Contribution à l’étude et au développement de techniques de gestion de fenêtres
Xu Quan

To cite this version:

HAL Id: tel-00671564
https://tel.archives-ouvertes.fr/tel-00671564
Submitted on 17 Feb 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Contribution à l’étude et au développement de techniques de gestion de fenêtres

THÈSE

présentée et soutenue publiquement le 15 décembre 2010

pour l’obtention du

Doctorat de l’Université des Sciences et Technologies de Lille
(spécialité informatique)

par

Xu Quan

Composition du jury

Président : XXX

Directeurs de thèse : Pr. Christophe Chaillou
Dr. Géry Casiez

Rapporteurs : Pr. Karin Coninx
Dr. Emmanuel Dubois

Examinateur : Dr. Olivier Chapuis
To my family and friends.
Abstract

Window switching is an integral part of our daily computing experience since computers allow multi-tasking. Window switching includes two subtasks: first, finding the window of interest, and second, bringing it to the foreground. With the advent and ubiquity of graphical user interfaces and the desktop metaphor, window switching has become one of the most common operations performed on a computer. Window switching is also one of the most frequent tasks of any window manager happening several hundred times per day. However this task can become laborious when the number of windows becomes important. The difficulties for finding the desired window come from two main reasons: 1) window overlapping, which hides window information; and 2) the visual similarity of windows. Yet the different techniques provided by modern overlapping window management systems such as Alt+Tab/Cmd+Tab, Taskbar/dock and Exposé only allow to switch between windows or applications and user studies have revealed their usability problems.

Furthermore, to reduce the number of switching operations, interaction techniques have been proposed to explicitly or implicitly define groups. Whether explicitly or implicitly defined, these grouping mechanisms have revealed usability problems. For explicitly defined groups, users may find it burdensome to explicitly classify windows into tasks and even may be hard-pressed to decide on an appropriate classification for each window. For automatically defined groups, the system can incorrectly infer groups and create groups that may not correspond to users expectations. These limitations have restricted their use.

To address these limitations, this dissertation aims at understanding and developing new window switching techniques to help users to improve task
switching. In order to understand how users manage their windows, we developed a tool we called WindowsOSLog to log user window management activity in mainstream Windows operating system by means of Windows messages and hooks. 26 participants participated in this study duration over 5-weeks. We also collected subjective data from participants by means of questionnaires.

Three window switching techniques: Push-and-Pull Switching, stack scanning and WindowsTagging were designed and developed based on the results of this data.

First, Push-and-Pull Switching, a window switching technique using window overlapping to implicitly define groups. Push-and-Pull Switching further allows switching between groups and restacking the focused window to any position to change its group affectation. The technique was evaluated in an experiment showing that Push-and-Pull Switching allows improving 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 while they found it easy to understand and use.

Second, stack scanning, a window switching technique based on a widget that combines generalized scrolling and crossing to control the stack order of layers of visible windows. The empirical evaluations showed that stack scanning was faster than other techniques when the number of windows is high and the visual similarity among windows is important. They also showed that Taskbar was the best choice when the number of windows is small, regardless of other visual factors conditions, and for users who always maximized each window, Alt+Tab was the best choice when the number of windows is important.

Finally, to theorize window/group switching, we defined types of switching operations and introduced window tagging mechanism to provide a new alternative to the existing windows grouping techniques, and then provided eleven design principles to help designers to design new switching techniques. WindowsTagging was implemented as a prototype system based on these design principles. It combines implicit and explicit definition of groups, spatial and visual memories to help users to quickly find windows or groups and switch.
between them. *WindowsTagging* represents windows as thumbnails, displaying all groups at the same time and allowing windows to exist simultaneously in multiple groups. *WindowsTagging* exploits users’ spatial and visual memories by providing a relative stable spatial layout with larger thumbnails using overlapping. Groups are first automatically created based on a window overlapping algorithm. *Window tagging* mechanism was used to allow users to explicitly create or modify groups by tags. The *Drag-and-drop* and *splitting group* functions also allow users to explicitly define or modify groups. A search function was presented to help users finding a window by its title or tags. An experiment was designed to compare the performance and error rate of *WindowsTagging* to *Exposé*. Results showed that *WindowsTagging* was faster than *Exposé* technique, and participants strongly preferred it.

To summarize, this dissertation contributes a literature review of related work in the domain of window and group switching; a longitudinal study to understand users window management activities; providing eleven design principles and introducing window tagging mechanism to help designers design new window switching techniques; and three window/group switching techniques: *Push-and-Pull Switching*, *stack scanning* and *WindowsTagging*. 
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nomenclature</td>
<td></td>
<td>xix</td>
</tr>
<tr>
<td>1 Introduction</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Problem Statement And Research Goal</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Research Approach</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Research Contributions</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Structure of the Dissertation</td>
<td>6</td>
</tr>
<tr>
<td>2 Related Work</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Introduction</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Window Switching Techniques</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Temporal Approach</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1.1</td>
<td>Alt+Tab</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1.2</td>
<td>RelAltTab</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Spatial Approach</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2.1</td>
<td>Taskbar/Dock</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2.2</td>
<td>Exposé</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2.3</td>
<td>EyeExposé</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2.4</td>
<td>Taskposé</td>
<td>16</td>
</tr>
<tr>
<td>2.2.2.5</td>
<td>SCOTZ</td>
<td>17</td>
</tr>
<tr>
<td>2.2.2.6</td>
<td>FST</td>
<td>18</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Hybrid Approach</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>Group Switching Techniques</td>
<td>19</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Explicit Window Grouping Techniques</td>
<td>20</td>
</tr>
<tr>
<td>2.3.1.1</td>
<td>Virtual Desktop Managers</td>
<td>20</td>
</tr>
</tbody>
</table>
CONTENTS

2.3.1.2  GroupBar  .............................................. 20
2.3.1.3  Elastic Windows .......................................... 21
2.3.1.4  Scalable Fabric ........................................... 22
2.3.1.5  SCWM ..................................................... 22
2.3.2  Implicit Window Grouping Techniques ............................... 23
2.3.2.1  WindowScape ............................................. 23
2.3.2.2  Stack leafing ............................................. 23
2.3.2.3  SWISH .................................................... 24
2.4  Tabs Switching Techniques ........................................ 24
2.5  Desktop Organization and Switching Techniques ....................... 25
2.6  Spatial Memory And Visual Search .................................. 26
2.6.1  Psychology of Visual Search and Memory ......................... 26
2.6.2  Spatial Memory and User Interfaces .............................. 27
2.6.3  Visual Search ............................................... 28
2.7  Finding obscured Windows ......................................... 29
2.8  Switching Techniques Time Model ................................... 29
2.8.1  Hick-Hyman And Fitt’s Laws .................................. 30
2.8.2  GOMS/KLM .................................................. 31
2.9  Log-based Empirical Methods ...................................... 32
2.10  Tags .......................................................... 33
2.11  Summary ...................................................... 34
2.11.1  Issues Associated With Log-based Empirical Methods on Window
        Switching ......................................................... 35
2.11.2  Issues Associated With Switching Techniques Evaluations ....... 35
2.11.3  Leveraging Natural Human Capabilities .......................... 35
2.11.4  The Combination of Implicit and Explicit Grouping Techniques Are
        The Best Way ..................................................... 36
2.11.5  Window Tagging ............................................... 36

3  Log-Based Longitudinal Study .......................................... 37
3.1  Introduction ...................................................... 37
3.2  Experiment Objectives ............................................ 38
3.2.1  Quantitative Goals ............................................ 38
CONTENTS

3.2.2 Qualitative Goals .................................................. 39

3.3 Longitudinal Study .................................................... 40
3.3.1 Definition ............................................................ 40
3.3.2 Participants And Apparatus ....................................... 41
3.3.3 Design ............................................................... 42
    3.3.3.1 WindowsOSLog .............................................. 42
    3.3.3.2 Log Information ........................................... 42
    3.3.3.3 Information Incompleteness ................................ 45

3.4 Results And Analysis ................................................ 45
3.4.1 Number of windows on the desktop ................................ 47
3.4.2 Window Event ........................................................ 48
3.4.3 Window Switching Techniques ...................................... 50
    3.4.3.1 Window Switching Techniques Used to Switch Windows . 50
    3.4.3.2 Cost of Error ............................................... 53
    3.4.3.3 Types of Switching ........................................ 54
    3.4.3.4 Tabbed Windows vs. Group Windows ...................... 55

3.4.4 Window Visibility .................................................. 56
    3.4.4.1 Number of Visible Windows ................................ 56
    3.4.4.2 Visible Windows vs. Window Switching Techniques .......... 57

3.4.5 Spatial Memory ..................................................... 58

3.4.6 Windows Layout .................................................... 59
    3.4.6.1 The Distribution of Window Size .......................... 60
    3.4.6.2 Windows Group ............................................. 60

3.4.7 TDI and MDI Applications ......................................... 63
    3.4.7.1 The Number of Tabs/Documents ............................ 66
    3.4.7.2 Switching Between Tabs/Documents ........................ 67

3.4.8 Active Window Sequences .......................................... 67

3.5 Conclusion ............................................................ 71

4 Push-and-Pull Switching: Window Switching based on Window Overlapping ........................................ 72
4.1 Introduction ........................................................... 72
4.2 Push-and-Pull Switching .............................................. 75
CONTENTS

4.2.1 Group Switching .................................... 75
4.2.2 Restacking the Focused Window ....................... 77
4.3 Experiments ........................................... 78
  4.3.1 Experiment 1: Group Switching ....................... 78
     4.3.1.1 Apparatus ................................... 78
     4.3.1.2 Participants ................................. 78
     4.3.1.3 Experimental Design ......................... 78
     4.3.1.4 Procedure .................................. 80
     4.3.1.5 Results .................................... 80
  4.3.2 Experiment 2: Restacking the Focused Window .... 82
     4.3.2.1 Apparatus and Participants .................. 82
     4.3.2.2 Experimental Design ......................... 82
     4.3.2.3 Procedure .................................. 82
     4.3.2.4 Results .................................... 82
4.4 Longitudinal User Study ................................ 83
4.5 Application ........................................... 85
4.6 Conclusion ............................................ 86

5 Stack Scanning Rules! .................................. 87
  5.1 Introduction ......................................... 87
  5.2 Stack Scanning ...................................... 89
  5.3 Window Switching Time Model ........................ 93
  5.4 Experiment .......................................... 95
     5.4.1 Visual Factors For Window Switching ............ 95
     5.4.2 Hypothesis .................................... 97
     5.4.3 Apparatus .................................... 97
     5.4.4 Participants .................................. 98
     5.4.5 Experimental Design ........................... 98
     5.4.6 Procedure .................................... 100
  5.5 Results ............................................... 100
     5.5.1 Switching Time ................................ 101
        5.5.1.1 Main Experiment .......................... 101
        5.5.1.2 Second Experiment ....................... 107
CONTENTS

6.4.3 Participants ................................................. 137
6.4.4 Experimental Design ...................................... 138
6.4.5 Procedure ................................................. 139
6.4.6 Results ................................................... 140
  6.4.6.1 Switching Time ................................... 140
  6.4.6.2 Error Rate ........................................ 142
  6.4.6.3 Qualitative Results ................................. 142
6.5 User Evaluation .............................................. 143
  6.5.1 Using Step-by-Step .................................. 144
  6.5.2 Results ................................................. 145
6.6 Discussion .................................................. 146
  6.6.1 Overlapping Mechanism ............................... 146
  6.6.2 Grouping Mechanism ................................. 146
  6.6.3 Search Mechanism .................................... 148
  6.6.4 Stable Spatial Layout strategy ..................... 148
6.7 Conclusion .................................................. 148
6.8 Future Work ................................................ 149

7 Conclusion And Future Work ................................. 150
  7.1 Research Objectives ..................................... 150
  7.2 Conclusion ................................................. 153
  7.3 Future Work .............................................. 155
    7.3.1 Window Tagging .................................... 155
    7.3.2 Theorize Window Switching ..................... 155
    7.3.3 Implicit Grouping Techniques ................... 156
    7.3.4 A Standardized Evaluation Framework .......... 156

References .................................................... 168
List of Figures

2.1 Alt+Tab dialog in Windows XP showing icons for opened windows. A title is displayed for only one icon at any time, and the same icon may appear again for each application window opened. ........................................... 11

2.2 Windows Vista or 7 Alt+Tab. It shows running application names along with their live thumbnails so that you can see the window content before switching to them. It also shows desktop in the list so that you can directly access desktop icons without minimizing all running applications manually. 11

2.3 Windows Flip 3D renders live thumbnail images of the exact contents of the opened windows. It allows users to switch between windows while dynamically displaying them in a 3D view. ................................. 12

2.4 RelAltTab includes two types of background coloring: salmon for the semantically related windows and light-blue for the temporally related windows. It also displays a red number on top of each semantically related window. This number allows the user to directly switch to the window by pressing Alt + NUMBER. ......................................................... 13

2.5 Ungrouped window buttons on the Taskbar ................................. 13

2.6 Grouped application buttons on the Taskbar ............................... 14

2.7 Dock tool in Mac OS, which is used to launch applications, and switch between running applications. ................................. 14

2.8 Visual TaskTip, it displays a thumbnail preview of the opened windows when the mouse is hovered over Taskbar. ................................. 15

2.9 Exposé view of open applications. ........................................... 15
LIST OF FIGURES

2.10 The Taskposé visualization arranges open windows in two dimensions when
the visualization is called up. Windows automatically size relative to their
importance, and closely-related windows appear together. .......................... 16
2.11 SCOTZ ........................................................................................................... 18
2.12 Size morphing ............................................................................................... 18
2.13 GroupBar is a prototype for demonstrating the use of window-grouping
features in an Windows XP TaskBar-like interface. ........................................... 21
2.14 Scalable Fabric showing the representation of three tasks as clusters of
windows, and a single window being dragged from the focus area into the
periphery. .............................................................................................................. 22
2.15 WindowScape represents windows as thumbnails and uses the timeline of
desktop states shown as a series of photograph-like snapshots. ..................... 24

3.1 Example box-and-whisker chart. ................................................................. 46
3.2 The number of opened windows on the desktop by each participant (some
participants did not close down their computers, so some minimum values
for the number of windows are not zero). ............................................................ 47
3.3 Cross participant means of the percentage of windows Event: Create, Ac-
tive and Destroy. Error bars show standard error (the standard error is too
small for dual monitors, so it may not display on the chart). ............................. 49
3.4 Cross participant means of the percentage of window move and maxi-
mize/minimize event in window activation event. Error bars show standard
error. ...................................................................................................................... 50
3.5 The percentage of window switching activated by Direct pointing, Taskbar,
Alt+Tab and Alt+Esc for 14 single monitor participants. ................................. 51
3.6 The percentage of window switching activated by Direct pointing, Taskbar,
Alt+Tab and Alt+Esc for 12 dual monitors participants. ................................. 51
3.7 The number of press of the tab key when using Alt+Tab switching technique
(because some values (min, lower quartile, median) are equal, so it results
in some degenerated charts). ........................................................................... 53
3.8 Cross participant means of the percentage of window switching activated by
Direct pointing, Taskbar, Alt+Tab and Alt+Esc. Error bars show standard
error. ...................................................................................................................... 54
3.9 Group Windows on Taskbar ................................................. 55
3.10 An example of tabbed windows technique .............................. 56
3.11 The number of visible windows kept on the desktop by each participant .............................. 57
3.12 Mean of the percentage of window switching techniques to switch windows when they are visible ................................................. 58
3.13 Mean of the percentage of window switching techniques to switch windows when they are invisible ................................................. 59
3.14 Mean of the percentage of SmallWindow, NormalWindow and LargeWindow accounts for the proportion of the total number of windows for single monitor users. Error bars show standard error. ................................................. 61
3.15 Mean of the percentage of SmallWindow, NormalWindow and LargeWindow accounts for the proportion of the total number of windows for dual monitors users. Error bars show standard error. ................................................. 61
3.16 The number of groups of single monitor users when the overlapping threshold is set to 25% ................................................. 64
3.17 The number of groups of dual monitors users when the overlapping threshold is set to 25% ................................................. 65
3.18 Spy++ displays the tabs/documents information of applications ................................................. 66
3.19 Mean of the percentage of three types interaction pattern under NTChild condition for each single monitor user. Error bars show standard error. ................................................. 69
3.20 Mean of the percentage of three types interaction pattern under YTChild condition for each dual monitors user. Error bars show standard error. ................................................. 70
3.21 Mean of the percentage of three types interaction pattern under NTChild condition. Error bars show standard error. ................................................. 70

4.1 A typical windows organization with two closely related windows in the foreground. To switch focus to the navigator window in the background and then give back the focus to the two related windows is a tedious task requiring multiple switching operation with current user interface switching techniques. ................................................. 74
4.2 Push-and-Pull Switching example representing an initial layout (a) with window W7 active (represented by the letter A). 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). ................................................................. 75

4.3 Example for restacking the focused window. Figure (a) represents the initial layout in which window W7 is active (represented by the letter A) 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 Ctrl and Shift keys activates window W5 (d). 77

4.4 The initial layout for the 4 scenarios used in experiment one to compare the switching time between Taskbar, Alt+Tab, Direct pointing and Push-and-Pull Switching. The letter A represents the active window. Windows are numbered according to their stacking order. Window numbers were replaced by real application windows in the experiment. 79

4.5 Mean switching time for SWITCHING TECHNIQUE and SCENARIO. Error bars represent 95% confidence interval. 81

4.6 Windows layout used in the second experiment with the initial layout on the left and the target layout on the right. The letter A represents the window in focus. Windows are numbered following their stacking order. Window numbers were replaced by real application windows in the experiment. . . . 83

4.7 Mean switching time for SWITCHING TECHNIQUE. Error bars represent 95% confidence interval. 84

4.8 Typical mouse with five keys, there are two shortcut keys in the right side of the mouse. In the default state, they are used to implement forward and backward operations. 86
5.1 Example for stack scanning. Figure (a) represents the initial layout with window w5 focused (represented by the letter F). Windows are numbered according to their stacking order. Pressing the mouse wheel (250 ms timeout) computes the following layers: (W5, W4), (W3) and (W2, W1). Moving the mouse to (W2, W1) layer and leaving the widget, all windows within this layer are brought closer to the foreground, the window with the highest Z-order receives the keyboard focus.

5.2 Example of stack scanning extend function. Figure (a) represents an initial layout with window W5 is focused (represented by the letter F) and where windows are numbered according to their stacking order, pressing the mouse wheel, bringing up the widget. Moving the mouse to cross the button, bring all windows within the layer closer to the foreground (b), the mouse leaves from the left side of widget, only bringing the clicked window within the layer to the foreground (c), the mouse leaves from the right side of widget, bringing all windows within the layer closer to the foreground (d).

5.3 Stack scanning with 7 layers, using the original windows as representation.

5.4 Exposé view of 8 windows, representing them as thumbnails.

5.5 Alt+Tab represents windows as icons.

5.6 Taskbar in each of the 8, 12 and 16 windows conditions with NVS, representing windows as buttons.

5.7 The visual similarity between (1) and (2) is NVS, (1) and (3) is LVS and (1) and (4) is HVS.

5.8 Mean switching time (ST) in s for NUM and TECHNIQUE. Error bars represent 95% confidence interval.

5.9 Mean switching time (ST) in s for VS and TECHNIQUE. Error bars represent 95% confidence interval.

5.10 Mean switching time (ST) in s for MLS and TECHNIQUE. Error bars represent 95% confidence interval.

5.11 Mean switching time (ST) in s for DisSize and TECHNIQUE.

5.12 Mean switching time (ST) in s for AMOVERLAP and TECHNIQUE.

5.13 Mean switching time(ST) in s for NUM and TECHNIQUE with the HVS. Error bars represent 95% confidence interval.
LIST OF FIGURES

5.14 Mean switching time (ST) in s for MLs and TECHNIQUE with the 16 windows. Error bars represent 95% confidence interval. .................................................. 107
5.15 Mean switching time (ST) in s for VS and TECHNIQUE with the 16 windows. Error bars represent 95% confidence interval. ................................. 108
5.16 Mean switching time (ST) in s for each TECHNIQUE, grouped by VS under the SIZE001 condition. Error bars represent 95% confidence interval. ... 109
5.17 Mean switching time (ST) in s for each TECHNIQUE, grouped by MLS for HVS under the SIZE001 condition. Error bars represent 95% confidence interval. ............................................................ 110
5.18 Error rate for each TECHNIQUE, grouped by NUM. Error bars represent 95% confidence interval. .......................................................... 111
5.19 Error rate for each TECHNIQUE, grouped by VS. Error bars represent 95% confidence interval. .................................................. 111

6.1 WindowsTagging presents windows and groups using thumbnails. Thumbnails are allowed to overlap within each group to identify groups more easily and improve thumbnails legibility. In this example all opened windows (16) were divided into ten groups. When moving the mouse, the group closest to the cursor is expanded to make them easy to recognize and the titles of all window within the group are displayed. ........................................ 122
6.2 A simple example to describe the differences between the creating group algorithm of Push-and-Pull switching and the new algorithm. For Push-and-Pull switching, computing the following groups: (W3, W1), (W2), and for the new algorithm, computing the following groups: (W3, W1), (W2, W1). .......................................................... 125
6.3 WindowsTagging’s context menu, tag item is expanded to allow users to select add/remove tag item. ........................................... 127
6.4 Tag Management Dialog. .................................................. 128
6.5 Example of group reorganization using *drag-and-drop*. Figure (a) represents the initial layout of groups. Moving a window to another group (b1 to e1) is done by pressing the left mouse button on the thumbnail window W (b1). Copying a window to another group (b2 to e2) is done by pressing the Ctrl key and holding it down with the left mouse button on the thumbnail window W (b2). The border color of the thumbnail window being moved or copied is continuously updated with the color of the group hovered (c1, d1; c2, d2). Releasing the left mouse button (or including the Ctrl key) resizes the dropped window to prevent overlapping with another group (e1, e2). .......................................................... 129

6.6 *Exposé* view of 12 windows, when moving mouse on the thumbnail of window, the window’s title is displayed. .......................................................... 136

6.7 *NSWindowsTagging* view of 12 windows, when moving mouse on the thumbnail group, all windows within the group are expanded and all windows’ title are displayed in columns from top to bottom by the window z-order. . 137

6.8 *SFWindowsTagging* showed the result that users entered ”ppt” in the text box, the search dialog located at the left-top (the red rectangle region). Each time a new character is entered or removed, the list of the windows corresponding to the search is updated and the windows that do not match are darken but their content remains visible through transparency. . . . . 138

6.9 Mean switching time (ST) in s for NUM and TECHNIQUE. Error bars represent 95% confidence interval. ................................. 142

6.10 Mean switching time (ST) in s for VS and TECHNIQUE. Error bars represent 95% confidence interval. ................................. 143

6.11 Error Rate for TECHNIQUE, grouped by NUM. Error bars represent 95% confidence interval. ................................. 144
# List of Tables

3.1 Participants Information .................................................. 43
3.2 Example of user configuration information. ............................. 44
3.3 Description of the WindowsOSLog output stream ........................ 45
3.4 Users opened more windows on large display for single monitor users (s.d. = standard deviation). ................................................. 48
3.5 Users opened more windows on dual monitor system than on the single monitor system (s.d. = standard deviation). ........................... 48
3.6 Users opened more windows on main monitor than on secondary monitor for dual monitor system users (s.d. = standard deviation). .................. 48
3.7 The average number of pressing tab when users used $Alt+Tab$ technique (s.d. = standard deviation). ................................................. 52
3.8 Users kept more windows visible simultaneously on dual monitor system than on the single monitor system (s.d. = standard deviation). ............................ 56
3.9 The distribution of window size. ............................................. 62
3.10 The mean of number of groups of all participants for each overlapping threshold condition (s.d. = standard deviation). ............................... 64
3.11 The mean of number of groups of main monitor and secondary monitor for dual monitor system users for each overlapping threshold condition (s.d. = standard deviation). ........................................... 65
3.12 The mean of number of tabs/documents in the TDI/MDI applications (s.d. = standard deviation). ................................................. 67
3.13 The mean of percentage of switching techniques for TDI/MDI applications (s.d. = standard deviation). ................................................. 67
4.1 User satisfaction averages for on a five point scale where 1 = useless, 5 = useful. ................................................. 85

5.1 Pairwise comparisons between techniques condition on switching time. A cell contains the means difference and the lower and upper bound of the confidence interval. Underlined cells are significant. ..................... 101

5.2 Pairwise comparisons between techniques condition on switching error rate. A cell contains the means difference and the lower and upper bound of the confidence interval. Underlined cells are significant. ..................... 112

6.1 Pairwise comparisons between techniques condition on switching time. A cell contains the means difference and the lower and upper bound of the confidence interval. Underlined cells are significant. ..................... 141

6.2 User satisfaction averages for on a five point scale where 1 = useless, 5 = useful. .................................................. 147
Chapter 1

Introduction

The personal computer (PC) has undergone dramatic changes over the past 30 years. The huge gains in processor speed and physical memory size and the large and multiple-monitor now allow people to use computers in an astonishing variety of ways. A key advantage of additional screen space is the ability to keep more windows open simultaneously, reducing the amount of resizing, repositioning, and other window-management activities (Hutchings has shown that users had more than eight windows open more than 78% of the time Hutchings et al. (2004)). However, additional windows also mean that more windows are competing for users’ attention D.Mackinlay & Royer (2007); Hutchings et al. (2004).

Window switching is an integral part of our daily computing experience since computers allow multi-tasking. Window switching includes two subtasks: first, finding the window of interest, and second, bringing it to the foreground. With the advent and ubiquity of graphical user interfaces and the desktop metaphor, window switching has become one of the most common operations performed on a computer. Window switching is also one of the most frequent tasks of any window manager happening several hundred times per day (takes place on average once every 20.9s on large displays Hutchings et al. (2004), the results were confirmed by D.Mackinlay & Royer (2007)). However this task can become laborious when the number of windows becomes important. The difficulties for finding the desired window come from two main reasons: 1) window overlapping, which hides window information; and 2) the visual similarity of windows.

Although tiled windows system may be more efficient for window switching Bly & Rosenberg (1986); Kandogan & Shneiderman (1996), the overlapping systems is the de
1. INTRODUCTION

facto standard for modern window management systems, and this situation will not dis-
appear with the advent of larger displays Chapuis & Roussel (2007); Hutchings & Stasko
(2004). With the increasing in the number of windows, it makes more difficult to switch
to the window of interest. Many works have been proposed to reduce the amount of over-
lapping between windows, including all tiling systems, leafing through stacked windows,
peeling them back Beaudouin-Lafon (2001), snip and snap Hutchings & Stasko (2007),
and Exposé Apple.

The importance and frequency of window switching has led to intensive research and
development into improving switching efficiency for window manager. There has been
many extensive researches in the area of window manager and task management Badros
et al. (2000); Beaudouin-Lafon (2001); Bell & Feiner (2000); D. Austin Henderson & Card
(1986); D.Mackinlay & Royer (2007); Dragunov et al. (2005); Faure et al. (2009); Kando-
gan & Shneiderman (1996); Robertson et al. (2000, 2004); Smith et al. (2003); Tak et al.
(2009a); Tashman (2006). However there has been little innovation in window switching.
Using the mouse to click on a region of the window of interest (Direct pointing, Xu &
Casiez (2010)) or using a key combination to navigate the list of windows or applications
(Alt+Tab/Cmd+Tab) or clicking on a representation of the windows or applications at
the bottom of the display with icons and text (Taskbar/dock) have been the de facto
standard for window switching for several years. Probably the most notable advance is
Exposé which tiles all opened windows or those of the focused application so that they are
all visible at once. However, as the number of opened windows increases, the legibility
of their content decreases, making them hard to distinguish (it solves the overlapping
problem but reduces windows to snapshot).

Many researches have hinted that traditional window switching techniques have po-
tential usability problems Czerwinski et al. (2003); Grudin (2001); Hutchings & Stasko
(2003). Another problem is that those window switching techniques have been used sev-
eral hundred times per day D.Mackinlay & Royer (2007); Hutchings et al. (2004) and
have been the de facto standard for window switching for several years, but we have not
found that they have been evaluated in any serious way. As a result, it is hard to assess
them whether they have affected the people experience on computers.

Furthermore, to reduce the number of switching operations, interaction techniques
have been proposed to explicitly or implicitly define groups (Kandogan & Shneiderman
(1996); Robertson et al. (2000, 2004); Smith et al. (2003); Tashman (2006)). Whether explicitly or implicitly defined, these grouping mechanisms have presented problems for users. For explicitly defined groups, users may find it burdensome to explicitly classify windows into tasks and even may be hard-pressed to decide on an appropriate classification for each window. For automatically defined groups, the system can incorrectly infer groups and create groups that may not correspond to users expectations. The limitation of grouping mechanism they provided has restricted their use.

A thorough characterization of users’ activities in real-world window management systems would significantly benefit interaction designers when developing new window switching techniques. It would provide insights into users common operations on windows, tasks and computer use.

To fill this knowledge gap, this dissertation aims at understanding and developing new window switching techniques to help users to improve task switching. To achieve this goal, we developed a tool we called WindowsOSLog to log window management activity in mainstream Windows operating system by means of Windows messages and hooks to understand how users manage their windows. 26 participants participated in this study duration over 5-weeks. We also collected subjective data from participants by means of questionnaires. Three window switching techniques: Push-and-Pull Switching, stack scanning and WindowsTagging were designed and developed based on the results of this data.

The remainder of this chapter formally defines the research goals, describes the approach for achieving these goals, provides scope for this work, presents the primary research contributions and outlines the structure of the dissertation.
1. INTRODUCTION

1. Understand users’ activities on window management and the reasons users choose to employ switching techniques. These results should also allow user activities to be generalized or classified. Successful completion of this goal will explain the activities observed in the long-term study. It will also provide insights into the limits of user knowledge of current switching techniques.

2. Understand current window/group switching techniques where and when they are effective and ineffective. This goal will provide the directions to design new switching techniques.

3. Theorize window/group switching and define types of switching operations which have been justified by read-world usage data, and then provide some design principles to help designers to design switching techniques. Successful goal completion will provide a basic theory to guide designers to develop new switching techniques.

4. Using the knowledge gathered in the previous goals, analyses, design and evaluate new switching techniques based on design objective. Successful goal completion will result in new switching techniques that empirically and subjectively outperforms currently available alternatives.

1.2 Research Approach

Window switching is one of the most common operations and the most frequent task of any window manager. This research focuses on understanding the switching techniques used when interacting in real-world and then providing improvements and some design principles to guide designers to develop new switching techniques. To implement the first of these goals, we need to observe user window switching actions during a long period of time. Techniques such as interviews and screen-recorders provide a contextual understanding of user activities, but require a large amount of time to analyze, leading to scalability issues in large studies. To achieve this goal, a log tool was developed to record window management activity in mainstream Windows operating system in real-world. We also collected subjective data from participants by means of questionnaires.

Having formed a good understanding of user window switching activity, the results are then used to help designing new window/group switching techniques: Push-and-Pull
1.3 Research Contributions

Switching, stack scanning and WindowsTagging. We also conducted and ran a set of experiments to compare the performance of our techniques to other techniques, the results showed that our switching techniques could help users to switch between windows/groups more effective than the traditional switching techniques, and users preferred our techniques.

Finally, to theorize window/group switching, we defined types of switching operations which have been justified by read-world usage data and introduced window tagging mechanism to provide a new alternative to the existing windows grouping techniques, and then provide a set of design principles to help designers to design new switching techniques. WindowsTagging was implemented as a prototype system based on those design principles.

1.3 Research Contributions

This dissertation makes seven primary contributions to the research knowledge in the domain of window/group switching techniques. These are:

1. A review of window/group switching techniques and related window management systems. This review allow other researchers to more quickly understand and analyze current switching techniques.

2. A log-based longitudinal study about user window management activity, a log tool called WindowsOSLog was developed to record user window management activity in mainstream Windows operating system. 26 participants’ window management activities were recorded during a period of 5-week.

3. Design and evaluation of Push-and-Pull Switching. Push-and-Pull Switching is a window switching technique using window overlapping to implicitly define groups. Push-and-Pull Switching further allows to switch between groups and restack the focused window to any position to change its group affectation. The technique was evaluated in an experiment showing that Push-and-Pull Switching allows to improve 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 while they found it easy to understand and use.
1. INTRODUCTION

4. Design and evaluation of stack scanning. Stack scanning is based on a widget that combines generalized scrolling and crossing to control the stack order of layers of visible windows. The empirical evaluations showed that stack scanning was faster than other techniques when the number of windows is high and the visual similarity among windows is important. They also showed that Taskbar was the best choice when the number of windows is small, regardless of other visual factors conditions, and for users who always maximized each window, Alt+Tab was the best choice when the number of windows is important.

5. Provide eleven design principles and introduce window tagging mechanism to help designers to design new window switching techniques. Those design principles provides a theory foundation to designers and researchers.

6. Design and evaluation of WindowsTagging. WindowsTagging was designed based on the eleven design principles. It combines implicit and explicit definition of groups, spatial and visual memories to help users to quickly find windows or groups and switch between them. The empirical evaluations showed that WindowsTagging was faster than Exposé technique, and participants strongly preferred it. The longitudinal study also showed that WindowsTagging was very effective and could improve task management.

1.4 Structure of the Dissertation

The remainder of this dissertation is organized as follows.

To begin, Chapter 2 reviews of the current related work in the domain of window and group switching and the state-of-the-art in window and group switching techniques.

Chapter 3 describes a log-based longitudinal study, a log tool called WindowsOSLog that was developed to record user window management activity in mainstream Windows operating system. It can record low level interactions such as “Alt+Tab pressed” and “the right mouse button pressed” as well as high level interactions such as windows selections. WindowsOSLog was then installed on the computer of 26 participants duration over 5-weeks. Finally, we describe the results and analysis of this study in details.
Chapter 4 presents a new window switching technique, *Push-and-Pull Switching*, is a window switching technique using window overlapping to implicitly define groups. Push-and-Pull Switching further allows to switch between groups and restack the focused window to any position to change its group affection. Two experiments were designed and implemented to compare the performance of *Push-and-Pull Switching* to other window switching techniques (*Direct pointing, Taskbar, Alt+Tab*) under different scenarios. Finally, a longitudinal user study was represented to show how users actually use it through log analysis and users feedbacks.

Chapter 5 presents a new window switching technique, *stack scanning*, a window switching technique based on a widget that combines generalized scrolling and crossing to control the stack order of layers of visible windows. We conducted an experiment to compare the performance and error rate of *stack scanning* to other four common window switching techniques under a variety of visual conditions (e.g. the number of windows, their visual similarity and windows layout). Results showed that stack scanning was faster than other techniques when the number of windows is high and the visual similarity among windows is important. They also showed that *Taskbar* was the best choice when the number of windows is small, regardless of other visual factors conditions, and for users who always maximized each window, *Alt+Tab* was the best choice when the number of windows is important.

Chapter 6 presents eleven design principles which are based on the presented issues on the existing window/group switching techniques. We then designed and developed a prototype system called *WindowsTagging* based on those design principles. It combines implicit and explicit definition of groups, spatial and visual memories to help users to quickly find windows or groups and switch between them. An experiment was designed to compare the performance and error rate of *WindowsTagging* to *Exposé*. Results showed that *WindowsTagging* was faster than *Exposé* technique, and participants strongly preferred it. A longitudinal study also showed that *WindowsTagging* was very effective and could improve task management.

Chapter 7 provides a discussion of the research and summarizes the findings and conclusions of the work presented in this dissertation and future work in this area.
Chapter 2

Related Work

This chapter presents a review of the current knowledge in the domain of window/group switching techniques. We first review the different window and group switching techniques provided by modern window management systems and proposed by previous researchers and their associated evaluations. We then describe related visual factors that can affect the performance of switching technique (e.g. spatial memory, window visibility and windows layout) and review the tags mechanism and their typical applications. Finally, we describe the studies about window switching techniques, including interview-based study and log-based study.

2.1 Introduction

Window switching is an integral part of our daily computing experience since computers allow multi-tasking. Window switching is also one of the most frequent tasks of any window manager happening several hundred times per day (takes place on average once every 20.9s on large displays \textit{Hutchings et al. (2004)}, the results were confirmed by \textit{D.Mackinlay & Royer (2007)}). Window switching includes two subtasks: first, finding the window of interest (visual search), and second, bringing it to the foreground. However this process can be harder when the number of windows becomes important.

Users of desktop computers are increasingly turning to large displays and multiple-monitor setups. A key advantage of additional screen space is the ability to keep more windows open simultaneously, reducing the amount of resizing, repositioning, and other
2.2 Window Switching Techniques

window-management activity (Hutchings has shown that users have more than eight win-
dows open more than 78% on the time Hutchings et al. (2004); Robertson et al. (2005)).
However, additional windows also mean that more windows are competing for users’ at-
tention and users need to frequently manage them.

The importance and frequency of window switching has led to intensive research and
development into improving switching efficiency for window manager. Many researchers,
organizations, and individuals have proposed a variety of window/group switching tech-
niques.

This chapter is organized as follows. We first review all existed window and group
switching techniques provided by modern window management systems and proposed by
previous researchers and their associated evaluations. We then describe related visual fac-
tors that can affect the performance of switching technique (e.g. spatial memory, window
visibility and windows layout) and review the tags mechanism and their typical appli-
cations. Finally, we describe the studies about window switching techniques, including
interview-based study and log-based study.

2.2 Window Switching Techniques

There are many different window switching techniques provided by modern window
management systems and proposed by previous researchers.

Kumar et al. have categorized window switching techniques into three approaches
Kumar et al. (2007): Temporal, Spatial and Hybrid. Temporal approaches sort windows
based on their time of last access, and therefore the order in which the windows are shown
to the user changes depending on which window was used last (e.g. Alt+Tab/Cmd+Tab).
Such techniques make best use of the user’s temporal memory and make switching among a
limited number of windows very efficient. Spatial approaches may use an initial ordering
based on temporal information or on where the window is located on the screen. The
relative order of the windows in the window switching view does not change, unless there
is a change in the number or spatial location of the windows (e.g. Taskbar/dock, Exposé);
Hybrid approaches use a combination of temporal and spatial techniques to determine
how to present the list of windows to the user (e.g. Window Vista Alt+Tab, Windows XP
PowerToys Microsoft (b)).
2. RELATED WORK

2.2.1 Temporal Approach

2.2.1.1 \textit{Alt+Tab}

\textit{Alt+Tab} is the typical example of a \textit{temporal} approach (Figure 2.1). It ranks the window icons or thumbnails based on Z-order \footnote{Windows XP/2000 Alt+Tab list algorithm: For each visible window, walk up its owner chain until you find the root owner. Then walk back down the visible last active popup chain until you find a visible window. If you’re back to where you’re started, then put the window in the Alt+Tab list. (http://blogs.msdn.com/b/oldnewthing/archive/2007/10/08/5351207.aspx)}, which is related to the order with which the windows were last accessed. Kumar \textit{et al.} have observed that \textit{Alt+Tab} is very efficient when the number of windows is low Kumar \textit{et al.} (2007), but researchers have also labelled the method ‘tedious’ Grudin (2001) and reported that a very small percentage of users regularly use the \textit{Alt+Tab} key combination for window switching Czerwinski \textit{et al.} (2003).

\textit{Alt+Tab} functionality in Windows Vista and 7 has been updated with Windows Flip and Flip3D Microsoft (a). Flip allows users to view a preview of each opened window instead of just the program icon as they press \textit{Alt+Tab} (Figure 2.2). In addition, Windows Flip 3D enables users to flip through a cascading stack of their opened windows using the mouse scroll wheel or the keyboard. Windows can be stacked and rotated in 3D to provide views of all of them simultaneously (Figure 2.3).

Some compositing window managers (Compiz Fusion \footnote{http://www.compiz.org/}) for the X Window System have differ in the implementation of the \textit{Alt+Tab} technique, when users select the icon, the selected window will be brought to the foreground with the animation. However there was no evaluation to compare with these two implementations.

2.2.1.2 \textit{RelAltTab}

The \textit{RelAltTab} is an enhanced \textit{Alt+Tab} prototype that assists users in switching windows Oliver \textit{et al.} (2008). Authors used semantic and temporal information to create a list of related windows to the window that the user is currently engaged in (the algorithm is the same as SWISH system Oliver \textit{et al.} (2006)). This ordering of the list was used to replace the ordering of windows in the \textit{Alt+Tab} user interface. The results have shown that the related window list contained the next window that the user switched to in over...
2.2 Window Switching Techniques

Figure 2.1: Alt+Tab dialog in Windows XP showing icons for opened windows. A title is displayed for only one icon at any time, and the same icon may appear again for each application window opened.

Figure 2.2: Windows Vista or 7 Alt+Tab. It shows running application names along with their live thumbnails so that you can see the window content before switching to them. It also shows desktop in the list so that you can directly access desktop icons without minimizing all running applications manually.

80% of the instances. The main assumption is that the user is more likely to switch to a related window than to any other window in the system.
2. RELATED WORK

Figure 2.3: Windows Flip 3D renders live thumbnail images of the exact contents of the opened windows. It allows users to switch between windows while dynamically displaying them in a 3D view.

2.2.2 Spatial Approach

2.2.2.1 Taskbar/Dock

The Taskbar (Figure 2.5 and 2.6) or dock (Figure 2.7) follows a spatial approach in its organization of window buttons (each containing an icon and a truncated piece of the window title). Users can access any open window by clicking on a button or iconic representation of the window. The location of icons or buttons on the Taskbar is fixed and therefore this approach takes advantage of the user’s spatial memory. In addition, the Taskbar can group by application when more than a certain number of windows belonging to one particular application are opened. In this case, they collapse into a single application button that activates a pop-up with a list of windows. Recent research has hinted that the Taskbar has potential usability problems Hutchings & Stasko (2003). One issue raised by some participants in a qualitative study is that when eight or more windows are open, and the Taskbar is in its default position at the bottom of the monitor, only a few (or none!) of the letters in the windows’ titles are visible (In the default case, once the number of windows reaches a certain amount (10 – 15 windows for a 1024 x 768 pixel resolution display), very little, if any, of the title bar can be read and only the
2.2 Window Switching Techniques

Figure 2.4: RelAltTab includes two types of background coloring: salmon for the semantically related windows and light-blue for the temporally related windows. It also displays a red number on top of each semantically related window. This number allows the user to directly switch to the window by pressing Alt + NUMBER.

icons remain). In that case, it would need more time to distinguish the windows which have the same icons when using the Taskbar for window switching (visual similarity is increased). Many researchers, including Microsoft themselves, have attempted to improve the Taskbar. Both Smith et al. Smith et al. (2003) and Robertson et al. Robertson et al. (2004) claim that the grouping by application behavior confuses many users because application windows may not be related to the same task. Smith et al. presented the GroupBar Smith et al. (2003) as a solution, which allows user to arbitrarily assign groups. Visual TaskTip (Figure 2.8) was developed to enhance the cognition of buttons on the Taskbar, when the mouse is hovered over Taskbar, it displays a thumbnail preview of the opened window 1. However there was no report of any form evaluation of the tool.

![Figure 2.5: Ungrouped window buttons on the Taskbar](http://www.pctipsbox.com/visual-task-tip-for-windows-xp/)
2. RELATED WORK

2.2.2.2 Exposé

Exposé \(^1\) is another example of a spatial approach, which tiles all opened windows or those of the focused application so that they are all visible at once Apple. But as the number of opened windows increases, the legibility of their content decreases, making them hard to distinguish. Exposé uses a non-stable spatial layout to arrange the opened windows in a layout: when a window is repositioned or resized between two invocations of the technique, the layout may completely change. When users want to switch back to a window previously switched, they may no longer find it at its last location, wasting the benefit of spatial memory.

2.2.2.3 EyeExposé

The EyeExposé system Kumar et al. (2007) presented an innovative extension of the Exposé system by allowing users to use a combination of keyboard and eye gaze. This approach combines the use of a two-dimensional layout visualization for showing to the user all opened applications and the use of eye gaze tracking for selecting the window of interest. The authors designed an experiment to evaluate EyeExposé and found Alt+Tab in Windows XP that was faster when the number of open windows was low (4), and EyeExposé had the lowest switching time when the number of open windows was high (12). For Alt+Tab, its performance scaled worse relative to the other methods evaluated as the number of open windows increased. However we were not able to find other research

\(^1\)http://www.apple.com/pro/tips.switch_expose.html
2.2 Window Switching Techniques

Figure 2.8: Visual TaskTip, it displays a thumbnail preview of the opened windows when the mouse is hovered over Taskbar.

Figure 2.9: Exposé view of open applications.

evaluating the relative performance of $Alt+Tab$ to support their findings. We could notice that they did not take into account the visual similarity of windows in the experiment, which directly impacts the visual search time for finding the target window.
2. RELATED WORK

Figure 2.10: The Taskposé visualization arranges open windows in two dimensions when the visualization is called up. Windows automatically size relative to their importance, and closely-related windows appear together.

2.2.2.4 Taskposé

Taskposé Bernstein et al. (2008b) (Figure 2.10) is another example of a spatial approach, which is a screen-filling visualization of user’s workspace in two-dimensions, representing opened windows by thumbnails and using a degree-of-relatedness to implicitly create task groupings. It uses the WindowRank algorithm and window switch history to define the window association, and then using this association heuristic to layout related windows near each other. It uses a non-stable spatial layout to arrange the opened windows and the size of the thumbnails can also be changed. When users want to revisit a window, the size and its last location may have changed, wasting the benefit of spatial memory. However when users work on multiple tasks simultaneously, Taskposé would move the tasks close together to make them difficult to distinguish. Meanwhile it does not adequately use the space to display thumbnails as big as they can (the big thumbnail can reduce the selection time). During longitudinal evaluation, authors observed that Taskposé was most useful when the number of opened windows outstripped the space available on the Windows Taskbar, In that case, the performance of Taskbar will greatly decrease (see Section 2.2.2.1).
2.2.2.5 **SCOTZ**

Keith Humm presented Spatially Consistent Thumbnails Zones (SCOTZ, see Figure 2.11) which is an example of a spatial approach and uses the stable zones to display window thumbnails (Humm 2007). SCOTZ divides the entire screen space into a grid of equally-sized zones. Each zone represents a group of thumbnails based on the same application, and each thumbnail only belongs to one zone. Especially, zones are always located in the same space and never move. Taken into account that windows are opened and closed frequently, spacing inside zones must be much more dynamic than the zones themselves. The author use a hybrid approach between zone sizing and space-filling techniques: a zone is sized based on the nearest square number to the number of items required. When resize occurs, the current grid is transposed to the upper left coordinates of the new grid. The author evaluated SCOTZ and found that the performance of SCOTZ performs faster than the other techniques (Grouped Taskbar (see Figure 2.6), Ungrouped Taskbar (see Figure 2.5), and Alt+Tab (see Figure 2.1)) for both low (4, 8) and high window density (16, 32). SCOTZ is more similar to Exposé than Taskbar and Alt+Tab (SCOTZ has group mechanism and the property of spatial constancy, and Exposé may have bigger size thumbnails), however the author did not compare with the Exposé, leading to results not convincing. Meanwhile the author did not take into account the visual similarity of windows in the experiment. This factor can also impact the results of the experiment.

Tak et al. added the size morphing (see Figure 2.12) operation to the SCOTZ (see Figure 2.11) technique, the size morphing operation allows for the addition of new items while maintaining as much spatial stability as possible and allocating more space to frequently-used applications and windows in order to reduce their selection time (Tak et al. 2009a). The authors investigated the performance impact of four different orderings (spatially stable, recency order, frequency order, and random order) for tasks involving acquisition of targets in a Zipf-like \(^1\) distribution and found that the stable layout was significantly faster than the recency layout, with performance benefits increasing with expertise. The authors evaluated the size morphing operation and found that gradual size ‘morphing’ of item areas in the display did not affect performance - importantly, morphing did not substantially harm the participant’s spatial memory for target locations.

---

\(^1\)http://en.wikipedia.org/wiki/Zipf%27s\_law
2. RELATED WORK

2.2.2.6 FST

FST technique has been proposed to view and manipulate hidden content through unimportant regions of overlapping windows Ishak & Feiner (2004), and using mouse-
over pie menu to display thumbnails of all windows that lie beneath that selected pixel. However, the techniques only considered the windows close to the mouse cursor, and when the pie menu displays multi items, it becomes difficult to display thumbnails of each window at a recognizable size.

2.2.3 Hybrid Approach

Hybrid approaches use temporal ordering but allow for random access. The Windows XP PowerToys Microsoft (b) shows thumbnails of all the desktop top-level windows and allows users to either cycle through the window by repeatedly pressing Alt+Tab or to use the mouse to click on the thumbnail for the window of interest. Another example of a hybrid approach is Windows Vista Alt+Tab (Task Switcher), which shows live thumbnail of all the desktop top-level windows \(^1\) and allows users to either cycle through them by repeatedly pressing Alt+Tab or to directly switch to any window by clicking on its thumbnail with the mouse.

2.3 Group Switching Techniques

To improve switching efficiency for window manager, researchers have in general researched in two directions, one way is to develop new window switching techniques (see section 2.2) and another research direction is to effectively organize windows (grouping windows). We review related works of this research direction in this section.

To reduce the number of switching operations to achieve the objective of reducing the switching time, switching techniques have been proposed to explicitly or implicitly define window groups. For explicit way (manual control way), although users can define groups in any way, users need to organize windows by themselves. The primary limitation of this way is that users must carry out explicit actions to gain potential benefits (e.g. GroupBar Smith et al. (2003) and Scalable Fabric Robertson et al. (2004)). For implicit

---

\(^1\)Windows Vista Alt+Tab (Flip/Flip3D) list algorithm: Windows Vista changed the default behavior (Classic Alt+Tab, under most default installations) with its Flip interface. The six most recently used items in the Flip tab-order work as described (thumbnails are still shown in Z-order), then remaining windows are ordered alphabetically by application path (and optionally grouped, depending on the ‘group similar Taskbar buttons’ setting which is enabled by default. (http://blogs.msdn.com/b/oldnewthing/archive/2007/10/08/5351207.aspx)
2. RELATED WORK

way (automatic way), by contrast, windows are organized to its most likely group based on upon evidence group creation and manipulation, such as semantic data about the windows (window title, window content), activation history. The big problem of this way is that the system can incorrectly predict the users’ intention and users can fail to understand the system prediction (e.g. WindowScape Tashman (2006)). In the following paragraphs, we first describe related works of explicit window grouping techniques, then reviewing current knowledge of implicit window grouping techniques.

2.3.1 Explicit Window Grouping Techniques

For explicit window grouping technique, users can define groups in any way, and it requires to plan in advance the number of groups and associate each window to a group. There are many techniques that have been proposed.

2.3.1.1 Virtual Desktop Managers

Virtual desktop managers (VDMs) have been one of the most popular solutions to explicitly define groups but at the cost of a strict separation between them, making it difficult to switch between windows from multiple groups. Meanwhile VDMs have limitations in the flexibility of grouping mechanisms they provide, and the means offered for finding particular windows. One of the earliest designs exploring a VDMs was Smalltalk Project Views Goldberg & Robson (1983). Rooms D. Austin Henderson & Card (1986) is probably the most famous of these kinds of systems. A number of these systems are currently available, and are described in XDESK. None of these systems have been evaluated in a stringent way and no detailed study demonstrate how easy to learn they are, or how well they integrate into real-world settings.

2.3.1.2 GroupBar

In addition to VDMs, a number of novel and more flexible solutions have been proposed, such as GroupBar Smith et al. (2003), Scalable Fabric Robertson et al. (2004) and SCWM Badros et al. (2000). GroupBar (Figure 2.13) used the same minimized window representation used by Windows Taskbar to task creation and maintenance by allowing users to create or remove a task by using a single dragging gesture and allowed users to simultaneously display any subset of windows, even if they should be assigned to
2.3 Group Switching Techniques

Figure 2.13: GroupBar is a prototype for demonstrating the use of window-grouping features in an Windows XP TaskBar-like interface.

different tasks. The design did not effectively leverage human spatial and visual recognition memory. GroupBar made limited use of spatial memory by allowing users to create multiple bars, limitations stem from the bar design, which is linear, list-based, and did not expose much virtual space in which to place tasks.

2.3.1.3 Elastic Windows

Kandogan et al. proposed Elastic Windows that was designed to provide an alternative to window management strategies for efficient personal role management based on hierarchical window organization, multi-window operations and space-filling tiled layout Kandogan & Shneiderman (1996). In Elastic Windows, window groups could be created by opening a container window and dragging and dropping the selected items inside this window. The authors compared Elastic Windows to an independent overlapping window
2. RELATED WORK

Figure 2.14: Scalable Fabric showing the representation of three tasks as clusters of windows, and a single window being dragged from the focus area into the periphery.

manager in terms of user performance time on task environment, switching, and four task execution types. They found almost in all cases that Elastic Windows resulted in faster task completion and task switching times.

2.3.1.4 Scalable Fabric

*Scalable Fabric* (Figure 2.14) makes use of the periphery of the display for spatial layout of tasks, in addition to leveraging users’ efficient visual recognition memory for images, allocating screen real estate in accordance with a user’s attention, using a focus-plus-context display, allowing users to group collections of windows that are used together and to leave windows and clusters of windows open and visible at all times via a process of scaling down and moving the windows and clusters to the periphery.

However users need to leverage the size of the focus area and the periphery in order to operate (the focus area) and recognize (the periphery) the windows.

2.3.1.5 SCWM

The key feature of *SCWM* is the user’s ability to interact with groups of windows by setting constraints on the windows, the constraints aim for windows layout, and lack of control over windows stacking. Users create these constraints by using a relationship panel that graphically represents the types of constraints that can be set. Using the panel hides the details of the constraint system. An example constraint is adjacency, where some
window A is constrained to be adjacent to some other window B. Then, whenever a user repositions either A or B, its partner is simultaneously repositioned without additional user interaction. Unfortunately, there is not enough information about how users currently use SCWM and where and how it is adequate.

A disadvantage faced by these systems is that requiring windows to be in a single group forces users to decide ahead of time where a new window belongs. These systems do not allow for windows to reside in multiple groups at the same time. Likewise, such a single-group approach is inflexible, in that it neglects the possibility that some windows might naturally be useful in several different groups.

### 2.3.2 Implicit Window Grouping Techniques

Compared to explicit window grouping techniques, implicit grouping techniques use different methods, such as machine learning, to automatically define groups. However the system can incorrectly infer groups and create groups that may not correspond to users expectations. There are also many techniques that have been proposed.

#### 2.3.2.1 WindowScape

WindowScape Tashman (2006) (Figure 2.15) is an example of implicit grouping technique which automatically creates groups by taking photograph-like snapshots each time a window is expanded or miniaturized, using the timeline of desktop states shown as a series of photograph-like snapshots. 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. Each snapshot group can include many windows and two close groups may include the same windows (windows have the same state (position, size)), this can lead to the snapshots that look like very similar, thereby it may cause difficulties to distinguish between the two groups. Similarly, WindowScape does not appear to have been evaluated.

#### 2.3.2.2 Stack leafing

Stack leafing Faure et al. (2009) technique is another example of implicit grouping techniques, which is based on widget that combines generalized scrolling and crossing to control the stacking order of layers of non-overlapping windows. This technique has the advantage of minimizing mouse navigation and preserving the size and position of
2. RELATED WORK

Figure 2.15: WindowScape represents windows as thumbnails and uses the timeline of desktop states shown as a series of photograph-like snapshots.

windows while still providing access to all of their content. However, this technique does not preserve the ordering by frequency and strict non-overlapping requirement also limits its use.

2.3.2.3 SWISH

SWISH Oliver et al. (2006) uses semantic features (windows associated with the same task share common words in their content, and, in particular, in their window titles) and temporal features (windows associated with the same task are accessed in temporal proximity to each other) to automatically divide windows into different groups. Their evaluations suggested 70% accuracy rates in assigning windows to task groups.

2.4 Tabs Switching Techniques

With more and more applications based on tabs, the performance of switching among the tabs directly impacts the user experience of application. The applications arrange the tabs by default in one row and usually support *drag and drop* operation, and when the number of tabs is over a threshold, they will add a horizontal scrollbar (e.g. Firefox) (For those applications, generally, the width of each tab is the same and can be changed) or add a pull-down menu (e.g. Visual Studio IDE) (For those applications, generally, the
width of each tab depends on its title and can not be changed) for hidden tabs. The spatial approach is used to organize the tabs.

The Direct pointing is the main interaction technique for switching among tabs, keyboard shortcuts are also supported (e.g. Ctrl+Tab), it is based on temporal approach as usual (like Alt+Tab), and a few applications can support the function like Exposé to display all tabs as thumbnails with the keyboard shortcut (e.g. Internet Explorer 8).

2.5 Desktop Organization and Switching Techniques

Users have different ways in which they organize screen space. Some experiments have shown that users can be divided into three groups depending on the way they organize screen space: MAXIMIZERS (simply maximize all windows or almost all windows), NEAR MAXIMIZERS (these users have one or more smaller windows with which they frequently interact or glance or leave a bank of desktop icons uncovered) and CAREFUL COORDINATORS (those who tend to have many windows visible simultaneously (meaning that none of them is maximized) or when they had a maximized window, were working in an application that itself has many sub-windows). Careful coordinators represent 50% of all users Hutchings & Stasko (2003).

Hutchings has shown that all maximizers (all were single monitor users and all used MS Windows) used a Taskbar or Alt+Tab to switch among windows. This outcome is expected as the Taskbar and keyboard can be easily accessed regardless of window size, making window switching very easy for this type of user. Each near maximizer used Direct pointing to switch between a nearly maximized window and a visible window elsewhere on the desktop. But near maximizers composed a variety of interaction techniques for switching among nearly maximized windows. Careful coordinators (many used multiple desktops and high screen resolutions) used Direct pointing more, they seemed to use the Taskbar only when the window to be switched to was not visible.

With the large displays and multiple-monitors, people have presented different interaction activities. A previous research has shown that multiple monitor users used Direct pointing more and used the Taskbar less than single monitor users Hutchings et al. (2004). Tak’s study has also shown the same results Tak et al. (2009a), and her study also showed that dual monitor users used Alt+Tab less than single monitor users, and most users rely heavily on mouse-based (Taskbar and Direct pointing) window switching methods.
2. RELATED WORK

2.6 Spatial Memory And Visual Search

In the real world, spatial memory (i.e. the ability to remember where you put something) helps us find things. For example, when we place a document on a pile in our office, we are likely to remember approximately where that document is for a long time. People hope that this ability works as well in the digital work environment as it does in the physical world. Locating a desired target item in a scene with a number of competing distractors is termed visual search. Visual search is an activity that we usually perform many times each day, including in the physical world and the digital work environment. When we want to operate with the target, the first thing is to find the target by visual search. The performance of visual search can directly impact users experience. Many researches have confirmed that performance with user interfaces is strongly predicted by spatial aptitude and many techniques have been introduced to take advantage of human spatial memory to improve users interactive capabilities.

Recent research in neuroscience has shown that object recognition in primates makes use of features of intermediate complexity that are largely invariant to changes in scale, location, and illumination (Tanaka (1997), Perrett & Oram (1998)). It is also known that object recognition in the brain depends on a serial process of attention to bind features to object interpretations, determine pose, and segment an object from a cluttered background Treisman & Kanwisher (1998).

2.6.1 Psychology of Visual Search and Memory

Recognition of place, navigation, and formation of cognitive maps are among the fundamental activities for any creature, and so it is not surprising that humans have considerable skills in these areas. For example, Standing and Haber have hinted that users who matched collections of hundreds or thousands of images are able to recognize the previously shown images with an accuracy of 90% and more after only one brief viewing Standing et al. (1970). Shepard has shown that memory for briefly shown pictures is greater than that for words Shepard (1967), and searching for a picture of a particular object among many is also faster than searching for the name of that object among other words PAIVIO (1974). More generally the availability of rich visual cues in the environment is believed to help in the formation of memory and play a role in developing
cognitive maps Searleman & Herrmann (1994); conversely, navigation in environments without distinctive visual appearance is difficult.

Some evidences have been provided that humans often prefer to use visual skills in organizing their works Lewis et al. (2004). Although haphazard "stacks" of papers would seem to be an inefficient organizational strategy, stacks allow search based on the remembered location and appearance of documents, rather than relying on their more easily forgotten textual labels Lansdale (1988); Maglio & Barrett (1997). Jones and Dumais Jones & Dumais (1986) provided some cautions on overreliance on spatial organization. Their evaluation shows that semantic labels provide stronger retrieval cues than spatial organization alone, but indicate that combinations of semantic and spatial organization enhance performance.

2.6.2 Spatial Memory and User Interfaces

Prior research has shown that the efficient use of graphical user interfaces strongly depends on human capabilities for spatial cognition Cockburn (2004). User interfaces can improve task performance by exploiting the powerful human capabilities for spatial cognition, this opportunity has been demonstrated by many prior experiments. Egan and Gomez Egan & Gomez (1985) showed that measures of spatial memory and age provided the best predictors of how well participants learned to use a text editor. Gagnon Gagnon (1985) reported the surprising result that computer game scores were not correlated with measures of hand-eye coordination, but were correlated with scores on a spatial memory test. Vicente et al. Vicente et al. (1987) and Leitheiser and Munro Leitheiser & Munro (1995) also concur that measures of spatial ability predict performance in hierarchical file browsing tasks and in a variety of file management tasks.

Spatial memory is a valuable tool in supporting efficient information organization. Some researches have shown that spatial organization of information allows efficient access to items in graphical user interfaces (GUI). For example, Data Mountain Robertson et al. (1998) is a user interface for document management designed specially to take advantage of human spatial memory, which has been empirically shown that task times and error rates were lower when retrieving web pages (from sets of 100 pages) using their 3D Data Mountain than when using a commercial program with a list-based interface (the standard 2D 'Favorites' mechanism of Internet Explorer). A subsequent evaluation
2. RELATED WORK

of the Data Mountain showed that subjects were able to rapidly retrieve pages six months after creating their spatial organization Maarten et al. (1999). Furthermore, replacing the thumbnail images with blank outlines did not detrimentally affect retrieval time. The document thumbnails were no more effective than icons in a recall test-thumbnails Maarten et al. (1999). The role of simulated location and context has been explored in Infocockpit, a multiple-monitor and multimodal interface testbed Tan et al. (2001). Subjects memorized word pairs shown at various points on a three-monitor display, and in one condition a "context" was created by projecting a photograph (e.g. of a museum interior) as a panoramic backdrop in the test room. The panoramic context greatly aided later recall Stefanucci & Proffitt (2002).

2.6.3 Visual Search

There is a large body of literature on visual search (e.g. Fisher et al. (1989); Treisman & Gelade (1980); Wolfe et al. (1989)). Some of these studies have culminated in a set of guidelines for designers of the digital work environment.

Feature integration theory Treisman & Gelade (1980) assumes that search proceeds in two stages: an initial stage of pre-attentive search, where low-level features (such as color) are efficiently extracted in parallel across the full field of view, and a stage of attentive search, where interesting points identified in the first stage are visited serially. If an item visually pops-out because a feature pre-attentively guides attention, the time required to find it is independent of the number of distracters. Numerous studies have thus tried to identify features that can guide attention. The most certain features are color, motion, orientation, and size Wolfe et al. (2004). Other likely features are shape and shading. Shading can enable the perception of depth which pops out Kleffner & Ramachandran (1992). Not all guiding features have the same visual span, however. For example, motion can be detected at greater distances from the point of fixation than color Bartram et al. (2003) or shape. In general, targets far from the point of fixation are detected slower and less accurately Wolfe et al. (1998). Furthermore it has been found that guiding features are not truly independent: Random variations of color interfere with the ability to find a target based on orientation Callaghan (1989) or shape Pashler (1988). The converse may also be true but to a lesser degree, suggesting that there exists a hierarchy with some features, such as color, dominating others, such as orientation or shape Callaghan (1989).
Complex backgrounds may also make it harder to separate items and thus also reduce search efficiency \cite{Wolfe2002}. In the presence of distracters, search becomes more efficient when target and distracters are less similar. Heterogeneity among distracters reduces search efficiency, but if distracters can be grouped by some criteria efficiency is improved \cite{Duncan1989}.

Raphael \textit{et al.} evaluated 9 graphical visual cues (five types of frames and mask around the target window and four trails leading to the window) for window switching on large screens \cite{Hoffmann2008}. They evaluated each cue in isolation and some cues combination model. The results have shown that combinations of trails and frames are more effective than the cues in isolation and resulted in high subjective satisfaction (the best cues were visually sparse-combinations of curved frames which use color to pop-out and tapered trails with predictable origin).

### 2.7 Finding obscured Windows

Selecting a virtual object is a prerequisite for a variety of interaction. The speed and accuracy to find a window can directly affect the subsequent interaction operation. But sometimes it is very difficult to find the target window. Pierre Dragicevic \cite{Dragicevic2004} proposed “fold-and-drop” technique to avoid using traditional techniques (Taskbar, \textit{Alt+Tab} and Exposé) to find the target window for Dragging and Dropping object between overlapping windows. He has hinted traditional techniques are not wholly satisfactory because they require switching back and forth between two different representation of the same window set. The biggest issue is that compact window visualization do not show sufficient information for the target window to be recognizable while they are too small to contain numerous drop targets. But using the “fold-and-drop” technique it is sometimes also very difficult to find the target window (The example of Manipulating Multiple Folds Interaction in \cite{Dragicevic2004}). In the WindowScape system \cite{Tashman2006}, the author allows users to bring all miniatures to the top of the z-order for finding obscured windows, when users simply drag the cursor over the desktop background and all miniatures and title bars appear, while everything else tints red to make the miniatures visually stand out. But if one window’s title bar is fully covered with other windows, this method is also very poor to find that window.
2. RELATED WORK

2.8 Switching Techniques Time Model

Window switching includes two subtasks: first finding the window of interest (visual search) and second bringing it to the foreground. Each technique requires first a visual search and second an action to select the window of interest. Then, interaction time should be $T_i(n, op) = T_v(n) + T_a(op)$, where $T_i(n, op)$ is the total time for interaction technique; $T_v(n)$ is the visual search time as a function of windows set size; $T_a(op)$ is the time for action to select the window of interest as a function of the technique used for the selection.

The time to perform this visual search depends on the number of items and the degree of visual similarity between these different windows (more similarity requires more time). The time to select the window of interest depends on the technique used, and the technique implementation depends on the input devices. It can be a number of keystrokes or a target selection task.

2.8.1 Hick-Hyman And Fitt’s Laws

The Hick-Hyman Law Hick (1952), Hyman (1953) describes human decision time as a function of the information content conveyed by a visual stimulus. The amount of time taken to process a certain amount of bits in the Hick-Hyman Law is known as the rate of gain of information. Fitts’ Law Fitts (1954) describes the movement time taken to acquire, or point to, a visual target. Both Hick-Hyman and Fitts’ Laws are derived from information theory Shannon & Weaver (1949), where the information content $H$ of an event, measured in bits, is inversely proportional to its probability - likely events have low information content; unlikely ones, high. The formula for information content is: $H = \log_2 \frac{1}{p}$, where $p$ is the probability of the event.

The Hick-Hyman Law can be generalized in the case of choices with unequal probabilities $p_i$ of occurring, to

$$T = a + b \times H$$

where $H$ is the information-theoretic entropy of the decision, defined as:
where \( p_i \) refers to the probability of the \( i^{th} \) alternative yielding the information-theoretic entropy. The constants \( a \) and \( b \) are determined by linear regression. According to Card et al. Card et al. (1983), the +1 is "because there is uncertainty about whether to respond or not, as well as about which response to make".

The target acquisition stage is well described by Fitts' law Fitts (1954) and is widely verified (for example, Card et al. (1987) and MacKenzie et al. (1991)) and has been heavily used in HCI research, largely because many expert tasks require low-level object selection of graphical screen elements or of physical keys. In essence, Fitts' law states that objects that are smaller or further away will take longer to acquire than those that are larger or closer. Equation 2.2 shows MacKenzie’s widely accepted improvement to Fitts’ law. The movement time, \( MT \), is proportional to the Index of Difficulty, \( ID \). \( ID \) is determined from \( A \), the amplitude of movement and \( W \), the target width, \( a \) represents the start/stop time of the device (intercept) and \( b \) stands for the inherent speed of the device (slope). The constants \( a \) and \( b \) are determined experimentally by fitting a straight line to measured data (linear regression). From the equation, we see a speed-accuracy trade off associated with pointing, whereby targets that are smaller and/or further away require more time to acquire.

\[
MT = a + b \times ID
\]  
where:

\[
ID = \log_2\left(\frac{A}{W} + 1\right)
\]

### 2.8.2 GOMS/KLM

GOMS is a modeling technique (more specifically, a family of modeling techniques) that analyzes the user complexity of interactive systems. It is used by software designers to model user behavior. The user’s behavior is modeled in terms of Goals, Operators, Methods and Selection rules, which are described below in more detail. Briefly, a GOMS model consists of Methods that are used to achieve Goals. A Method is a sequential list of
2. RELATED WORK

Operators that the user performs and (sub) Goals that must be achieved. If there is more than one Method which may be employed to achieve a Goal, a Selection rule is invoked to determine what Method to choose, depending on the context Card et al. (1983).

The Keystroke-Level Model (KLM) is a simplified version of GOMS. It was proposed by Card et al. as a method for predicting user performance Card et al. (1983). The KLM is an 11-step method that can be used by individuals or companies seeking ways to estimate the time it takes to complete simple data input tasks using a computer and mouse. Using KLM, execution time is estimated by listing the sequence operators and then summing the times of the individual operators. The original KLM had six classes of operators: K for pressing a key, P for pointing to a location on screen with the mouse, H for moving hands to home position on the keyboard, M for mentally preparing to perform an action, and R for system response where the user waits for the system. For each operator, there is an estimate of execution time. Additionally, there is a set of heuristic rules to account for mental preparation time.

GOMS and KLM models are limited for two reasons: they are confined to expert performance of routine tasks, and they use a 1.10s average time for pointing the mouse to an object on screen, which is crude compared to the precision accessible through the Fitts’ law.

2.9 Log-based Empirical Methods

Recent researchers have performed many longitudinal studies using logging tools to collect data. Hutchings (2004) used VibeLog tool to log each window management activity that occurred on mainstream Windows platform. The main feature of VibeLog is the maintenance of two logs of window system information: events and windows. They logged these information in order to understand and analyze the display space usage and window management operation comparisons between single monitor and multiple monitor users Hutchings et al. (2004). Oliver et al. (2008) also used VibeLog tool to log window activity (window titles and all the window switching events) and tried to find semantic and temporal related windows. Then they used related windows to automatically adapt to users activities Oliver et al. (2006, 2008). Xiaojun et al. (2009) ran VibeLog and MouseLog tools to log window management activities and mouse activities (including each window event that occurred, detailed information of each running window and each mouse
activity) on different levels resolution display and compared users’ activities to understand
the differences and similarities between large high-resolution displays and single or dual
desktop displays for daily work  Bi & Balakrishnan (2009). Through this study, Xiaojun et al. discovered the differences and gave some advice to design new tools to support large high-resolution displays. Keith Humm (2007) used TrayLog tool to log task switching actions in Windows XP to understand task switching and then using the understanding gained to help them to design SCOTZ interface that can better utilize spatial memory in task switching  Humm (2007). Mackinlay and Royer (2004) used the Glass Box Analysis environment custom software that runs on Microsoft Windows platform to capture a large amount data including mouse events, keyboard events, windows events, and so on. They used collected data to analyze window thrashing and presented results as a timeline D. Mackinlay & Royer (2007). Renaud and Gray (2004) used UAR tool based around GRUMPS project to collect low-level usage data such as window focus events, mouse click events and key press events. The GRUMPS system provides a mixture of data describing different events at the user interface to support a variety of diverse investigations for HCI practitioner to understand user activities. They also present a novel method of data cleaning and analytical preparation and techniques to deal with missing actions  Renaud & Gray (2004). Tak et al. (2009) have carried out the longitudinal study to understand users’ patterns of revisitation to windows and applications  Tak et al. (2009a), after getting the observed results, authors strongly suggested that supporting revisitation should be a main design goal in window switching tools (see Section 2.2.2.5).

2.10 Tags

A tag is a non-hierarchical keyword or term assigned to a piece of information (such as an Internet bookmark, digital image, or computer file). This kind of metadata helps to describe an item and allows it to be found by browsing or searching. Tags are generally chosen informally and personally by the item’s creator or by its viewer, depending on the system. Tagging was popularized by websites associated with Web 2.0 and is an important feature of many Web 2.0 services. 1

1http://en.wikipedia.org/wiki/Tag_(metadata)
2. RELATED WORK

With the extensive application of Web 2.0 and the emergence and rapid proliferation of social resource sharing, social bookmarking tools become more and more popular nowadays, such as BibSonomy \(^2\), Flickr \(^3\), Delicious \(^4\), tagging has become a popular way to organize the information and help people to recall or search the resources. People can freely assign tags to any resource object (such as image, video, blog, media, photos, articles), this mechanism is reshaping the way people manage and access visual content Li et al. (2010). One might expect tag-based retrieval to be a natural and good starting point for search. Compared to content-based image retrieval Datta et al. (2008), tag-based retrieval copes more easily with semantic queries. Moreover, its scalability has been verified by text retrieval research Bay et al. (2008).

Tags may be a ”bottom-up” type of classification, compared to hierarchies, which are ”top-down”. In a traditional hierarchical system (taxonomy), the designer sets out a limited number of terms to use for classification, and there is one correct way to classify each item. In a tagging system, there are an unlimited number of ways to classify an item, and there is no ”wrong” choice. Instead of belonging to one category, an item may have several different tags. Some researchers and applications have experimented with combining structured hierarchy and ”flat” tagging to aid in information retrieval Heymann & Garcia-Molina (2006).

Social tagging has been popularized by ubiquitous and diverse content sharing services, from bookmarks, articles, to photos and videos. Tagging systems have become major infrastructures on the Web. Tagging is widely used to organize information and allow users to recall or search the resources. Mark et al. presented a study for searching videos that showed in the current context, social tags yield an effective retrieval process, whereas automatically generated metadata do not Melenhorst et al. (2008). They had also found some evidence that tagging is effective in the retrieval process.

Wetzker et al. have hinted that tags allow users to create tags that annotate and categorize content and share them with other users, very helpful in particular for searching multimedia content Wetzker et al. (2010).

SwingStates uses tags on 2D graphical objects to represent groups of 2D objects and manipulate all the objects in a group at once Appert & Beaudouin-Lafon (2008). The

\(^2\)http://www.bibsonomy.org/
\(^3\)http://www.flickr.com/
\(^4\)http://www.delicious.com/
SwingStates includes two types of tags: extensional tags, which are explicitly added to or removed from a shape and intentional tags, which are specified by a predicate that tests whether or not a shape has this tag. Presto uses tags on documents interaction to tag data items with any relevant contextual information. Giornata uses tags on activities to describe their semantics. Each activity in Giornata can be annotated with optional, freeform tags to describe its semantics and an activity can have one or more tags associated with it. An activity’s tags help users to identify the activity in which they are currently working on and distinguish among background activities. The active activity’s tags are persistently visible, rendered over the desktop wallpaper.

2.11 Summary

This chapter has presented a review of window and group switching techniques and their associated evaluations and users activities of management windows longitudinal studies based on logging tools. In order to improve the efficiency of windows switching, a variety of new switching techniques are proposed.

2.11.1 Issues Associated With Log-based Empirical Methods on Window Switching

To understand window and group switching, researchers have conducted interviews of a variety of users with a variety of window managers and developed a variety of tools to observe and record users’ activities. The results have shown users’ use of these window and group switching techniques, but it is short of detailed analyzing and users feedback about why and where users use these switching techniques (habit? or window layout? or other cases?).

2.11.2 Issues Associated With Switching Techniques Evaluations

Many switching techniques have not been evaluated or only informally, and only a few of them have been evaluated. As a result, it is hard to assess the extent to which ad-
dressed problems actually exist in window/group switching practices. For group switching techniques which have been evaluated, the authors mainly compared the performance of their group switching technique to other window switching techniques (e.g. GroupBar, Elastic Windows), no researchers have compared the performance of their group switching technique to other group switching techniques, so it is difficult to conclude.

2.11.3 Leveraging Natural Human Capabilities

For switching techniques, the first step is to find the targets, this step depends on people’s spatial memory and cognition capabilities. For Taskbar, users get the target window according to the content of button (icon + text) and the location of the button on the Taskbar; for Alt+Tab, according to the z-order or the icon and the title of the target window; for Exposé, according to the thumbnail of the target window; for Direct pointing, according to the appearance of the target window. From the traditional window switching techniques, the location of window has not been considered closely (Jones et al. has hinted name + location is better than name-only and location-only to recall an object Jones & Dumais (1986)) and the appearance of the target window is used only in the Direct pointing (when the target window is invisible, most time this technique is not used). Replacing to use the appearance of the target window, users need to use other representations of the same window to find the target window, users require switching back and forth between two different representations of the same window set.

2.11.4 The Combination of Implicit and Explicit Grouping Techniques Are The Best Way

Grouping windows techniques fall into two main categories: implicitly defined group and explicitly defined group. The primary limitation of implicitly defining a group is that they can incorrectly predict the user’s intention and that users can fail to understand or anticipate the system’s adaptation. The combination of our analysis of the related work on switching techniques, we believe such automation is critical (implicit grouping technique), as users are typically disinclined to organize their personal information up-front Bernstein et al. (2008a) and The new switching techniques should allow users to control groups by manual (explicitly grouping technique), so the combination of implicit
and explicit methods is the best way. The strategies for implicitly defining group are very important, this can effectively reduce users manual control, but a variety of used strategies are short of theory foundation. The simple and good way is to analyze users’ activities to build the rules for creating groups.

2.11.5 Window Tagging

Tags was used in a variety of fields, including document management, activity, 2D graphics, image searching, and so on. But this concept and mechanism are never used to windows management, we explore the concept of window tagging instead of groups-as-lists to support window switching.
Chapter 3

Log-Based Longitudinal Study

This chapter describes a log-based longitudinal study, a log tool called WindowsOSLog was developed to record user window management activity in mainstream Windows operating system. It can record low level interactions such as “Alt+Tab pressed” and “the right mouse button pressed” as well as high level interactions such as windows selections. The activity of 26 participants was monitored by WindowsOSLog during 5-weeks. We describe the results and analysis of this study in details.

3.1 Introduction

Researchers investigating techniques for improving window switching would benefit from a thorough empirical characterization that describes how current window management systems are used by people in daily activities. Insights into the windows used and patterns of window switching would provide evidence and motivation for the design and development of new window switching techniques.

The log-based longitudinal study described in this chapter provides a foundation to help designers to design new window switching techniques. This chapter is organized as follows. To begin, we describe the study objectives, participants and methodology. We then detail the design and implementation of the longitudinal study to collect data from users. These data explicitly detail their window activities (including a questionnaire investigation). Finally, we analyze this data to gain knowledge of how users manage their windows and tasks.
3.2 Experiment Objectives

The aims of this study are to understand how users manage their windows. On the one hand, the aims are to identify trends observed within the participant group that can be used alongside data from other studies (see Related Work), and to provide insight into the causality of these trends. On the other hand, we want to get data that have not been analyzed by other studies. Our aims include two ways:

1. Quantitative Goals: concerning quantifiable measurements of user activity
2. Qualitative Goals: using log data and subjective responses to help explain and validate our measurements

3.2.1 Quantitative Goals

We will collect and analyze log data to provide statistical evidence to answer the following questions:

1. How many windows are simultaneously opened?
   *Do users only keep the windows/applications about the current task or keep all previously opened windows/applications?*

2. Which window switching techniques do users often use?
   *Do users have some predilections to use some window switching techniques?*

3. How many windows/applications do users frequently use?
   *Do users often switch to a large or a small number of windows/applications?*

4. How do users arrange their windows/applications on the desktop?
   *Do users often change window size or use default window size?*

5. How many visible windows/applications do users frequently keep?
   *Do users often keep some windows visible to quickly access or in order to get more contents simultaneously visible?*

6. How much time each window/application is active?
   *Do users frequently switch windows/applications?*
3.2.2 Qualitative Goals

We also collect subjective data from participants by means of a questionnaire to be used in conjunction with the log data to both validate the log data and answer. Because it is sometimes different to understand users’ strategies only through log data, the questionnaire is a good complementarity to understand users activities:

1. Which factors can affect the usage of window switching techniques? What do user depend on when they choose window switching techniques?
   The factors include: Position, Window Visibility, Recency Order, Number of windows and device used to switch (Mouse/Keyboard)

2. Which factors do users remember best for a window?
   The factors include: Content/Title/The type of application, Position, Recency Order,....

3. When a technique is used and a wrong window/application appear, what technique is used next?
   the same technique or another one

4. How often windows/applications are open and closed, or windows are switched?
   Creating/Closing Event vs. Switching Event

5. How individual work sessions differ?

6. Do users often coordinate windows to finish a task or do they only keep default window size?

7. Getting general comments about window switching techniques;
   Problems, improving, new functions?

Our questionnaire contained also 5-point Likert scales for rating Alt+Tab, Taskbar, Direct pointing, including speed, accuracy, ease of use and user preference.
3.3 Longitudinal Study

In order to fully understand users activities, we employed a broader user population and a longer period of time. On the one hand users of desktop computers are increasingly turning to large displays and multiple-monitor setup (over the past decade, the default size of a desktop monitor has increased from 15” to 21” Grudin (2001)), and on the other hand the display of laptop has not changed in the same way. Previous studies mainly focused on the differences and similarities of the users for different monitor configuration conditions (single monitor users vs. multiple monitor users) and they rarely took into account the differences between the different display resolution with the same monitor configuration condition. Taken into account this situation, our study employ some users who had a wider range of differences in monitor resolution (from 1024 x 768 to 1680 x 1050). We would like to understand whether there are some similarities or differences between these users. Taken into account that some users simultaneously use multiple monitor and single monitor (laptop) in their daily work, our study includes those participants and we want to understand the differences and similarities of the same user who simultaneously works on single monitor and multiple monitor computer.

3.3.1 Definition

Windows OS uses “click-to-focus focus model” (“focus follows click”), that means the window clicked by the mouse can get focus and becomes the active window (i.e. the only window receiving user input). It is brought to foreground and receives keyboard input. We use input focus to refer to the window that has system focus, i.e. the window that exclusively receives input from the user. We use user focus to refer to the window that the user is actively viewing, which may or may not have input focus.

For a window w, there are several ways to make w active (also called switching to w), including clicking on any part of w (Direct pointing), pressing a key combination Alt+Tab and selecting w from the resulting window list, or clicking on w’s Taskbar button. Taskbar is a special area that aids in window management by displaying a list of buttons (icon + title), one for each application window. The user has the option of making the Taskbar always on top so that other windows cannot occlude it, or autohide, so that Taskbar slides out of focus (and out of sight) when not being activated. Special operations for windows include minimize (which Myers refers to as iconify Myers (1988)) which hides...
3. LOG-BASED LONGITUDINAL STUDY

A window from view but maintains its Taskbar button, and maximize, which grows a window to the size of its current monitor. Windows can overlap, two windows are said to overlap or intersect if the windows share at least one common pixel. Two windows are adjacent if a section of each of their borders touch (so that there are no pixels in between the two windows), but the two windows do not intersect. If two windows are intersecting or overlapping, that covered windows have some valuable information which is invisible for user. If the user wants to browse or check the information, (s)he must move the upper window or bring the covered window to the front. A key attribute of a window is its z-order, each window is assigned an integer (z) as its depth; the window with the highest z-order is at the top of window stacking and the window with the lowest z-order is at the bottom of window stacking. A window has a width (x-dimension), height (y-dimension), and a depth (z-dimension), all of which can be manipulated directly by the user. A window can be repositioned in many ways (width, height and depth, one of which is changed). Resize means to change the height or width of a window, move means to change the left-to-right or up-to-down position of a window, and stacking means to change the z-order of a window.

An open window can be invisible to the user for two main reasons: (1) the user has hidden the window, for example by minimizing it, and (2) the window is occluded because another window or set of windows higher in the z-order obscure it.

We used the same definition as Oulasvirta & Saariluoma (2006) for sessions that are periods of work within a day, separated by a long interruption. We used an idle time tracker to record each time the computer has received no input. Because the large majority of our participants are Ph.D student, engineer (from our lab), we will define long interruption as anything longer than 30 minutes as an acceptable time for a meeting or lunch break.

3.3.2 Participants And Apparatus

26 participants participated in this study with a mean age of 27 (SD = 2.4), 18 participants (Group1) are from within the computer science department (8 Ph.D. students, 5 software engineers, 1 researcher, 4 electronic engineers), the remaining 8 participants (Group2) were students, but they do not come from the computer science department. Most of them use computers more than 8 hours every day. All participants are Windows
Operating System users, including Windows XP (22), Vista (2), Windows 7 (2). Display resolution ranged from 1024 x 768 to 1680 x 1050 for single monitor participants, and from 1024 x 768 to 1680 x 1050 per monitor for dual monitor participants. They were 12 participants using dual monitor in Group1. All participants in Group2 are single monitor users. The study duration is over 5-weeks (eight of them is over 10-weeks). The detailed information of participants is displayed in the Table 3.1.

3.3.3 Design

Participants were categorized into different groups according to their normal computing environments. We first divided participants into two groups according to their monitor configuration condition: single monitor or dual monitors users. We then divided single monitor participants into two groups according to their display resolution. Taking into account that some participants simultaneously use single monitor and multiple monitor systems, we specially considered these participants as one group.

We use a tool we called WindowsOSLog to log window management activity in mainstream Windows operating system by means of Windows messages and hooks. After running WindowsOSLog, then we collected users data and use a statistical analysis to understand users activities. A questionnaire investigation helped us to understand users activities.

3.3.3.1 WindowsOSLog

The WindowsOSLog application was built in Visual C# and can run on current mainstream Windows platform. It hooks the Win32 API to listen to and publish window events. The tool is used to log windows system information (events and windows) by means of Windows messages and hooks. Static configuration information is collected at the first startup, including the version of operating system, physical memory, the number of monitors and coordinates of each monitor registered by the operating system (Table 3.2).

3.3.3.2 Log Information

The event log has an entry for each window management activity that occurred. These activities include window opening and closing, window activation, focus, movement, re-
3. LOG-BASED LONGITUDINAL STUDY

Table 3.1: Participants Information

<table>
<thead>
<tr>
<th>Par.</th>
<th>Sex</th>
<th>Age</th>
<th>Profession</th>
<th>Display Resolution</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>24</td>
<td>geoscientist</td>
<td>1024x768</td>
<td>Windows Vista</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>26</td>
<td>computer scientist</td>
<td>1024x768</td>
<td>Windows XP</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>26</td>
<td>electronic engineer</td>
<td>1024x768</td>
<td>Windows Vista</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>25</td>
<td>mechanical engineer</td>
<td>1280x800</td>
<td>Windows XP</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>25</td>
<td>geoscientist</td>
<td>1280x800</td>
<td>Windows 7</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>24</td>
<td>geoscientist</td>
<td>1366x768</td>
<td>Windows 7</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>31</td>
<td>civil engineer</td>
<td>1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
<td>29</td>
<td>electronic engineer</td>
<td>1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td>29</td>
<td>civil engineer</td>
<td>1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>10</td>
<td>M</td>
<td>26</td>
<td>electronic engineer</td>
<td>1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>11</td>
<td>M</td>
<td>32</td>
<td>civil engineer</td>
<td>1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>12</td>
<td>M</td>
<td>29</td>
<td>civil engineer</td>
<td>1680x1050</td>
<td>Windows XP</td>
</tr>
<tr>
<td>13</td>
<td>M</td>
<td>26</td>
<td>electronic engineer</td>
<td>1680x1050</td>
<td>Windows XP</td>
</tr>
<tr>
<td>14</td>
<td>M</td>
<td>24</td>
<td>computer scientist</td>
<td>1680x1050</td>
<td>Windows XP</td>
</tr>
<tr>
<td>15</td>
<td>M</td>
<td>25</td>
<td>computer scientist</td>
<td>1280x1024/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>16</td>
<td>M</td>
<td>25</td>
<td>computer scientist</td>
<td>1280x1024/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>17</td>
<td>M</td>
<td>26</td>
<td>computer scientist</td>
<td>1280x1024/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>18</td>
<td>M</td>
<td>23</td>
<td>computer engineer</td>
<td>1280x1024/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>19</td>
<td>M</td>
<td>27</td>
<td>computer scientist</td>
<td>1280x1024/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>20</td>
<td>F</td>
<td>26</td>
<td>computer scientist</td>
<td>1280x1024/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>21</td>
<td>M</td>
<td>25</td>
<td>computer scientist</td>
<td>1280x1024/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>22</td>
<td>M</td>
<td>27</td>
<td>computer scientist</td>
<td>1600x1200/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>23</td>
<td>F</td>
<td>27</td>
<td>computer scientist</td>
<td>1600x1200/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>24</td>
<td>M</td>
<td>32</td>
<td>computer scientist</td>
<td>1600x1200/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>25</td>
<td>M</td>
<td>26</td>
<td>computer scientist</td>
<td>1680x1050/1280x1024</td>
<td>Windows XP</td>
</tr>
<tr>
<td>26</td>
<td>M</td>
<td>27</td>
<td>computer scientist</td>
<td>1680x1050/1280x1024</td>
<td>Windows XP</td>
</tr>
</tbody>
</table>
Table 3.2: Example of user configuration information.

<table>
<thead>
<tr>
<th>Operating System Version:</th>
<th>Windows XP Professional Service Pack 3 v5.1.2600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical:</td>
<td>2042 MB</td>
</tr>
<tr>
<td>Display Number:</td>
<td>2</td>
</tr>
<tr>
<td>Primary Screen Resolution:</td>
<td>1600 * 1200</td>
</tr>
<tr>
<td>Second Screen Resolution:</td>
<td>1280 * 1024</td>
</tr>
</tbody>
</table>

sizing, minimizing, maximizing, and so no. The window state includes window’s ID (identified by window’s handle), title, host application, coordinates, z-order, monitor information, size state, styles, class name, process ID and executable name of the application (e.g. Explorer.EXE). The size state has three kinds of cases: maximized, minimized, normal. The window styles define a series of attributes of the window, including whether the window is (1) visible, (2) a popup window (typically used for dialog boxes), (3) a toolbar, and (4) always on top. The monitor information identifies which monitor the window locates on (the window may locate on more than one monitor, but this case is rare). For single monitor system, it always returns the main monitor; for multiple monitor system, the result is calculated by the coordinates of the window and coordinates of monitor system (each monitor has a coordinates range).

Because the switching events are not included in the event generated by the Windows OS, and there is no window-switching and task-switching interface provided by Windows OS, we track the input separately (thanks to alternative input devices abstracted by the Windows OS, so the input type too appear as one of keyboard or mouse). We infer the device used for switching by analyzing the keyboard and mouse events.), and attribute the most recently generated input event to the window event, taking care to clear the most recent input event as appropriate. We also used activation event to indicate a window switch Humm (2007). If a user activates a window by using the key combination Alt+Tab, the switching technique is Alt+Tab; if a user activates a window by clicking in the Taskbar region, the switching technique is Taskbar; if a user activates a window by clicking outside Taskbar region, the switching technique is Direct pointing; if a user activates a window by using the key combination Alt+Esc, the switching technique is Alt+Esc; when a window/dialog automatically pop-ups (e.g. message prompt dialog), the switching technique is OtherSW.
3. LOG-BASED LONGITUDINAL STUDY

Table 3.3: Description of the WindowsOSLog output stream

<table>
<thead>
<tr>
<th>Name of Event</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Number</td>
<td>Identify Event</td>
<td>3</td>
</tr>
<tr>
<td>Number of Windows</td>
<td>Current number of windows opened</td>
<td>8</td>
</tr>
<tr>
<td>Number of visible windows</td>
<td>Current number of visible windows</td>
<td>3</td>
</tr>
<tr>
<td>Event Name</td>
<td>CREATE, ACTIVE, etc.</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>Switching Technique</td>
<td>which techniques are used to bring this window to the foreground</td>
<td>Taskbar</td>
</tr>
<tr>
<td>HWND</td>
<td>Win32 window handle</td>
<td>132312</td>
</tr>
<tr>
<td>Title</td>
<td>Current title of window</td>
<td>Windowstack</td>
</tr>
<tr>
<td>Class Name</td>
<td>Current window class name</td>
<td>ExploreWClass</td>
</tr>
<tr>
<td>Process Name</td>
<td>Name of executable which owns the window</td>
<td>Explorer.EXE</td>
</tr>
<tr>
<td>Position</td>
<td>Current window coordinates (x,y)</td>
<td>(-4,-4) - (1684,1024)</td>
</tr>
<tr>
<td>Size State</td>
<td>maximized/minimized/normal</td>
<td>Normal</td>
</tr>
</tbody>
</table>

3.3.3.3 Information Incompleteness

For some applications, especially multi/tabbed document interface (MDI/TDI) application such as Firefox, it is impossible to generate correct message streams or window sets by means of OS windows message mechanism (it uses its own internal message mechanism to operate). However it does not hinder us to obtain their window titles, positions and other information. Meanwhile for technical reasons such as OS optimization of event generation and dispatch, a few windows may fail to generate correct message streams (such as they fail to produce the activation event), but such cases are extremely infrequent ([Hutchings et al. (2004), Humm (2007)]), therefore they will not affect our statistical results.

3.4 Results And Analysis

We are interested in understanding how users operate and manage windows with respect to the type digital work environment (operating system, single monitor, dual
monitors, display resolution). We hope to find some similar (or different) activities by analyzing users actions in order to design and develop better interface to improve users experience.

When summarizing data from many users, it is important to not let one participant’s actions be lost or overshadow by those of the other participants. Statistics are generally reported as the mean of participant means, this value is calculated by first determining the mean value for each participant and then averaging those means to calculate a value. In descriptive statistics, box-and-whisker charts\(^1\) provide a convenient way of graphically depicting groups of numerical data through their five-number summaries: the smallest observation (sample minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (sample maximum). These are especially useful when users show diverse behavior. For clarity, Figure 3.1 shows the values that are used on box-and-whisker charts in this dissertation. The minimum, lower quartile, median, upper quartile and maximum values are displayed, some charts will also illustrate mean values with the use of a diamond. In the following parts, we present results that we got from the log.

\[\text{Figure 3.1: Example box-and-whisker chart.}\]

\(^1\)http://en.wikipedia.org/wiki/Box_plot
3. LOG-BASED LONGITUDINAL STUDY

The number of windows on screen is an important factor that can impact users’ activities. Sometimes this factor is even decisive Kumar et al. (2007). In this subsection, we present information on how many windows users often keep on the desktop with respect to monitor configuration.

Figure 3.2 shows that the number of windows opened by each participant during this study. For single monitor user, the difference among the mean number of windows opened is small with display resolution from 1024 x 768 to 1280 x 1024, but there was a difference for 1680 x 1050, for which users opened more windows simultaneously (the mean number of windows opened is 10) (Table 3.4).

Many researches have indicated that multiple monitor systems can help users be more productive Czerwinski et al. (2003), one reason is that users who used multiple monitor can open more windows simultaneously, reducing the amount of resizing, repositioning, and
Table 3.4: Users opened more windows on large display for single monitor users (s.d. = standard deviation).

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 * 768</td>
<td>5.7</td>
<td>5.0</td>
<td>0.01</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>1280 * 800</td>
<td>4.9</td>
<td>5.0</td>
<td>0.02</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>1280 * 1024</td>
<td>5.1</td>
<td>5.0</td>
<td>0.12</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>1680 * 1050</td>
<td>10.4</td>
<td>10.0</td>
<td>0.02</td>
<td>1</td>
<td>29</td>
</tr>
</tbody>
</table>

other window-management activity. The study data in Table 3.5 supports this conclusion, the mean number of windows opened was 7 for single monitor user and 10 for dual monitors user. Users also kept more windows on main monitor than on secondary monitor for the dual monitors users (Table 3.6).

Table 3.5: Users opened more windows on dual monitor system than on the single monitor system (s.d. = standard deviation).

<table>
<thead>
<tr>
<th>Display</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single monitor</td>
<td>6.8</td>
<td>5.9</td>
<td>0.01</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>Dual monitors</td>
<td>10.3</td>
<td>11</td>
<td>0.02</td>
<td>0</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 3.6: Users opened more windows on main monitor than on secondary monitor for dual monitor system users (s.d. = standard deviation).

<table>
<thead>
<tr>
<th>Display</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main monitor</td>
<td>6.9</td>
<td>7.0</td>
<td>0.01</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>Secondary monitor</td>
<td>3.3</td>
<td>3.0</td>
<td>0.01</td>
<td>0</td>
<td>11</td>
</tr>
</tbody>
</table>

3.4.2 Window Event

This analysis is conducted to understand users basic window management activities, such as which window management activity frequently happens, whether users often move windows or minimize or maximize windows, and so on.
Figure 3.3: Cross participant means of the percentage of windows Event: Create, Active and Destroy. Error bars show standard error (the standard error is too small for dual monitors, so it may not display on the chart).

Figure 3.3 shows that window activation activity happens more frequently than window creation and destruction under both single monitor and dual monitors conditions (A previous research has shown than window switching took place on average once 20.9s on large displays Hutchings et al. (2004)). The frequency of window activation makes the switching techniques to be particularly important for a window management system, because its efficiency directly affects the efficiency of window management and the user experience.

For dual monitors users, they could coordinate windows between two screens, so the window move event should be more frequent than single monitor users. Our data prove this result (Figure 3.4). We also observed that the single monitor users rarely moved windows and there was no substantial difference for maximize/minimize event between single monitor and dual monitors users.
3.4 Results And Analysis

Figure 3.4: Cross participant means of the percentage of window move and maximize/minimize event in window activation event. Error bars show standard error.

3.4.3 Window Switching Techniques

We analyze the main four window switching techniques currently used to switch windows: Direct pointing, Taskbar, Alt+Tab, Alt+Esc. This analysis was conducted to determine whether users made similar or divergent use of those techniques.

3.4.3.1 Window Switching Techniques Used to Switch Windows

Figure 3.5 shows single monitor users’ use of those techniques and Figure 3.6 shows dual monitors users’ use of those techniques. We also observed some important results from this data. First, there is substantial difference among users in their use of those techniques. All users never used Alt+Esc. To understand why users did not use it, we checked this problem in the questionnaire and we got the reason that most users (92.3%) did not know this technique, only two users knew it, but they thought it was not useful for them.

Second, we observed that users used less Alt+Tab than Direct pointing and Taskbar.

---

1 OtherSW switching technique data is removed from this analysis
3. LOG-BASED LONGITUDINAL STUDY

Figure 3.5: The percentage of window switching activated by Direct pointing, Taskbar, Alt+Tab and Alt+Esc for 14 single monitor participants.

Figure 3.6: The percentage of window switching activated by Direct pointing, Taskbar, Alt+Tab and Alt+Esc for 12 dual monitors participants.
under both single monitor and dual monitors environment. Some users (5 female participants and three male participants) reported that they could not understand very well this technique, because the order of presentation of items in Alt+Tab always changes, they do not like this technique. Another reason is that some users (57.7%) prefer to use mouse to switch windows, but no similar function is integrated into mouse.

Third, we observed that there was a difference among users in their use of Alt+Tab. Male users use it more frequently than female users, and all 7 female users use it very lightly or not at all (less than 1.87%, three of them did not use it). Ten male users also use it very lightly or not at all (less than 1.0%, two of them did not use it). Three male users use it fairly often (7.72%) and one single monitor user use it frequently (23.64%). When users used Alt+Tab technique to switch windows, the average number of pressing tab of all users is 1.4 by one time (Table 3.7), this result hints that users often use this technique to switch between the current window and the last active window. A previous research has also confirmed that Alt+Tab was very effective in this case Kumar et al. (2007). Figure 3.7 shows the number of pressed tab key for each user.

Table 3.7: The average number of pressing tab when users used Alt+Tab technique (s.d. = standard deviation).

<table>
<thead>
<tr>
<th>Display</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single monitor</td>
<td>1.3</td>
<td>1.2</td>
<td>0.04</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>Dual monitors</td>
<td>1.6</td>
<td>1.2</td>
<td>0.07</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>Total</td>
<td>1.4</td>
<td>1.2</td>
<td>0.04</td>
<td>1</td>
<td>26</td>
</tr>
</tbody>
</table>

Finally, there is a difference in the percentage of both using Direct pointing and Taskbar technique to switch windows between single monitor and dual monitor users (Figure 3.8). One main reason is that dual monitor users often switch from one display’s window to other display’s window, when a window is visible, users often use the mouse to click. Another reason is that dual monitor users often open more windows simultaneously (on average 10), and recent research has hinted that the Taskbar has potential usability problems Hutchings & Stasko (2003) (Chapter 2 section 2.2.2.1). Five dual monitors users reported that sometimes using Alt+Tab could lead to confusion, because they often use direct pointing to switch between one display and another display, so when they used
3. LOG-BASED LONGITUDINAL STUDY

![Number of pressing tab key when using Alt+Tab](image)

Figure 3.7: The number of press of the tab key when using Alt+Tab switching technique (because some values (min, lower quartile, median) are equal, so it results in some degenerated charts).

Alt+Tab to switch among windows, they might neglect/forget clicking action, and then get a wrong window.

### 3.4.3.2 Cost of Error

When users use one technique and get a wrong window, most of the time they do not change the technique, especially when they use Alt+Tab and Taskbar. When users used Alt+Tab, if they got a wrong window, they said that they would still use Alt+Tab to switch window, but they then could press the tab key slowly and more carefully to find the desired window. When users use Direct pointing, if they switch a wrong window, this might be fatal, because this can lead to make the desired window invisible. In this case, users often choose Taskbar technique to switch to the desired window.

Through our investigation, some factors can impact users to use switching techniques, such as: window visibility, current users work status (i.e. users are using mouse to...
3.4 Results And Analysis

Figure 3.8: Cross participant means of the percentage of window switching activated by Direct pointing, Taskbar, Alt+Tab and Alt+Esc. Error bars show standard error.

do something, then they may use mouse to switch; if users are using keyboard to do something, they can use Alt+Tab to switch).

3.4.3.3 Types of Switching

Although many researchers, organizations and individuals have proposed many group switching techniques, users rarely used these group switching techniques to manage their windows (only two users used VDMs, but not often). The main reason is that many of those systems were only prototype systems and there were no evaluation in any strict ways. However many users (65.4%) reported that they needed group switching techniques to support them to organize and manage their windows. Users reported that they needed at least three types of switching: 1) between windows; 2) between groups; and 3) between selected windows of groups. So that when we design new window switching techniques, those types of switching should be supported.
3. LOG-BASED LONGITUDINAL STUDY

3.4.3.4 Tabbed Windows vs. Group Windows

When the number of windows on the Taskbar reaches a threshold value (see Chapter 2 Section 2.2.2.1), the title on each button can not be read and only the icons remain. In this case, users often change the default configuration of Taskbar to use two rows to display items. Most users (73.1%) do not like the group windows function of Taskbar (Figure 3.9). They prefer to use tabbed windows technique (Figure 3.10) ¹.

They made this conclusion mainly based on the experience of their use of some browsers based on tabs metaphor (e.g. Firefox ² and Google Chrome ³). However there is an important difference between windows and the web pages. Windows often need to be resized, moved, and rearranged in order to make the best use of their contents and of the rest of the desktop. When users browse the web, they can not do the same operations on tabs. One of the big disadvantages of tabs compared to windows is that it is not possible to see the content of two tabs simultaneously by default (although some applications can support this operation). With windows, it is possible to arrange two windows side-by-side, or only partially overlapping. While this is useful in some cases (comparing two documents, typing in one windows while using another for reference, etc.), it is not often needed when browsing the web. We can not confirm whether users like it or not in desktop usage.

![Six windows were grouped in one tab](image)

Figure 3.9: Group Windows on Taskbar.

¹The first application for Windows OS that introduced tabbed browsing was QT Tab Bar ([http://qttabbar.wikidot.com/](http://qttabbar.wikidot.com/)) which added that functionality to Windows OS Explorer, but no participants used this tool and there was also no any study for this tool.
³[http://www.google.com/chrome](http://www.google.com/chrome)
3.4 Results And Analysis

Figure 3.10: An example of tabbed windows technique.

3.4.4 Window Visibility

One of the major advantages of a multitasking window system is the ability to both run and display many applications simultaneously. Since screen space is a limited resource, it seems likely that any opened window with some part visible at a particular point in time is of some importance to the user. We thus developed a line of analysis that focus on this measure of importance by calculating the percentage of visible area of a window for each window.

3.4.4.1 Number of Visible Windows

Users of multiple monitor system are expected to keep more windows visible simultaneously than single monitor system users Huchings et al. (2004), and our results confirmed that expectation (Table 3.8). Here, we used the same definition as Huchings et al. (2004) for window visible that a window is not entirely covered (not obscured).

Figure 3.11 shows that the number of visible windows kept simultaneously during this study. For single monitor user, the difference among the mean number of visible windows is small with display resolution from 1024 x 768 to 1280 x 1024, but there is a difference for 1680 x 1050, where users kept more windows visible simultaneously for 1680 x 1050.

Table 3.8: Users kept more windows visible simultaneously on dual monitor system than on the single monitor system (s.d. = standard deviation).

<table>
<thead>
<tr>
<th>Display</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single monitor</td>
<td>1.8</td>
<td>1.5</td>
<td>0.02</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Dual monitors</td>
<td>4.2</td>
<td>4.1</td>
<td>0.01</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>
3. LOG-BASED LONGITUDINAL STUDY

Figure 3.11: The number of visible windows kept on the desktop by each participant.

3.4.4.2 Visible Windows vs. Window Switching Techniques

The visible window is very meaningful for users. On the one hand users can see information from them, and on the other hand when users want to switch to them, it is easy to use Direct pointing technique to finish this action. Our results (Figure 3.12 and 3.13) confirmed this deduction. When the target window is visible, users used Direct pointing more than other techniques under both single monitor and dual monitors conditions, when the target window is invisible, users mainly used Taskbar technique (a few users also used frequently Alt+Tab to switch windows) and Direct pointing was not used in this case.

The visible windows can reduce users visual search time. In this situation users do not need to switch back and forth between two different representations of the same window set (e.g. when users use Exposé to switch windows, users need to switch back and forth between windows and thumbnails) (A previous search has shown that there was
3.4 Results And Analysis

Figure 3.12: Mean of the percentage of window switching techniques to switch windows when they are visible.

a potential problem for users to switch back and forth between different representations of windows Dragicevic (2004)). Another advantage to keep windows visible is to help users remember their positions. This can also help users to quickly revisit them (spatial memory).

3.4.5 Spatial Memory

People’s spatial memory ability is a valuable characteristic in supporting efficient information organization. Some evidences have been provided that humans often prefer to use this skill to organize their tasks in the workspace Lewis et al. (2004) (See Related Work). We would like to understand what content people often remember when they use some windows and whether they hope to use this information to help them to revisit those windows. We finished this study through our questionnaire investigation.

Users reported that they could often remember the topic of their tasks, and also remember the main application (e.g. Word, Excel), but it was difficult to remember which windows they used for their tasks, sixteen of them said that they did not take into account this matter. For the current task, they reported that they often remember some keywords of related windows, such as partial characters from the window’s title and
window’s content. They could also remember the type and icon of applications that they used frequently. The position can become an important factor for them to switch when windows are visible. Almost all users said that they did not want to have the burden of memory, so the stability is very important to them. When designing new switching techniques, designers should adopt the stability strategy as much as they can to reduce the users’ memory burden. This can be easy to explain why users used \textit{Alt+Tab} technique less often. From this study, we can get and sort users’ memory factors used to switch windows is: content (some semantic keywords, including title, window’s content) > the type of application > position > recency order. This is very useful as we can use this information to help users to quickly revisit the desired windows.

### 3.4.6 Windows Layout

Aforementioned, we have mentioned that users have different ways in which they organize screen space, and window switching techniques heavily depends on the windows layout. To understand how users organize their windows on the desktop, we need to analyze each window layout represented by users. A window layout can be summarized as these three main factors: 1) the number of windows; 2) window size; and 3) relationship
3.4 Results And Analysis

between window position (e.g. the order of window z-order, overlapping). We have described the factor of the number of windows in section 3.4.1, so we are going to analyze in detail the other two factors in the following subsection.

3.4.6.1 The Distribution of Window Size

Hutchings has shown that many users often show just a small portion of a window to use its information Hutchings & Stasko (2004). Meanwhile many applications use by default a small window to display information, such as instant messenger application, where users set the size and position of those windows, but rarely maximize. We use the concept SmallWindow to define the window size is less than 25% of screen resolution (the total pixels of the window is less than 25% screen pixels), LargeWindow to define the percentage is more than 75% and NormalWindow to define the percentage between 25% and 75%. We could understand the distribution of different sizes of window and know whether there are some differences among SmallWindow, NormalWindow and LargeWindow.

Figure 3.14 and 3.15 show that the distribution of SmallWindow, NormalWindow and LargeWindow in single monitor system and dual monitors system. We also observed some important results from this data. First, we observed that there were some differences among users in their use of different sizes of windows. Five users used the proportion of large window more than 70% of the time, six users used them around 60%, two users less than 10%, nine users used between 20% and 30% of the time. Twelve users used the proportion of small window less than 20% of the time, and nine users used them more than 40% of the time. Second, single monitor users used more large windows and less small windows than dual monitors users (Table 3.9).

3.4.6.2 Windows Group

Hutchings simply divided users into three groups by interviewing users how to organize screen space Hutchings & Stasko (2003): Maximizers, Near maximizers and Careful coordinators (Chapter 2 section 2.5). This simple categorization can not effectively detail real situation of users on how they organize windows. Users often work on more than one task simultaneously, and they often coordinate some windows to finish their tasks (Careful coordinators represent 50% of all users Hutchings & Stasko (2003)).
3. LOG-BASED LONGITUDINAL STUDY

Figure 3.14: Mean of the percentage of SmallWindow, NormalWindow and LargeWindow accounts for the proportion of the total number of windows for single monitor users. Error bars show standard error.

Figure 3.15: Mean of the percentage of SmallWindow, NormalWindow and LargeWindow accounts for the proportion of the total number of windows for dual monitors users. Error bars show standard error.
### 3.4 Results And Analysis

Table 3.9: The distribution of window size.

<table>
<thead>
<tr>
<th>Display</th>
<th>Window size type</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single monitor</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SmallWindow</td>
<td>25.3%</td>
<td>16.7%</td>
<td>0.05%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>NormalWindow</td>
<td>20.6%</td>
<td>16.7%</td>
<td>0.04%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>LargeWindow</td>
<td>54.1%</td>
<td>50.0%</td>
<td>0.05%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Dual monitors</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SmallWindow</td>
<td>44.2%</td>
<td>40.0%</td>
<td>0.07%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>NormalWindow</td>
<td>13.9%</td>
<td>25.0%</td>
<td>0.05%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>LargeWindow</td>
<td>41.9%</td>
<td>50.0%</td>
<td>0.07%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SmallWindow</td>
<td>29.6%</td>
<td>25.0%</td>
<td>0.04%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>NormalWindow</td>
<td>19.1%</td>
<td>12.5%</td>
<td>0.03%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>LargeWindow</td>
<td>51.3%</td>
<td>50.0%</td>
<td>0.05%</td>
<td>0%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Hutchings and Stasko have also shown that a significant group of users tend to have many windows opened simultaneously Hutchings & Stasko (2004). We would like to analyze all windows by each window’s position and z-order to understand whether users keep some windows (group) belonging to the same activity visible by trying to avoid or minimize the amount they overlap. Groups are defined by considering windows in decreasing Z order, from foreground to background, and creating a new group each time when the amount of overlapping for a window is beyond a given overlapping threshold (we would like to get this threshold value from this analysis, so that we could use this value to help users to define groups). Here the amount of overlapping (AmOVERLAP) for a given window is computed as the percentage of pixels occluded. For the dual monitors system, groups are defined by each monitor (each monitor is as an independent unit), the total number of groups is the sum of the number of groups of each monitor. We tested AmOVERLAP with 4 levels (0%, 25%, 50%, 75%), and 0% AmOVERLAP indicates that windows within a group do not overlap each other.

The algorithm of computing windows group as following:
3. LOG-BASED LONGITUDINAL STUDY

**Algorithm 1**: Compute the grouped windows

*Input*: the ungrouped windows  
*Output*: the group windows

```plaintext
define Window Wi in the windows stacking from foreground to background
    flag ← false
    Create a new group G_k and add W_i to this group
    foreach Window W_j in the windows stacking from W_{i+1} to background
        Create temporary rectangle group TG
        foreach W_t in the group G_k
            if W_j has intersection with W_t then
                Add intersection rectangle R_{jt} to the group TG
                if TG ∩ W_j ≥ setvalue * W_j then
                    flag ← true break
                end
            end
        end
    end
    if !flag then
        Add W_j to this group G_k
    end
end
```

Figure 3.16 and 3.17 show that the number of groups of single monitor users and dual monitors users when the overlapping threshold is set to 25%. Table 3.10 shows that the mean number of groups of both single monitor and dual monitor participants for each overlapping threshold condition. Table 3.11 shows that the distribution of the number of groups on the main monitor and secondary monitor for dual monitor for each overlapping threshold condition.

### 3.4.7 TDI and MDI Applications

Tabs have become the standard in web browsers to browse Internet pages (refer to tabbed document interfaces (TDI) applications). Meanwhile many multiple document interfaces (MDI) applications are also used every day. MDI is similar to TDI, they allow multiple documents to be contained within a single window, using tabs as a navigational
3.4 Results And Analysis

![Bar chart showing the number of groups of single monitor users when the overlapping threshold is set to 25%.](image)

Figure 3.16: The number of groups of single monitor users when the overlapping threshold is set to 25%.

<table>
<thead>
<tr>
<th>Display</th>
<th>AMOVERLAP</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SINGLE MONITOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0%</td>
<td>3.9</td>
<td>3</td>
<td>0.00</td>
<td>1</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>25%</td>
<td>3.6</td>
<td>3</td>
<td>0.00</td>
<td>1</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>3.2</td>
<td>3</td>
<td>0.00</td>
<td>1</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>3.0</td>
<td>3</td>
<td>0.00</td>
<td>1</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td><strong>DUAL MONITORS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0%</td>
<td>9.0</td>
<td>9</td>
<td>0.01</td>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>25%</td>
<td>7.5</td>
<td>8</td>
<td>0.01</td>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>7.0</td>
<td>7</td>
<td>0.01</td>
<td>1</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>6.7</td>
<td>7</td>
<td>0.01</td>
<td>1</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.10: The mean of number of groups of all participants for each overlapping threshold condition (s.d. = standard deviation).

The number of groups of single monitor users when the overlapping threshold is set.

Table 3.10: The mean of number of groups of all participants for each overlapping threshold condition (s.d. = standard deviation).

widget for switching sets of documents, but TDI does not form part of the Microsoft Windows User Interface Guidelines. However there were few studies in this area to know how people manage and switch with tabs and MDI applications. Only Patrick Dubroy & Balakrishnan (2010) did a study of tabbed browsing among Mozilla Firefox users.
3. LOG-BASED LONGITUDINAL STUDY

Figure 3.17: The number of groups of dual monitors users when the overlapping threshold is set to 25%.

Table 3.11: The mean of number of groups of main monitor and secondary monitor for dual monitor system users for each overlapping threshold condition (s.d. = standard deviation).

<table>
<thead>
<tr>
<th>Display</th>
<th>AmOVERLAP</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAIN MONITOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0%</td>
<td>6.2</td>
<td>6</td>
<td>0.01</td>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>25%</td>
<td>5.0</td>
<td>5</td>
<td>0.01</td>
<td>1</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>4.7</td>
<td>5</td>
<td>0.01</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>4.5</td>
<td>4</td>
<td>0.01</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td><strong>SECONDARY MONITOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0%</td>
<td>3.1</td>
<td>3</td>
<td>0.01</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>25%</td>
<td>2.8</td>
<td>2</td>
<td>0.00</td>
<td>1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>2.6</td>
<td>2</td>
<td>0.01</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>2.6</td>
<td>2</td>
<td>0.00</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

The biggest challenge in this field is that these applications are impossible to generate correct message streams by means of OS windows message mechanism (section 3.3.3.3), so researchers have to develop different tools based on their own internal message mechanism.
3.4 Results And Analysis

In this subsection, we would like to understand some basic activities for a few specific MDI and TDI applications (Firefox, Visual Studio series, Internet Explorer (IE), Google Chrome and all other MDI applications), such as the number of tabs/documents, which switching technique users often use to switch among tabs/documents, because we can get this information by means of Windows OS windows message mechanism. Figure 3.18 shows that Microsoft Spy++ \(^1\) displays the partial tabs/documents information of applications (this information was recorded in the log).

### 3.4.7.1 The Number of Tabs/Documents

The number of tabs/documents of a TDI/MDI application is basic and very important factor. Table 3.13 shows that the mean of number of tabs/documents in our study applications.

\(^1\)http://msdn.microsoft.com/en-us/library/aa264396(VS.60).aspx
3. LOG-BASED LONGITUDINAL STUDY

Table 3.12: The mean of number of tabs/documents in the TDI/MDI applications (s.d. = standard deviation).

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>8.4</td>
<td>8</td>
<td>0.03</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>Visual Studio series</td>
<td>7.2</td>
<td>7</td>
<td>0.02</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>IE</td>
<td>4.3</td>
<td>4</td>
<td>0.02</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>5.1</td>
<td>5</td>
<td>0.02</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>Other MDI applications</td>
<td>3.5</td>
<td>3</td>
<td>0.04</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3.13: The mean of percentage of switching techniques for TDI/MDI applications (s.d. = standard deviation).

<table>
<thead>
<tr>
<th>Switching Techniques</th>
<th>Mean</th>
<th>Median</th>
<th>s.d.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct pointing</td>
<td>99.86%</td>
<td>100%</td>
<td>0.00</td>
<td>99.42%</td>
<td>100%</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>0.06%</td>
<td>0.05%</td>
<td>0%</td>
<td>0.00</td>
<td>0.12%</td>
</tr>
</tbody>
</table>

3.4.7.2 Switching Between Tabs/Documents

There are two main types of switching techniques to be used for switching between tabs/documents: Direct pointing and Ctrl+Tab (some applications may use different keyboard shortcuts) (see Chapter 2 section 2.4).

3.4.8 Active Window Sequences

Hutchings has shown the average amount of time that any window was active was 20.9 seconds and half of all window activation lengths are quite short Hutchings et al. (2004). Thanks to users switch frequently among windows, it is very important to understand what kind of windows users switch frequently. The following rules are processed to construct the active windows sequences to get this result:

- according to time sequence;
- each session constructs one sequence (the definition of session 3.3.1);
- using the top-level windows activation event to construct.
Many TDI and MDI applications fail to produce correct message streams (section 3.3.3.3), some applications may adjust the title bar of parent window when the tab or child window’s title changes (e.g. Firefox), and we can not get this information by means of Windows OS windows message mechanism, so we use a polling loop every 150ms to detect title text, and when the new title text does not match the last stored text, this may indicate that the tab or MDI child window may have changed or open a new tab or child window. When we define the active window sequence, we will take into account this case (IsTChild). On one hand we do not take into account those tabs or child windows (NTChild) (we will mainly analyze this case), only using their parent window to construct it. On the other hand we take into account those tabs or child windows as a separate window (YTChild). In order to deal with those windows sequences, we use a simple symbol to identify a window (e.g. w1), using the same symbol to identify the same window in a sequence, so the sequence can be like this:

w1 w2 w3 w1 w2 w1 w2 w4 w3 w5 w4 w6 w5......

This sequence means: the first active window is w1, then w2 is active, after w3 is active, then w1 is active again, the rest can be done in the same manner. We can consider this sequence as a string sequence, each active window is as a basic element in the string, so we can deal with this sequence by using string processing method. We can easily find the windows that users switch frequently in a period of time by means of the sequence. We use the term \( A_n W_s \) to define that users switch frequently among \( n \) windows, so \( A_2 W_s \), \( A_3 W_s \), \( A_4 W_s \) correspond respectively to switching frequently among two windows, three windows, four windows. The sequence string, \( A_2 W_s \), \( A_3 W_s \), \( A_4 W_s \) are defined detailedly as follow:

- \( A_2 W_s \): the substring includes only two elements and the length of substring is greater than 3 (e.g. w1 w2 w1 w2 w1);
- \( A_3 W_s \): the substring includes only three elements and the length of substring is greater than 5 (e.g. w1 w2 w3 w1 w3 w2);

\(^1\)Sometime the problem still exists, because some applications allow users to open the tabs or documents with the same title.
3. LOG-BASED LONGITUDINAL STUDY

- $A_4Ws$: the substring includes only four elements and the length of substring is greater than 7 (e.g. w1 w2 w3 w4 w1 w2 w4 w3 w1);

We use regular expression to parse the active windows sequences. Figure 3.19 and 3.20 show that the results of the mean of percentage of three types interaction pattern under $NTChild$ condition for each single monitor and dual monitors user. We observed some important results from this data. First, users frequently switched among a small number of windows ($A_2Ws + A_3Ws + A_4Ws$ are over 60.0%). Second, we observed that there were small differences among single monitor users in interaction pattern $A_3Ws$ and $A_4Ws$ and there were only small differences among dual monitors users in all interaction patterns. Third, there was a small difference between single monitor and dual monitors users in interaction pattern (Figure 3.21).

![Figure 3.19: Mean of the percentage of three types interaction pattern under $NTChild$ condition for each single monitor user. Error bars show standard error.](image-url)
3.4 Results And Analysis

Figure 3.20: Mean of the percentage of three types interaction pattern under YTChild condition for each dual monitors user. Error bars show standard error.

Figure 3.21: Mean of the percentage of three types interaction pattern under NTChild condition. Error bars show standard error.
3. LOG-BASED LONGITUDINAL STUDY

3.5 Conclusion

A log-based longitudinal study was described in this chapter. We developed a log tool called WindowsOSLog to record users window management activity in mainstream Windows operating system. WindowsOSLog was then installed to 26 participants during 5-weeks. After 5 weeks, we collected data from users and analyzed this data to gain knowledge on how users manage their windows and tasks.

This chapter lays the foundation for further research in this area. As the future work, we would like to use the understanding gained from the log data and questionnaire investigation to help us to design new window and task switching techniques in order to improve people to more efficiently and conveniently interact with computer.
Chapter 4

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

This chapter proposes Push-and-Pull Switching, a window switching technique using window overlapping to implicitly define groups. Push-and-Pull Switching further allows to switch between groups and restack the focused window to any position to change its group affectation. The technique was evaluated in an experiment showing that Push-and-Pull Switching allows to improve 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 while they found it easy to understand and use.

4.1 Introduction

Window switching is one of the most frequent tasks of any window manager happening several hundred times per day (takes place on average once every 20.9s on large displays Hutchings et al. (2004)). 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 by using the mouse to click on a region of the window of interest, Alt+Tab /
4. PUSH-AND-PULL SWITCHING: WINDOW SWITCHING BASED ON WINDOW OVERLAPPING

Cmd+Tab that consists in using a key combination to navigate the list of windows or applications, Taskbar/dock that 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 all visible at once.

These techniques allow 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. Smith et al. (2003), grouping windows by application confuses many users because application windows may not be related to the same task Robertson et al. (2004). Virtual desktops and Task gallery Robertson et al. (2000) 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 Robertson et al. (2004) and GroupBar Smith et al. (2003) allow to interact with windows from multiple groups at once without affecting the group structure. However the user has to plan in advance the number of groups needed to accomplish his tasks and affect each window to a group to leverage the benefit of the window grouping system. WindowScape Tashman (2006) proposes to automatically create groups by taking photograph-like snapshots each time a window is maximized or minimized. 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 a same activity visible by trying to avoid or minimize the amount of overlapping. As a result groups are implicitly created by the user. The following scenario illustrates such a situation:

Peter is editing some code using his favorite editor and uses a terminal window to compile and run his program. The terminal window is positioned to avoid occlusion of any line of code displayed in the editor window. He uses a third maximized window to repeatedly search information on 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 Internet window to one group and the editor and terminal windows to a second group but he does not want a strict separation.
4.1 Introduction

between these groups as he wants to be able to read some information on the Internet window while editing his code (Figure 4.1). He could use Scalable Fabric, GroupBar or WindowScape to display the two groups at once but he does not want to explicitly manage groups nor he wants to waste some display space required to operate these techniques.

Figure 4.1: A typical windows organization with two closely related windows in the foreground. To switch focus to the navigator window in the background and then give back the focus to the two related windows is a tedious task requiring multiple switching operation with current user interface switching techniques.

To address these limitations, we have designed Push-and-Pull Switching, a switching technique based on window overlapping to implicitly define groups and help users quickly switch between them. We first describe the windows stacking model. We then giving an overview of Push-and-Pull Switching technique, after this 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 and conclusions.
4. PUSH-AND-PULL SWITCHING: WINDOW SWITCHING BASED ON WINDOW OVERLAPPING

Figure 4.2: Push-and-Pull Switching example representing an initial layout (a) with window W7 active (represented by the letter A). 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).

4.2 Push-and-Pull Switching

In the following we describe how the Push-and-Pull Switching technique is used to create and switch between groups. We then present a variation of the algorithm to easily change the Z order of the focus window.

4.2.1 Group Switching

When invoked, our algorithm first creates groups of non overlapping windows. 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. Faure et al. (2009) 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%.

The algorithm of computing windows group as following:

<table>
<thead>
<tr>
<th>Algorithm 2: Compute the grouped windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> the ungrouped windows</td>
</tr>
<tr>
<td><strong>Output:</strong> the group windows</td>
</tr>
<tr>
<td><strong>foreach</strong> <em>Window</em> <em>W</em>&lt;sub&gt;i&lt;/sub&gt; in the windows stacking from foreground to background** do</td>
</tr>
<tr>
<td>flag←false</td>
</tr>
<tr>
<td>Create a new group <em>G</em>&lt;sub&gt;k&lt;/sub&gt; and add <em>W</em>&lt;sub&gt;i&lt;/sub&gt; to this group</td>
</tr>
<tr>
<td><strong>foreach</strong> <em>Window</em> <em>W</em>&lt;sub&gt;j&lt;/sub&gt; in the windows stacking from <em>W</em>&lt;sub&gt;i&lt;/sub&gt;+1 to background** do</td>
</tr>
<tr>
<td>Create temporary rectangle group TG</td>
</tr>
<tr>
<td><strong>foreach</strong> <em>Window</em> <em>W</em>&lt;sub&gt;t&lt;/sub&gt; in the group <em>G</em>&lt;sub&gt;k&lt;/sub&gt; do</td>
</tr>
<tr>
<td>if <em>W</em>&lt;sub&gt;j&lt;/sub&gt; has intersection with <em>W</em>&lt;sub&gt;t&lt;/sub&gt; then</td>
</tr>
<tr>
<td>Add intersection rectangle <em>R</em>&lt;sub&gt;jt&lt;/sub&gt; to the group TG</td>
</tr>
<tr>
<td>if <em>TG</em> ∩ <em>W</em>&lt;sub&gt;j&lt;/sub&gt; ≥ setvalue * W*&lt;sub&gt;j&lt;/sub&gt; then</td>
</tr>
<tr>
<td>flag←false break</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>if !flag then</td>
</tr>
<tr>
<td>Add <em>W</em>&lt;sub&gt;j&lt;/sub&gt; to this group <em>G</em>&lt;sub&gt;k&lt;/sub&gt;</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

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 calls the above algorithm to create groups. Pushing and pulling consist in 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 foreground. Pushing a group does the opposite.
4. PUSH-AND-PULL SWITCHING: WINDOW SWITCHING BASED ON WINDOW OVERLAPPING

Figure 4.3: Example for restacking the focused window. Figure (a) represents the initial layout in which window W7 is active (represented by the letter A) 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 Ctrl and Shift keys activates window W5 (d).

During push and pull operations, only the first group created (closer to foreground) can be pushed or pulled (Figure 4.2). Upon release of the Ctrl key, the window with the highest Z order gets the keyboard focus. We chose to give the keyboard focus to that window as it is the last accessed one within the group and we consider the user more likely to interact with it. In the following sections, we will refer to this frontmost window with the keyboard focus as the active window.

4.2.2 Restacking the Focused Window

Push-and-Pull Switching can also be used to change the Z order of the focused window. Upon invocation using Ctrl+Shift keys instead of Ctrl, our algorithm creates groups by considering only the windows in intersection with the focused window (considering all windows would end with pushing or pulling operations with no visible effect). Pushing or pulling moves it in front or behind the related group. Releasing Ctrl+Shift gives the keyboard focus to the window with the highest Z order from the frontmost group (Figure
4.3 Experiments

Moving a window to the background is a feature proposed by some X window managers but no modern window manager allows to precisely define the Z order of a window. This technique can be used to affect a window to another group and it is also interesting for restacking a window 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.

4.3 Experiments

We conducted an experiment to compare the performance of Push-and-Pull Switching to other techniques (Direct pointing, Taskbar and Alt+Tab) in different scenarios.

4.3.1 Experiment 1: Group Switching

4.3.1.1 Apparatus

The Push-and-Pull Switching technique was implemented in C# on Windows XP/Vista. We used a PC running Microsoft Windows XP using a 22 inch LCD monitor with a 1680 × 1050 resolution. The mouse is a standard optical one with two buttons and a clickable wheel that can be used as a middle button.

4.3.1.2 Participants

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 reported to spend at least 8 hours a day working on a Microsoft Windows system. Most participants reported to mainly use Direct pointing and Taskbar (with the group by application option disabled) and three reported to often use Alt+Tab.

4.3.1.3 Experimental Design

A repeated measure design was used. The independent variables were switching technique (SwT) with the three switching techniques available on Windows XP (Taskbar,  

\footnote{http://code.google.com/p/push-and-pull-switching}
4. PUSH-AND-PULL SWITCHING: WINDOW SWITCHING BASED ON WINDOW OVERLAPPING

Figure 4.4: The initial layout for the 4 scenarios used in experiment one to compare the switching time between Taskbar, Alt+Tab, Direct pointing and Push-and-Pull Switch-ing. The letter A represents the active window. Windows are numbered according to their stacking order. Window numbers were replaced by real application windows in the experiment.

Alt+Tab, Direct pointing) and the Push-and-Pull Switching technique, and Scenario with four levels. The first scenario consists in switching back and forth between window W7 and window W8 (Figure 4.4a). 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 4.4b. The third scenario consists in switching back and forth between the group represented by windows (W6, W7, W8) and window W5 in Figure 4.4c. 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 4.4d. Each scenario consists of 8 windows. 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 run each scenario 10 times with the four switching tech-
niques before moving to the next one. The scenario were run from $a$ to $d$ but the techniques were counter-balanced across participants using a balanced Latin square. Before starting the experiment, participants had a 5-10 minutes training period to get used to the switching techniques and windows content. Before each scenario, participants were clearly explained the layouts they had to switch between. The experiment lasted approximately 25 minutes.

4.3.1.4 Procedure

The task was to switch back and forth between windows presented in different scenarios. Each trial started with an initial layout (Figure 4.4). After pressing the space bar, the task was to switch to a specific layout and then switch back to the initial layout before pressing 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 he gave no indication how to correct it.

4.3.1.5 Results

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

During the experiment we observed that participants were more error prone with $\text{Alt+Tab}$ which can explain the more important 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. The error rate for
Alt+Tab is equal to 5% while it is equal to 2% for Taskbar, 1% for Direct pointing and 0% for Push-and-Pull Switching. Overall, participants had no problem understanding and using Push-and-Pull Switching.

![Figure 4.5: Mean switching time for Switching Technique and Scenario. Error bars represent 95% confidence interval.](image)

In this experiment, we focused on the 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.
4.3.2 Experiment 2: Restacking the Focused Window

4.3.2.1 Apparatus and Participants

This second experiment was run after the first experiment with the same participants and the same hardware configuration.

4.3.2.2 Experimental Design

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.

4.3.2.3 Procedure

The task was to change the Z order of the focused window represented in Figure 4.6a to get the layout represented in Figure 4.6b. A trial started and finished by pressing the space bar. Participants had to successfully get the target layout before moving to the next trial. The task was repeated 10 times.

4.3.2.4 Results

The dependent variable is restacking time as the time measured between two presses on the space bar. Repeated measures analyses of variance showed a significant main effect for SWT ($F_{3,21}=5.11$, $p=0.008$) on restacking time. Pairwise comparisons found significant difference between Push-and-Pull (1.4s) and Alt+Tab (3.0s) ($p=0.026$) and Push-and-Pull and Taskbar (2.4s) ($p<0.001$). 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 found Push-and-Pull switching as the most direct and intuitive technique to perform this task and most commented that the technique mimics how people classify files in piles of documents.
4. PUSH-AND-PULL SWITCHING: WINDOW SWITCHING BASED ON WINDOW OVERLAPPING

4.4 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 mechanist, 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 how useful they found it to be. In addition, the application recorded any switching operation and the corresponding technique used.

Single monitor users had on average 5 windows simultaneously opened on their desk-
top. They used mainly Direct pointing (47%) and Taskbar (36%) whereas Alt+Tab was used 2% of the time and Push-and-Pull Switching 15%. Dual monitors users kept on average 8 windows simultaneously opened on their desktop. They used mainly 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 represents 10% of Push-and-Pull Switching invocations.

Participants reported to 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 they know is just behind another. Using a 5 points Likert scale, participants rated the technique as useful (averaged response = 3.9) (1=disagree, 5=agree) and easy to use (4.1). Half participants (mostly single monitor users) reported to reposition and resize windows more frequently than usual to leverage full benefit from Push-and-Pull Switching.

Participants used a 5 points Likert scale to answer the following questions:

---

**Figure 4.7:** Mean switching time for Switching Technique. Error bars represent 95% confidence interval.
4. PUSH-AND-PULL SWITCHING: WINDOW SWITCHING BASED ON WINDOW OVERLAPPING

Table 4.1: User satisfaction averages for on a five point scale where 1 = useless, 5 = useful.

<table>
<thead>
<tr>
<th>Questionnaire Item</th>
<th>Average Response (1 = useless, 5 = useful)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usefulness</td>
<td>3.88</td>
</tr>
<tr>
<td>Comments about the window that get focus once Ctrl is released: is it the right window</td>
<td>3.63</td>
</tr>
<tr>
<td>Easiness</td>
<td>4.13</td>
</tr>
<tr>
<td>It is useful to be able to rearrange some windows to finish one task with less switching time</td>
<td>3.88</td>
</tr>
<tr>
<td>It is useful to be able to switch more windows at once</td>
<td>4.38</td>
</tr>
<tr>
<td>It is useful to be able to restack the focused window</td>
<td>3.63</td>
</tr>
<tr>
<td>It is useful to be able to work with more tasks simultaneously</td>
<td>3.63</td>
</tr>
</tbody>
</table>

4.5 Application

The users longitudinal studies provide important finding that most users rely heavily on mouse-based window switching methods (Taskbar and Direct pointing). In order to effectively use Push-and-Pull Switching and enhance the interaction function of mouse, we would like to add this technique to the common mouse. Figure 4.8 is a typical mouse with five keys (There are two shortcut keys in the right side of the mouse), this type of mouse is becoming increasingly common (Logitech/Microsoft). In general, these two shortcut keys are used to implement forward and backward operations for browsing the web or accessing folder, some advanced mice even allow users to define the function for these two shortcut keys by themselves. Before they were used for navigating within application, now we can integrate Push-and-Pull Switching technique to enhance their functions so that users can use them to switch windows, using up key to implement Push operation and down key to
implement \textit{Pull} operation. During the interview, we told users this idea, most users were very interested in this idea and thought this would be very useful for them.

Figure 4.8: Typical mouse with five keys, there are two shortcut keys in the right side of the mouse. In the default state, they are used to implement forward and backward operations.

4.6 Conclusion

Push-and-Pull Switching provides a lightweight alternative to other grouping techniques. We demonstrated with our \textit{in situ} and laboratory user studies 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. In our experiments, participants were explained how to utilize Push-and-Pull switching. Future work includes adding visual feedback for novice users to help visualize groups by changing the color of the windows border when the technique is invoked.
Chapter 5

Stack Scanning Rules!

This chapter presents stack scanning, a window switching technique based on a widget that combines generalized scrolling and crossing to control the stacking order of layers of visible windows. With stack scanning, the visual information for each window is maximized as each window remains at its original size while the ordering by frequency is preserved. We conducted an experiment to compare the performance and error rate of stack scanning technique to other four common window switching techniques under a variety of visual factor conditions (e.g. the number of windows, their visual similarity). Results showed that stack scanning is faster than other techniques when the number of windows is high and the visual similarity among windows is important. They also showed that Taskbar is the best choice with a small number of windows condition, regardless of other visual factor conditions, and for users who always maximize their windows, Alt+Tab is the best choice when the number of windows is important.

5.1 Introduction

Users of desktop computers are increasingly turning to large displays and multiple-monitor setups. A key advantage of additional screen space is the ability to keep more windows open simultaneously, reducing the amount of resizing, repositioning, and other window-management activity (Hutchings has shown that users have more than eight windows open more than 78% on the time) Hutchings et al. (2004); Robertson et al. (2005). However, additional windows also mean that more windows are competing for users’ attention. This can make it harder for users to switch the desired target. For many window
interaction operations (e.g. copy-and-paste), one important thing is to find the desired window. The speed and accuracy to find the window of interest can directly affect the subsequent interaction operations. But sometimes it is very difficult to find the desired window, and this process can become a laborious visual task when the number of windows becomes important. The difficulties for finding the desired window come from two main reasons: 1) window overlapping, which hides window information; and 2) the visual similarity of windows. However, the overlapping model is now the dominant model for modern window systems and this situation will not disappear with the advent of larger displays Chapuis & Roussel (2007); Hutchings & Stasko (2004).

Window switching is one of the most frequent task of any window manager and can occur several hundred times per day (on average, once every 20.9s Hutchings et al. (2004) on large displays). Switching windows first requires to find the desired window, and second, bringing it to the foreground. There exist different techniques for window switching where the most common techniques are Direct pointing, Taskbar/dock, AltTab/CmdTab and Exposé Xu & Casiez (2010).

However, many researches have hinted that traditional window switching techniques have potential usability problems. Hutchings has hinted that it becomes difficult to recognize buttons on Taskbar when the number of windows is high Hutchings & Stasko (2003). Kumar et al. have observed that Alt+Tab is very efficient when the number of windows is low Kumar et al. (2007), but researchers have also labelled the method ’tedious’ Grudin (2001) and reported that a very small percentage of users regularly use the Alt+Tab key combination for window switching Czerwinski et al. (2003). A previous research Dragicevic (2004) has hinted that some window switching techniques (Taskbar, Alt+Tab, Exposé) are not wholly satisfactory because they require switching back and forth between two different representations of the same window set (e.g. Exposé switches back and forth between thumbnails and windows, and Taskbar 1 switches back and forth between buttons and windows).

Although many window switching techniques have been proposed, only a few of them have been evaluated. The biggest challenge is that window switching techniques are heavily affected by the window layout. Kumar et al. proposed an experiment to compare EyeExposé to Alt+Tab, Exposé and Taskbar Kumar et al. (2007). However they only considered the number of windows condition and ignored visual similarity and window

---

1Windows 7 uses the big icon to replace the traditional button on the Taskbar
5. STACK SCANNING RULES!

layout. As a result, it is hard to assess the extent to which addressed problems actually exist in window switching practices.

To reduce the number of switching operations, interaction techniques have been proposed to explicitly or implicitly define groups (some previous work also refer to task) (e.g. virtual desktop managers). Although grouping mechanism can reduce partially the problem of switching operations, users still need to switch windows within one group. So window switching techniques continue to be important when window grouping is used.

In this chapter, we propose a window switching technique, stack scanning, a window switching technique based on a widget that combines generalized scrolling and crossing to control the stacking order of layers of visible windows. We describe an experiment to compare the performance and error rate of stack scanning technique to other four commonly window switching techniques (Direct pointing, Taskbar, AltTab and Exposé) under a variety of combination conditions (the number of windows, visual similarity and windows layout). Results from the experiment showed that stack scanning was faster than other techniques when the number of windows is high and the visual similarity among windows is important. They also show that Taskbar was the best choice when the number of windows is small, regardless of other visual factor conditions, and for users who always maximize their windows, Alt+Tab was the best choice when the number of windows is important.

5.2 Stack Scanning

Our technique aims are:

- Providing as much visual information as possible for each window (that is no creation of thumbnails) to help the visual search

- Increasing the user expectancy to find a given window at a given position to help the visual search (if the user knows a window is at the bottom left then he will look at bottom left)

- Reducing the number of steps to find and select the window of interest to reduce the search time and the selection time
5.2 Stack Scanning

- Preserving an ordering by frequency: the most recent windows come first. This is intended to reduce the search time (the user expects to get a window more quickly if it was recently used)

The aims of our proposal are preserved through the scanned stack algorithm. When invoked, all windows in the stacking are scanned in decreasing Z-order, from foreground to background, and layers are created at the same time. Windows are added to the current layer as long as they remain visible. Windows are taken in the stack order (when the overlap of a window is less than a given allowable overlap within the current layer, we refer as that the window is visible). The overlap for a given window is computed as the percentage of pixels which are covered by the windows with the higher Z-order in the current stack. The default overlap threshold is set to 75%, that is, when the overlap of a window is less 75%, the window is considered to be visible for users and it can be easily recognized. With this algorithm, each layer includes all the windows which are ‘visible’. If the layer is brought closer to the foreground, all windows within the layer are visible for users. This is different from Xu & Casiez (2010), for the algorithm of Push-And-Pull Switching, once a window overlaps with the current group (or the overlap of a window is over the allowable overlap), a new group will be created and other windows which have the lower z-order than this window in the stack will be not computed for others group (they will be computed for the new group). For our algorithm, when a window is not visible, we will continue to scan other windows which have the lower z-order than this window until the bottom of stack.
Algorithm 3: Compute the scanned stack

**Input**: the unscanned stack  
**Output**: the scanned stack

**foreach** Window $W_i$ in the stack $S_t$ taken from foreground to background  
do  
Create a new layer $L_j$ and add $W_i$ to the layer $L_j$  
**foreach** Window $W_k$ in the stack $S_t$ from $W_{i+1}$ to background  
do  
Compute the overlap of $W_k$ with $L_j$  
if The overlap of $W_k$ is less than a given allowable overlap parameter  
then  
| Add $W_k$ to the layer $L_j$;  
end  
end  
Get a visible layer $L_j$ and remove all windows from the current layer in the stack $S_t$  
Update $S_t$ and $j++$  
end  

* $W_{i+1}$ is the next window of $W_i$ in the stack $S_t$  

With this technique, the visual information for each window is maximized as each window remains at its original size. The expectancy to find a given window at a given position are high as windows positions do not change (compared to Exposé), and then the number of steps to get the window of interest is reduced as the scanned stack reduces the size of the stack compared to the traditional Alt+Tab (but at the cost of a pointing action). The windows are still ordered from the most recently used to the least recently used. In this way, we hope to reduce the time to find the window of interest and then select it.

Stack scanning is invoked using the mouse wheel. A long click on the mouse wheel (250 ms timeout Faure et al. (2009)) brings up the widget which is made of buttons (each button represents one layer, the buttons are arranged from top to bottom by the Z-order), and the above algorithm is invoked to create layers. Moving the mouse to cross one button brings all windows within the layer closer to the foreground. Once the mouse leaves the widget, the widget disappears, the window with the highest Z-order of that layer receives the keyboard focus (Figure 5.1).
5.2 Stack Scanning

Figure 5.1: Example for stack scanning. Figure (a) represents the initial layout with window w5 focused (represented by the letter F). Windows are numbered according to their stacking order. Pressing the mouse wheel (250 ms timeout) computes the following layers: (W5, W4), (W3) and (W2, W1). Moving the mouse to (W2, W1) layer and leaving the widget, all windows within this layer are brought closer to the foreground, the window with the highest Z-order receives the keyboard focus.

Figure 5.2: Example of stack scanning extend function. Figure (a) represents an initial layout with where window W5 is focused (represented by the letter F) and where windows are numbered according to their stacking order, pressing the mouse wheel, bringing up the widget. Moving the mouse to cross the button, bring all windows within the layer closer to the foreground (b), the mouse leaves from the left side of widget, only bringing the clicked window within the layer to the foreground (c), the mouse leaves from the right side of widget, bringing all windows within the layer closer to the foreground (d).
5. STACK SCANNING RULES!

When the mouse leaves the widget, the mouse can be moved from the right side or the left side of the widget. To conveniently and efficiently allow users to use stack scanning technique, we expand the stack scanning function to allow two actions (out from right side or left side, visual feedback is added to prompt that the mouse is at right or left side). Moving the mouse to get out from the right side of the widget allows the action described before and getting out from the left side of the button allows the following operation: when moving the mouse to cross one button, the action is the same, all windows within this layer are brought closer to the foreground, but the mouse leaves the widget from the left side, users click the window of interest and only this window is brought to the foreground. The other windows go back to their original stack position (Figure 5.2 (c)).

5.3 Window Switching Time Model

Each window switching technique requires first a visual search and second an action to select the window of interest. Then, the interaction time should be $T_t(n, op) = T_v(n) + T_a(op)$, where $T_t(n, op)$ is the total time for the interaction technique; $T_v(n)$ is the visual search time as a function of the number of windows; $T_a(op)$ is the time to select the window of interest depending on the technique used.

The visual search usually involves two kind of search which may or may not be related to each other (referred to as "mixed visual"). This is because window switching techniques represent windows as various forms which are often composed of image and text. Thus, search is both over the images (referred as "image visual") and over the text attached to the image, termed "semantic search". For some window techniques (such as Alt+Tab, Taskbar), the window of interest may have the same icon as other windows simultaneously present on the display. This makes necessary to read the text attached to the window most or all the time in order to compensate for the fact that the icon alone is ambiguous in that it may not represent the target window.

The time to perform this visual search depends on the number of items (which time can be predicted by Hick’s law) and the degree of visual similarity between these different windows (more similarity requires more time). The time to select the window of interest depends on the technique used. It can be a number of keystrokes or a target selection task predicted by Fitts’law.
5.3 Window Switching Time Model

There is some evidence Byrne (1993); Fisher et al. (1989) that mixed search is actually a two-stage process: the user uses an image search to narrow down the semantic search. For example, the user first finds the icon on the Taskbar with icons that match the target window and then reads the text only on those icons. Then, visual search time should be \( T_v(n) = T_i(n) + T_s(k) \), where \( T_v(n) \) is the total time for the visual search (mixed search), \( T_i(n) \) is the image search time as a function of the number of windows, and \( T_s(k) \), the semantic search time as a function of the average number of images matching the target window image \( k \). The semantic search is a standard self-terminating serial search Byrne (1993), since there is no visual guidance for the search, one can reasonably expect that the user will read each text attached to the window and decide if it is the target window, so in that case, search time should be a function of the average number of windows to search, which is \( n/2 \). That is, \( T_s(k) \) for the number of windows \( k \), should be a linear function of the form \( T_s(k) = m_s \ast (k/2) + b_s \). If the user knows the text of target window, the decision process should be quicker, and therefore the slope \( m_s \) shallower. But if the user only knows partial or no characters of the target window text, and meanwhile other windows have similar characters of their text, the slope \( m_s \) will be larger.

Times for \( T_{i}(n, op) \) and \( T_{a}(op) \) can be empirically estimated, and then it should be possible to estimate \( T_v(n) \) by simple subtracting \( T_a(op) \). So we can compare \( T_a(op) \) and \( T_v(n) \) for each window switching technique to know which is the dominant time factor, so as to improve them and help us to design the new window switching techniques. It can also be possible to estimate \( T_i(n) \) time by \( T_v(n) - T_s(k) \), to know which image type is more effective to search (icon vs. thumbnail).

Taskbar involves a visual search (on Windows, only the icon and the begin of each title are visible in the Taskbar) and then a target selection.

Direct pointing includes two cases: first when the window of interest is directly visible so the time can be easily predicted by Fitts’ law; another way is to minimize (in this case, moving is less used) the current window and selecting the desired window. It involves a serial visual search and window minimization.

For Alt+Tab, the visual search depends on the visual content provided: images or/and text. The selection is done by a number of keystrokes where the maximum number is the total number of windows minus one (\( n \) is the number of windows, the average number of keystrokes is \( \sum_{i=1}^{n} (i - 1)/(n - 1) \)).
5. STACK SCANNING RULES!

Expose requires first pressing a key on the keyboard to call it, then a visual search and a target selection.

For Stack scanning, first pressing the mouse wheel to call it, and moving the mouse to bring windows closer to the foreground (the distance to cover with the mouse depends on the layer order), then a visual search and a target selection.

5.4 Experiment

We conducted an experiment to compare the performance and user preferences for the Taskbar (Figure 5.6), Alt+Tab (Figure 5.5), Direct pointing, Exposé (Figure 5.4) and stack scanning (Figure 5.3) techniques in different scenarios.

![Figure 5.3: Stack scanning with 7 layers, using the original windows as representation.](image)

![Figure 5.4: Exposé view of 8 windows, representing them as thumbnails.](image)

5.4.1 Visual Factors For Window Switching

There are many visual factors that can impact the performance of window switching techniques, including the number of windows, windows layout, visual similarity.

*Number of windows.* Probably the best-studied factors in window switching techniques is this factor.

*Windows Layout.* The performance of window switching techniques heavily depends on the windows layout, which is also the most difficult to consider. To quantify this factor,
Figure 5.5: *Alt+Tab* represents windows as icons.

Figure 5.6: *Taskbar* in each of the 8, 12 and 16 windows conditions with NVS, representing windows as buttons.

we use three factors: *window size*, *amount of overlapping* and *window Z-order* to define a windows layout.

- **Window size.** It can directly impact the time of visual search and selection. We divided windows into three categories by window size: *SmallWindow* (the window size is less than 25% of screen resolution), *NormalWindow* (between 25% and 75%), *LargeWindow* (greater than 75%);

- **Amount of overlapping.** It may impact the visibility of windows, and this can directly impact the performance of visual search. Meanwhile the amount of overlapping determines the number of layers in *stack scanning* technique;

- **Window Z-order.** It can affect the duration of operations when the technique depends on this factor (such as, *Alt+Tab*)

*Visual similarity.* Window switching techniques use various visualizations to represent windows (such as, icon+title, thumbnail), different visualizations have different visual stimulus for users. We divide visual similarity between windows into three types: **UN-SIMILAR** (NVS) (different application type, the title, icon and the content of window are
5. STACK SCANNING RULES!

different, and it can be easy to distinguish them), LOW SIMILAR (LVS) (same application type, the icon is the same, and the title characters are partial same, the content is different), HIGH SIMILAR (HVS) (same application type, the same icon and most of the title characters are similar, the content of window is also similar) (Figure 5.7).

![Figure 5.7](image)

Figure 5.7: The visual similarity between (1) and (2) is NVS, (1) and (3) is LVS and (1) and (4) is HVS.

5.4.2 Hypothesis

H1 The stack scanning is expected to reduce the search and selection time compared to other techniques, especially when the number of windows is high and the visual similarity is important because the other techniques may fail due to the difficulty to identify a specific window (Exposé, Taskbar, Alt+Tab, Direct pointing) or when the number of keystrokes becomes too high (Alt+Tab).

5.4.3 Apparatus

The stack scanning technique was implemented in C# on Windows XP/Vista. We used our implementation of an Exposé clone to perform the experiment in a Windows environment and our implementation is similar to the Apple’s Exposé (Figure 5.4). We used a PC running Microsoft Windows XP using a 22 inch LCD monitor with a 1680 × 1050 resolution. The mouse is a standard optical one with two buttons and a clickable wheel that can be used as a middle button.
5.4 Experiment

5.4.4 Participants

10 people (6 male, 4 female) with a mean age of 27.4 (SD=2.95) participated. They were recruited from the university (2 civil engineer, 1 mechanic, 1 electronic engineer, 1 chemical engineer and 5 computer scientists) and said they spent at least 8 hours a day working on a Microsoft Windows system. The female participants reported that they mainly used Direct pointing and Taskbar (with the group by application option disabled) and four reported that they often used Alt+Tab.

5.4.5 Experimental Design

A repeated measures within-subjects design was used. The independent variables were Technique with 5 levels (Direct pointing, Exposé, Taskbar, Alt+Tab, stack scanning), number of windows Num with 3 levels (8, 12, 16), distribution of window size DISSIZE with 4 levels (SmallWindow, NormalWindow, LargeWindow, the proportion of three types of window in the total number of windows, 2:1:1 (Size211), 1:2:1 (Size121), 1:1:2 (Size112), 0:0:1 (Size001) (all windows are maximum size, but they have not been maximized, because when the window is maximized, it can be not moved by the mouse)), For example, when the number of windows is 8, Size211 shows that four of them are SmallWindow, two of them are NormalWindow and other two windows are LargeWindow. Amount of overlapping (AmOverlap) with 4 levels (0%, 25%, 50%, 75%). The visual similarity (VS) between windows with 3 levels (NVS, LVS, HVS). The size of the match set MATCH levels (MLs) with 3 levels (1, 2, or 3), the match level (1, 2, or 3) had a different meaning in each number of windows, With Num equal to 8, the match set is 1, 2, or 4, corresponding to the match levels of 1, 2, and 3. With Num of 12 and 16 match sets of 1, 3, 6 and 1, 4, 8, respectively. The MLs defines the number of windows in total Num (distractors) is similar with the target window. For example, when Num is eight, MLs is the third level (match set is four) and VS is HVS, this means that four of eight windows have high similar with the target window.

The two main factors are the window switching techniques (Taskbar, Alt+Tab, Direct pointing, Exposé, stack scanning) and the scenario conditions (Num, MLs, DISSIZE, AmOverlap). The main measure is the completion time and error rate to perform a window switching technique to find the window of interest in different scenarios. Our experiment used real application windows such as Notepad, Word and PDF document as
5. STACK SCANNING RULES!

the target windows. We believed that participants would easily be able to recognize real application windows. In the Taskbar condition, the number of windows on the Taskbar never exceeded a threshold that would cause it to add a second line with a scroll button and the group by application option was disabled.

When the distribution of window size condition is SIZE001 (all windows are maximum size, the overlap of any window (except the foreground window) is 100%, the AMOVERLAP condition only has one level (75%), so in order to ensure a balance design, the (SIZE001) condition will be handled as a separate part of our main experiment, and we analyzed it separately.

The experiment consists of 1620 \((\text{NUM} \times \text{VS} \times \text{MLS} \times \text{DISSIZE} \times \text{AMOVERLAP} \times \text{TECHNIQUE})\) (Main experiment) + 135 \(((\text{NUM} \times \text{VS} \times \text{MLS} \times \text{TECHNIQUE}))\) \((\text{DISSIZE} \text{is SIZE001})\) (Second experiment) trials. Orders for techniques, number of windows, visual similarity and amount of overlapping conditions were counter-balanced across participants using a balanced Latin-square. For distribution of window size condition, each trial first used the distribution of SIZE001, order for other three distribution conditions were counter-balanced across participants following a Latin-square. Taking into account the total time for this experiment (over 5 hours), we divided the experiment into three parts by the number of windows.

The performance of some techniques depends on the target window’s position in the stack (Z-order) and the performance difference would be great between different window Z-order in the stack (e.g. for Alt+Tab, the difference can be important between pressing one time Tab key and pressing ten times Tab key to get the target window). So if the target window is randomly presented in the stack, the performance may be very volatile for a small number of trials (4 trials).

To overcome this issue, the target window can be randomly positioned in the stack but the average Z-order for one subgroup trial has to respect the theoretical median Z-order. We used the mean time of that subgroup to compute one successful trial. For example, participants used Alt+Tab to switch windows, the number of windows is 8 and the theoretical average number of pressing the tab keyboard is \(\sum_{i=1}^{8}(i-1)/(8-1) = 4\) (median Z-order position), the target window is randomly presented in the stack, but the average number of pressing Tab key in one subgroup should be 4 times. Stack scanning also depends on the target window Z-order. So the number of layers is very important
parameter for stack scanning. With 8 windows, the number of layers of stack scanning is 5 on average, and for 12 and 16 windows, it is 7 and 9 on average.

### 5.4.6 Procedure

A trial consists of a series of five window switching techniques. During each trial, the participants' task was to find the window of interest by using five window switching techniques in the different scenarios. Before starting each part of the experiment, participants had a 15-20 minutes training period to get used to the switching techniques and windows content (participants allowed to train as long as they want, training until they feel at ease with the technique). Participants are instructed to "perform as fast as possible without error". Participants first pressed a "Layout" button to initialize a windows layout, and the target window was presented to participants (the title + icon), then pressing the "start" button to start a trial, participants were asked to find the target window with one switching technique. After finishing this process, they were asked to press the space bar to start the next trial until (s)he had completed 4 successful trials. Then the system gave the prompt to change the technique to continue. When participants had successfully performed five techniques in one scenario, the system gave the prompt for another scenario. We recorded the amount of time it took to select the target window, starting from the time a participant clicks the "start" button. If the participant switched to a wrong window, we recorded an error.

### 5.5 Results

The dependent variables are the switching time ($ST$ in seconds) and the error rate. The switching time is measured from the time the participant clicked the "start" button to the time the participant brought the window of interest to foreground. We first performed a repeated measures analysis of variance (ANOVA) for our main experiment, participant as a random effect factor and $\text{Num}$, $\text{VS}$, $\text{MLs}$, $\text{DisSize}$, $\text{AmOverlap}$ and $\text{Technique}$ as fixed effect factors. We first analyze the main experiment, then the second experiment.
5. STACK SCANNING RULES!

5.5.1 Switching Time

5.5.1.1 Main Experiment

The results of ANOVA showed a significant main effect for Num (F_{2,18} = 59.19, p<0.001), VS (F_{2,18} = 54.96, p<0.001), AmOverlap (F_{3,27} = 147.16, p<0.001) and Technique (F_{4,36} = 42.13, p<0.001) on switching time, and MLS (F_{2,18} = 3.46, p=0.054) and DisSize (F_{2,18} = 2.44, p=0.116) were not significant on switching time.

We use the LSD (Least Square Difference) test with \( \alpha = 0.05 \) for the pairwise comparisons between techniques (We mainly focus on techniques). Table 6.1 shows the results of those tests in detail.

Table 5.1: Pairwise comparisons between techniques condition on switching time. A cell contains the means difference and the lower and upper bound of the confidence interval. Underlined cells are significant.

<table>
<thead>
<tr>
<th></th>
<th>AltTab</th>
<th>Direct pointing</th>
<th>Expose</th>
<th>Stack scanning</th>
<th>Taskbar</th>
</tr>
</thead>
<tbody>
<tr>
<td>AltTab</td>
<td>0</td>
<td>-2.060</td>
<td>-0.237</td>
<td>0.212</td>
<td>0.489</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-2.589</td>
<td>-0.872</td>
<td>-0.209</td>
<td>0.040</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-1.531</td>
<td>0.398</td>
<td>0.633</td>
<td>0.939</td>
</tr>
<tr>
<td>Direct p.</td>
<td>2.060</td>
<td>0</td>
<td>1.823</td>
<td>2.272</td>
<td>2.549</td>
</tr>
<tr>
<td></td>
<td>1.531</td>
<td>0</td>
<td>1.074</td>
<td>1.744</td>
<td>2.010</td>
</tr>
<tr>
<td></td>
<td>2.589</td>
<td>0</td>
<td>2.572</td>
<td>2.799</td>
<td>3.088</td>
</tr>
<tr>
<td>Expose</td>
<td>0.237</td>
<td>-1.823</td>
<td>0</td>
<td>0.449</td>
<td>0.726</td>
</tr>
<tr>
<td></td>
<td>-0.398</td>
<td>-2.572</td>
<td>0</td>
<td>0.063</td>
<td>0.435</td>
</tr>
<tr>
<td></td>
<td>0.872</td>
<td>-1.074</td>
<td>0</td>
<td>0.834</td>
<td>1.018</td>
</tr>
<tr>
<td>Stack s.</td>
<td>-0.212</td>
<td>-2.272</td>
<td>-0.449</td>
<td>0</td>
<td>0.278</td>
</tr>
<tr>
<td></td>
<td>-0.633</td>
<td>-2.799</td>
<td>-0.834</td>
<td>0</td>
<td>0.054</td>
</tr>
<tr>
<td></td>
<td>0.209</td>
<td>-1.744</td>
<td>-0.063</td>
<td>0</td>
<td>0.501</td>
</tr>
<tr>
<td>Taskbar</td>
<td>-0.489</td>
<td>-2.549</td>
<td>-0.726</td>
<td>-0.278</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-0.939</td>
<td>-3.088</td>
<td>-1.018</td>
<td>-0.501</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-0.040</td>
<td>-2.010</td>
<td>-0.435</td>
<td>-0.054</td>
<td>0</td>
</tr>
</tbody>
</table>

More interestingly we found some significant interactions between those visual factors and Technique (we mainly focused on the interactions between visual factors and
technique):

- Two-way interactions: between Num and Technique (F_{8,72} = 6.22, p<0.001), between VS and Technique (F_{8,72} = 28.58, p<0.001), between MLS and Technique (F_{8,72} = 3.83, p<0.001), between DisSize and Technique (F_{8,72} = 3.33, p<0.001), between AmOverlap and Technique (F_{12,108} = 158.05, p<0.001);

- Three-way interactions: between Num, VS and Technique (F_{16,144} = 7.49, p<0.001), between Num, MLS and Technique (F_{16,144} = 2.30, p<0.005), between VS, MLS and Technique (F_{16,144} = 1.98, p<0.018).

Interpreting two-way interactions

Pairwise comparisons is used to analyze and interpret those two-way interactions.

**Num x Technique.** For all Num conditions, Direct pointing is significantly more than 40% slower compared to other techniques. For the 8 windows and 12 windows conditions, Taskbar is significantly (p<0.013) faster than the other techniques. On average, for 8 windows, Taskbar is 49% faster, and for 12 windows, Taskbar is 35% faster. We also observed a significant difference (p<0.035) between stack scanning and other techniques for 16 windows condition. On average stack scanning is 17% faster than Taskbar, Exposé and Alt+Tab, and 65% faster than Direct pointing (H1 is supported) (Figure 5.8).

**VS x Technique.** There is a significant difference between stack scanning and other techniques (p<0.018) for HVS. On average stack scanning is 24% faster than Taskbar, Exposé and Alt+Tab, and 49% faster than Direct pointing. For LVS and NVS conditions, we observed significant differences (p<0.005) between Taskbar and other techniques. On average Taskbar is 26% faster than stack scanning, Exposé and Alt+Tab, and is 97% faster than Direct pointing for these two VS conditions (Figure 5.9).

**MLs x Technique.** There is a significant difference between Direct pointing and other techniques (p<0.001) for all match level conditions with Direct pointing being 49% slower on average. For the first match level, we observed a significant difference between Taskbar and other techniques (p<0.001). On average, Taskbar is 17% faster than stack scanning, Exposé and Alt+Tab, and is 72% faster than Direct pointing (Figure 5.10).

**DisSize x Technique.** There is a significant difference between Taskbar and other techniques (p<0.045) for all the distributions of window size. On average Taskbar is 12% faster than stack scanning, Exposé and Alt+Tab, and 63% faster than Direct pointing for those three DisSize conditions (Figure 5.11).
Figure 5.8: Mean switching time (ST) in s for Num and Technique. Error bars represent 95% confidence interval.

**AMOverlap x Technique.** We found that stack scanning was significantly (p<0.033) faster than the other techniques for the 0% AMOverlap (this also hinted that Direct pointing is faster than other techniques when the target window is visible). On average stack scanning is 21% faster than Taskbar, Exposé and Alt+Tab, and 79% faster than Direct pointing. For 75% and 50% AMOverlap we observed a significant difference (p<0.035) between Taskbar and other techniques. On average Taskbar is 15% faster than Exposé and Alt+Tab, and is 34% faster stack scanning, and is 63% faster than Direct pointing (Figure 5.12).

**Interpreting three-way interactions**

We had observed some significant differences between visual factors and techniques in the three-way interactions, we now interpret those significant interactions.

**Num, VS and Technique.** The Num x VS interaction is larger for Taskbar (F_{4,12951} = 402.43, p<0.001) than other techniques. With stack scanning, the difference between
5.5 Results

Figure 5.9: Mean switching time (ST) in s for VS and Technique. Error bars represent 95% confidence interval.

VS by Num is very small. With Taskbar, the difference between Num is large, especially when the VS is important (Figure 5.13). With Direct pointing, the difference between VS by Num is small whereas there is a large difference between Num. With Exposé, the difference between Num for each VS condition is larger, when the VS is important, the difference is more obvious.

Num, MLS and Technique. The Num x MLS interaction is larger for Taskbar ($F_{4,12951} = 31.75$, $p<0.001$) than other techniques. With Exposé, stack scanning and Direct pointing, the difference between MLS is small for each the Num condition. With Taskbar, the difference between the Num by the MLS is larger than other techniques, and the difference between the MLS is smaller for 8 and 12 windows than for 16 windows condition (Figure 5.14).

VS, MLS and Technique. The VS x MLS interaction is larger for Taskbar ($F_{4,12951} = 14.13$, $p<0.001$) than other techniques. With Taskbar, the difference between the VS is larger than other techniques (Figure 5.15). With stack scanning and Direct pointing
Figure 5.10: Mean switching time (ST) in s for MLs and TECHNIQUE. Error bars represent 95% confidence interval.

Figure 5.11: Mean switching time (ST) in s for DiSSIZE and TECHNIQUE.
Figure 5.12: Mean switching time (ST) in s for AMOVERLAP and TECHNIQUE.

Figure 5.13: Mean switching time (ST) in s for NUM and TECHNIQUE with the HVS. Error bars represent 95% confidence interval.
5. STACK SCANNING RULES!

Figure 5.14: Mean switching time (ST) in s for MLs and TECHNIQUE with the 16 windows. Error bars represent 95% confidence interval.

The differences between the MLs are small for each the VS condition. With Alt+Tab, the difference between VS is large for each the VS condition.

5.5.1.2 Second Experiment

For the SIZE001 of the distribution of window size condition, we performed an ANOVA similar to our main ANOVA. The results of ANOVA showed a significant main effect for NUM (F\(_{2,18} = 75.01, p<0.001\)) , VS (F\(_{2,18} = 40.75, p<0.001\)) and TECHNIQUE (F\(_{4,36} = 53.17, p<0.001\)) on switching time. Comparing to the main experiment, there was a difference for MLs condition. For the SIZE001, it showed a significant main effect for MLs (F\(_{2,18} = 5.21, p<0.048\)) on switching time. There was only one significant two-way interactions between VS and TECHNIQUE (F\(_{8,72} = 19.93, p<0.001\)) (Figure 5.16).

For the SIZE001 condition, each layer of stack scanning only includes one window, participants need to cross more layers to get the target window, so it is slower than other DisSIZE conditions. With Exposé, the similar window size would result in similar thumbnail size, and the thumbnail of window is smaller than other DisSIZE condition.
5.5 Results

![Bar chart showing mean switching time (ST) in s for VS and TECHNIQUE with the 16 windows. Error bars represent 95% confidence interval.](image)

Figure 5.15: Mean switching time (ST) in s for VS and TECHNIQUE with the 16 windows. Error bars represent 95% confidence interval.

So with the visual similarity increasing, it is slower than other DISSIZE conditions.

For the HVS condition, Exposé is slower than other techniques for each NUM condition. Taskbar is still the best choice under the low NUM condition, it is significantly faster than other techniques for 8 and 12 windows, and AltTab is the best choice when the number of windows is high. It is faster than other techniques (Figure 5.17). For the LVS and NVS conditions, Taskbar is faster than other techniques for each NUM condition, so it is the best choice for users.

5.5.2 Error Rate

5.5.2.1 Main Experiment

The overall error in the experiment was 4.83%. ANOVA showed a significant main effect for NUM ($F_{2,18} = 20.89$, $p<0.001$), VS ($F_{2,18} = 19.35$, $p<0.001$), MLs ($F_{2,18} = 7.48$, $p<0.008$) and TECHNIQUE ($F_{4,36} = 18.70$, $p<0.001$) on switching errors. We also observed some significant interactions between visual factors and techniques, including between NUM and TECHNIQUE ($F_{8,72} = 18.70$, $p<0.012$) (Figure 5.18), between VS and
5. STACK SCANNING RULES!

Figure 5.16: Mean switching time (ST) in s for each Technique, grouped by VS under the Size001 condition. Error bars represent 95% confidence interval.

Technique \( (F_{8,72} = 3.25, p<0.032) \) (Figure 5.19), between MLS and Technique \( (F_{8,72} = 3.99, p<0.015) \). There are also three-ways significant interactions between Num, VS and Technique\( (F_{16,144} = 2.78, p<0.027) \), between Num, MLS and Technique\( (F_{16,144} = 3.99, p<0.009) \), between VS, MLS and Technique\( (F_{16,144} = 3.93, p<0.009) \). The results reveal that Direct pointing is less error prone than other techniques and Exposé is more error prone than other techniques for each combination of levels of the other visual factors conditions.

We use the LSD (Least Square Difference) test with \( \alpha = 0.05 \) for the pairwise comparisons between techniques condition on switching error rate. Table 5.2 shows the results of those tests in detail.
5.5 Results

Figure 5.17: Mean switching time (ST) in s for each Technique, grouped by MLs for HVS under the Size001 condition. Error bars represent 95% confidence interval.

5.5.2.2 Second Experiment

The overall error for the Size001 condition was 6.31% (it is higher than other DisSize conditions). However, a new ANOVA only shows that Expose is more error prone than other techniques: the Num \( (F_{2,18} = 3.95, p<0.050) \) and MLs \( (F_{2,18} = 5.88, p<0.038) \) are significant factors, but the techniques are not and there is no evidence of an interaction between visual factors and techniques.

5.5.3 Qualitative Results

At the end of experiment, participants were asked to rate those five window switching techniques and visual factor conditions. Nine participants said that they preferred stack scanning when the number of windows and the visual similarity is high or the number of layers is low, and four of them liked the overlapping mechanism of stack scanning very much, they said that it provided a good way to support them to organize their windows by
Figure 5.18: Error rate for each TECHNIQUE, grouped by NUM. Error bars represent 95% confidence interval.

Figure 5.19: Error rate for each TECHNIQUE, grouped by VS. Error bars represent 95% confidence interval.
Table 5.2: Pairwise comparisons between techniques condition on switching error rate. A cell contains the means difference and the lower and upper bound of the confidence interval. Underlined cells are significant.

<table>
<thead>
<tr>
<th></th>
<th>Alt+Tab</th>
<th>Direct pointing</th>
<th>Expose</th>
<th>Stack scanning</th>
<th>Taskbar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Tab</td>
<td>0</td>
<td>0.0012</td>
<td>-0.0054</td>
<td>-0.0012</td>
<td>-0.0010</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.0006</td>
<td>-0.0080</td>
<td>-0.0021</td>
<td>-0.0020</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.0019</td>
<td>-0.0027</td>
<td>-0.0003</td>
<td>0.0000</td>
</tr>
<tr>
<td>Direct p.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.0012</td>
<td>0</td>
<td>-0.0066</td>
<td>-0.0024</td>
<td>-0.0023</td>
</tr>
<tr>
<td></td>
<td>-0.0019</td>
<td>0</td>
<td>-0.0095</td>
<td>-0.0033</td>
<td>-0.0033</td>
</tr>
<tr>
<td></td>
<td>-0.0006</td>
<td>0</td>
<td>-0.0037</td>
<td>-0.0015</td>
<td>-0.0013</td>
</tr>
<tr>
<td>Expose</td>
<td>0.0054</td>
<td>0.0066</td>
<td>0</td>
<td>0.0042</td>
<td>0.0044</td>
</tr>
<tr>
<td></td>
<td>0.0027</td>
<td>0.0037</td>
<td>0</td>
<td>0.0017</td>
<td>0.0016</td>
</tr>
<tr>
<td></td>
<td>0.0080</td>
<td>0.0095</td>
<td>0</td>
<td>0.0067</td>
<td>0.0071</td>
</tr>
<tr>
<td>Stack s.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0012</td>
<td>0.0024</td>
<td>-0.0042</td>
<td>0</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>0.0003</td>
<td>0.0015</td>
<td>-0.0067</td>
<td>0</td>
<td>-0.0008</td>
</tr>
<tr>
<td></td>
<td>0.0021</td>
<td>0.0033</td>
<td>-0.0017</td>
<td>0</td>
<td>0.0011</td>
</tr>
<tr>
<td>Taskbar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0010</td>
<td>0.0023</td>
<td>-0.0044</td>
<td>-0.0001</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.0000</td>
<td>0.0013</td>
<td>-0.0071</td>
<td>-0.0011</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.0020</td>
<td>0.0033</td>
<td>-0.0016</td>
<td>0.0008</td>
<td>0</td>
</tr>
</tbody>
</table>

groups (task) to allow windows overlapping, and they considered that this could improve the efficiency of their work. There were different opinions for Exposé, three participants said they liked Exposé, two participants said they did not like Exposé, because the thumbnails were small and it was difficult to distinguish them. They could only use the label attached to the thumbnails to recognize the target window. All participants commented on Exposé that the location of windows was not fixed made them very frustrated. Six participants (4 female) reported that they preferred the techniques based on the mouse (Direct pointing, Taskbar and stack scanning).

All participants said that they did not like the high visual similarity condition. Eight participants said that they used Direct pointing less often with the window of interest which was obscured, but seven participants (six from those eight participants) cited Direct pointing as their preferred way with the window which is visible (the experiment has
proven that it was faster than other techniques when the target window is visible). Five participants cited Taskbar as their preferred technique with the low number of window: the button location fixed on the Taskbar was interesting for them. Participants who used Alt+Tab never used 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. Participants said that it was inconvenient and uncomfortable to press the Shift key in this case (simultaneously press the Alt and Tab key). Three participants who used Alt+Tab technique said that the changing of windows order in the list could make them feel confused, because the stability is very important for them.

5.6 Discussion

Taskbar is fast when the number of windows and the visual similarity are low. When the number of windows is constant, the fixed button location on the Taskbar can also help users to revisit the window (when the number of windows is changed, the button location will also change). The main problem with Taskbar is that its performance becomes worse and worse with a high number of windows and a high visual similarity. One possible solution is to allow the user to define a relevant region on the Taskbar to put some important windows (drag-and-drop operation is added to help users to rearrange the windows) and this region size is not affected by the change in the number of windows on the Taskbar. In this way, all windows in the special region can be very easy to find (fix location and the icon and title are visible).

The main problem with Exposé is the high visual similarity between windows and non-stable spatial layout. The visual similarity mainly includes two aspect: 1) the window size (the thumbnail size is decided by layout algorithm); 2) the window content, and when the thumbnail size is small, the content is difficult to recognize. To reduce the visual similarity, one possible solution is to use real windows to replace thumbnails, the title bar of these windows is the same as the original windows, and the content of these windows use the same content as the last time access of the original windows. For non-stable spatial layout mechanism, the direct way uses a stable spatial layout mechanism. Other ways, include adding visual feedback to identify frequently accessed window (i.e., using the number 1, 2, 3 to identify frequently accessed three windows or adding the trails between them Hoffmann et al. (2008)).
For *Alt+Tab*, in addition to the performance issues (the high number of keystrokes), the interaction way based on the keyboard is another problem for some users. They are more accustomed to using the mouse, so we can consider integrating this technique with mouse shortcuts.

The performance of *stack scanning* mainly depends on the number of layers and the amount of overlapping of the target window. Most text documents being left-aligned, overlapping them on the right side usually leaves more content visible than on the left side, we can develop some placement strategies to keep windows to overlap on the right side as possible as we can. In this case, we may easily recognize each window for using *stack scanning*. Push-And-Pull Switching has been proved that it is efficient to quickly switch between the current group and the closer, but it appears more difficult to switch to any group Xu & Casiez (2010). *Stack scanning* has a potential possible and advantage to be used to group switching, it can address that problem (switch to any group) very well, and we believe it will have a good performance for group switching (it does not need to implement the pointing action).

### 5.7 Conclusion

In this chapter, we examined the problems related to switch windows under various visual factors conditions. We proposed a new window switching technique, *stack scanning*. It provides a lightweight alternative to other window switching techniques. We conducted an experiment to compare the performance and error rate of *stack scanning* to other 4 window switching techniques. Results showed that stack scanning was faster than other techniques when the number of windows is high and the visual similarity is important, and participants strongly preferred it in this condition. They also showed that Taskbar was the best choice when the number of windows is small, regardless of other visual factor conditions and for users who always maximize their windows, *Alt+Tab* was the best choice when the number of windows is important. For the potential problems with the current window switching techniques, we provide some solutions to improve them.
Chapter 6

WindowsTagging: Quick Task Switching Using Tags

In this chapter, we first discuss and give eleven design principles which are based on the presented issues on the existing task switching techniques. Those design principles provide a theory foundation to help designers to develop new window switching techniques. We then designed and developed a prototype system called WindowsTagging based on those design principles. It combines implicit and explicit definition of groups, spatial and visual memories to help users to quickly find windows or groups and switch between them. WindowsTagging represents windows as thumbnails, displaying all groups at the same time and allowing windows to exist simultaneously in multiple groups. WindowsTagging exploits users’ spatial and visual memories by providing a relative stable spatial layout with larger thumbnails using overlapping. Groups are first automatically created based on a window overlapping algorithm. Window tagging mechanism was used to allow users to explicitly create or modify groups by tags. The Drag-and-drop and splitting group functions also allow users to explicitly define or modify groups. A search function is presented to help users finding a window by its title or tags. An experiment was designed to compare the performance and error rate of WindowsTagging to Exposé. Results showed that WindowsTagging was faster than Exposé technique, and participants strongly preferred it. Finally, we showed in a user longitudinal study that WindowsTagging was very effective and can improve the efficiency of users task management.
6.1 Introduction

Throughout the history of personal computing, the limited display space has always been a problem. On the one hand users are increasingly turning to use larger display and multiple-monitors system, on the other way users keep more opened windows simultaneously with the additional display space Smith et al. (2003). This results in that users may still face the same window management and interaction problems that occur on the small displays when using modern overlapping window management systems with large displays.

Virtual desktop managers (VDMs) have been one of the popular solutions to the space management problem, such works can be traced back to Card and Henderson’s Rooms which used ‘rooms’ to organize windows and in an entirely manual way D. Austin Henderson & Card (1986). In Rooms, Card and Henderson D. Austin Henderson & Card (1986) observed that people tend to use windows in groups. They also identified desirable properties of task management systems (to correspond with windows group), including 1) fast task switching, 2) fast task resumption, and 3) easy reacquisition of the cognitive context associated with a task.

Card and Henderson proposed those desirable properties mainly based on the performance of task switching. However we did not find that they provided some design principles to help researchers to implement those properties except an example (Rooms). Another problem is that they only focused on task switching, they did not discuss the problem how to create tasks which is also very important for task management systems. It is a prerequisite for the task switching and can directly affect subsequent interaction.

Since then, researchers have spent considerable effort on this topic and have proposed a variety of task management systems (e.g., Elastic Windows Kandogan & Shneiderman (1996), Task Gallery Robertson et al. (2000), Scalable Fabric Robertson et al. (2004), GroupBar Smith et al. (2003) and WindowScape Tashman (2006)) and each has exhibited some of these properties. Those systems have proposed two main ways to help users define tasks: 1) automatically group windows into tasks; and 2) give users manual control over these groupings. Whether groups are explicitly or implicitly defined, however, these grouping mechanisms present problems for users. For groups explicitly defined, users may find it burdensome to explicitly classify windows into tasks and even may be hard-pressed to decide on an appropriate classification for each window. For groups automatically
defined, the system can incorrectly infer groups and create groups that may not correspond to users expectations.

But in spite of those system successes, the limitation of grouping mechanism they provided has restricted their use. Tagging has proven a powerful alternative to existing top-down categorization techniques, tags are more flexible in organizing objects Wetzker et al. (2010).

Tags are an important information source in Web 2.0. They can be used to describe user’s topic preferences as well as the content of items to make personalized recommendations. A tag is a non-hierarchical keyword or term assigned to a piece of information. It helps to describe an item and allows it to be found again by browsing or searching. Tagging has emerged as a powerful mechanism that enables users to find, organize and understand online entities Sen et al. (2009). Tagging systems have already showed their value in many areas, such as query expansion Biancalana et al. (2008), web search Bao et al. (2007), personalized search Schmidt et al. (2009); Xu et al. (2008), resource classification Yin et al. (2009) and clustering Ramage et al. (2009), retrieval process Melenhorst et al. (2008) and information memory and organization.

Since tags are created by users, they represent concepts meaningful for them. Tagging allows users to choose the labels that match their real needs, tastes, or language, which reduces the required cognitive efforts Wetzker et al. (2010). Because tags are easily comprehended by users, tags serve as a bridge enabling users to better understand an unknown relationship between an object and themselves. Their advantages can be summarized as follows:

- Allows users to classify their objects in the ways that they find useful (classification);
- Allow for more flexibility in organizing content. They can mark and represent the thinking of users, users can easily resume and recognize objects (users can use semantic information to mark their groups, because it can be difficult to reacquire a task based on the content of windows only, especially after a long time interruption);
- Facilitate content retrieval (search by tags is faster and more accurate than search by the content of objects);
- Represent the correlation between object and object. One object may have some different attributes (tag), and an attribute (tag) can be marked on different objects.
6.2 Design Principles to Support Task Switching

This gives us the ability to link different objects, which is difficult to achieve with the title only;

- Facilitate the annotation process because little or no knowledge about the system is required (easy to use and understand for users).

Taking into account their advantages, we can easily find that the tagging mechanism is a very powerful and can be used in window management to enhance grouping mechanism for modern window management systems.

So far, although many task management systems have been proposed, no system has described in details what kinds of operations we should provide to users and how to efficiently perform those operations. Twenty years ago, Bannon et al. gave some suggestions to support task switching Bannon et al. (1983). However the personal computer (PC) has undergone dramatic changes over the past 30 years. The huge gains in processor speed and physical memory size and the large and multiple-monitor have allowed people to keep more windows on the desktop. we re-examine those suggestions and give some new design principles to support task switching.

In this chapter, we would like to "theorize" task switching and define types of task switching operations which have been justified by real-world usage data. First, we analyze the problems posed by the proposed task management systems and use the results of longitudinal study in the aforementioned chapter to give some basic design principles to support task switching. After describing the design principles, we present a prototype system called WindowsTagging to implement those basic design principles and use the tagging mechanism to address the limitations of other task management in the grouping mechanism, window tagging, which uses tags to define groups. We then detail the operations that WindowsTagging provides to users. Finally, we present the results of an informal study and discuss some disadvantages of tagging mechanism and future work.

6.2 Design Principles to Support Task Switching

The first thing a window management system should provide to manage groups is the following operations:

- group creation;
6. WINDOWSTAGGING: QUICK TASK SWITCHING USING TAGS

- group edition;
- group access (switch between groups);
- group deletion.

In the following, we analyze each step and give some design principles to help researchers design new task management systems.

6.2.1 Group Creation

Separating explicitly and implicitly definition of groups present problems, and a previous research has also shown that users were typically disinclined to organize their personal information up-front Bernstein et al. (2008a), so we would like a system with the following property:

C1. Allow implicit and explicit methods to create groups.

We wish to implicitly define groups to free users from the initial task of explicitly defining groups.

The position relationship among windows is very important for users, and users often coordinate windows to finish their works Hutchings & Stasko (2004), so we would like to maintain it to decrease the manual operations of rearranging groups. The second design principle is:

C2. Keep the position relationship among windows as much as possible with automatically defining groups.

With the large displays and multiple-monitors popularization, users can keep more windows open simultaneously Smith et al. (2003), so that we should not neglect the possibility that some windows might naturally be useful in several groups. Researches have hinted that a disadvantage faced by many systems (e.g. virtual desktop managers (VDMs), Scalable Fabric Robertson et al. (2004), Groupbar Smith et al. (2003)) is that requiring windows to be in a single group forces users to decide ahead of time where a new window belongs. So our the third principle is:

C3. Allow a window to be in multiple groups at the same time.

\(^1\)C is the first character of create and ‘1’ represents the first principle for creating groups operation. And the following symbols have the similar meanings.
6.2 Design Principles to Support Task Switching

6.2.2 Group Edition

Bernstein et al. Bernstein et al. (2008b) observed that tasks can evolve constantly, and what might have been a correct groups an hour ago may be inappropriate now. As users change their tasks, a system should provide some mechanisms to help users quickly rearrange windows to adapt to the new task, so we wish a system that has the property to help users edit groups:

E1. Allow users to explicitly change groups structure, including adding/removing windows in groups and creating new groups using existing groups.

6.2.3 Group Access

Twenty years ago, Bannon et al. observed that people often switch between concurrent tasks. Today, this conclusion was confirmed again and quantified (window switching takes place on average once every 20.9s on large displays Hutchings et al. (2004), the results were confirmed by D.Mackinlay & Royer (2007)). This frenetic frequency of window switching illustrates that task switching is still the focus problem of task management systems, and the quality of interaction efficiency can directly impact on users’ experience and tasks productivity.

With modern window management systems, users want at least three types of switching: 1) between windows; 2) between groups; and 3) between selected windows of groups (these operations have been justified by real-world usage data (the longitudinal study)). So the first design principle for switching groups is:

A1. Allow users to switch between windows or groups or selected windows of groups.

One limitation of VDMs is a strict separation between windows group, making it difficult to switch between windows from multiple groups at the same time. In the new design, we should avoid this problem, so the second design principle for switching groups is:

A2. Allow users to switch between windows from multiple groups at once and without affecting the grouping structure.

Although users may switch among groups in a self-guided manner, a significant portion of group switching is caused by external interruptions Czerwinski et al. (2004). To quickly resume the tasks, it is sometimes difficult to do it by only according to the content of windows. The reasons mainly come from two aspects: 1) some tasks may share different
windows, and 2) the content of windows may not represent the task topic well. So we wish that the semantic information can be introduced to mark tasks in order to easily recognize them, the third design principle for switching groups is:

A3. Allow users to add semantic information to mark their tasks.

Search function is widely used in people’s physical environment and digital environment and it has become the most basic way for people to get information from Internet and Desktop environment. A variety of search services (such as Desktop Search and Web Search) have been developed to support people requests. However, we are not aware of any of modern window management systems that have provided the search function for windows/groups and little work has been done by the HCI community on this problem (We only found that Ishak proposed CAL approach that uses the content of windows to automatically rearrange their windows Ishak (2007). It is only a prototype system and we did not find any evaluation about it). So we would like a system have the property:

A4. Allow users to search windows or groups by some keywords.

Prior research has shown that the efficient use of graphical user interfaces strongly depends on human capabilities for spatial cognition Cockburn (2004). User interfaces can improve task performance by exploiting the powerful human capabilities for spatial cognition. This opportunity has been demonstrated by many prior experiments Egan & Gomez (1985); Gagnon (1985); Leitheiser & Munro (1995); Vicente et al. (1987). Many techniques have been introduced to take advantage of human spatial memory to improve users interactive capabilities Robertson et al. (1998); Tan et al. (2001). Meanwhile some techniques have exposed this problem, such as Exposé Apple. So we would like a system that can maintain a stability of the layout as much as possible, following the design principle:

A5. Keep a stable groups layout to allow users to make effective use of their spatial memory.

Taken into account the task as the basic organization way in the task management systems, we wish a system that can provide some operations for groups similar to windows, so we have the following design principle:

A6. Provide some operations for groups, such as move, resize.
6.2.4 Group Deletion

When users finish a task or they have to close a task to do other things, a system should allow users to use a simple way to close it, so we would wish the system have the property:

\( D1. \) Allow users to delete groups.

We have developed a prototype system called WindowsTagging based on the design principles aforementioned. We explore the concept of window tagging instead of groups-as-list in WindowsTagging. Each window in WindowsTagging can be annotated with optional, freeform tags to describe its semantics. All windows in WindowsTagging can be tagged, and a window can have several tags (it is easy to address the problem which allows one window to reside in different groups) and a given tag can be attached to several windows (define a group). With the application of tags in WindowsTagging, users can find windows or groups by searching their tags and it can help users to resume tasks according to the tags without regard to the content of tasks.

In the following we describe the WindowsTagging operations and implementation in details.

6.3 WindowsTagging

WindowsTagging is a screen-filling visualization of the user’s workspace in two dimensions which displays all groups and windows at the same time using thumbnails (Figure 6.1). The different operations to automatically define groups, switch to a group or a window, select windows of groups, explicitly reorganize groups and finding windows are presented below.

6.3.1 Group Creation

WindowsTagging combines implicit and explicit definition of groups (\( C1 \) is supported). When WindowsTagging is initially invoked, groups are implicitly created.
6. WINDOWSTAGGING: QUICK TASK SWITCHING USING TAGS

6.3.1.1 Implicit Group Definition

Push-and-Pull switching proposed to use window overlapping to implicitly define groups Xu & Casiez (2010) based on the observation that users try to keep windows belonging to the same activity visible by trying to avoid or minimize the amount they overlap. This is also supported by Hutchings and Stasko who found similar results Hutchings & Stasko (2004). However Push-and-Pull switching did not allow a window to be in multiple groups at the same time. To address this limitation, we propose a new algorithm to implicitly create groups based on overlapping windows and allow a window to be in multiple groups at the same time. Groups are also created by considering windows in decreasing Z order, from foreground to background, and creating a new group each time.

Figure 6.1: WindowsTagging presents windows and groups using thumbnails. Thumbnails are allowed to overlap within each group to identify groups more easily and improve thumbnails legibility. In this example all opened windows (16) were divided into ten groups. When moving the mouse, the group closest to the cursor is expanded to make them easy to recognize and the titles of all window within the group are displayed.
the amount of overlapping for a window is beyond an overlapping threshold (\textit{C2 and C3 are supported}). The algorithm is defined as follow:

\begin{algorithm}
\caption{Compute Windows Group}
\begin{algorithmic}
\State \textbf{Input:} windows on the desktop
\State \textbf{Output:} windows group
\State Initialization: Create a group container $G$
\For {\textit{Window $W_i$ in the window stacking taken from top to bottom}}
\State \textbf{flag} $\leftarrow$ false
\For {\textit{group $G_i$ from top to bottom in $G$}}
\If {the intersection between $W_i$ and the group $G_i$ is not over the overlapping threshold value}
\State Add $W_i$ to the group $G_i$
\State \textbf{flag} $\leftarrow$ true
\EndIf
\EndFor
\If {!flag}
\State Create a new group and add $W_i$ to this group
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

When initially invoked, our algorithm \ref{alg:compute-windows-group} first creates groups of overlapping windows. Here the amount of overlapping for a given window is computed as the percentage of pixels occluded. We set the default overlapping threshold to 25\% (users can define this value by themselves). The biggest difference between our algorithm and the algorithm of Push-and-Pull switching \cite{XuCasiez2010} is that our algorithm allows a window to reside in multiple groups simultaneously (Figure \ref{fig:windows-tagging}). When a window is computed and added to a group, the window does not take part in other groups computing for the algorithm of Push-and-Pull switching, and it will continue to be computed with other groups for our algorithm.

6.3.1.2 Explicit Group Definition

We allow two means to manually reorganize groups in WindowsTagging, one is to use the \textit{drag-and-drop} operation and another is to use tags. \textit{Drag-and-drop} provides a natural way to allow users to explicitly define and change groups. The cost is that users need to
Figure 6.2: A simple example to describe the differences between the creating group algorithm of Push-and-Pull switching and the new algorithm. For Push-and-Pull switching, computing the following groups: (W3, W1), (W2), and for the new algorithm, computing the following groups: (W3, W1), (W2, W1).

use drag-and-drop operations. Tags allow users more flexibility to explicitly define and modify groups. Users can add tags to windows (we call this process as tagging) in order to make them become one group and remove tags of the window in order to remove the window from the group. This method can easily make one window reside in multiple groups simultaneously (attaching the tags to groups), compared to drag-and-drop, it does not need to copy the window (E1 is supported).

Window Tagging

A tag is a collection of windows that can be enumerated using an iterator. Each window in WindowsTagging can be tagged, and can have several tags. A given tag can be attached to several windows. Windows are initially created without tags in WindowsTagging. A tag can include any valid character. When users add a tag to a window, the window position information (screen coordinates) is saved automatically and attached to this tag. When users add a tag to a group, each window within the group is tagged with this tag. The tag structure is like: \text{chi:}(200, 200, 800, 800), the ”chi” is the semantic information
6.3 WindowsTagging

that is tagged by users, 

\((200, 200, 1200, 800)\)

is the window’s coordinates (left, top, width, height). It is created by the system and automatically attached to the tag. With the coordinates information, a window’s tag can correspond to a specific position of the window, so when a window is in multiple groups at the same time, it can anchor at the different position in different groups (it can be better to maintain the position relationship) \((A3 \text{ is supported})\).

WindowsTagging uses two types of tags: extensional tags, which are explicitly added to or removed from a window and intentional tags, which are specified by a predicate that tests whether or not a window has this tag and are used to define a group windows according to a given criterion.

**Extensional tags.** Users need to explicitly add or remove extensional tags for a window. In WindowsTagging, users can click the right mouse button on the target window to open a popup hierarchical menu and select the Window Tagging item from the menu (Figure 6.3). Users can also add or remove tags for all windows within one group at the same time, replacing the Window Tagging item, users use the Group Tagging item to implement this function. When users add a tag, users can select an existing tag or create a new tag. If users choose to create the new tag, a popup window with a text box is opened to type characters. For Window Tagging, the default content in the text box is the window’s title. For Group Tagging, the default content in the text box is the title of the window which is the topmost z-order in this group.

**Intentional tags.** Intentional tags are used to define group of windows according to a given criterion (e.g. all ”pdf” windows). Each time an intentional tag is used, the group of tagged windows is recomputed. In WindowsTagging, we implement it in a simple way with searching function (see Finding Windows), the criterion is given by entering the search keywords in the text box and the search results will be as a group, when users click the Tagging button (at the right of text box), the criterion is tagged to this group. For example, users want to bring all windows with “ pdf ” in the title closer to the foreground, users can type “ pdf ” in the text box, and then clicking the Tagging button. Finally, clicking Foreground item from the tag function menu will bring all windows with “ pdf ” in the title closer to the foreground (We plan to enhance this function to support more general operations, such as, users can enter “ pdf ”, and bring all windows without “ pdf ” in the title closer to the foreground).
6. WINDOWSTAGGING: QUICK TASK SWITCHING USING TAGS

Figure 6.3: WindowsTagging’s context menu, tag item is expanded to allow users to select add/remove tag item.

Tag management. The tag management window is used to implement the tag management function (Figure 6.4). The tag management includes basic operations for tags and methods to deal with the identified group by a tag. The basic operations, such as: add tag(s) to window(s), remove tag(s) to window(s) and edit tags. The methods mainly include: bring a window to the foreground, bring group closer the foreground, close group (if a window has other tags, this window can not be closed). Users can see all tags that have been used and each window or group has been tagged. Users can check each tag to see which windows or groups are tagged with the same tag, this process is similar as searching windows by window’s tag (see Finding Windows via window’s tag), the windows which do not include the tag are darken but their content remains visible through transparency. By default, tags do not display on thumbnails, users can click the Display Tags button to display all tags, and they will be displayed on thumbnails. We use different colors to identify different tags. The same tags have the same color.

Drag-and-drop

WindowsTagging allows users to explicitly modify groups using drag-and-drop. Groups modification includes window reorganization within a group: users can change the position
and Z order of each window within the group by dragging or selecting the corresponding thumbnail. Thumbnails can also be dragged-and-dropped between groups to change the group belonging of a window. Windows can also belong to different groups using the copy function.

Moving a window to another group is done using drag-and-drop with the left mouse button. The border color of the thumbnail begin dragged is first orange when it is hovered and it changes to red when it starts being dragged (Figure 6.5 (b1)). The thumbnails within each group then appear with a different color to clearly show the different groups (Figure 6.5 (c1)). The border color of the dragged thumbnail is then updated with the color of the group for which the intersection area is the most important (Figure 6.5 (c1) and (d1)).

If the dragged thumbnail does not intersect with any group or if the area of intersection with two other groups is exactly the same then the color border of the thumbnail is not changed.

Once the border color of the thumbnail changes to the color of the destination group the user can drop it or continue to adjust its position within the group before dropping it (Figure 6.5 (d1)). Once dropped, the dragged window is clipped if it is in intersection.
Figure 6.5: Example of group reorganization using drag-and-drop. Figure (a) represents the initial layout of groups. Moving a window to another group (b1 to e1) is done by pressing the left mouse button on the thumbnail window W (b1). Copying a window to another group (b2 to e2) is done by pressing the Ctrl key and holding it down with the left mouse button on the thumbnail window W (b2). The border color of the thumbnail window being moved or copied is continuously updated with the color of the group hovered (c1, d1; c2, d2). Releasing the left mouse button (or including the Ctrl key) resizes the dropped window to prevent overlapping with another group (e1, e2).

with other groups (Figure 6.5 (e1)).

Copying a thumbnail window is similar to moving a window (Figure 6.5 (b2)) except the users press the Ctrl key and hold it down after the drag operation has been initiated to indicate the copy operation (Figure 6.5 (c2)).

Visible windows snapshot

Drag-and-drop and attaching tags provide two simple way to allow users to explicitly define and change groups. In the process of drag-and-drop, users can coordinate the group structure and position relationship among the thumbnails (using tags can only coordinate the group structure), but the position relationship among the thumbnails can not map nicely to the windows on the desktop (thumbnails and windows have different sizes) and users can not effectively control it. However sometimes the position relationship among the windows (windows layout) is very important and significative for users as it directly
affect the visibility of the contents of the windows (windows overlapping). Hutchings has shown that 50% users tend to have many windows visible simultaneously Hutchings & Stasko (2003). If the user’s organization results of windows can be saved and be resumed by a simple way, it can evidently improve the interaction efficiency of users.

WindowsTagging provides a simple operation to support this requirement based on window tagging. Users can press the F10 key to save all the current visible windows to one group (if one window is covered by no more than 75%, we refer to the window as visible window), after pressing the the F10 key, the tag management window will be opened, and users can select one existing tag (this group will include other windows) or enter a new tag to be attached to those visible windows. This functionality provides another way to explicitly define group and can protect windows layout stability as much as possible.

6.3.2 Window & Group Access

6.3.2.1 Window&Group Switching

WindowsTagging is invoked using a keyboard shortcut (F9) or clicking the WindowsTagging icon in the system tray. When the WindowsTagging is first invoked, the algorithm 4 is called to create groups. For the following invocations, the implicit group definition algorithm will only be executed when a new window is opened or when there was an important number of modifications on windows such as closing, resizing and moving windows. In practice, the implicit group definition algorithm is called when there was at least three windows that were modified. Otherwise the groups structure and spatial layout from the last invocation are preserved (A5 is supported).

We consider several factors to decide which groups participate in the implicit group definition algorithm. First we consider that groups explicitly modified by users (refer to "Fixed Group") should not be altered by an automatic process. Users can get frustrated if the time they spent defining groups is lost. As a result the groups for which windows were added are not implicitly changed. When a new window is opened or there was at least three windows that were modified, the implicit group definition algorithm is called, and the explicitly defined groups will be taken into account as a whole (like a big 'window') and the highest Z order of the window within that group will be as the Z order of that
group. The new algorithm is as follow:

**Algorithm 5:** Compute Windows Group

**Input:** windows on the desktop  
**Output:** windows group

Initialization: Create a group container $G$

foreach Window $W_i$ in the window stacking taken from top to bottom do
    flag ← false  
    foreach group $G_i$ from top to bottom in $G$ do
        if $W_i$ belongs to a "Fixed Group" FG then
            if the intersection between FG and the group $G_i$ is not over overlap threshold value then
                Add FG to the group $G_i$
                flag ← true
            end
        else
            if the intersection between $W_i$ and the group $G_i$ is not over overlap threshold value then
                Add $W_i$ to the group $G_i$
                flag ← true
            end
        end
    end
    if !flag then
        Create a new group and add $W_i$ FG or to this group
    end
end

The WindowsTagging visualization then overlaps user’s screen and displays thumbnails of all opened windows at once (Figure 6.1). Thumbnails are organized in groups with overlapping between thumbnails within each group. Groups are visually separated by preventing any overlapping between the thumbnails belonging to different groups. When users move the mouse pointer on a group, all thumbnails within the group are expanded and their borders are highlighted in orange. These two feedbacks allow a better identification of the windows belonging to a group and the thumbnails expansion allows to improve their legibility and visual recognition.

Switching to a window and bring it to foreground is done by clicking on the corre-
sponding thumbnail using the left mouse button. Switching to a group and bringing all its windows to foreground is done by pressing the Alt key and clicking on any thumbnail within the group using the left mouse button. Switching to selected windows of groups is done by pressing the Ctrl key and using the left button to select the windows of interest (they can come from different groups), pressing Ctrl + G brings them to foreground. During group switching, the Z order of all windows matches the one of thumbnails. The window with the highest Z order within that group 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 (A1 and A2 are supported).

6.3.2.2 Finding Windows

Via window’s title

WindowsTagging additionally provides a search technique for finding a window using its title. This functionality can be useful when the number of windows gets really important or for users who have more facilities for remembering a window title than its visual content. After the invocation of WindowsTagging, the search window appears at the left-top with a text box to type characters (get the focus by default). Each time a new character is entered or removed, the list of the windows corresponding to the search is updated and the windows that do not match are darken but their content remains visible through transparency. If only one thumbnail remains visible then the user can press the enter key to bring the corresponding window to foreground or if several remains visible he can use the mouse to select the thumbnail of interest (A4 is supported).

Via window’s tag

This process is similar to finding windows via their title. Users can search windows via window’s tag, users can enter characters to match the possible tags and select the desired tags, or users directly select the tag from the list where all used tags are displayed, and users can select multiple tags at the same time, then using those tags to search the windows of interest. Each time a new tag is added or removed, the list of the windows corresponding to the search is updated and the windows which do not include the tags are darken but their content remains visible through transparency. The search results display all the windows which include the input tags (A4 is supported).
6.3.3 Group Edition

6.3.3.1 Adding/Removing windows in groups

Drag-and-drop operation is also an efficient way to add/remove windows in groups. Another way is to use tags to implement this operation when the group has been tagged, users only need to add/remove the tags to the object windows, then the windows can be added/removed in this group. Drag-and-drop operation is more natural way to define groups than tags (*A6 is supported*).

6.3.3.2 Fixing Group

In the aforementioned subsection, we mentioned groups explicitly modified by users that can not be altered by an automatic process. However, if users do not want to modify the structure of an implicitly defined group and do not want this group to be modified by an automatic process, users can use fixing group function. Users can click the right mouse button on the target group to open a popup menu, and selecting the *Fixing Group* item to fix group.

6.3.3.3 Splitting Group

Splitting group function is used to create new groups. This is done by selecting, within a group, one or more windows that are going to belong to the new group. The selection is done by pressing the Ctrl key and selecting each thumbnail window. Pressing Ctrl+S creates a new group with the selected thumbnails.

The splitting algorithm make some room next to the group where the different thumbnails were selected by resizing and translating the other groups. The thumbnails of the original group are scaled down and the new group appears next to it.

6.3.4 Window & Group Deletion

WindowsTagging allows users to close one window or group with right button popup menu. When users close a group, if the windows within this group also belong to other groups (they are tagged by other tags), the closing action can not close those windows, but the system will automatically delete the tag from this group, the windows remain in place (*D1 is supported*).
6.3 WindowsTagging

6.3.5 Implementation

WindowsTagging prototype has been implemented on Windows operating system using C++, in the following subsections, we describe WindowsTagging’s implementation in details, focusing on the main algorithms for group layout, clipping rectangle and splitting group.

6.3.5.1 Group Layout

The algorithm of group layout was inspired by the Exposé patent in Apple’s Mac OS X operating system \textit{patent: US 2004/0261038 A1 (2004)}. We first compute the minimum enclosing rectangle for each group ($MERects$), then we invoke the similar algorithm of Exposé to detect the best position and size of those rectangles ($BPRects$). Finally, we compute the position and size for each window according to the position of two rectangles mapping relations (the transformation matrix from $MERects \rightarrow BPRects$).

\begin{algorithm}
\caption{Group Layout}
\textbf{Input}: windows on desktop  \\
\textbf{Output}: The position and size of each thumbnail  \\
Compute the minimum enclosing rectangle for each group ($MERects$)  \\
Invoke the similar algorithm of Exposé windows layout, get the position and size of those rectangles ($BPRects$)  \\
Compute each transformation matrix $M_i$ from $MERects \rightarrow BPRects$  \\
\textbf{foreach} window $W_i$ on the desktop  \\
do  \\
\text{\hspace{0.5cm}} $W_i$ to do matrix $M_i$ transformation  \\
end
\end{algorithm}

6.3.5.2 Clipping Rectangle

When users use \textit{drag-and-drop} operation, once dropped, the dragged window is clipped if it intersect with other groups Figure 6.5 (e1). In order to make the clipped window
keep the size as big as possible, the clip algorithm is invoked.

**Algorithm 7: Clipping Rectangle**

**Input:** the dragged window  
**Output:** the best window size  
Initialization: Create a rectangle container *vecRectangles*

```
foreach group $G_i$ which intersects with the dragged window do
    foreach window $W_i$ within the $G_i$ do
        if $W_i$ intersects with the dragged window then
            Get the intersection rectangle $r_i$
            Detect all the largest empty-space rectangles when $r_i$ is added to the dragged window \cite{Bell2000}, then adding all largest empty-space rectangles to *vecRectangles*
        end
    end
end
```

Sort *vecRectangles* by the rectangle area

```
foreach rectangle $esr_i$ within the *vecRectangles* do
    if $esr_i$ intersects with the target group then
        Clip the dragged window
        return
    end
end
```

### 6.3.5.3 Splitting Group

This process is similar with the initialization group layout process, we first detect the largest empty-space rectangles and sort them by area. We then find out the rectangle from the largest empty-space rectangles which contains the target group. Finally, we implement the algorithm of group layout 6 in this rectangle region.

### 6.4 Experiment

We conducted an experiment to compare the performance and users’ preferences of WindowsTagging and *Exposé* (Figure 6.6) in different scenarios. To understand if the
search function can improve the performance of switching, we use two prototypes for WindowsTagging, one is with search function (refer to as Prototype SFWindowsTagging (Figure 6.8)), and another one is not (refer to as Prototype NSWindowsTagging (Figure 6.7)).

Figure 6.6: Exposé view of 12 windows, when moving mouse on the thumbnail of window, the window’s title is displayed.

6.4.1 Hypothesis

H1 WindowsTagging is expected to reduce the switching time compared to Exposé. Because WindowsTagging uses thumbnails overlapping to intuitively represent groups, this representation of groups will improve the finding of individual windows through the visual recognition of other windows belonging to the same group giving hint towards the right window.

H2 SFWindowsTagging is expected to reduce the switching time compared to NSWindowsTagging, especially when the number of windows is high.
6. WINDOWSTAGGING: QUICK TASK SWITCHING USING Tags

Figure 6.7: NSWindowsTagging view of 12 windows, when moving mouse on the thumbnail group, all windows within the group are expanded and all windows’ title are displayed in columns from top to bottom by the window z-order.

6.4.2 Apparatus

WindowsTagging prototype has been implemented on Windows operating system using C++. We used our implementation of an Exposé clone to perform the experiment in a Windows environment and our implementation is similar to the Apple’s Exposé (WindowsTagging and Exposé clone use the similar layout algorithm) (Figure 6.6, 6.8 and 6.7). We used a PC running Microsoft Windows XP using a 22 inch LCD monitor with a 1680 × 1050 resolution.

6.4.3 Participants

9 people (6 male, 3 female) with a mean age of 27.5 (SD=2.87) participated. They were recruited from our lab and university (2 civil engineer, 1 mechanic, 1 electronic engineer, 1 chemical engineer and 4 computer scientists) and said they spent at least 8 hours a day...
6.4 Experiment

Figure 6.8: *SFWindowsTagging* showed the result that users entered "ppt" in the text box, the search dialog located at the left-top (the red rectangle region). Each time a new character is entered or removed, the list of the windows corresponding to the search is updated and the windows that do not match are darken but their content remains visible through transparency.

working on the computer. Among them, eight use a Microsoft Windows system and one uses a Microsoft Windows system and Apple Mac OS X simultaneously.

6.4.4 Experimental Design

A repeated measures within-subjects design was used. The independent variables were TECHNIQUE with 3 levels (Exposé, *SFWindowsTagging* and *NSWindowsTagging*), number of windows NUM with 3 levels (8, 12, 16), distribution of window size DIS_SIZE with 3 levels (SmallWindow: NormalWindow: LargeWindow (the definitions are the same as in Chapter 5 Section 5.4.5), 2:1:1 (Size211), 1:2:1 (Size121), 1:1:2 (Size112)), VISUAL SIMILARITY (VS) with 3 levels (NVS, LVS, HVS).
The two main factors are the window switching techniques (Exposé, SFWindowsTagging and NSWindowsTagging) and the scenario conditions (Num, VS, DisSize). The main measure is the completion time and error rate to perform a window switching technique to find the window of interest in different scenarios. Our experiment used real application windows such as Notepad, Word and PDF document as the target windows. We believed that participants would easily be able to recognize real application windows.

The number of groups may affect the performance of WindowsTagging, and the amount of overlapping determines the number of groups in our technique WindowsTagging. So in here we use the number of groups to replace this factor in our experiment. With the 8 windows, the number of groups in the WindowsTagging is 5 on average, and for 12 and 16 windows, it is 7 and 9 on average. The size and position of each window are random but they have to respect the number of groups in a trial.

The experiment consists of 81 (Num x VS x DisSize x Technique) trials. Orders for the techniques, the number of windows, visual similarity, and distribution of window size conditions were counter-balanced across participants using a balanced Latin-square.

6.4.5 Procedure

A trial consists of a series of three window switching techniques. During each trial, the participants’ task was to find the window of interest by using three window switching techniques in different scenarios. For Exposé, first pressing the F9 key on the keyboard to call it, then finding the window of interest and selecting it. For NSWindowsTagging, first pressing the F10 key on the keyboard to call it, then finding the window of interest and click it. For SFWindowsTagging, first pressing the F11 key on the keyboard to call it, then typing characters to select the window. Each time a new character is entered or removed, the list of the windows corresponding to the search is updated and the windows that do not match are darken but their content remains visible through transparency. When only one thumbnail remains visible then the user can press the enter key to bring the corresponding window to foreground or if several remains visible he can use the mouse to select the thumbnail of interest.

Before starting each part of the experiment, participants had a 15-20 minutes training period to get used to the switching techniques and windows content (participants allowed
to train as long as they want, training until they feel at ease with the technique). Participants are instructed to "perform as fast as possible without error". Participants first pressed a "Layout" button to initialize a windows layout, and the target window was presented to participants (the title + icon), then pressing the "start" button to start a trial, participants were asked to find the target window with one switching technique. After finishing this process, they were asked to press the space bar to start the next trial until (s)he had completed 4 successful trials. Then the system gave the prompt to change the technique to continue. When participants had successfully performed three techniques in one scenario, the system gave the prompt for another scenario. We recorded the amount of time it took a participant to select the target window, starting from the time a participant to click the "start" button. If the participant switched to a wrong window, we recorded an error.

6.4.6 Results

The dependent variables are the switching time (ST in seconds) and the error rate. The switching time is measured from the time the participant clicked the "start" button to the time the participant brought the window of interest to foreground.

6.4.6.1 Switching Time

A repeated measures analysis of variance (ANOVA) showed a significant main effect for NUM (F(2,16) = 54.19, p<0.001), VS (F(2,16) = 90.26, p<0.001), DISIZE (F(2,16) = 40.60, p<0.001) and TECHNIQUE (F(2,18) = 30.53, p<0.001) on switching time.

We use the LSD (Least Square Difference) test with $\alpha = 0.05$ for pairwise comparisons between techniques condition on switching time (We mainly focus on techniques). Table 6.1 shows the results of those tests in detail.

More interestingly we found some significant interactions between NUM and TECHNIQUE (F(4,32) = 12.42, p<0.008), between VS and TECHNIQUE (F(4,32) = 7.36, p<0.027). Pairwise comparisons were used to analyze and interpret these two-way interactions.

NUM x TECHNIQUE. For all levels of NUM, SFWindowsTagging is faster than other techniques (H2 is supported). For the 8 windows, we found no significant difference among them and the difference among means is small (SFWindowsTagging was 10.5%
Table 6.1: Pairwise comparisons between techniques condition on switching time. A cell contains the means difference and the lower and upper bound of the confidence interval. Underlined cells are significant.

<table>
<thead>
<tr>
<th></th>
<th>Exposé</th>
<th>NSWindowsTagging</th>
<th>SFWindowsTagging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exposé</td>
<td>0</td>
<td>0.074</td>
<td>0.829</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.512</td>
<td>1.273</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.950</td>
<td>1.718</td>
</tr>
<tr>
<td>NSWindowsTagging</td>
<td>-0.950</td>
<td>0</td>
<td>0.562</td>
</tr>
<tr>
<td></td>
<td>-0.512</td>
<td>0</td>
<td>0.761</td>
</tr>
<tr>
<td></td>
<td>-0.074</td>
<td>0</td>
<td>0.959</td>
</tr>
<tr>
<td>SFWindowsTagging</td>
<td>-1.718</td>
<td>-0.959</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-1.273</td>
<td>-0.761</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-0.829</td>
<td>-0.562</td>
<td>0</td>
</tr>
</tbody>
</table>

faster than NSWindowsTagging and 2.8% faster than Exposé). For the 12 and 16 windows conditions, SFWindowsTagging was significantly faster than NSWindowsTagging and Exposé (p<0.001) and we also observed a significant difference between NSWindowsTagging and Exposé (p<0.028). On average, for the 12 windows, SFWindowsTagging was 25.9% faster than NSWindowsTagging and 42.1% faster than Exposé, NSWindowsTagging was 12.8% faster than Exposé, and for 16 windows, SFWindowsTagging was 27.5% faster than NSWindowsTagging and 60.7% faster than Exposé, NSWindowsTagging was 25.5% faster than Exposé (Figure 6.9) (H1 is partially supported, when the number of windows is high (12/16), it is supported).

VS × Technique. There was a significant difference (p<0.017) between SFWindowsTagging and other techniques for all visual similarity conditions. On average SFWindowsTagging was 16.2% faster than NSWindowsTagging and 28.2% faster than Exposé for the 8 windows, and SFWindowsTagging was 14.7% faster than NSWindowsTagging and 33.3% faster than Exposé for the 12 windows, and SFWindowsTagging was 31.1% faster than NSWindowsTagging and 44.3% faster than Exposé for the 16 windows. For HVS and LVS conditions, we found no significant difference between NSWindowsTagging and Exposé (On average NSWindowsTagging was 13.2% faster than Exposé) (Figure 6.10).
6.4.6.2 Error Rate

The overall error in the experiment was 3.1%. A new ANOVA only showed a significant main effect for Num ($F_{2,16} = 5.75$, $p<0.043$) on switching errors. We found no significant main effect for other visual factors and we also found no significant interactions among Num, VS, DisSize and Technique. When the number of windows is high, SFWindowsTagging was less error-prone than other techniques (Figure 6.11).

6.4.6.3 Qualitative Results

One final question was presented for those three techniques, inquiring whether they prefer to use Exposé or SFWindowsTagging or NSWindowsTagging.

All of 9 participants reported that they preferred SFWindowsTagging when the number of windows or the visual similarity is high. They said they often remembers some characters of a window’s title, even if sometimes it was not precise match with the target.
6. WINDOWSTAGGING: QUICK TASK SWITCHING USING TAGS

![Figure 6.10: Mean switching time (ST) in s for VS and TECHNIQUE. Error bars represent 95% confidence interval.](image)

windows, they also would like to filter down some windows using the search function, then using the mouse to select the target window. When the number of windows or visual similarity is high, it was really difficult to recognize the thumbnails.

6.5 User Evaluation

To understand how WindowsTagging might be used in real situation, we performed a longitudinal field study on a small number of users over 15 days.

5 people (3 male, 2 female), age between 25 and 32, participated in the study. There were 2 computer scientists, 1 system administrator and 2 geoscientists. Among them, one used a Ubuntu Linux system \(^1\) and a Microsoft Windows system simultaneously, three used a Microsoft Windows system and one uses a Microsoft Windows system and

\(^1\)http://www.ubuntu.com/
Apple Mac OS X simultaneously. For three Windows system users, they know the Exposé function on Apple Mac OS, but they never used it. The user who used a Ubuntu Linux and Windows system and used a similar function (Scale plugins) with Compiz Fusion ¹ on the Ubuntu Linux. Two of them used a two monitors, and other used a single monitor.

### 6.5.1 Using Step-by-Step

Participants were instructed how to use WindowsTagging technique step-by-step and were given an executable. At the beginning to the study, participants were instructed to use WindowsTagging with those functions (automatic management): implicitly define groups (pressing F9 to automatically implement), switching window or group and the function of searching windows by their title. After five days, participants were interviewed to collect details and comments about how they utilized WindowsTagging and how useful

---

¹http://www.compiz.org/
they found it to be. Meanwhile participants were instructed that they could use some new functions (manual management), including drag-and-drop operation, visible windows snapshot operation, edit group, splitting group and delete group. After another five days, we interviewed them and collected details and comments about how they utilized WindowsTagging and how useful they found it to be. Meanwhile we introduced the window tagging mechanism to them and instructed how to use it (window tagging), including adding tags, editing tags, deleting tags and searching windows or groups by tags. Finally, participants were interviewed to collect details and comments about how they utilized WindowsTagging and whether they found it useful.

6.5.2 Results

All participants found that WindowsTagging was very useful, and easy to use to define and switch between windows/groups. One participant who used both Windows and Mac OS simultaneously said that WindowsTagging was much improved compared to Exposé, the stable windows layout strategy was very useful. Another one who used both Windows and Ubuntu simultaneously said that it was very cool. He often arranged windows on desktop to finish tasks and WindowsTagging provided a very powerful tool. All participants reported that the search function was very good and important for them, it was easy to use and could quickly find the window of interest.

All participants commented that the window tagging mechanism provided a good way to help them mark their tasks that could help to quickly resume and find tasks, and it was easy to use and understand. They also hoped that we can provide a prompt dialog on the desktop to display some of recently accessed tasks. Two participants reported that this grouping mechanism is very interesting, they often use it, but other two participants said that they preferred to use drag-and-drop operation, because this operation provided an intuitive way to define groups.

WindowsTagging has also revealed potential usability problems. One participant said that the algorithm of implicitly creating groups sometimes did not work very well. She did not like to rearrange some windows manually and she hoped the system could improve the automatic creation of groups. Secondly, when the number of window is high, the legibility of some thumbnails was poor. Although the search function was a good way to help them find the window of interest, but sometimes as one of their hands was always...
on the mouse, moving the mouse to click the target window was more convenient than typing the character.

Using a 5 points Likert scale, participants rated the technique as useful (averaged response = 4.02) (1=disagree, 5=agree) and easy to use (3.83). Participants reported that the technique is useful to implicitly create groups (2.87) and switch between groups (4.04). Meanwhile Participants also rated that the technique’s search function is useful to find windows and groups (4.25). Table 6.2 shows all questions and users’ average response.

6.6 Discussion

6.6.1 Overlapping Mechanism

WindowsTagging share similarities with other window switching / grouping techniques. Like Exposé, Taskposé, WindowScape and Scalable fabric, WindowsTagging uses thumbnails to represent windows. In contrast to these techniques where thumbnails legibility can be really low, and WindowsTagging uses the overlapping mechanism, it can have large groups of thumbnails, and it can improve the legibility of groups.

We use thumbnails overlapping to intuitively represent groups. Compared to other grouping techniques like Groupbar where buttons represent groups, we think it can improve the recognition of groups by their visual content. In addition we expect that this representation of groups will improve the finding of individual windows through the visual recognition of other windows belonging to the same group giving hint towards the right window. Compared to Exposé, this may improve the efficiency of window switching.

6.6.2 Grouping Mechanism

WindowsTagging is based on automatic group definition, but also supports users to explicitly define groups by drag-and-drop, splitting group and window tagging operations. Therefore the layout of groups may be very different but reflecting users expectations and it can have potential advantages to help users to find the window/group of interest. Users can also organize groups using different shapes to help them reducing the visual search time.

Like Taskposé Bernstein et al. (2008b) and WindowScape Tashman (2006), WindowsTagging allows windows to reside in multiple groups simultaneously, so users can
Table 6.2: User satisfaction averages for on a five point scale where 1 = useless, 5 = useful.

<table>
<thead>
<tr>
<th>Questionnaire Item</th>
<th>Average Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is useful for users</td>
<td>4.02</td>
</tr>
<tr>
<td>It is easy to use</td>
<td>3.83</td>
</tr>
<tr>
<td>It is useful to be able to predict groups by the implicitly define group mechanism</td>
<td>2.87</td>
</tr>
<tr>
<td>It is useful to be able to rearrange some windows to change the groups structure</td>
<td>3.78</td>
</tr>
<tr>
<td>It is useful to be able to switch between groups</td>
<td>4.04</td>
</tr>
<tr>
<td>It is useful to be able to use window tagging mechanism to define groups</td>
<td>3.40</td>
</tr>
<tr>
<td>It is useful to be able to use visible window snapshot function to define groups</td>
<td>2.82</td>
</tr>
<tr>
<td>It is useful to be able to operate groups</td>
<td>3.23</td>
</tr>
<tr>
<td>It is useful to be able to use search function to find windows or groups</td>
<td>4.25</td>
</tr>
<tr>
<td>It is useful to be able to allow one window in multiple groups at the same time</td>
<td>3.23</td>
</tr>
</tbody>
</table>
6.7 Conclusion

Easily switch windows from multiple groups at once, and without affecting the grouping structure.

Window tagging provides a powerful and flexible windows grouping mechanism, especially, when it combines the search function. This mechanism has shown a powerful advantage for window switching. It can easily address the problem that one window can be in multiple groups at the same time. It provides the important semantic information for groups, making task resumption easier. Although when tags are widely used in the Internet, it has shown some issues \cite{Golder2006}; \cite{Guy2006}; \cite{Mathes2004}, such as semantic ambiguity. Compared to the Internet, people only need to manage very few tags on the desktop and we can also provide some mechanisms to help users to distinguish the used tags (e.g. when typing tags, we can prompt the used tags and give some candidate keywords).

6.6.3 Search Mechanism

We did not find any commercial window management systems to support searching windows by their title. However the experiment and the longitudinal study have proved that this is very effective to switch windows and groups, especially when we add the tags to windows and groups, this is more effective.

6.6.4 Stable Spatial Layout strategy

The stable spatial layout has proved that is very useful for users revisiting a target \cite{Tak2009b}. But considering the existing grouping and switching techniques, only WindowScape \cite{Tashman2006} uses a stable spatial layout to help finding previously switched windows. Taskposé \cite{Bernstein2008b}, GroupBar \cite{Smith2003} and most similar systems do not support this feature, and WindowsTagging implement the algorithm of creating groups by keeping a relative stable spatial layout.

6.7 Conclusion

In this chapter, we first discussed and gave eleven design principles which were based on the presented issues by the existed task switching techniques and real-world usage data (a longitudinal study). Those design principles provided a theory foundation to help
designers to develop new window switching techniques. We then designed and developed a prototype system called WindowsTagging based on those design principles. WindowsTagging combines implicit and explicit definition of groups, spatial and visual memories to help users to quickly find windows or groups and switch between them, providing a lightweight alternative to other grouping techniques.

Windows are represented as large thumbnails which can be dragged and dropped by the user. WindowsTagging allows groups to be defined implicitly and explicitly and windows can exist in multiple groups simultaneously. Window tagging mechanism provides a more powerful and flexible grouping techniques to help users to explicitly define groups. By using tags, users allow to go through the semantic information to find the windows and resume their tasks. The drag-and-drop and splitting group functions were provided to allow users to intuitively and explicitly define or modify groups. A search function is also provided to help users to find the desired windows by their titles or tags.

We conducted an experiment to compare the performance and error rate of WindowsTagging to Exposé. Results showed that WindowsTagging was faster than Exposé technique, and participants strongly preferred it. We also showed in a user longitudinal study that WindowsTagging was very effective and can improve task management.

6.8 Future Work

We will perform a long-term longitudinal field study with about 20 participants over 4-weeks in order to understand how people actually use the WindowsTagging through log analysis and users feedbacks. We will also continue to complete the "theory" of task switching and expand the window tagging mechanism to improve the efficiency of task switching.
Chapter 7

Conclusion And Future Work

In this dissertation, we have described a set of contributions to the study and development of window management techniques. The work presented in this dissertation contributes to understand users’ activities on window management, build the theory of window switching and conceptualize some operations, and provide some design principles to improve window switching techniques. In this final chapter we first re-examine and discuss the research objectives presented at the beginning of this dissertation. We then summarize the findings and conclusions of the work. Finally, we provide several potential avenues for future work.

7.1 Research Objectives

The over-arching goal of this dissertation was to understand and improve window switching techniques. More specifically, the dissertation set out to achieve four goals:

1. Understand users’ activities on window management and the reasons users choose to employ switching techniques.

2. Understand current window/group switching techniques where and when they are effective and ineffective.

3. Theorize window/group switching and define types of switching operations which have been justified by read-world usage data, and then provide some design principles to help designers to design new switching techniques.
4. Using the knowledge gathered in the previous goals, analyses, design and evaluate new switching techniques based on the design objective aforementioned.

We defined these four goals at the beginning of this dissertation, now we examine and discuss the implementation of these goals with the alignment of the research outputs.

Objective 1 was completed via a log-based longitudinal study and a questionnaire investigation. To implement this study, WindowsOSLog was developed to log user actions in mainstream Windows operating system. The window management activities of 26 participants were monitored over 5-weeks. Chapter 3 presented the process and results of this study in details. A large range of activities statistics were reported including the number of windows opened simultaneously on the desktop, the distribution of window size, the occurrence ratio of the main window events, the distribution of window switching techniques, types of switching of users requirement, the number of visible windows and the number of window groups.

Objective 2 was completed via a review of previous work and an experiment to compare current window switching techniques in different scenarios. Chapter 2 described a review of the current knowledge in the domain of window and group switching techniques, the disadvantages and advantages of some window switching techniques were also presented in this chapter. However the previous work could not accomplish this goal, so an experiment was conducted to help us to achieve this goal. Chapter 5 presented the experiment to compare all mainstream window switching techniques in different scenarios. The results showed that Taskbar was the best choice when the number of windows is small and for users who always maximize their windows, Alt+Tab was the best choice when the number of windows is important.

Objective 3 required an explanation of the observations from the longitudinal study and a review of previous work. The previous work was presented in Chapter 2. Chapter 6 presented the eleven design principles, including: 1) allow implicit and explicit methods to create groups, 2) keep the position relationship among windows as much as possible with automatically defined groups, 3) allow a window to be in multiple groups at the same time, 4) allow users to explicitly change groups structure, including adding/removing windows in groups and creating new groups using existing groups, 5) allow users to switch between windows or groups or selected windows of groups, 6) allow users to switch between windows from multiple groups at once and without affecting the grouping structure, 7)
allow users to add semantic information to mark their tasks, 8) allow users to search windows or groups by some keywords, 9) keep a stable groups layout to allow users to make effective use of their spatial memory, 10) provide some operations for groups, such as move, resize, and 11) allow users to delete groups.

Objective 4 was deemed successful if the newly designed window switching techniques was significantly faster and subjectively preferred over the traditional window switching techniques. Chapter 4 presented the analysis, design and evaluation of the Push-and-Pull Switching, a window switching technique using window overlapping to implicitly define groups. The empirical evaluations found that Push-and-Pull Switching was 50% faster than other switching techniques in different scenarios. The longitudinal user study indicates that participants invoked this switching technique 15% of the time on single monitor displays while they found it easy to understand and use. Chapter 5 presented stack scanning, a window switching technique based on a widget that combines generalized scrolling and crossing to control the stack order of layers of visible windows. The empirical evaluations found that stack scanning was faster than other techniques when the number of windows is high and the visual similarity among windows is important. Chapter 6 presented WindowsTagging, a task switching techniques that combines implicit and explicit definition of groups, spatial and visual memories to help users to quickly find windows or groups and switch between them. Window tagging mechanism was first used in the domain of task switching techniques to allow users to explicitly create or modify groups by tags. The empirical evaluations found that WindowsTagging was faster than Expose technique, and participants strongly preferred it. The longitudinal study also showed that WindowsTagging is very effective and can improve task management.

Finally, in line with the overall research objective, this dissertation has presented characterizations that have significantly improved the research knowledge of window switching techniques. The value of this knowledge to improve window switching techniques was demonstrated in the design of the Push-and-Pull Switching, stack scanning and WindowsTagging.
7. CONCLUSION AND FUTURE WORK

7.2 Conclusion

The motivation of this dissertation was to understand user window management activity and develop new switching techniques to support people more efficiently and conveniently to interact with computer. To achieve these goals, a log tool was developed to record user window management activity in mainstream Windows operating system. 26 participants participated in this study and the duration was over 5-weeks. Push-and-Pull Switching and stack scanning switching techniques were developed based on the results of log-based longitudinal study, and a series of experiments were designed and implemented to evaluate them. To theorize window/group switching, we defined types of switching operations and introduced window tagging mechanism to provide a new alternative to the existing windows grouping mechanisms, and then provide eleven design principles to help designers design new switching techniques. Finally, we proposed a prototype system, WindowTagging, was designed and implemented based on those design principles. The empirical evaluations showed that WindowTagging was faster than Exposé technique and participants strongly preferred it.

Our contributions are:

1. A review of window/group switching techniques and related window management systems. This review allow other researchers to more quickly understand and analyze current switching techniques.

2. A log-based longitudinal study about user window management activity, a log tool called WindowsOSLog was developed to record user window management activity in mainstream Windows operating system. 26 participants’ window management activities were recorded during a period of 5-week.

3. Design and evaluation of Push-and-Pull Switching. Push-and-Pull Switching is a window switching technique using window overlapping to implicitly define groups. Push-and-Pull Switching further allows to switch between groups and restack the focused window to any position to change its group affectation. The technique was evaluated in an experiment showing that Push-and-Pull Switching allows to improve 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 while they found it easy to understand and use.

4. Design and evaluation of stack scanning. Stack scanning is based on a widget that combines generalized scrolling and crossing to control the stack order of layers of visible windows. With stack scanning, the visual information for each window is maximized as each window remains at its original size and while the ordering by frequency is preserved. The empirical evaluations showed that stack scanning was faster than other techniques when the number of windows is high and the visual similarity among windows is important. They also showed that Taskbar was the best choice when the number of windows is small, regardless of other visual factors conditions, and for users who always maximized each window, Alt+Tab was the best choice when the number of windows is important.

5. Provide eleven design principles and introduce window tagging mechanism to help designers design new window switching techniques. Those design principles provides a theory foundation to designers and researchers.

6. Design and evaluation of WindowTagging. WindowsTagging was designed based on the eleven design principles. It combines implicit and explicit definition of groups, spatial and visual memories to help users to quickly find windows or groups and switch between them. WindowsTagging represents windows as thumbnails, displaying all groups at the same time and allowing windows to exist simultaneously in multiple groups. WindowsTagging exploits users’ spatial and visual memories by providing a relative stable spatial layout with larger thumbnails using overlapping. Groups are first automatically created based on a window overlapping algorithm. Window tagging mechanism was used to allow users to explicitly create or modify groups by tags. The Drag-and-drop and splitting group functions also allow users to explicitly define or modify groups. A search function was presented to help users finding a window by its title or tags. An experiment was implemented to compare the performance and error rate of WindowsTagging to Exposé. Results showed that WindowsTagging was faster than Exposé technique, and participants strongly preferred it.
7. CONCLUSION AND FUTURE WORK

This research provides the foundation for further work in this area. Researchers can use these findings as a platform for future observations of switching techniques and designers can use these observations and design principles to design new switching techniques.

7.3 Future Work

Window switching is a well known problem in overlapping windows environments and is far from being solved. This dissertation has provided to the best of the author’s knowledge. The remainder of this section suggests possible areas for future work.

7.3.1 Window Tagging

*Push-and-Pull Switching* and *stack scanning* are based on windows stacking and only allow to implicitly define groups. They provide effective operations to manage users windows stacking, and their grouping mechanisms may limit their use. Window tagging provides a powerful windows grouping mechanism, adding this mechanism to *Push-and-Pull Switching* and *stack scanning* can expand their use. Window tagging mechanism can easily add to these two techniques. For example, adding a menu item to system context menu, then users can click the right button on the target window to popup a menu and add tags to this window. Users can select the content from the target window or input any valid character as tags. For *stack scanning*, the label of each layer button is only used to identify the number of layers. When we add tags, we can use them to display tags, this can help users to quickly find the group with the tags.

7.3.2 Theorize Window Switching

This dissertation has provided eleven design principles to help designers to develop new window switching techniques. We would like to continue to theorize window switching and related problems. A further longitudinal study could be implemented to wider participants to understand the different types of window switching operations that might be desirable for different group participants. Then we should think about the ways these operations can be conceptualized. *Push-and-Pull Switching, stack scanning, WindowsTagging* and
future techniques should be seen as partial and potentially combinable solutions with this context. They should be described in terms of the operations they support and the concept they use to present them.

### 7.3.3 Implicit Grouping Techniques

Researchers have proposed various implicit grouping techniques, such as *Push-and-Pull Switching* Xu & Casiez (2010), based on overlapping windows; *WindowScape* Tashman (2006), using all windows on the desktop except minimize windows; Taskpósé Bernstein *et al.* (2008b), based on the results of a ‘WindowRank’ algorithm; and SWISH Oliver *et al.* (2006), using temporal relationships between window focus events as well as window titles to establish semantic relationships (its evaluations suggested 70% accuracy rates in assigning windows to task groups). There was not any study and experiment to compare the accuracy rates of predicting groups structure between them. So we do not know which method is the best or in which conditions they are useful.

The direct and effective method is to use the real-world data to compute the accuracy rates of predicting groups by their grouping algorithms. A further longitudinal study could be implemented to employ a wider range of users to log their window management activities. Then we construct active windows sequences and use each grouping algorithm to construct groups to analyze the accuracy rates of predicting. The analysis of active windows sequences also provides an effective way to create new implicit grouping techniques.

### 7.3.4 A Standardized Evaluation Framework

Window management researchers would benefit from a standardized methodology for evaluating newly developed window or group switching techniques, as is available in other areas of HCI. All window and group switching techniques evaluations are currently performed in an ‘ad hoc’ manner, with individual researchers selecting windows, tasks and conditions that they believe to be a fair evaluation of their system. Unfortunately, this makes it difficult for later comparison of switching techniques without reproduction of the original evaluation.
7. CONCLUSION AND FUTURE WORK

A switching techniques evaluation framework would create standardized windows and groups (based different grouping algorithm). It would provide some standardized scenarios (tasks) that can be used to evaluate switching techniques (Chapter 4).

The framework would integrate current standardized window switching techniques, such as Direct pointing, Alt+Tab, Taskbar and Exposé. Researchers can easily set up experiments to compare the performance of their new techniques to those standard techniques. Taken into account the performance of window and group switching techniques depend on the windows layout, visual similarity, and so on, so the framework would allow researchers to set visual factor parameters to create a window layout, include the number of windows, distribution of window size, the amount of overlapping, window z-order and position and visual similarity between windows.

However, the frameworks must still be supported by standardized evaluation methodologies, such as the ISO9241 Douglas et al. (1999) standard that specifies the evaluation conditions for Fitts’ law Fitts (1954) experiments.
References


REFERENCES


REFERENCES


161
REFERENCES


REFERENCES


163
REFERENCES


164
REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


