Bumping Windows between Monitors

Tim Regan
Mary Czerwinski
Brian Meyers
Greg Smith

3/12/2003

Technical Report
MSR-TR-2003-13

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Bumping Windows between Monitors

Tim Regan, Mary Czerwinski, Brian Meyers, and Greg Smith
Microsoft Research, 1 Microsoft Way, Redmond, WA 98052 USA
timregan@microsoft.com

Abstract: Users' move from single to multiple monitors so that they can use more screen real estate. This increase enables them to keep a greater number of windows open and visible at the same time. But there is a cost: arranging a window takes far longer since there is more screen space to traverse and more relationships between windows to take into account. To address this we added 'bumping' to an application and tested it in a user study. Bumping allowed users to automatically move a window across multiple monitors. In this paper, we present our experiment contrasting three styles of bumping. We found less window repositioning required when users were given bumping in contrast to their existing practices. However we also found that simple implementations of intelligent bumping are not predictable, causing problems for our participants.

Keywords: Windows management, multimon, multiple monitors

1 Introduction

As increasing numbers of users turn to multiple monitors to solve their problems of limited screen real estate, new problems with windows management arise, problems not dealt with by the operating system. To quote Grudin (Grudin, 2001):

"Multimonitor development has focused on getting the system display software and the application program interface to work, with little attention to the human computer interface or intelligent exploitation by system or applications"

One such problem is the increased effort in arranging windows. Multiple monitor users can (and do) keep more windows open, but as they drag windows to arrange them there is more screen space to traverse. The traditional single monitor method of minimizing the window or moving it further back in the z-order of open windows is often not used on multiple monitors, as there is the space available to keep the window visible. We know from Fitt's law that the time taken to move the cursor to an item on the screen is proportional to the log of the distance to the item over the items size.

To make windows arrangement faster and easier on multiple monitors we added a new feature that we called bumping to an application. Bumping took a window and automatically moved it to a new position. We implemented three different bumping methods:

1. Faithful bumping maintains the relative position of the window. If the source and target screen are the same size (in pixels squared) then no resizing will take place.
2. Dark-space bumping moves and resizes the window to cover a currently unobscured region of the desktop.
3. Unobscured bumping moves and resizes the window to cover only unobscured desktop or windows that are already partially obscured.

To test the efficacy of these bumping methods against users' existing practices we conducted a user study, the results of which are presented in this paper. Firstly we present related work and give more detail of our design and implementation.

2 Related Work

Data visualization techniques often have to deal with packing a large number of visible items (data points) into a limited screen space. For example, in Valence (Fry, 2002) Fry looks at building representations that explore the structures and relationships inside very large sets of information in a limited space. But unlike generalized visualizations, windows are simple objects: they are rectangular, they need to be large enough for their contents and surrounding tools to be legible, they live on a rectangular background, and there are not that many of them open at any one time. Hence we can restrict our attention to algorithms for manipulating rectangles. In (Bell and Feiner, 2000), Bell and Feiner detail the algorithms they have used to manage the layout of rectangles, covering notions of empty space and its dual, and how to add and remove rectangles from each.

There are a number of interaction techniques and representational metaphors that work on, and exploit, large displays. Example display metaphors include fish-eyes (Furnas, 1986), zoomable UIs (Bederson, 2000), and focus plus context screens (Baudisch et al, 2001). Flow menus (Guimbretière et al, 2001) and gestural interactions (Myers et al, 2002) are good examples of large screen interaction techniques. Whilst informative, these do not deal directly with the problem of automatic layout of information or windows.

One solution to the windows arrangement problem is to swap from overlapping windows to other possibilities, for example tiling windows (Bly and Rosenberg, 1986). Today, innovative ways of handling windows are examined to handle new tasks and new ways of working. For example in (Kandogan and Schneiderman, 1996), Kandogan and Shneiderman revisit tiled windows to address tasks like photo sorting, where masses of images need to be addressed by the user at once. A categorization of such windows coordination actions is contained in (North and Schneiderman, 1997). Similarly in (Beaudouin-Lafon, 2001), Beaudouin-Lafon proposes rotating and peeling back windows as a way to address the increasing number of open windows made possible by today's powerful PCs. Through our notion of bumping we seek to tackle the problem of window arrangement without abandoning overlapping windows, since they are the established norm.

The closest work we have found to our notion of bumping is by Hutchings and Stasko in (Hutchings and Stasko, 2002). Their 'expand and shove' techniques allow users to expand a window in such a way that other windows on the screen are not further obscured or shrunk, though they may be moved. Any movement maintains windows' relative positioning. Expand and shove work together to give more screen space to the window that the user is currently focused on, but they result in older windows collecting towards the periphery of the display. While this may work well on large homogeneous display surfaces, multiple monitors are used differently. Multiple monitor users will turn from one screen to another, changing their experience of which screen is the focus and which peripheral. Expand and shove never make a window (or its visible region) smaller, thus leading to a number of large windows vying for desktop space. Bumping allows users to take up unused desktop with the bumped window – which may involve making it smaller. Bumping is also useful for discarding windows in such a way that they may be easily reacquired later. In summary expand and shove are useful for increasing the visible area of the window a user is currently attending to without sacrificing relative positioning of windows and without complex windows arrangement. Bumping is useful when a user's attention is swapping from one window to another.

3 Arranging Windows

When people move windows aside, where do they want to put the window? A field study to answer this question accurately is beyond the scope of this paper, but we can analyze the possibilities.

Move and refer to

Here the user wants to move a window away and start work in another window, while keeping the first visible for reference. Suppose that a user is working on a spreadsheet and becomes stuck trying to perform a complex sequence of data manipulation. He may turn to the web to find relevant newsgroup threads and having read them; move the newsgroup browser aside to return to my spreadsheet. However he still needs to refer to the suggested solution, so the browser must remain visible. The reference may be purely visual or may involve transferring information (e.g. through cut-and-paste or drag-and-drop) between open windows.

Move and keep working

Here the user wants to move the window to a new position and keep working on it there. For example, a user may turn to his secondary screen and start writing an email, but if the email becomes complicated he may want to move it to his primary work area. This would be a semantic-move in that it positions the window following the user's semantic designation of the screen space. Alternatively a user may turn from some code one is writing on her new 21" monitor to perform some SQL queries on her old 15" screen. As the data becomes hard to view on the smaller screen she may move the window to the larger screen. This would be a quality-move in that it positions the window following qualitative differences within the screen space. There is overlap between the notions of semantic-move and quality-move; users may designate one screen as their secondary monitor on which they conduct peripheral tasks purely because of qualitative differences between that and their primary screen. These qualitative differences extend beyond the technical specification of the screen and graphics card. One of the participants in the experiment described later in this paper who ran a three screen set-up reported using the left hand screen for instant messaging and emailing friends because the left monitor was not visible to people walking past him, and thus maintained his privacy.

Move and return to later

Single screen users often minimize windows they wish to stop working on now but return to later. For example a user may turn from authoring a report to answer incoming email. Once the email is dealt with she returns to the report. There are disadvantages with minimizing windows, disadvantages associated with the limited size of the Microsoft Windows start-bar or the Mac task-switcher. Because both are small they quickly get crowded with icons if many applications are running. Because they are small they cannot effectively leverage users' spatial navigation skills when users seek to return to an application. Placing a window on another screen allows users to get on with their work and more easily go back and get the window when it is required.

Remove

Having become acquainted with using window moves in preference to minimize, multiple monitor users may also associate the 'window move' action with the 'get window out of the way' intent, thus making use of fast muscle memory. Hence some moves may in fact be in place of closing windows.

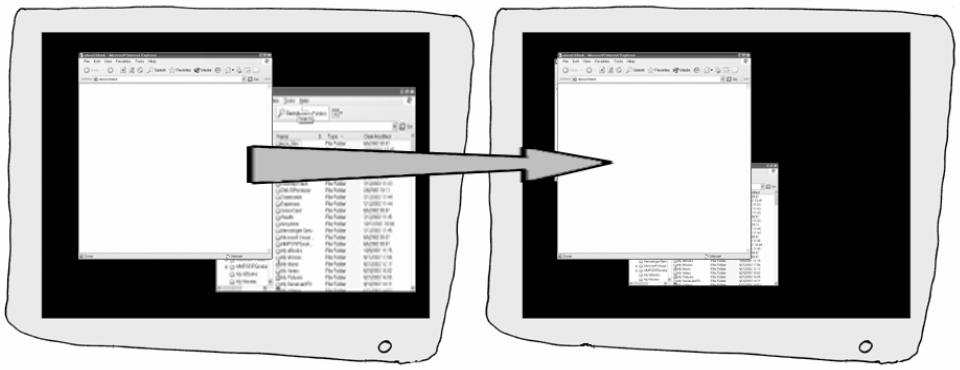

Figure 1: Faithful Bumping

Figure 2: Dark-space Bumping

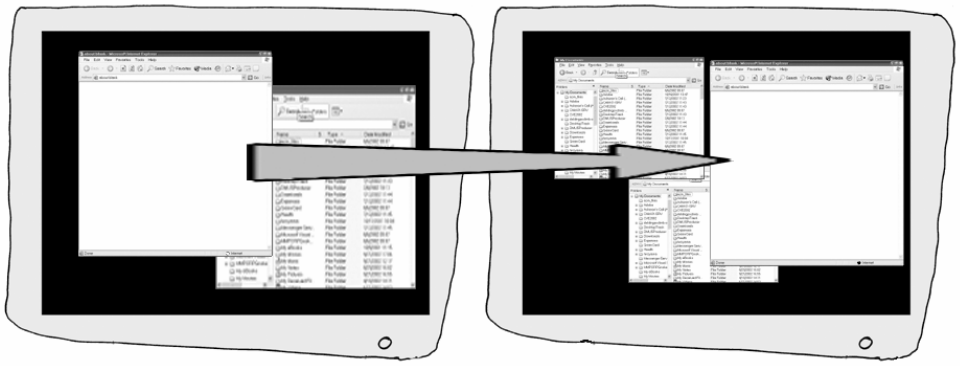

Figure 3: Unobscured Bumping

4 Bumping Functionality

In this section we will explain the bumping mechanism and discuss our design choices.

As mentioned in the introduction we implemented three bumping methods:

1. Faithful bumping,
2. Dark-space bumping, and
3. Unobscured bumping.

Faithful bumping is the easiest to understand and is shown in Figure 1. The bumped window moves from one screen to another, and maintains its relative position. In Figure 1 the clear IE window on the left hand screen is moved as indicated by the grey arrow to the same size and position on the right hand screen. We implemented two styles of faithful bumping: one that retains the absolute size of the window and another that resizes the window so that it occupies the same proportion of the new screen. This is particularly useful for multiple monitor users who have

used an old discarded monitor as their second screen. This second screen is often physically smaller and runs at a lower resolution than the primary monitor. In these conditions a faithful bump of a window, from the primary to the secondary screen, may result in obscuring all of the second screen without displaying the whole window. Resizing during faithful bumping is also useful for users with homogeneous screen who run their start bar as a wide horizontal bar at the vertical edge of one of the monitors, as this reduces the available desktop space on that monitor.

Dark-space bumping takes a bumped window and places it inside the largest rectangle of unobscured desktop available in the direction of the bump, resizing if necessary. Figure 2 shows the result of dark-space bumping. The clear IE window on the left hand screen is bumped to the clear IE window on the right hand screen, in order to avoid the two open Windows Explorer windows, its size is reduced.

Like dark-space bumping, unobscured bumping searches for new rectangles of unobscured space in which to position the bumped window. In addition to uncovered desktop it includes windows that are already partly obscured by other windows. Figure 3 shows how the clear IE window on the left hand screen will be bumped to the right hand screen using unobscured bumping. Because one of the Windows Explorer windows is partially obscured by the other, the bumped IE window assumes its new position over a combination of the already obscured window and unobscured desktop space.

Note that if there is sufficient space available, a dark-space or an unobscured bump may not result in the window switching screens but moving on the current screen instead.

Our prototype application was a text editor with two buttons added to a toolbar with arrows signifying 'bump left' and 'bump right'. In most windows based operating systems (e.g. Microsoft Windows, Mac, and X Windows) buttons associated with managing the positioning or size of a window are placed in the window's title bar. Most systems provide alternatives to this. For example, on Microsoft Windows PC the keyboard shortcut Alt-Space M acquires a window for moving, so that pressing the arrow keys will move the window until the user presses Enter to leave the mode. Our bump buttons' placed in the toolbar just below the menus was a pragmatic choice for ease of implementation.

Gestural interaction styles have been successfully applied to large screens (Guimbretière et al, 2001) and to small screens (Perlin, 1998). Instead of a button we could, for example, have had users acquire a window and then flick the mouse to send the window in the direction of the flick. We decided against this in favour of a button push for three reasons:

- Gestural input has not taken off for standard personal computing size screens (i.e. 15" to 21" diagonal screens) which multiple monitors extend
- Gestural input would be a substantial departure from current window manipulation techniques and hence less likely to be adopted

A keyboard shortcut to bump the window was also included in our prototype.

5 Experiment

To test our bumping idea, both quantitatively and qualitatively, we ran a user study. In this section we present the study and discuss the results.

1.1 Research Questions

The specific questions we wished to address were:

1) Is the bumping button effective in reducing the amount of window dragging required of users?

2) Does the bumping button improve productivity?

3) Do users prefer a bumping button?

With reference to the above questions we wanted to know which bumping algorithm performed best.

1.2 Experimental Setup

Participants

17 volunteers (5 female and 12 male) from the greater Puget Sound area were recruited from our company's usability database to participate in the study. Unfortunately 3 cancelled and 3 interpreted the screening question "Do you currently use Windows XP with two or more monitors?" differently from us, bringing our final number of participants to 11 (2 female and 9 male). We wanted to use only people experienced with multiple monitor use for two reasons. Firstly, we have found that the learning effects of multiple monitor use can dominate other factors in studies. Secondly, we wanted to get suggestions from the users on how a bump button should work in practice. Participants' jobs were mainly (but not all) technical. Participants had been using multimon for an average of 2 ½ years (SD = 2 years 5 months). This was not evenly spread: 7 users had been using multiple monitors for less than 1 ½ years while the remaining 4 had been using multiple monitors for more than 4 years. The participants were screened to be intermediate to expert Windows and Office users, as per validated internal screening tools.

Task & Design

We had participants do two tasks four times on a two monitor PC. The two monitors were identical 21" CRTs (i.e. not flat LCD screens) each at 1024 by 768 pixel resolution with the start bar along the bottom edge of the left hand screen. Participants completed a brief questionnaire after each of the four sessions and a longer one at the end. Before each pair of tasks the participants had a short practice session to familiarize themselves with the changes in behaviour of the bump button. The ordering of the conditions was fully balanced across the subjects. The four conditions were the three bumping methods discussed already (faithful bumping, dark-space bumping, and

unobscured bumping) as well as a condition with the bumping buttons removed.

The prototype application we chose was a simple text editor, but with the minimize and the maximize buttons disabled.

Task 1: The Reconstruction Task involved opening 9 Rich Text Format (RTF) files. The contents of 5 of the files were to be found, jumbled line by line, in 3 of the other files. The files opened in the same place and with the same size on left hand screen so that each file is initially obscured by the previous one. The last file to be opened was the instructions. Users had to reconstruct the missing contents of the 5 files using the other 3. So, for example, the line starting "H-10" is line 10 from file H and the user, having located it, would cut-and-paste it back into position in file H. After 3 minutes the users were stopped and asked to start the second task.

Task 2: The Alphabet Task instructions gave users two random lines of ten letters and asked them to recreate them across the two screens using the letter files provided so that the letters were not obscured. Figure 4 shows one screen during the second task in progress.

The Reconstruction Task was designed to involve a lot of switching back and forth between windows: 3 files were required for repeated searching for lines to paste into the other 5 files. The Alphabet Task was designed to require a lot of careful window positioning and resizing. As we have argued already, positioning windows so that they are readable (unobscured) and so that they are available are key windows management tasks on multiple monitor systems and so our tasks are typical, if abstract, windows management tasks.

These tasks abstract the two main behaviours encountered when users move and refer to windows, as discussed in Section 3. Task 2 abstracts visual reference (i.e. the task of looking at the information in multiple windows simultaneously). Task 1 abstracts transferring information between open windows (e.g. cut-and-paste). We could instead have chosen more realistic, domain specific, tasks instead of abstract ones (e.g. building a stock report in Word from a PowerPoint deck and a number of company and financial websites, or predict tomorrow's weather from a number of current and recent weather charts). We chose not to for two reasons, one pragmatic and one theoretical. Pragmatically our bumping button was implemented as part of an application (not added to an existing application) and so sticking to a simple WordPad kept the programming required manageable. Theoretically we believed that the results from abstract tasks could more easily be generalised, precisely because of the abstraction.

1.3 Experimental Results

Partly due to our small sample size the differences and distinctions given in this section are mostly not significant as shown by the test values reported. The tests are one-way ANOVAs unless otherwise stated. The discussion therefore hangs on trends inferred from the data and should not be interpreted as statistically significant.

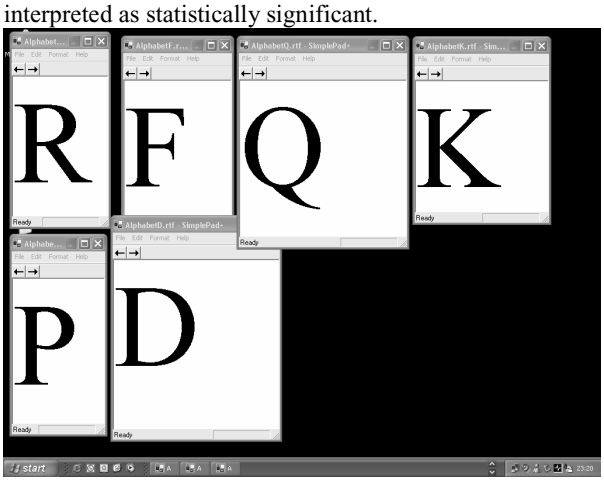

Figure 4: Task 2 in Progress

Preference was measured by the question "If you had to do the tasks again, which version would you use?" 9 of the 11 participants choose a bumping case (χ^2 (1, n=11) = 4.45, p = 0.04). They were also asked "Which task did you enjoy the most?" and 7 of the 11 participants choose a bumping case (χ^2 (1, n=11) = 0.82, p = 0.36). Although the presence of a bumping button was preferred, the preference is not entirely explained by enjoyment (perceived performance gains etc. could be other factors considered by participants). Of those who chose the bumping cases there was no appreciable difference shown between the faithful, dark-space, and unobscured bumping conditions. We also asked if participants felt a bumping button to move a window automatically was useful. On a seven point scale, with 1 representing "not at all" and 7 representing "yes very much", the average answer was 5.73 (STD 1.85).

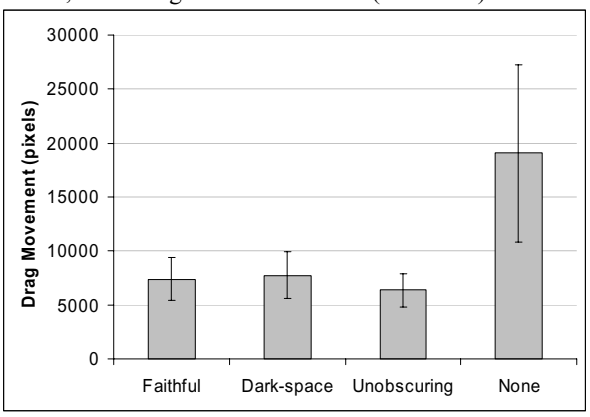

Figure 5: Reconstruction Task, Average Total Windows Dragging

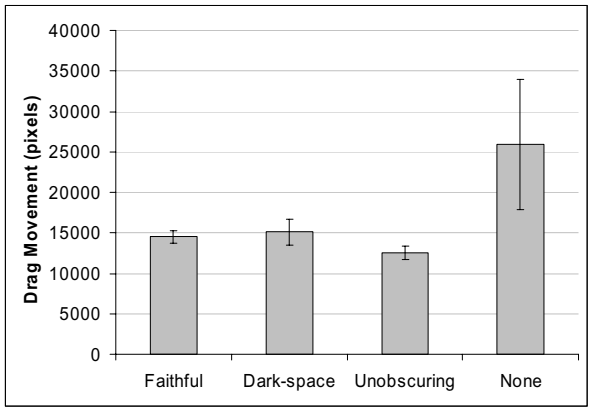

Figure 6: Alphabet Task, Average Total Windows Dragging

The basic purpose of the bumping button was to reduce the amount of window dragging required. Figure 5 and Figure 6 show the average total number of pixels that windows are dragged for the Reconstruction Task and the Alphabet Task. The values for the Reconstruction Task are averages of 7391, 7729, 6351, and 19039 pixels for the faithful, dark-space, unobscured, and no bumping conditions respectively ($F(3,30)=1.82$, $p=0.17$). The values for the Alphabet Task are 9914, 9549, 8502, and 19039 pixels for the faithful, dark-space, unobscured, and no bumping conditions respectively ($F(3,30)=1.97$, $p=0.14$). In both cases we see a difference between the case without a bumping button and the cases with: there is more windows dragging required without the bumping button. But this is misleading. The reduction in window dragging was only of benefit if the bumped window alighted in a position the user was happy with. If the user had to immediately move their cursor to the window and correct its position, then the bumping did not reduce mouse movement.

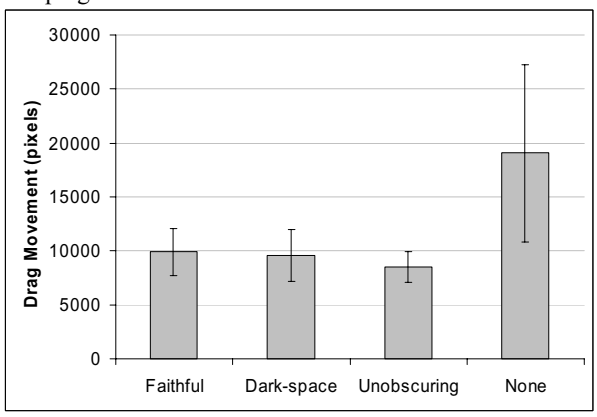

Figure 7: Reconstruction Task, Average Adjusted Total Windows Dragging

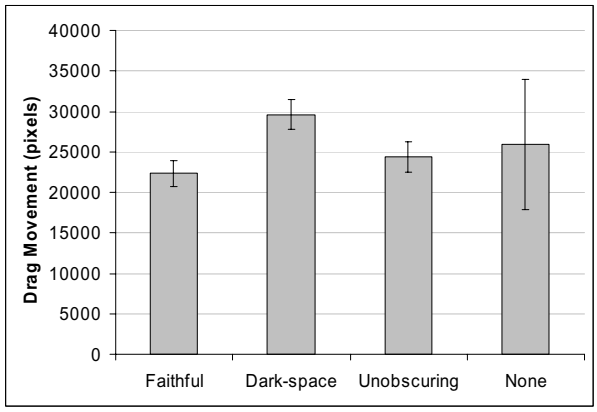

Figure 8: Alphabet Task, Average Adjusted Total Windows Dragging

Figure 7 and Figure 8 show a similar measure: the average total number of pixels windows are dragged for the Reconstruction Task and the Alphabet Task, but this time with position tweaking taken into account. Bumped window moves that require immediate repositioning are added to the total window dragging number. The values for the Reconstruction Task are averages of 14498, 15120, 12511, and 25877 pixels for the faithful, dark-space, unobscured, and no bumping conditions respectively ($F(3,30)=1.21$, $p=0.32$). The values for the Alphabet Task are 22311, 29600, 24398, and 25877 pixels for the faithful, dark-space, unobscured, and no bumping conditions respectively ($F(3,30)=0.52$, $p=0.67$). The prominent difference, between the case without a bumping button and the cases with, is retained in the Reconstruction Task as we go from the drag figures to these adjusted drag figures but lost in the Alphabet Task. It is replaced with a slight increase in the dark-space dragging over the other conditions. In the Reconstruction Task users needed to swap between windows, but the windows positioning was left to the users' own preferences. In this case the bumping buttons reduced the amount of dragging required. The Alphabet Task was about moving and resizing windows where their exact placement was important and largely prescribed. Hence any initial advantage gained by bumping for this task is lost as a window's positions and size are adjusted until perfect.

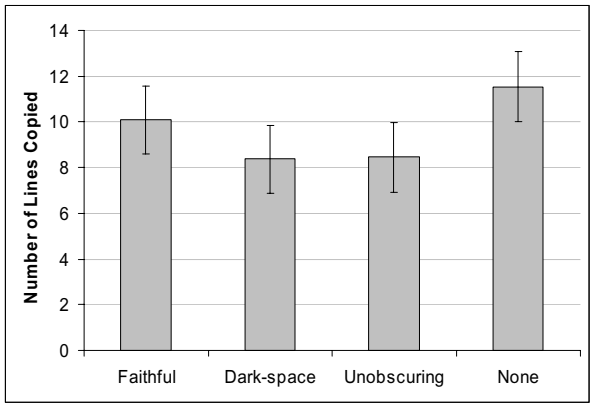

Figure 9: Reconstruction Task, Average Number of Lines Completed in 3 Minutes

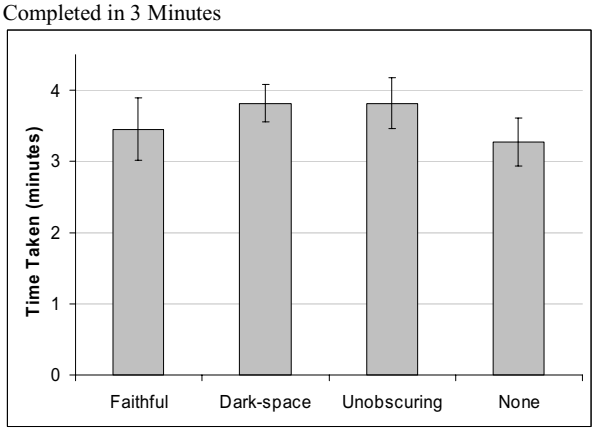

Figure 10: Alphabet Task, Average Time Taken

Figure 9 and Figure 10 show measures of task performance. Figure 9 shows the number of lines cut-and-paste in the 3 minutes of the Reconstruction Task. The values are averages of 10, 8, 8, and 12 lines for the faithful, dark-space, unobscured, and no bumping conditions respectively ($F(3,30)=3.53$, $p=0.03$). Figure 10 shows the time taken to complete the Alphabet Task in minutes. The values are averages of 3.45, 3.82, 3.82, and 3.27 minutes for the faithful, dark-space, unobscured, and no bumping conditions respectively ($F(3,30)=1.32$ $p=0.29$). Both charts show an increase in productivity for the faithful and the no bumping conditions over the dark-space and the unobscured bumping conditions.

This distinction between the more complex conditions (dark-space and unobscured) and the simple conditions (faithful and no bumping) are reflected in the participants' comments. Typical comments include:

- "made it quicker to move them, especially when I knew where they would go",
- "it is useful, but the bump button needs to place the window in an expected place", and
- "[it needs to be] simple to guess what it does".

When asked where a bumping button should place the window, the participants gave a variety of responses, but many people wanted it to retain its size (of the 8 participants whose comments included sizing information, 7 of them suggested no resizing).

We observed some unexpected uses of the bumping buttons, especially in the Alphabet task. One was the use of the 'bump left' button. Windows were fixed to initially open towards the left of the left hand screen. So whilst in the conditions which allowed automatic resizing clicking the bump left button had the effect of making the window smaller. Although this did not save participants time, several tended to prefer using it over manual resizing. Another unexpected use was the sizing and positioning of a window before clicking the bump button. Because of its easy predictability, some users positioned and sized a window on the left hand screen before bumping it over to the right hand screen using in the faithful bumping condition.

1.4 Discussion of Experimental Results

The bumping button was clearly a popular addition for multimon users: our participants chose it above the no bumping condition and enjoyed using it more. They declared that the addition of a bumping button was useful. It reduced the amount of dragging users needed to perform, but the total amount of window management related cursor movement was only reduced in the Reconstruction Task. The Reconstruction Task was typical of tasks where multiple windows provide the sources and targets for content. The fine-grained windows positioning required in the Alphabet Task exceeded our bump methods' abilities.

Performance was better with the simple behaviours: without bumping or using faithful bumping. Each version of the bumping had been explained to participants and practiced by them before the tasks, but dark-space bumping and unobscured bumping was too complex for participants to predict. It may be that longer term usage of the bump button would allow users time to develop an effective mental model of the more complex bumping algorithms. It also seems that the single aspect of the more complex bumping that participants found least useful was the resizing. One participant suggested that we keep our algorithms for intelligent placement of windows but just remove the resizing element.

However, the best bumping button may be the most complex. One participant's advice to us on the best bumping method to encode was that it should place the window "where I want it to go".

6 Conclusions

We have explained how the benefits afforded as users adopt multiple monitors come with an associated cost in terms of windows management. On single monitor PCs complex window arrangements are not desirable since the resulting windows are too small to work with. In multiple monitor systems users may lay windows side by side, and allow

each window ample space to read or work in. This arrangement involves users dragging and resizing windows across large screen distances. We showed how automating this arrangement by bumping windows can be an advantage. We implemented three simple methods of bumping and tested them in a user experiment. Bumping reduced the amount of windows dragging required, though not for all tasks: some tasks requiring exact windows placement still need to be accomplished manually.

Bumping may be added to the multiple monitor UI in a variety of ways. It could be implemented within an application, in an OS, or as part of a set of windows management functions (e.g. Ultramon http://www.realtimesoft.com/ultramon/) has a notion similar to our faithful bumping in their multiple monitor management software). For those working on such enhancements our recommendations are:

- Include bumping – we have shown that it can be effective in reducing the windows drag required for windows management.
- Keep it simple – we have shown that the semantics of the bumping button must be transparent to users for the benefits to emerge.

7 Next Steps

Although each bumping method we implemented was obviously algorithmic, and hence predictable, two of the algorithms proved too complex for the user to accurately and quickly predict. To address this we intend to re-implement similar algorithms but without the windows resizing element. This implies that the bumped window would obscure more of the screen (since we cannot make it smaller) and so we will try new methods of determining valid areas to obscure. For example we can analyze windows bitmaps to find large areas of white-space adjacent to a windows edge.

We will also add animation to the window bump to test if that helps users gain an understanding of the underlying mechanism and hence find the bump more predictable.

We intend to add the bumping button to a greater array of applications (or all applications) so that we can study a richer media mix of tasks in our experiment, for example pasting pictures into reports or presentations or data between spreadsheets. We can then install the enhancement in participants' workspaces to gain a longitudinal understanding of the usage of bumping.

8 References

Baudisch, P., Good, N., and Stewart, P. (2001). Focus Plus Context Screens: Combining Display Technology with Visualization Techniques, in Proceedings of UIST2001.

Beaudouin-Lafon, M. (2001). Novel Interaction Techniques for Overlapping Windows, in Proceedings of UIST2001.

Bederson, B. (2000). Jazz: an extensible zoomable user interface graphics toolkit in Java, in Proceedings of UIST2000.

Bell, A.B. and Feiner, S.K. (2000). Dynamic Space Management for User Interfaces, in Proceedings of UIST2000.

Bly, S.A. and Rosenberg, J.K. (1986). A Comparison of Tiled and Overlapping Windows, in Proceedings of CHI86.

Furnas, G. (1986). Generalized Fisheye Views, in Proceedings of CHI86.

Fry, B. (2002). Valence, http://acg.media.mit.edu/people/fry/valence/

Grudin, J. (2001). Partitioning Digital Worlds: Focal and Peripheral Awareness in Multiple Monitor Use, in Proceedings of CHI2001.

Guimbretière, F., Stone, M., and Winograd, T. (2001). Off the wall: Fluid interaction with high-resolution wall-size displays, in Proceedings of UIST2001.

Hutchings D.R. and Stasko, J. (2002). QuickSpace: New Operations for the Desktop Metaphor, in Proceedings of CHI2002.

Kandogan, E. and Shneiderman, B. (1996). Elastic Windows: Improved Spatial Layout and Rapid Multiple Window Operations, in Proceedings of AVI96.

Myers, B.A., Bhatnagar, R., Nichols, J., Peck, CH., Kong, D., Miller, R., and Long, C.A. (2002). Input Devices: Interacting at a distance, in Proceedings of CHI2002.

North, C. and Shneiderman, B. (1997). A Taxonomy of Multiple Window Coordinations, Technical Report, University of Maryland.

Perlin, K. (1998). Quikwriting: Continuous Stylus-Based Text Entry, in Proceedings of UIST99