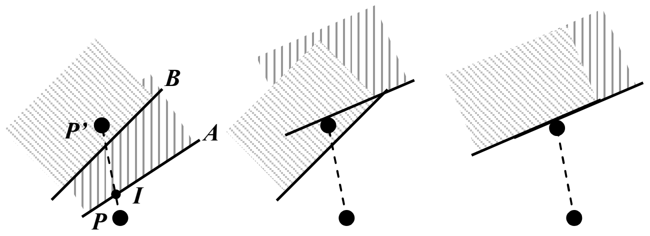

Figure 6: Folds computation after a mouse move. The grayed areas indicate the window side.

• *User manipulation*. We could not use Swing's mouse enter/leave events for handling user input because they do not carry enough information (we need the previous mouse location) and inadequately handle crossing of multiple boundaries (e.g. in Figure 6, *A* does not receive the "enter" event). Because Swing's event model is complex to extend we had to do it the "dirty way", i.e. by giving all responsibilities to the layered pane and making him listen to all mouse events. The underling mechanism is nevertheless the same: when the mouse moves from *P* to *P'*, folds intersecting [*P*, *P'*] are computed. The fold whose intersection *I* is the closest from *P* (fold *A* in Figure 6) is translated by *IP'* plus a small delta so that *P'* remains on the same side. The fold is also slightly rotated towards the direction perpendicular to (*P*, *P'*). After this, other folds are moved so that geometrical coherency constraints are verified i.e. a fold never intersects an upper window.

CONCLUSION

This paper described fold-and-drop, a new interaction technique for seamlessly dragging and dropping objects between overlapping windows. It also discussed three known interaction paradigms related to fold-and-drop and briefly described its implementation. We believe this technique can fill an important need in today's desktops. It also shows novel interaction techniques can still be introduced to improve the desktop but there is a huge need for more flexibility in GUI Toolkits and window managers to implement them.

The Java demo of the fold-and-drop technique can be downloaded with other related material at:
`http://liihs.irit.fr/dragice/foldndrop`

ACKNOWLEDGMENTS

Many thanks to Jean-Daniel Fekete for his ideas and his helpful comments on this paper, to Sandra Steere for reviewing the English and to David Navarre for his geometry formula.

REFERENCES

1. Accot, J. and Zhai, S., More than dotting the i's - foundations for crossing-based interfaces. In *Proc. of CHI 2002*. pp. 73-80.
2. Baudisch, P., *et al*. Drag-and-Pop and Drag-and-Pick: Techniques for Accessing Remote Screen Content on Touch- and Pen-operated Systems. In *Proc. of Interact 2003*. pp. 57-64.
3. Beaudouin-Lafon, M. Novel interaction techniques for overlapping windows. In *Proc. of UIST 2001*. pp. 153-154.
4. Denoue L., Nelson L., Churchill E.F. A fast, interactive 3D paper-flier metaphor for digital bulletin boards. In *Proc. Of UIST 2003*. pp. 169-172.
5. Kurtenbach, G., Sellen, A. and Buxton, W. An Empirical Evaluation of Some Articulatory and Cognitive Aspects of Marking Menus. *Human-Computer Interaction* 8, 1, 1993, pp. 1-23.
6. Mertz, C., Vinot, J.L. and Etienne, D. Entre interaction directe et reconnaissance d'écriture : les gestes écologiques. (between direct interaction and writing recognition: ecological gestures). In *Proc. of ergo-IHM2000*, Biarritz, France, pp 145-152.
7. Pook, S., Lecolinet, E., Vaysseix, G., Barillot, E. Control Menus: Execution and Control in a Single Interactor. In *Proc. of CHI 2000*. pp. 263-264.
8. Rekimoto, J. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. In *Proc. of UIST 97*. pp. 31-39.
9. Roussel, N. Ametista: a mini-toolkit for exploring new window management techniques. In *Proc. of CLIHC 2003, Latin American Conference on Human-Computer Interaction*, ACM Press, August 2003, pp. 117-124.
10. Tomitsch, M. *Trends and Evolution of Window Interfaces*. Diploma thesis, University of Technology, Vienna, December 2003, 132 pages.