Push-and-Pull Switching: Window Switching based on Window Overlapping

Quan Xu and Géry Casiez
LIFL & INRIA Lille
University of Lille, FRANCE
{quan.xu, gery.casiez}@lifl.fr

ABSTRACT

We propose Push-and-Pull Switching, a window switching technique using window overlapping to implicitly define groups. Push-and-Pull Switching enables switching between groups and restacking the focused window to any position to change its group membership. The technique was evaluated in an experiment which found that Push-and-Pull Switching improves switching performance by more than 50% compared to other switching techniques in different scenarios. A longitudinal user study indicates that participants invoked this switching technique 15% of the time on single monitor displays and that they found it easy to understand and use.

Author Keywords

Window management, window switching, overlapping.

ACM Classification Keywords

H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General Terms

Design, Performance.

INTRODUCTION AND PREVIOUS WORK

Window switching is one of the most frequent tasks of any window manager and can occur several hundred times per day (on average, once every 20.9s on large displays [2]). Window switching includes two subtasks: first, finding the window of interest, and second, bringing it to the foreground. Unlike other operations on windows like moving and resizing that do not vary much across window managers, different techniques exist for window switching. The most common techniques are *direct pointing* using a mouse to click on a region of the window of interest, *Alt+Tab* / *Cmd+Tab* which consists of a key combination to navigate the list of windows or applications, *taskbar*/*dock* which provides a representation of the windows or applications at the bottom of the display with icons and text, and *Exposé* which tiles all opened windows so that they are visible at once.

CHI 2010, April 10 – 15, 2010, Atlanta, Georgia, USA

These techniques allow one to directly select the window of interest by clicking on the window itself or one of its representations (*direct pointing*, *Alt+Tab*, *taskbar* and *Exposé*) or first select a group of windows (*Cmd+Tab* and *dock*) and then select the window of interest. For the two latter techniques on Mac OS X, groups are defined by the windows belonging to the same application. However according to Robertson et al. grouping windows by application confuses many users because application windows may not be related to the same task [4]. Virtual desktops alleviate these problems by allowing users to explicitly define groups of windows, but at the cost of a strict separation between them. In contrast Scalable Fabric [4] allows interaction with windows from multiple groups at once without affecting the group structure. However the users have to plan in advance the number of groups needed to accomplish their tasks, and add each window to a group to leverage the benefit of the window grouping system. WindowScape [5] automatically creates groups by taking photograph-like snapshots each time a window is expanded or miniaturized. However, when a user wants to resume a group, it may no longer be visible or the user may have to explicitly define favorite snapshots.

Our work builds upon the informal observation that users try to keep windows belonging to the same activity visible by trying to avoid or minimize the amount they overlap. As a result groups are implicitly created by the user. This is also supported by Hutchings and Stasko who showed that a significant group of users tend to have many windows visible simultaneously [3]. The following scenario illustrates such a situation: *Peter edits code using a text editor and uses a terminal window to compile and test his program. The terminal window is positioned to avoid occlusion of any line of code displayed in the editor. He uses a third maximized window to repeatedly search information on the Internet.*

Switching back and forth between the group represented by the editor and terminal windows and the Internet window is time consuming and error prone with *direct pointing*, *Alt+Tab*/*Cmd+Tab*, *taskbar* or *Exposé* techniques. Peter could use a virtual desktop to explicitly assign the web browser window to one group, and the editor and terminal windows to a second group. However, he does not want a strict separation between these groups because he wants to be able to read some information on the web browser window while editing his code. He could use Scalable Fabric or WindowScape to display the two groups at once but he

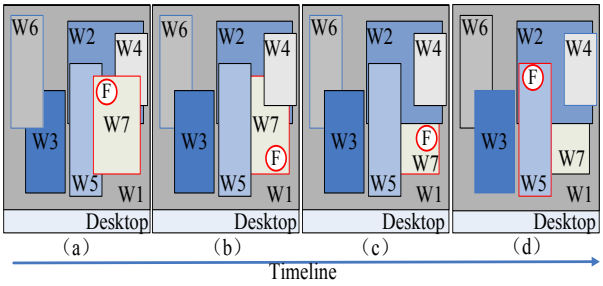

Figure 1. Push-and-Pull Switching example representing an initial layout (a) with window W7 focused (represented by the letter F). Windows are numbered according to their stacking order. Pressing the Ctrl key computes the following groups: (W6, W7), (W3, W4, W5), (W2) and (W1). Pushing one time the frontmost group moves all its windows behind the ones from the second group while respecting the relative stacking order within each group (b). Pushing one more time moves the group behind the third one (c). Releasing the Ctrl key gives the keyboard focus to the window with the highest Z order (d).

might not want to explicitly manage groups or use up more display space required to operate these techniques.

To address these limitations, we designed Push-and-Pull Switching, a switching technique based on window overlapping to implicitly define groups and help users quickly switch between them. After giving an overview of the technique, we present an experiment comparing Push-and-Pull Switching to other techniques in different scenarios. Finally we present the results of a longitudinal user study.

PUSH-AND-PULL SWITCHING

Group Switching

When invoked, our algorithm first creates groups of non overlapping windows automatically. Groups are created by considering windows in decreasing Z order, from foreground to background, and creating a new group each time a window overlaps with one of the windows of the current group. This algorithm is similar to the stack leafing algorithm proposed by Faure *et al.* [1] to facilitate drag-and-drop between overlapping windows. In addition, our algorithm uses a configurable *allowable overlap* parameter when considering whether a window will be included in the current group or not. The overlap for a given window is computed as the percentage of pixels which are covered by the rest of the group. Preliminary tests helped us to adjust the default overlap threshold to 15%.

Push-And-Pull Switching is invoked using keyboard shortcuts or the mouse wheel. For keyboard shortcuts, we use Ctrl+↑ to push a group and Ctrl+↓ to pull a group. Pressing the Ctrl key and rotating the mouse wheel forward pushes a group and rotating backward pulls a group. A press on the Ctrl key initiates the above algorithm to create groups. Pushing and pulling is accomplished by swapping all the windows from one group to the other while preserving the relative Z order within each group. Pulling a group brings all windows within the group closer to the foreground. Pushing a group does the opposite. During push and pull operations, only the first group which was created (closer to foreground) can be pushed or pulled (Figure 1). Once the Ctrl key is released, the window with the highest Z order receives the keyboard focus. We chose to give the keyboard focus to that window since it was the last one accessed within that group, so we consider the user more likely to interact with it.

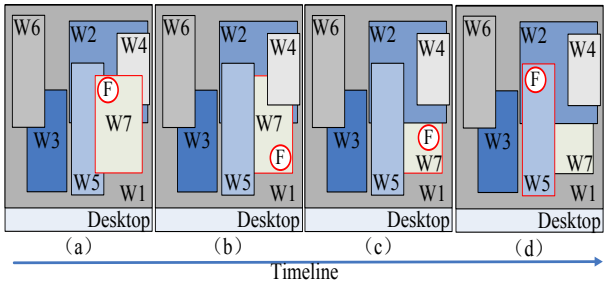

Figure 2. Example for restacking the focused window. Figure (a) represents the initial layout in which window W7 is focused (represented by the letter F) and where windows are numbered according to their stacking order. Pressing Ctrl+Shift computes the following groups for the windows intersecting window W7: (W4, W5), (W2), (W1). Pushing one time moves window W7 behind the first group (b) and pushing one more time moves it behind window W2 (c). Releasing the Crtl and Shift keys gives keyboard focus to window W5 (d).

Restacking the Focused Window

Push-and-Pull Switching can also be used to change the Z order of the focused window. When invoked using the Ctrl+Shift keys instead of Ctrl, our algorithm creates groups by considering only the windows which intersect with the focused window (if we considered all windows, pushing or pulling operations would have no visible effect). Thus, pushing or pulling moves the window in front or behind the related group. Releasing Ctrl+Shift gives the keyboard focus to the window with the highest Z order in the frontmost group (Figure 2).

Sending a window to the back of the Z order is a feature proposed by some X window managers and Windows (using Alt+Esc keys). Note that no modern window manager enables this type of precise setting of the Z order of a window. This technique can be used to add a window to another group. It could also be used for restacking a window back to its original position without modifying the Z order of the other windows. A typical example is to restack an instant messaging window after chatting and return to previous work.

EXPERIMENTS

The Push-And-Pull Switching technique was implemented in C# on Windows XP/Vista[1]. We wished to compare the performance of Push-and-Pull Switching to other techniques in different scenarios.

Experiment 1: Group Switching

We used a PC running Microsoft Windows XP using a 22 inch LCD monitor with a 1680 × 1050 resolution. The task was to switch back and forth between windows presented in different scenarios. Each trial started with an initial layout (Figure 3). After pressing the space bar, the participant was asked to switch to a specific layout, switch back to the initial

[1] `http://code.google.com/p/push-and-pull-switching`

layout, and then press the space bar again to end the trial. To help participants, the initial and target layouts for each scenario were printed on a paper positioned under the screen. Participants had to successfully reach the target layout and successfully return to the initial one before moving to the next trial. The experimenter warned the participants when a wrong layout occurred but gave no indication how to correct it.

8 people (5 male, 3 female) with a mean age of 27 (SD=2.2) participated. They were recruited from the computer science department and said they spent at least 8 hours a day working on a Microsoft Windows system. Most participants reported that they mainly use *direct access* and *taskbar* (with the group by application option disabled) and three reported that they often use *Alt+Tab*.

A repeated measure design was used. The independent variables were switching technique (SWT) with the three switching techniques available on Windows XP (*Taskbar*, *Alt+Tab*, *Direct pointing*) and the *Push-and-Pull Switching* technique, and SCENARIO with four levels. The first scenario consists of switching back and forth between window W7 and window W8 in Figure 3a. In the second scenario, participants were asked to switch back and forth between the group represented by windows (W7, W8) and window W6 in Figure 3b. The third scenario consists in switching back and forth between the group represented by windows (W6, W7, W8) and window W5 in Figure 3c. In the fourth scenario, participants were asked to switch back and forth between the groups represented by windows (W7, W8) and windows (W5, W6) in Figure 3d. Each scenario consists of 8 windows. Note that the scenarios were composed of real Windows applications arranged in layouts corresponding to realistic activities. For each application, we chose documents that participants could easily distinguish.

Participants were asked to complete each scenario 10 times with all four switching techniques before moving to the next one. The scenarios were run from *a* to *d* but the techniques

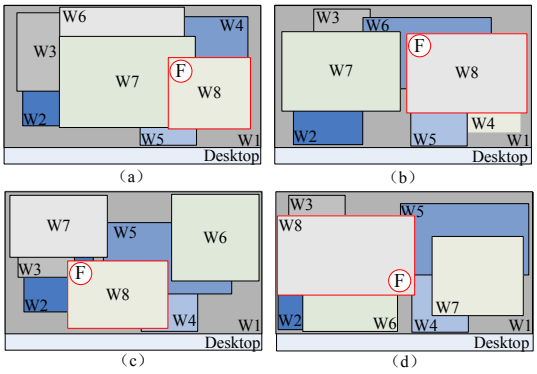

Figure 3. The initial layout for the four scenarios used in experiment 1 to compare the switching time between *Taskbar*, *Alt+Tab*, *Direct pointing* and *Push-And-Pull Switching*. The letter F represents the focused window. Windows are numbered according to their stacking order. Window numbers were replaced by real application windows in the experiment.

were counter-balanced across participants using a balanced Latin square. Before starting the experiment, participants had a 5-10 minute training period to get used to the switching techniques and windows content. Before each scenario, the experimenter clearly explained which layouts the participant had to switch between. The experiment lasted approximately 25 minutes.

Results

The dependent variable is switching time, the time measured between two presses on the space bar. Repeated measures analyses of variance showed a significant main effect for SWT ($F_{3,21}$=13.5, $p<0.001$) and SCENARIO ($F_{3,21}$=28.6, $p<0.001$) on switching time. More interestingly we also found a significant interaction between SWT and SCENARIO ($F_{9,63}$=7.5, $p<0.001$). Pairwise comparisons found a significant difference between *Taskbar* and all other techniques ($p<0.02$) for scenario *a* with *Taskbar* being 45% slower on average. For scenario *b*, *c* and *d* we observed significant differences ($p<0.03$) between *Push-and-Pull Switching* and the other techniques although the difference is marginal ($p<0.08$) with *Direct pointing* for scenarios *c* and *d*. On average *Push-and-Pull switching* is 50% faster than *Direct pointing* and *Taskbar*, and 70% faster than *Alt+Tab* for these three scenarios.

During the experiment we observed that participants were more error prone with *Alt+Tab* which can explain the more lengthly switching time for this technique. In fact, participants did not use the Shift key to move back in the window list when they missed the target window, and preferred to iterate through the entire list instead. The error rate for *Alt+Tab* is 5%, *Taskbar* 2%, *Direct pointing* 1% and *Push-and-Pull Switching* 0%. Overall, participants had no problem understanding and using *Push-and-Pull Switching*.

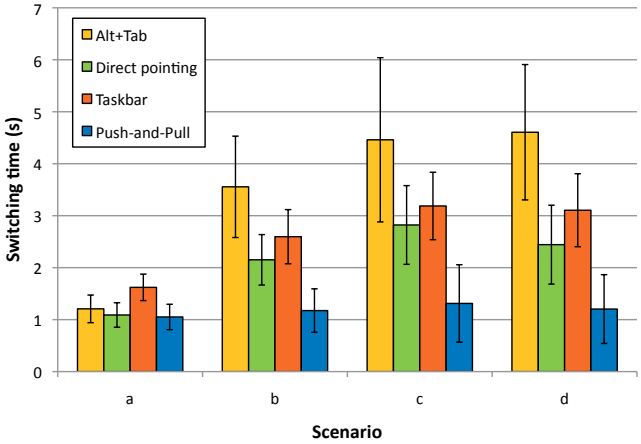

Figure 4. Mean switching time for SWITCHING TECHNIQUE and SCENARIO. Error bars represent 95% confidence interval.

In this experiment, we focused on *Push-and-Pull Switching* performance in different scenarios where the implicit creation of groups based on window overlapping is realistic. The results confirm that *Push-and-Pull Switching* can significantly improve switching time compared to other techniques when users switch between groups containing two or more windows.

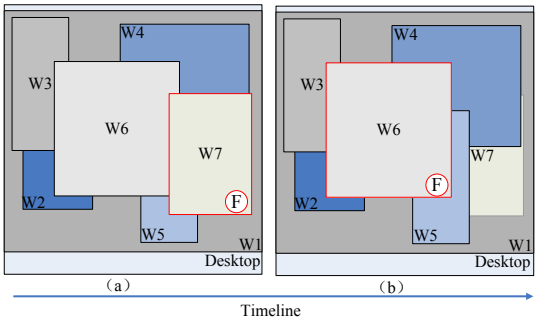

Figure 5. Window layout used in the second experiment with the initial layout on the left (a) and the target layout on the right (b). The letter F represents the window in focus. Windows are numbered following their stacking order. Window numbers were replaced by real application windows in the experiment.

Experiment 2: Restacking the Focused Window

This second experiment was run after the first experiment with the same participants and the same hardware configuration. The task was to change the Z order of the focused window represented in Figure 5a to create the layout represented in Figure 5b. As before, a trial started and finished by pressing the space bar and participants had to successfully create the target layout before moving to the next trial. The task was repeated 10 times.

A repeated measure design was used. The independent variables were switching technique SWT with the three switching techniques available on Windows XP (*Taskbar*, *Alt+Tab*, *Direct pointing*) and the *Push-and-Pull Switching* technique. The techniques were counterbalanced across participants and we used the same applications as in the first experiment. The experiment lasted approximately 5 minutes.

Results

As before, the dependent variable is restacking time which is the time measured between the two presses on the space bar. Repeated measures analyses of variance showed a significant main effect for SWT ($F_{3,21}$=5.11, p=0.008) on restacking time. Pairwise comparisons found significant differences between *Push-and-Pull* (1.4s), *Alt+Tab* (3.0s) (p=0.026), and *Taskbar* (2.4s) (all p<0.001). The difference observed with *Direct pointing* (2.1s) is marginal (p=0.07). *Push-and-Pull switching* reduces restacking time by 52% compared to *Alt+Tab* and 40% compared to *Taskbar*. All participants reported that they found *Push-and-Pull switching* to be the most direct and intuitive technique and most commented how the technique mimics the way in which people classify files in piles of documents.

LONGITUDINAL USER STUDY

In order to understand how people actually use the Push-And-Pull Switching technique, we performed a longitudinal field study on a small number of participants over one week.

8 people (7 male, 1 female), aged between 24 and 31, participated in the study. There were 1 civil engineer, 1 mechanic, 1 electronic engineer and 5 computer scientists. Half of the participants used a single monitor and the other half two monitors. Participants were instructed how to use the Push-And-Pull Switching technique and were given an executable. After one week, participants were interviewed to collect details and comments about how they utilized Push-And-Pull Switching and whether they found it useful. In addition, the application logged all switching operations and the corresponding technique used.

Single monitor users had, on average, 5 windows opened simultaneously on their desktop. They mainly used *Direct pointing* (47%) and *Taskbar* (36%) wheras *Alt+Tab* was used 2% of the time and *Push-and-Pull Switching* 15%. Dual monitor users had, on average, 8 windows opened simultaneously on their desktop. They mainly used *Direct pointing* (64%) and *Taskbar* (26%) while *Alt+Tab* was used 3% of the time and *Push-and-Pull* 7%. The restack of the focused window represented 10% of *Push-and-Pull Switching* invocations.

Participants reported that they mainly use *Push-and-Pull Switching* when they want to keep two or more windows grouped. They also use the restack functionality as a replacement for *Alt+Tab* when they want to access a window which they know is just behind another. Using a 5 point Likert scale, participants rated Push-and-Pull Switching as useful (averaged response = 3.9) (1=disagree, 5=agree) and easy to use (4.1). Half of the participants (mostly single monitor users) reported that they reposition and resize windows more frequently than usual to leverage full benefit from *Push-and-Pull Switching*.

CONCLUSION

Push-and-Pull Switching provides a lightweight alternative to other grouping techniques. We demonstrated with a longitudinal user study and two formal experiments that the definition of groups based on windows overlapping constitutes a valid approach and helps to drastically reduce switching time compared to traditional switching techniques. Future work includes adding visual feedback for novice users to help visualize groups by changing the color of the window borders when windows are being moved.

ACKNOWLEDGEMENTS

We thank Nicolas Roussel for his wise comments on early revisions of the paper and Daniel Vogel for proofreading.

REFERENCES

1. G. Faure, O. Chapuis, and N. Roussel. Power tools for copying and moving: useful stuff for your desktop. In *Proc. CHI '09*, pages 1675–1678, 2009.
2. D. R. Hutchings, G. Smith, B. Meyers, M. Czerwinski, and G. Robertson. Display space usage and window management operation comparisons between single monitor and multiple monitor users. In *Proc. AVI '04*, pages 32–39, 2004.
3. D. R. Hutchings and J. Stasko. Revisiting display space management: understanding current practice to inform next-generation design. In *GI '04*, pages 127–134, 2004.
4. G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. R. Hutchings, B. Meyers, D. Robbins, and G. Smith. Scalable fabric: flexible task management. In *Proc. AVI '04*, pages 85–89, 2004.
5. C. Tashman. Windowscape: a task oriented window manager. In *Proc. UIST '06*, pages 77–80, 2006.