Copy-and-Paste Between Overlapping Windows

Olivier Chapuis, Nicolas Roussel

Copy-and-Paste Between Overlapping Windows

Olivier Chapuis[1,2] **Nicolas Roussel**[1,2]
{chapuis,roussel}@lri.fr

[1]LRI - Univ. Paris-Sud & CNRS
Bât. 490, F-91405 Orsay, France

[2]INRIA
Bât. 490, F-91405 Orsay, France

ABSTRACT
Copy-and-paste, one of the fundamental operations of modern user interfaces, can be performed through various means (e.g. using the keyboard, mouse-based direct manipulation or menus). When users copy and paste between two different windows, the process is complicated by window management tasks. In this paper, we propose two new window management techniques to facilitate these tasks in the particular case of partially overlapping windows. We describe an experiment comparing four commonly used copy-and-paste techniques under four window management conditions – non-overlapping windows, partially overlapping windows, and partially overlapping ones with one of our two window management techniques. Results show that our new window management techniques significantly reduce task completion time for all copy-and-paste techniques. They also show that X Window copy-and-paste is faster than the other three techniques under all four window management conditions.

Author Keywords
Copy-and-paste, Window Management, Overlapping Windows.

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: Interaction styles, Windowing systems.

INTRODUCTION
Copy-and-paste (or copy-paste) is the basic mechanism for replicating part of a document in the same or another document. Already available in early systems such as Sketchpad [28] or NLS [11], copy-paste is one of the fundamental services provided by modern graphical user interfaces. Copy-paste requires the user to specify two things: the object(s) to copy and the destination. These can be done in different orders and using various means such as the keyboard, mouse-based direct manipulation or menus. Over the years, several "standard" techniques have emerged, such as the use of Ctrl-C to copy previously-selected objects and Ctrl-V to paste them. But although these techniques are used by millions of people several times a day, the interaction is still poorly understood. The techniques are implemented differently across operating systems and among applications[1] but most importantly, to our knowledge, they have never been formally evaluated.

Copy-paste operations between two different windows usually require users to perform additional window management tasks. If the source and destination windows overlap, for example, the user often has to temporarily change the stacking order to specify the objects to copy and the destination. Yet again, the interactions and potential interferences between copy-paste and window management operations have received very little attention. A notable exception is Dragicevic's work on the Fold n' Drop technique [9] that could be applied to the particular case of drag-and-drop copy-paste.

In this paper, we propose two new window management techniques, *restack* and *roll*, to facilitate copy-paste between partially overlapping windows. We describe an experiment comparing four commonly used copy-paste techniques (keyboard shortcuts, a context menu, drag-and-drop and a technique specific to X Window) under four window management conditions: non-overlapping windows, partially overlapping windows, and partially overlapping ones with one of our two new window-management techniques. Results from this experiment show that restack and roll significantly reduce the task completion time for all copy-paste techniques. They also show that the X Window technique is faster than the three others under the four window management conditions.

The paper is organized as follows. In the next three sections, we review some of the related work, describe some common copy-paste techniques and report on a series of interviews that illustrate how they are used in practice. We then present our restack and roll techniques, detail the experiment that was conducted to evaluate them and compare the four copy-paste techniques. Finally, we discuss some implications of our results, propose some solutions to the problems identified in the paper and generalize our key ideas into the concept of *fine-grained window management*.

O. Chapuis and N. Roussel. Copy-and-Paste Between Overlapping Windows. In CHI '07: Proceedings of the SIGCHI Conference on Human Factors and Computing Systems, pages 201-210, ACM, April 2007.
Authors Version

[1]On Microsoft Windows XP, for example, selecting text with the mouse in an overlapped window works with NotePad but not with WordPad.

RELATED WORK

Although tiled windows may be more efficient for certain tasks [6, 14], the overlapping model is the *de facto* standard for all modern window systems and plays an essential part in the realization of the *desktop metaphor*. The overlapping approach supports both time-based and space-based multiplexing of the screen space by *switching* (between windows) and *splitting* (the screen) [16]. However, as the number of windows increases, it imposes time-consuming and potentially complex management tasks on the user. The goal of the work presented in this paper is to reduce this overhead. Examples of related work include techniques for leafing through stacked windows, peeling them back or making them selectively transparent to access windows underneath [3, 9, 13], and dynamic space management algorithms to reduce overlapping [4].

Users with large displays tend to leave more applications running and associated windows open [21]. Like Hutchings and Stasko [12], we believe that overlapping windows will not disappear with the advent of larger displays. First, a variety of devices will keep using small or medium-size screens. More fundamentally, although large displays make it easier to develop tiling strategies, interactions across large screen distances may become more complex and time-consuming than keeping windows together on a smaller space. Large displays are probably better used to differentiate primary and peripheral activities, i.e. for tiling tasks, not windows [21]. We anticipate that larger displays will lead to fewer maximized (full-screen) windows that completely hide others. In some cases, the previously-obscured windows may become tiled on a larger display, but in many others, they will partially overlap. Therefore, copying and pasting between partially overlapping windows will remain important.

Modifying an existing document or combining pieces from several is always easier than creating a new one. Designers of the Xerox Star said it elevated the concept of copying to the higher level of "a paradigm for creating" [25]. We indeed believe that the combined use of overlapping windows and copy-paste supports innovation and creativity [24]. Copy-paste has been studied in specific domains such as programming environments [31, 15], graphical editors [8] or ubiquitous computing environments [19, 20]. Much previous research has tried to make it "smarter" by analyzing the selected data. Citrine [27], for example, can recognize that structured text has been copied, and paste it in multiple form fields in a single operation. Other systems have been proposed to support fast copy-paste of multiple selections or text entities like phone numbers [18, 5]. In this work, we are not interested in the objects being copied, or in optimizing copy-paste for a particular domain. Rather we are interested in the low-level interactions between copy-paste and window management operations.

COPY-PASTE TECHNIQUES

In this section, we describe what we believe are currently the four most common copy-paste techniques. Note that although we focus on copy-paste, most of these techniques can also be used for cut-and-paste operations, the main difference being that the selection is deleted after the copy has been made. Other differences between copy and cut will be further explained, as needed.

Copy-paste usually starts by using the mouse to select one or more object using one or more click(s) or a press-drag-release gesture[2]. This selection might be assisted, for example by automatically snapping to the edges of objects for example. The user must then (1) activate the copy command, (2) specify the destination – in the same window or another one – and (3) activate the paste command. We will now describe several ways of accomplishing these three operations.

Using the Keyboard

Sketchpad and the Xerox Star had specific Delete, Copy and Move keys that could be used in conjunction with the pointing device. Pressing the Copy key on a Star, for example, attached the selection to the cursor, and then a mouse click specified the destination. Modern systems do not have specific keys for these functions but support keyboard-based copy-paste in a less modal way: (1) a first shortcut, e.g. Ctrl-C, causes the selection to be copied; (2) the user navigates to the destination using the mouse and/or the keyboard; (3) a second shortcut, e.g. Ctrl-V, performs the paste.

We refer to the use of keyboard shortcuts to activate the copy-paste commands as KEY *copy-paste*.

Using Menus

In addition to being accessible through keyboard shortcuts, copy-paste commands are usually found in the standard menu bar of applications, e.g., under the *Edit* item, as icons in palettes and toolbars, and in context menus accessible from the selected objects, e.g., using a right click.

Menu bars are very similar to context menus but impose additional mouse travel to reach them after selecting objects and after indicating the insertion point. Copy and paste icons in toolbars or palettes have the same problem, so we decided to focus on the use of context menus to activate the copy and paste commands.

We refer to the use of context menus to copy-paste as MENU *copy-paste*.

Using Drag-and-Drop

Drag-and-drop offers a more direct way of performing a copy-paste operation. The user simply has to press a mouse button on one of the selected objects, drag the mouse pointer to the destination and release the button. However, this technique has several problems. First, its semantics are not always easy to determine: although one can reasonably assume that dropping something on a trash icon deletes it, dropping it somewhere else might copy it or move it. As a consequence, application designers often disagree with users on what the drag-and-drop operation should do [10].

[2]As a notable exception, users of the Xerox MESA programming environment had to first specify the destination and then the text to be copied [29]. Note that objects might also be selected using the keyboard, but this does not affect the descriptions that follow.

A second problem is that the drag-and-drop requires continuous pressure on the mouse button. Besides being fatiguing and error-prone, this can make it difficult to navigate between windows to reach the destination. While keyboard shortcuts may make it possible to switch between and close windows, other functions such as minimizing, opening or moving them may be difficult if not impossible. Some applications support initiating a drag in an inactive window without bringing it to the foreground, which makes it easier to arrange the source and destination windows before the drag-and-drop operation. Another interesting solution is the use of time-based navigation techniques. As an example, the "spring-loaded" windows of the Mac OS X Finder automatically move to the foreground during a drag-and-drop if the pointer stays more than a certain time over them, go back to their original place if the pointer leaves them and stay on top if the object is dropped.

A third problem occurs when users make a too-large text selection and try to correct it [24]. In this case, pressing the mouse button inside the selected text initiates the drag-and-drop instead of initiating a new selection process. We refer to this problem as the *drag vs. subselection problem*. Note that an easy workaround is to perform a simple click to cancel the selection and then press-drag-release to make a new one.

We refer to the use of drag-and-drop to copy-paste as DND *copy-paste*.

The X Window Case

The X Window system features a simple copy-paste technique: a click on a window with the middle mouse button[3] pastes the last selection at the insertion point of that window (applications may decide to move the insertion point under the mouse pointer before pasting). This technique minimizes the number of user actions: it works as if the copy command was implicitly executed after each selection, and the mouse action that pastes also specifies the destination. In addition to this mouse-controlled *primary selection*, X also features an explicit clipboard usually accessible through the standard keyboard shortcuts we already described (i.e. Ctrl-C, Ctrl-V). Both mechanisms can be used at the same time.

One drawback of the implicit copy approach is the *volatility* of the primary X selection, as illustrated by the following scenario:

> The user selects a URL to paste in the location field of a Web browser. The field holds a previous URL. The user decides to clear it before pasting the new one: he triple-clicks on it, which selects it, and presses the Delete key. When he presses the middle mouse button, the URL he just deleted reappears...

We refer to the use of the X primary selection to copy-paste as X *copy-paste*.

[3]The X protocol was originally designed for mice with up to three buttons: left, right and middle [23]. The middle one was sometimes simulated by pressing the two others simultaneously or pressing one of them with a modifier key. Most mice now have three or more buttons, a clickable scroll wheel being often used as the middle one.

COPY-AND-PASTE PRACTICES

We interviewed twenty-one people on their copy-and-paste and cut-and-paste habits, specifically of text. We consider these people as "expert users", most of them being computer science students or engineers. Among them, ten use the X Window system, eight use Microsoft Windows and three use Apple Mac OS X.

Before asking specific questions on copy-paste, we questioned the participants on how they arrange their windows. The use of partially overlapped windows was quite common. Eleven said they either use maximized windows or partially overlapped ones, depending on the applications they run and their tasks. Four said they primarily use maximized windows, and four that they primarily use partially overlapped windows. Only two said they carefully arrange their windows by resizing and moving them following a tiling approach. These two participants also use maximized windows.

Three participants said they rarely use copy-paste. All the others said they use it very often between windows. OS X and Windows users mostly use KEY copy-paste. X Window users mostly use KEY and X copy-paste, two having said they only use X copy-paste (for text). One said he uses X copy-paste only in terminal applications where Ctrl-C is used to interrupt programs and the replacement shortcut requires both hands. This participant was a long time Windows user who switched to X Window two years ago. Only two participants said they use MENU copy-paste more than KEY copy-paste, both being Windows users and one of them having said he rarely uses copy-paste. Most people said they use MENU copy-paste. Three explained that it seemed safer (i.e. less error-prone) than the other techniques.

Three participants said they use DND cut-and-paste for text from time to time, but two said they only use it in a single window. The other participants were unable to say if a drag-and-drop moves or copies text. The *drag vs. subselection problem* was never mentioned. But when explained to the participants, thirteen said they had run into it.

Among the ten users of the X copy-paste, five mentioned the volatility problem described in the previous section. One participant said he often loses selections as he likes to select text to highlight it as he reads it. Another said he uses X copy-paste only when both source and destination windows are visible because he fears to lose the selection when he performs "complex" window management tasks like virtual desktop switching. The five X copy-paste users that didn't mention the volatility problem said they ran into it after we described it. Some said this was one of the reasons why they also use KEY copy-paste and not only X copy-paste.

We asked a few specific questions about *clipboard history tools* – tools that keep track of copy operations and support easy reuse of previously copied items. Nine participants said they had tried such tools, but do not use them anymore (six tried with Microsoft Word, three tried the KDE Klipper). We will come back to this topic in the DISCUSSION section.

All participants said they are generally happy with the way copy-paste works on their system. Some of them complained about the fact that the selection snaps to words. Others complained about the fact that they sometimes get unexpected results because of silent data conversion or formatting between the source and destination applications (e.g. between a Web browser and a word processor).

RESTACK AND ROLL COPY-AND-PASTE

Partial overlapping allows one to work on a document while keeping parts of related windows at hand (Figure 1, top). However, as illustrated by the following scenario, it quickly introduces potentially complex window management tasks when combined with even the simplest copy-paste operation:

> Héloïse is editing a text document in a window that partially covers a Web browser. She wants to copy part of the text visible in the browser into her document. She selects the relevant text in the browser with a press-drag-release gesture. As she presses the mouse button, the browser is brought to the foreground. When she releases it, the browser stays on top, partially covering her document. Héloïse presses Ctrl-C to copy the selection. She clicks on her document to bring it to the foreground and presses Ctrl-V to paste the text.

Looking at this example, one might think that the window management overhead is small, consisting in a single click on the document to bring it back to the front. But reaching this document might be difficult, since it is now behind the browser. It might even be impossible if it is fully covered (in that case Héloïse would probably use Alt-Tab to reach for it). Clicking on the document might move the insertion point and require further navigation inside it. Clicking on the window decorations solves this problem, but they are small and thus difficult to select (they may also contain dangerous controls such as the window-close button). Finally, the overall copy-paste operation changes the stacking order of the browser which, as the window second from the top of the window stack, might now cover other windows.

The root cause of these problems is that as soon as Héloïse starts interacting with a window, the system assumes it to be her primary interest. Current windowing systems provide little support to indicate secondary interest. The "focus follows mouse" policy implemented by some systems, as opposed to "click to focus", is a good example of what can be done, but it is limited to keyboard interaction. In order to further reduce this problem, we propose the following design principle:

> A left button click in a secondary window should make it of primary interest, i.e. make it active by raising it and giving it the keyboard focus. Any other interaction with a secondary window should be treated as a temporary interest indication and should be handled in a specific, appropriate way.

We propose two new window management techniques based on this design principle in order to facilitate copy-paste operations. Our previous scenario can be used to illustrate them. The first technique, *restack*, operates as follows:

> When Héloïse presses the mouse button to initiate the text selection in the browser, it is brought to the foreground. As she keeps the button pressed and starts dragging the mouse, the system infers that she is not indicating primary interest. As a consequence, when she releases the button, the browser returns to its original place in the stacking order, behind the document. It keeps the keyboard focus though, so that Héloïse can drag the selection but also use Ctrl-C to copy it. Had she decided to use a context menu, it would have been displayed in the foreground, but the browser would have stayed in its place. She can now easily drag-and-drop the selection or paste it using Ctrl-V or a context menu in the document.

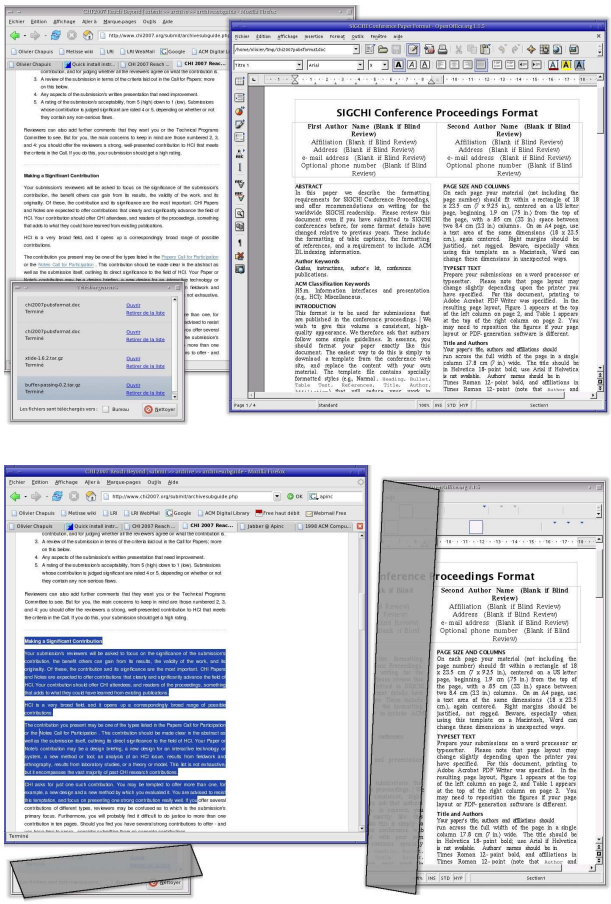

Figure 1. Rolling windows to reveal an overlapped one.

The second technique we propose, *roll*, uses a variant of the folding operation described in [3, 9] instead of automatic restacking:

> When the system infers that Héloïse is not indicating primary interest in the browser, all the windows that cover it are rolled back with a fast animation so as to fully reveal it (Figure 1, bottom). When, Héloïse finishes her selection, the windows roll back to their original state, again with an animation.

The rolling metaphor was chosen over the folding one because it hides less window content (Figure 2) and leads to less obtrusive animations. We believe that an advantage of using the restack and roll techniques for copy-paste is that

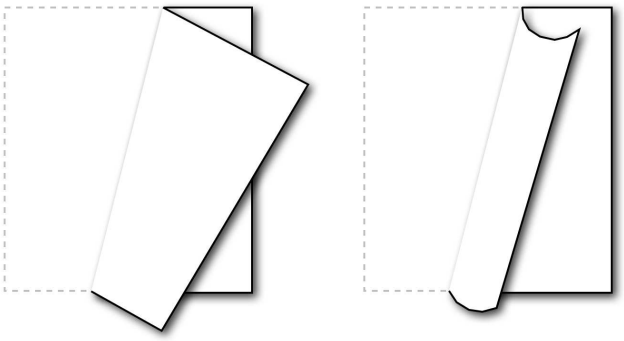

Figure 2. Folding (left) vs. rolling (right).

the window of primary interest is covered by auxiliary windows for a minimum time (during the selection), which may help users stay focused on their primary task. A drawback of these techniques is that users who want to switch focus and immediately make a selection must first click on the window to change its status. However, this limitation seems acceptable as it is quite similar to the strict "click to focus" implemented by many systems and applications.

Our design principle is based on the idea that augmenting the window management complexity will probably lead to more powerful user interfaces. One may argue that this additional complexity should be provided for expert users only.

EXPERIMENT

We conducted an experiment to compare completion times and user preferences for the KEY, MENU, DND and X copy-paste techniques between two windows under four window management conditions. We first distinguish the non-overlapping (NONOVERLAPPING) and the overlapping cases. In the overlapping case, we further distinguish three cases corresponding to the window management techniques available: the usual set of techniques (OVERLAPPING), and the restack (RESTACK) and roll (ROLL) techniques described in the previous section.

We decided to use what we thought were the most efficient variants of the MENU, DND and X techniques. Our implementation of the X technique pastes the selection under the mouse pointer (there is no notion of insertion point in the experiment). Similarly, the MENU technique pastes where the right mouse button was clicked to open the context menu. In the case of the DND technique, the initiation of a drag on a window in the background does not raise it but windows are immediately raised when the dragged object enters them in the OVERLAPPING condition. In this condition, when the drag-and-drop is not used, the only other way to raise a window is to click on it. This was decided to simplify the experiment and seemed reasonable since the two windows overlap only partially and are quite big.

Hypothesis

X copy-paste is the technique that requires the least elementary operations (free mouse movements, mouse drags, button clicks and key presses). KEY copy-paste requires two additional key presses (e.g. Ctrl-C and Ctrl-V) and DND copy-paste requires some slower mouse dragging [17] in addition to free mouse movement. These elements lead us to our first hypothesis:

H1: X *copy-paste is faster than* KEY *and* DND *copy-paste.*

This hypothesis is not so obvious in the X vs. KEY case as pressing Ctrl-C can be done while moving the mouse and X paste requires a middle button press which can be delicate to perform (e.g. in the case of a mouse wheel button). We see no obvious reason to separate KEY and DND copy-paste, and MENU copy-paste requires a lot of elementary operations (two right clicks and some menu navigation). So we proposed a second hypothesis:

H2: KEY *and* DND *(and* X*) copy-paste are faster than* MENU *copy-paste.*

Concerning the window management conditions, since the RESTACK and ROLL techniques do not require the user to raise the destination window, we proposed a third hypothesis:

H3: RESTACK *and* ROLL *techniques are faster than the* OVERLAPPING *technique for all the copy-paste techniques.*

As NONOVERLAPPING doesn't require the user to raise the destination window either, one can reasonably assume that it should be faster than OVERLAPPING. However, we see no evidence to separate (RESTACK,ROLL) and NONOVERLAPPING as RESTACK and ROLL lead to less mouse travel, but NONOVERLAPPING makes the source and destination points always visible. Finally, it is not clear whether the animations accompanying the ROLL technique are better or worse than the immediate restacking of the RESTACK technique in terms of completion time and user preferences.

Experimental Design

A repeated measures within-subject 4×4 factorial design was used. The two main factors are the copy-paste techniques (KEY, MENU, DND, X) and the window management conditions (OVERLAPPING, RESTACK, ROLL, NONOVERLAPPING). The main measure is the completion time to perform a copy-paste between two windows (Figure 3). The experiment was conducted with 18 volunteers and unpaid Computer Science students and engineers (16 males and 2 females): nine X Window users, six Windows users and three Mac OS X users. All but one had also participated in the interviews on copy-paste practices.

The experiment consists of 16 trial groups, each group consisting itself in a series of at least 4 trials. Within each group, the copy-paste technique and the window management condition are fixed. Copy-paste techniques are not intermixed. As an example, the subject performs four trial groups of X copy-paste with NONOVERLAPPING, OVERLAPPING, RESTACK and ROLL; then four groups of DND copy-paste with ROLL, RESTACK, OVERLAPPING and NONOVERLAPPING; etc. Orders of the techniques and of the window management conditions

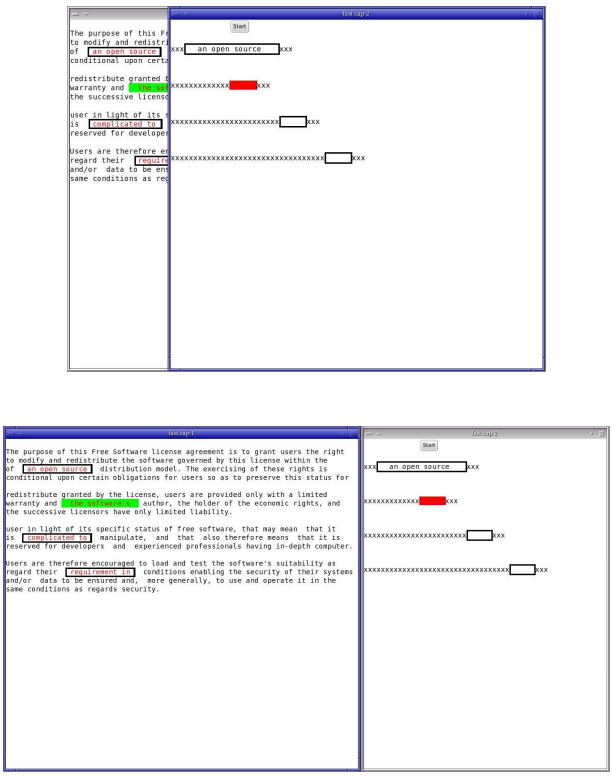

Figure 3. Overlapping and non overlapping conditions. Images show the second copy-paste of a trial with current text source and destination highlighted. Other sources and destinations have been framed for the convenience of the reader.

were pseudo-randomly balanced across participants following a Latin-square design.

A trial consists of a series of four copy-paste actions. The subject first presses a "start" button at the top of the right window. The first text to copy appears highlighted in green in the left window and the corresponding paste area in red in the right one. The subject selects the text and executes the copy-paste operation. As soon as the text is pasted, the next copy-paste areas are highlighted in the two windows, below the ones that were just used (see Figure 3). When the fourth copy-paste is done, the chronometer is stopped. The number of successful operations for the current trial is presented to the subject with some indication of his progression in the experiment. The subject can take a short break and then move on to the next trial by clicking the "start" button again. Note that we do not consider these repeated copy-pastes as a natural task. This is simply a way of obtaining as many copy-paste completion times as possible in a minimum amount of time. These times should be representative, similarly to the classical Fitts' pointing experiments that use back and forth pointing.

The four texts to copy during the trials have the same number of characters and the same length (a monospace font is used). Their position and the position of their corresponding paste area are fixed. These positions were chosen so as not to be favorable to a particular window management condition (the first and second copy-paste are favorable to RESTACK and ROLL, the third and fourth ones to NONOVERLAPPING and OVERLAPPING). In the overlapping case, two are fully visible while the two others are initially half-covered by the right window (Figure 3, top). In the NONOVERLAPPING condition, the right window is just moved further to the right to suppress overlapping and resized to keep it fully on screen.

Each trial group starts with a training period used to explain the technique to the subject. Subjects are allowed to train as long as they want ("train until you feel at ease with the technique"). The first trial actually starts when the subject presses a button. Subjects are instructed to "perform as fast as possible without errors". The group finishes when the subject has successfully performed four copy-paste of each of the four texts (i.e. at least four trials and sixteen successful copy-paste). Pasting can be done anywhere in the paste area, which is made of several spaces. The selection mechanism is also space-tolerant. The error policy is otherwise strict: the subject must perform the perfect interaction.

Apparatus

The restack and roll window management techniques were implemented inside the Metisse window system [7]. The software used for the copy-paste experiment was written in C using the GTK+ toolkit. The experiment ran on a 2.66 GHz bi-processor PC running Linux with a powerful graphics card connected to a 1280x1024 LCD display. The mouse used was a standard optical one with two buttons and a clickable wheel that could be used as a third (middle) button. The default linear X Window mouse acceleration was used. Instructions and source code needed to reproduce the experiment are available from `http://insitu.lri.fr/metisse/rock-n-roll/`.

Results

We analysed the data using a repeated measures analysis of variance (ANOVA), with completion time to perform one copy-paste (CT in milliseconds) as measure, subject as a random effect factor, copy-paste techniques and window management conditions as fixed effect factors. Our major interest is in the interactions between copy-paste techniques and window management conditions. We used JMP [22] to perform the ANOVA with the Restricted Maximum Likelihood (REML) method. Erroneous copy-paste operations were removed from our data: on a total of 5229 trials, 4608 error-free completion times were taken into account (18 participants $\times$ 4 copy-paste techniques $\times$ 4 management conditions $\times$ 4 texts $\times$ 4 repetitions).

Figure 4 shows almost all the results of this analysis. The copy-paste techniques reveal significant effects ($F_{3,4575} = 2334$, $p < 0.001$) as do the window management conditions ($F_{3,4575} = 248$, $p < 0.001$), and there is no evidence of an interaction between these factors ($F_{9,4575} = 1.01$, $p = 0.43$). As no interaction appears to be present, we focus our analysis on the main effects. We use the Tukey HSD (honestly significant difference) test with $\alpha = 0.05$. Figure 5 details the results of these tests for our two main factors.

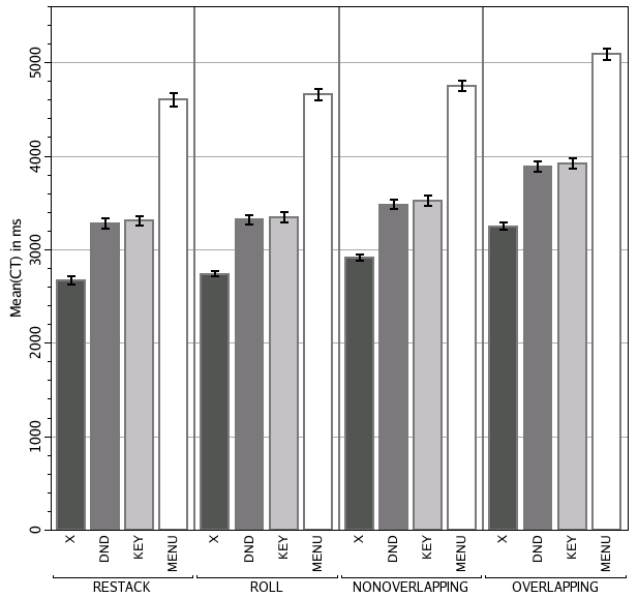

Figure 4. Completion time for each copy-paste technique, grouped by window management condition. Error bars show standard error.

	X	DND	KEY	MENU
X	0 0 0	-599 -659 -539	-633 -693 -574	-1883 -1943 -1824
DND	599 539 659	0 0 0	-34 -93 25	-1284 -1343 -1224
KEY	633 574 693	-34 -25 -93	-0 -0 -0	-1249 -1309 -1190
MENU	1883 1824 1943	1284 1224 1343	1249 1190 1309	-0 -0 -0

	RESTACK	ROLL	NONOVER.	OVER.
RESTACK	0 0 0	-50 -109 9	-200 -259 -140	-570 -629 -510
ROLL	50 -9 109	0 0 0	-150 -209 -90	-520 -579 -460
NONOVER.	200 140 259	150 90 209	-0 -0 -0	-370 -429 -310
OVER.	570 510 630	520 460 579	370 310 429	0 0 0

Figure 5. Tukey crosstab: techniques (top) and window management conditions (bottom). A cell contains the means difference and the lower and upper bound of the confidence interval. Underlined cells are significant.

First, we examine the copy-paste techniques. X copy-paste is significantly faster than KEY and DND, and KEY and DND are significantly faster than MENU (H1 and H2 are supported). We found no significant difference between KEY and DND (this is also the case for each window management condition). A practical equivalence test [30] with a threshold of 100 ms (3% of the means) gives a positive result ($p = 0.002$). However, as we explained, our implementation of the DND technique immediately raises a window in the OVERLAPPING condition when the dragged text enters it. So, one may assume that with a "spring-loaded"-like implementation, we would have found a significant difference between means under OVERLAPPING, caused by the delay. It is interesting to note that X is 18% faster than KEY which is probably the most popular technique among "expert" users.

We now examine the window management conditions. H3 is supported: ROLL and RESTACK are significantly faster than OVERLAPPING (and NONOVERLAPPING is also faster than OVERLAPPING). We found no significant difference between RESTACK and ROLL[4]. More surprisingly, ROLL and RESTACK are significantly faster than NONOVERLAPPING. The difference between means is small. However, X copy-paste is 10% faster with RESTACK than with NONOVERLAPPING. On the other hand, RESTACK is 18% faster than OVERLAPPING (for all techniques but MENU, where RESTACK gives only a 10% speed-up). Finally, we note that X with RESTACK is 32% faster than KEY in the OVERLAPPING condition. For such an elemental operation as copy-paste, this is a huge improvement. See Figure 6 for more numbers regarding the combinations of copy-paste techniques with the window management conditions.

Level								CT Mean
MENU,OVERLAPPING	A							5085
MENU,NONOVERLAPPING		B						4746
MENU,ROLL		B						4656
MENU,RESTACK		B						4600
KEY,OVERLAPPING			C					3919
DND,OVERLAPPING			C					3883
KEY,NONOVERLAPPING				D				3522
DND,NONOVERLAPPING				D	E			3475
KEY,ROLL					E	F		3342
DND,ROLL						F		3316
KEY,RESTACK						F		3305
DND,RESTACK						F		3277
X,OVERLAPPING						F		3244
X,NONOVERLAPPING							G	2906
X,ROLL							H	2737
X,RESTACK							H	2667

Figure 6. Means and significance for the combination of the copy-paste techniques and of the window management conditions. Levels not connected by the same letter (A,B,C, ...) are significantly different.

The overall error in the experiment was 5.58% (Figure 7). However, a new ANOVA (similar to our main ANOVA, but with errors as the measure) only shows that XS and DND are less error-prone than MENU: the technique is a significant factor, but the window management condition is not and there is no evidence of an interaction between the two factors.

In the OVERLAPPING condition, as we explained, two of the texts to select in the left window are half covered by the right one, while the two others are not. Since the lengths of the texts are equal, this allows us to compare the selection time

[4]However, with the less robust Student's t test we found a significant difference ($t = 2.16$, $p = 0.03$). The difference between the means is of 50 ms which can be compared to the roll back animation which takes 150 ms.

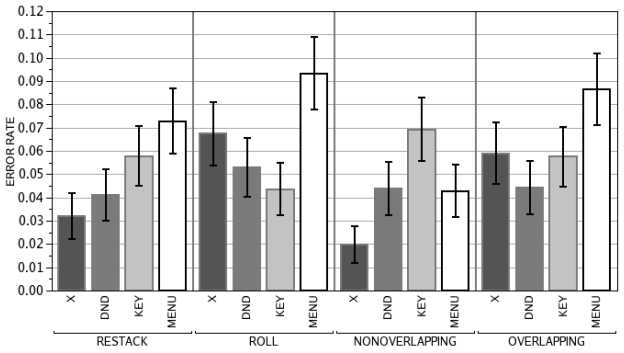

Figure 7. Error rate for each copy-paste technique, grouped by window management condition. Error bars show standard error.

for overlapped vs. non overlapped text. We performed an ANOVA similar to our main ANOVA, but we removed the NONOVERLAPPING data, added "text overlap" as a new fixed effect factor and took selection time as the measure. Text overlap has a strong effect. Subjects performed the text selection faster when the text was not overlapped (with a significant difference between means of 93 ms for a mean selection time of 858 ms). The only other effect is a small interaction between the text overlap condition and the window management conditions. The difference between means is more important with ROLL (129 ms) because the selection time is faster with ROLL for non overlapped text (non significant on its own) and slower for overlapped text (significant vs. OVERLAPPING, but not RESTACK).

KEY copy-paste is bi-manual: the left hand can be used to activate the keyboard shortcuts while the right hand controls the mouse. We tried to see how far the subjects did use synchronised bi-manual interaction. The average time between the moment subjects place the cursor in the paste area and the moment they press Ctrl-V is of 454 ms in average. This is big, but goes down to a value closer to 200 ms for some subjects. The average time between the moment subjects finish the selection and the moment they press Ctrl-C is 368 ms. During this time, some subjects move the mouse: we measured an average distance of 97 pixels in the non overlapping case (for some subjects this value is close to 0, but for some others, it is closer to 300). Subjects also move the mouse between the Ctrl-C press and release: we measured an average distance of 269 pixels in the non overlapping case (with no huge difference between subjects).

To estimate the time lost by pressing Ctrl-C, we performed an ANOVA similar to our main ANOVA but with only the X and KEY techniques (and all the window management conditions) and where the measure is the time between the end of the text selection and the moment where the subject presses the middle mouse button (in the X case) or positions the text cursor with a left click (in the KEY case) in the paste area. The copy-paste techniques and the window management conditions are significant factors and there is no evidence for an interaction between the factors. We get a significant difference between means for the X technique vs. the KEY technique of 223 ms ($p < .001$).

Qualitative Results

At the end of the experiment, participants were asked to rate the copy-paste techniques and window manager conditions.

Of the eighteen subjects, fifteen said they preferred X copy-paste. Two said they preferred DND (both liked the feedback given by this technique). Only one said he preferred KEY. Thirteen subjects cited MENU as the worst technique, two cited DND and three cited KEY. KEY was cited ten times as the second preferred technique, DND was cited nine times at this place and MENU three times (with some ex aequo). X is clearly the technique that the subjects preferred. KEY and DND are the second preferred techniques and MENU is clearly disliked. It is interesting to note that between the nine Windows or OS X users, seven said they would like to have the X technique available on their system.

Eight subjects preferred to perform copy-paste under the NONOVERLAPPING condition (eight subjects cited it in second place). Seven subjects cited RESTACK as their preferred way to run the experiment (eight subjects cited it in second place). Most subjects said that they were disturbed by the animation of the ROLL techniques, three subjects placed it first (they liked the animation and the graphics), three subjects at the second position and ten at the third position. All the subjects preferred RESTACK and NONOVERLAPPING to OVERLAPPING. Only two subjects did not cite OVERLAPPING as the worst technique. Both found the animation produced by ROLL strongly disturbing. RESTACK was thus accepted: because of this technique a reasonable number of subjects preferred an overlapping context to a non overlapping one, and most of the others placed it second. Moreover, two subjects asked if RESTACK could be made available on their system.

ROLL was not similarly appreciated, which is unfortunate because it gives more feedback about what is going on than RESTACK (only two subjects placed ROLL before RESTACK, and one placed both techniques second). One possible reason for this is that the subjects were told to perform the experiment as fast as possible, which the 150 ms animation didn't help (especially in our repeated task). Moreover, in the case of a MENU or DND copy-paste on an overlapped text, the unrolling of the right window over the text can make it difficult to open the context menu or drag it. Two subjects mentioned this as a problem. However, a third one claimed it helped him move to the left to grab the text.

The subjects easily answered our questions regarding their preferred copy-paste techniques and their preferred window management conditions. We also asked whether they had specially liked or disliked any combinations of them. We got very few answers. Two subjects who preferred the X technique said they preferred the DND technique under the NONOVERLAPPING condition. Three subjects made some remarks regarding the interaction between DND/MENU and ROLL (see the previous paragraph). A few subjects who said they disliked MENU added that they particularly disliked it in the OVERLAPPING condition. Two subjects who already preferred the X technique, said it was really better in the OVERLAPPING condition.

DISCUSSION

X copy-paste is fast, simple and can be used in conjunction with KEY and MENU copy-paste. Most people who tried it like it. X Window has unfortunately no equivalent technique for cut-and-paste. One could probably implement a drag-select-and-cut command, map it to a specific mouse button and reuse the middle button for pasting. However, as the interviews confirmed, the volatility of the selection clearly poses some problems. Selection history tools could certainly help, but none of the interviewed people were in the habit of using them. We believe the main problem with these tools is that they are usually accessible from a system or application menu bar, but not directly from the place where the user wants to paste. In the case of X copy-paste, we suggest that a long middle button press should pop up a context menu presenting the selection history. This idea can also be applied to KEY copy-paste: pressing Ctrl-V and holding the V key pressed could also pop up the history. The user could then circulate in it by repeatedly pressing the V key.

Restack and roll are currently used on a daily basis by the first author of this paper and a student. Both use the techniques several times a day and the first author even developed some placement strategies to take advantage of them. Most text documents being left-aligned (sometimes justified), overlapping them on the right side usually leaves more content visible than on the left side. As a consequence, when writing a paper, the author usually displays auxiliary documents (e.g. other papers, Web pages) on the left side of the screen, partially covered by a window on the right side showing the paper. The student often uses the restack technique to paste command line templates found on Web pages in a terminal that overlaps the browser and is sometimes fully surrounded by it. One interesting point is that both users diverted the techniques. As an example, they often make arbitrary selections just to temporarily expose an overlapped window. This can be viewed as the counterpart of the folding operation described in [3] which was designed to temporarily look behind a single window by grabbing one of its corners and peeling it back.

The idea of differentiating user interactions with the primary window from those with secondary windows opens an interesting design space. With restack and roll, we proposed specific actions that temporarily expose a window when the user selects some of its content. One might ask what should happen when the user interacts in other ways with a secondary window. What should happen, for example, when the user starts dragging the scrollbar of a partially covered window? One possibility would be to uncover it when the drag starts (e.g. using restack or roll), let the user manipulate the scrollbar and either put it back to its original place if the button is released outside the window or make it of primary interest otherwise.

Similar questions could be asked for other interactions, other types of widgets, unused window space or even the desktop. Selecting an icon on the desktop, for example, could temporarily expose its surroundings by rolling nearby windows, or rendering them using selective transparency [13] or multi-blending [2]. In an attempt to generalize these ideas, based on the design principles we previously described and similar arguments given in [1, 9, 13], we propose the concept of *fine-grained window management*, defined with the following goal:

> Take into account the context of user actions on windows (e.g. the window type or content, or its surroundings) to execute the most appropriate window management command.

Creating a fine-grained window manager poses a number of technical problems. It should, for example, have some knowledge about the relations between windows and their internal structure, e.g., the widgets they contain and the potential user actions on them. In order to work with a wide range of applications, our implementation of the restack and roll techniques relies on several "hacks" to monitor mouse activity and the various X selection mechanisms. Accessibility APIs provide clean ways to figure out internal window structures that can help implement fine-grained window management techniques [26]. But more than this, we believe there is a need for new, richer, bi-directional communication protocols between the window manager and the applications.

CONCLUSION

In this paper, we examined the problems related to copy-paste between partially overlapping windows. We proposed two new window management techniques, *restack* and *roll*, to solve some of these problems. We described an experiment that was conducted to evaluate these techniques and compare four common copy-paste techniques. Results show that X Window copy-paste is faster than the use of keyboard shortcuts, context menus or drag-and-drop. They also show that restack and roll significantly improve the four copy-paste techniques. Restack and roll were designed according to the idea that user interactions with windows of primary interest could differ from those with secondary windows. We intend to continue exploring this idea to develop the more general concept of fine-grained window management that takes the context of user actions into account.

ACKNOWLEDGEMENTS

We would like to thank the participants of the interview and of the experiment for there time and cooperation. We also thank J.D. Fekete, P. Dragicevic, J.B. Labrune, H. Goodell, A. Cockburn, M. Beaudouin-Lafon, W. Mackay, C. Appert, E. Pietriga and Y. Guiard for helpful discussions about this work. Finally, we thank the anonymous reviewers for their useful comments and suggestions about this paper.

This work has been partially funded by the French *ACI Masses de données* (Micromégas project).

REFERENCES

1. C. Appert, M. Beaudouin-Lafon, and W. Mackay. Context Matters: Evaluating Interaction Techniques with the CIS Model. In *Proceedings of HCI'04*, pages 279–295. Springer Verlag, 2004.
2. P. Baudisch and C. Gutwin. Multiblending: displaying overlapping windows simultaneously without the draw-

backs of alpha blending. In *Proceedings of CHI '04*, pages 367–374. ACM Press, 2004.

3. M. Beaudouin-Lafon. Novel interaction techniques for overlapping windows. In *Proceedings of UIST '01*, pages 153–154. ACM Press, 2001.

4. B. Bell and S. Feiner. Dynamic space management for user interfaces. In *Proceedings of UIST'00*, pages 239–248. ACM Press, 2000.

5. E. A. Bier, E. W. Ishak, and E. Chi. Entity quick click: rapid text copying based on automatic entity extraction. In *Extended abstracts of CHI '06*, pages 562–567. ACM Press, 2006.

6. S. Bly and J. Rosenberg. A comparison of tiled and overlapping windows. In *Proceedings of CHI '86*, pages 101–106. ACM Press, 1986.

7. O. Chapuis and N. Roussel. Metisse is not a 3D desktop! In *Proceedings of UIST '05*, pages 13–22. ACM Press, 2005.

8. W. Citrin, D. Broodsky, and J. McWhirter. Style-based cut-and-paste in graphical editors. In *Proceedings of AVI '94*, pages 105–112. ACM Press, 1994.

9. P. Dragicevic. Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. In *Proceedings of UIST '04*, pages 193–196. ACM Press, 2004.

10. E. Dykstra-Erickson and D. Curbow. The role of user studies in the design of OpenDoc. In *Proceedings of DIS '97*, pages 111–120. ACM Press, 1997.

11. D. C. Engelbart. Authorship provisions in augment. In *Proceedings of COMPCON '84*, pages 465–472. IEEE, 1984.

12. D. Hutchings and J. Stasko. Revisiting Display Space Management: Understanding Current Practice to Inform Next-generation Design. In *Proceedings of GI '04*, pages 127–134. Canadian Human-Computer Communications Society, 2004.

13. E. W. Ishak and S. K. Feiner. Interacting with hidden content using content-aware free-space transparency. In *Proceedings of UIST '04*, pages 189–192. ACM Press, 2004.

14. E. Kandogan and B. Shneiderman. Elastic Windows: evaluation of multi-window operations. In *Proceedings of CHI '97*, pages 250–257. ACM Press, 1997.

15. M. Kim, L. Bergman, T. Lau, and D. Notkin. An ethnographic study of copy and paste programming practices in OOPL. In *Proceedings of ISESE '04*, pages 83–92. IEEE, 2004.

16. B. Lampson. Personal distributed computing: the alto and ethernet software. In *Proceedings of the ACM Conference on The history of personal workstations*, pages 101–131. ACM Press, 1986.

17. I. S. MacKenzie, A. Sellen, and W. A. S. Buxton. A comparison of input devices in element pointing and dragging tasks. In *Proceedings of CHI '91*, pages 161–166. ACM Press, 1991.

18. R. C. Miller and B. A. Myers. Multiple selections in smart text editing. In *Proceedings of IUI '02*, pages 103–110. ACM Press, 2002.

19. B. A. Myers. Using handhelds and PCs together. *Communications of the ACM*, 44(11):34–41, 2001.

20. J. Rekimoto. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *Proceedings of UIST '97*, pages 31–39. ACM Press, 1997.

21. G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. Hutchings, B. Meyers, D. Robbins, and G. Smith. Scalable Fabric: Flexible Task Management. In *Proceedings of AVI '04*, pages 85–89, 2004.

22. SAS Institute Inc. *JMP Version 6*. SAS Institute Inc., Cary, NC, 1989-2005.

23. R. Scheifler and J. Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, 1986.

24. B. Shneiderman. Creating creativity: user interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction*, 7(1):114–138, 2000.

25. D. Smith, C. Irby, R. Kimball, W. Verplank, and E. Harslem. Designing the Star user interface. *Byte*, 7(4):242–282, 1982.

26. W. Stuerzlinger, O. Chapuis, D. Phillips, and N. Roussel. User Interface Façades: Towards Fully Adaptable User Interfaces. In *Proceedings of UIST'06*, pages 309–318. ACM Press, 2006.

27. J. Stylos, B. A. Myers, and A. Faulring. Citrine: providing intelligent copy-and-paste. In *Proceedings of UIST '04*, pages 185–188. ACM Press, 2004.

28. I. Sutherland. *Sketchpad, a man-machine graphical communication system*. PhD thesis, Massachusetts Institute of Technology (USA), 1963.

29. R. Sweet. The MESA programming environment. In *Proceedings of the ACM SIGPLAN 85 symposium on Language issues in programming environments*, pages 216–229. ACM Press, 1985.

30. W. W. Tryon. Evaluating statistical difference, equivalence, and indeterminacy using inferential confidence intervals: An integrated alternative method of conducting null hypothesis statistical tests. *Psychological Methods*, 6:371–386, 2001.

31. G. Wallace, B. Robert, and E. Tempero. Smarter cut-and-paste for programming text editors. In *Proceedings of AUIC '01*, pages 56–63. IEEE, 2001.