Quantifying the Performance Effect of Window Snipping in Multiple-monitor Environments

Dugald Ralph Hutchings[1] and John Stasko[2]

[1] Computer Science Department, Bowling Green State University
Bowling Green, OH 43403 USA
drhutch@cs.bgsu.edu

[2] GVU Center, College of Computing, Georgia Institute of Technology
Atlanta, GA 30332 USA
stasko@cc.gatech.edu

Abstract. Snip is a tool that allows a user to constrict the view onto any window. We report on a controlled study of the snip tool in the context of a multiple-monitor environment. The study was designed based on observed user behavior in a field study of multiple-monitor users' snipping habits. Analysis provided results that indicate that users can expect to reference information approximately 15% to 30% faster from snipped windows than from non-snipped windows. Further, users need to pay only a small overhead cost to perform the snip operation. The result extends to other recently presented region-based interface tools that aim to assist multiple-monitor users interact effectively and employ additional monitor space for information-referencing activities.

Keywords: multiple monitors, window management, evaluation, snip

1 Introduction

Over the last five to ten years there has been an increasing amount of interest in and attention paid to research about *multiple-monitor systems*, which are defined as a traditional, single computer with one or more graphics adaptors driving a total of two or more physical monitors. Multiple-monitor systems exhibit a physically separated display space (the physical area between monitor frame boundaries) but provide the user with a virtually contiguous space, often at a cost less than a single large display. With a single set of input devices such as keyboard and mouse, users can set the position of information to be on any of the monitors or across monitor boundaries. Initial results about multi-display systems are positive, showing that users can expect to experience rises in productivity [5, 6] despite frequent occurrences of inconsistent or counterproductive interface behaviors [6, 7]. Recent research has focused on ways to overcome or eliminate the behavior or further improve the interaction experience.

One of the noted uses of multiple monitors is in support of the display of reference materials in one or more inactive windows (*i.e.* windows not receiving input from the user) while interaction takes place in the active window [7, 9, 10]. Applications are

typically designed with an active user in mind, with little or no attention paid to how people might use the information displayed by an application when the window is not active. Field research further indicates that users often try to set a group of windows to overlap in just the right way to show or hide information in each of the windows in the group when they are in "inactive use" because users have no direct way to force a window to show only a specific subregion [11]. The challenge of keeping just the right amount of information visible can become more difficult when multiple monitors are present since there is an increased chance that information that should stay hidden will become visible.

In response to the evident desire to control the information displayed by a window when in inactive use and the increasing percentage of display space showing inactive windows in the presence of multiple monitors, we proposed the notion of a window snipping operation that would allow a user to select a region of a window and show only that region [12]. Figure 1 illustrates the difference between snipping a window and resizing a window.

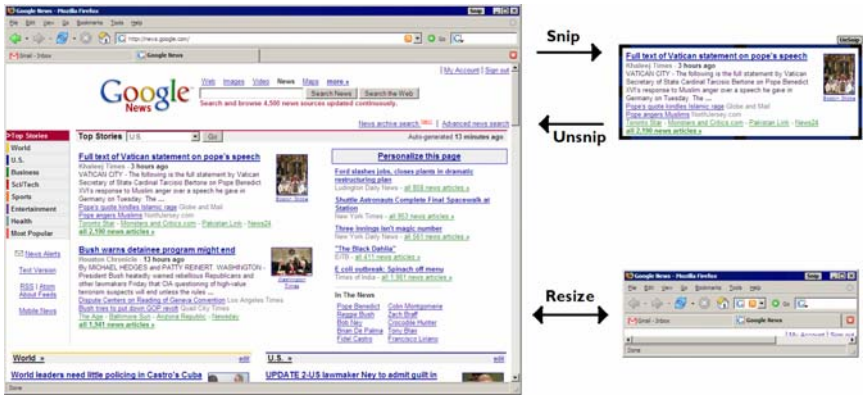

Fig. 1. Window snipping is portrayed in the upper-right corner. For comparison, window resizing is portrayed in the lower-right corner. Notice the difference in favoring the display of information (snip) or favoring the display of interaction components (resize).

Since proposing the idea of a region-based interface to control displayed information in windows [11, 12], we have developed a tool that implements the operation for everyday use in the Microsoft Windows XP system. We have also deployed the tool to a group of multiple-monitor participants in a field study of their window management practices with snip and without it [9]. The key finding from that study is that as compared to periods of time when the snip tool was not available, participants used snip to create space for more visible windows. In other words, multiple-monitor participants actually did use snip to simultaneously show more pieces of information, which can be described as a space-efficiency gain.

The question that remained was whether users could likewise expect to experience a complementary time-efficiency gain, *i.e.*, whether users could reference information more quickly from snipped windows than from non-snipped, normally-arranged windows. If so, we wanted to understand how much overhead was involved in using snip

(the time needed to snip versus the time saved from each reference to the snipped window) to measure the expected true time-efficiency gain from employing window snipping. As a result, we developed a controlled laboratory study to measure the length of time needed to snip each member of a representative set of windows and the length of time needed to answer questions about information in those windows in both their normal, full-size states and their snipped states. Before we proceed to report the results of the study, we briefly provide some additional detail about the current methods users employ to snip a window.

2 How Snip Works

We built snip as a third party application for Microsoft Windows XP, which is both a window manager and operating system. When a user desires to snip a window, the user must set the window to be active. This results in a button labeled *Snip* appearing near the window's other operation interface buttons (close, maximize/restore, and minimize). The user then clicks the Snip button, which temporarily interrupts any interaction between the user and the interface components in the window's application.[1] In a fashion similar to drawing a rectangle in a graphical editing program, the user then indicates the region to be shown (to result in hiding the remainder of the window). The window becomes snipped and interaction resumes as normal. In particular, since the snip occurs directly on the window, interaction can still take place in the constrained region. For example, if a user snips a piece of a window containing a research document, then the user can copy the text from the snipped window's application and paste it into another window without having to unsnip the document window.

In addition to the window being "snipped," the tool provides a small border around the snipped region to allow the user to distinguish it from other windows. Since Windows XP generally allows users to move windows only through interaction with the title bar, the small border also facilitates the movement of the snipped window elsewhere on-screen. The border is larger when the snipped window is active to facilitate window movement and smaller when the window is inactive so as to reflect the normal border size of a window. Figure 1 (previous page) illustrates the result of window snipping.

3 Controlled Study of Snip

In this study we compared the time cost paid to make a reference to each of two sets of windows on a second monitor: (1) a set of snipped windows and (2) a corresponding set of non-snipped windows (in this section, we call these *regular* windows). We placed all snipped reference windows or regular reference windows on the second monitor because of a pattern that we observed in the aforementioned field study [9], namely that snipped windows tended to appear on a specific monitor. As a result, snipped windows did not overlap but regular windows did overlap. Thus a user poten-

[1] Refer to Hutchings' detailed description of snip for technical details [9].

tially pays a higher cost in accessing a regular window because it might need to be brought forward before the reference can be made. Simultaneously, there is an up front cost to reference a snipped window: it must be snipped. We arranged the regular windows in such a way that they could all be accessed with a single click by ensuring a fixed portion of each window was always visible. Further, this visible portion allowed the used to quickly identify the type of information in the window, making it as easy as possible for the participant to determine the correct window. In essence, we constructed the study to give as much advantage to the regular windows as was possible but also reflect the tradeoffs users make in choosing whether to snip a window.

This is a crucial point and deserves reiteration. We fully expect that users will be able to access information from snipped, non-overlapping windows more easily than from non-snipped, overlapping windows. This does not confound the study but rather completely describes the heart of the matter: is the initial, one-time overhead of snipping worth avoiding the repeated overhead of finding the overlapped window of interest and possibly bringing it to the top of the stack? In the study, we are addressing the tradeoff that users can consider but also making any possible additional adjustments to ensure that users can access the overlapped window as quickly as possible. We discuss other, future experimental strategies in Section 3.5.

The study examines the average time needed to make a snip to a window and the relative difference in time needed to make references to sets of snipped windows and regular windows. For the first part, participants snipped a variety of differently shaped windows in different on-screen locations. In the second part, participants answered a series of questions displayed on a left-hand monitor about content of a set of snipped windows and a corresponding, equivalent set of regular windows, both pre-arranged on a right-hand monitor.

3.1 Method

We recruited participants by word of mouth at Georgia Institute of Technology. We required that they were fluent in English and had never interacted with the snip operation. All interaction during the study occurred on a computer system with two monitors arranged side-by-side, a state-of-the-art video card designed to support 2D graphics for a dual-display configuration, and a standard optical desktop mouse. The system ran Microsoft Windows XP and was set to use the default mouse pointer speed. Each monitor was a 17" flat-panel LCD display running at native landscape resolution of 1280 × 1024 pixels for a total resolution of 2560 × 1024 pixels. Figure 2 (next page) is a photograph of the experimental setup. Note that the left-hand monitor was not used in the experiment.

In Phase 1 participants responded to 8 sets of 12 statements: 2 practice sets followed by 6 timed sets. Each set had the following structure. To begin a set, the participant clicks a "begin set" button on the left-hand monitor. A group of windows appears on the right-hand monitor in predefined locations. Participants have the opportunity to alter the *z*-order to see what is contained in each window but cannot otherwise move or resize the windows. After a fixed period of time (5 seconds per window; a group of four windows yields 20 total seconds of viewing time), the right-hand screen becomes blank so the user cannot see the windows. The left-hand monitor

then displays a window containing a statement, the name of the window on the right-hand monitor to which the statement refers, and a "ready" button. The user reads the statement, understands the necessary reference window to respond to the statement, and clicks the ready button (again, note that the user knows *in advance* which window to use to respond to the statement). Simultaneously, (1) the right-hand screen re-appears and the user locates the reference window and (2) the left-hand screen displays a "true" button and a "false" button. Once the user ascertains whether the statement is true or false by viewing the appropriate window on the right-hand screen, the participant clicks the appropriate button on the left-hand screen. The right-hand screen then goes blank and the process repeats for each of the remaining 11 statements in the set.

Fig. 2. A photograph of the input and output devices used for the experiment.

An example statement is "The top news story concerns a pharmaceutical drug company." The statement would be accompanied by the window description "news Web browser" so that the user would know where to look before clicking "ready." Note that complementary sets of windows contain equivalent questions. For example, if the answer to the example statement is true for a set of snipped windows, then there will be a statement about the news Web browser that will also be true. In this way, each statement should be able to be answered in the same amount of time.

Participants are timed only between clicking "ready" and clicking the response ("true" or "false"). During the practice trials we instruct participants that information in the window is likely to change in between statements and that they need to verify a response by actually looking at the window first. They are further instructed that upon introduction of the new group of windows they should become familiar with the configuration of the windows and the type of information contained in the windows, not the actual information, since the information will be different the next time that they look at it. Changing the information each time eliminates any potential advantage gained by memorizing information while answering. Figure 3 (next page) demonstrates a group of four snipped windows and a group of four regular windows.

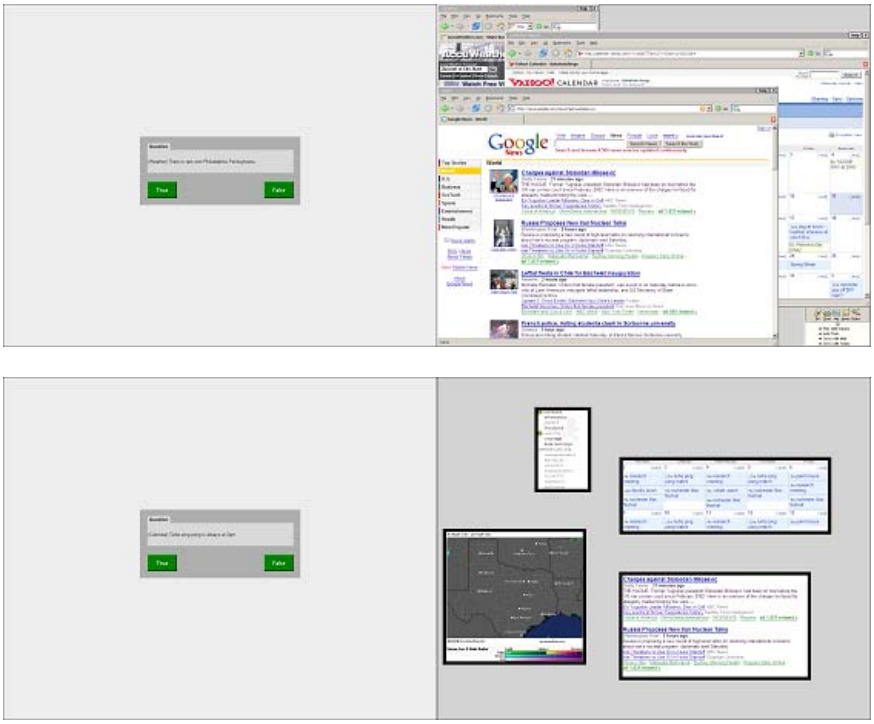

Fig. 3. A comparison of an overlapping group of four windows and a snipped group of four windows. Notice that each window in the overlapping group is visible.

There are three timed primary groups of windows: (*G*2) a group of two windows, which is a personal calendar and a news Web page; (*G*4) a group of four windows, which is group *G*2 plus an instant message buddy list and a weather map; and (*G*6) a group of six windows, which is group *G*4 plus an outline of a document and a road map. Each of the three primary groups *Gi* has two secondary groups Gi_r of regular windows and Gi_s of snipped windows, for a grand total of six timed window groupings and statement sets. Gi_r and Gi_s are equivalent in that for each question in Gi_r, there is a corresponding question in Gi_s that should take an equal time to answer. However, the *content* displayed in the sets does not overlap in any way. For example, if set Gi_r contained the statement "The top news story is about the Yellow Jackets" and the headline in the news window contained the word "Yellow Jackets" (thus the answer was true), then in the set Gi_s, for the corresponding news headline (say, "Falcons edge Rockets in overtime thriller"), there would be a statement containing a keyword from the headline (say, "The top news story is about the Falcons"). Both of these statements should thus be able to be answered in the same amount of time, so the entire set of statements is equivalent. Figure 3 illustrates $G4_r$ and $G4_s$.

An equal number of participants receive the primary window groups in each of the six possible orders. Within each ordering, half of the participants always receive the snipped windows first and the regular windows second and the other half receive the

reverse ordering. Twelve participants are thus necessary to fill each variation in ordering and balance the study.

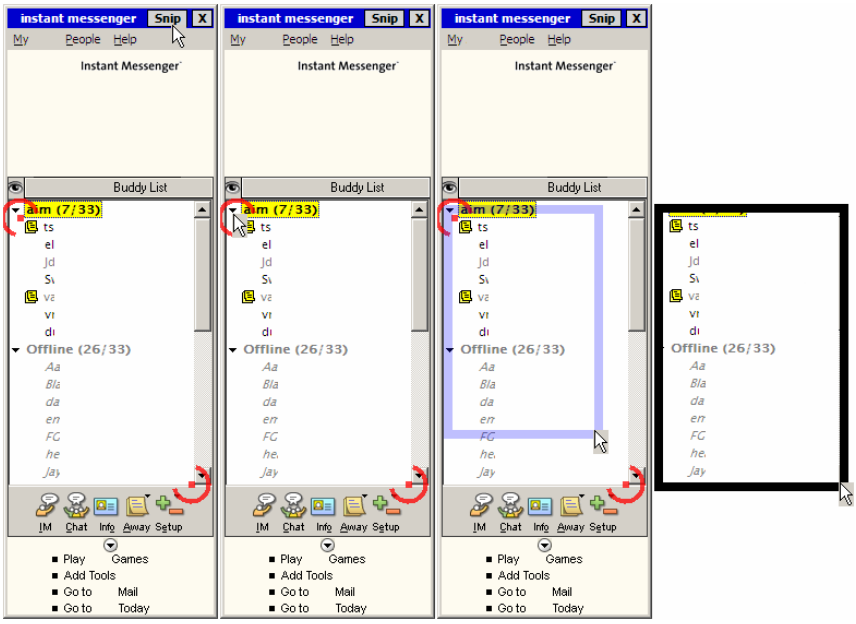

Fig. 4. A step-by-step breakdown of snipping: clicking the snip button, clicking the first region point, dragging toward the second region point, and releasing the button to complete the snip.

Participants are not allowed to move the windows in the groups when responding to the statements in order to avoid confounding the timing data (further, participants are disallowed from accessing windows through other means, such as the TaskBar or keyboard shortcuts like pressing alt+tab). The group of snipped windows is always small enough to be placed such that they do not overlap. The group of regular windows is always too large to be arranged in a non-overlapping fashion. As a result, the regular groups are always pre-arranged such that (1) each window always has a piece visible regardless of the global *z*-ordering and (2), the "always visible piece" of each window reflects the type of information it contains. To further provide participants opportunities to most quickly respond to statements about regular window groups, when the group has *n* windows and since there are 12 questions per group, *n*/12 of the questions refer to a window that is at the top of the *z*-order and the participant does not need to bring any window forward to answer the question.

In Phase 2 participants are introduced to the actual window snipping operation (as opposed to simply the display of snipped windows) and given 5 practice trials to snip a series of different regions on a single window. Snip region points appear on the window to guide the user in snipping the desired region (Figure 4). This visual setup simulates an actual snip operation, where a user knows what region should be snipped before actually moving to snip it. After the practice trials, participants proceeded to

make one snip in each of a series of seven windows and then repeated the sequence for a total of 14 timed snips. These seven windows were ones that appeared in the different reference sets in Phase 1. We measured the time from pressing the snip button in the window title bar to the end of region definition. This phase allowed us to compute the average time needed to snip a window.

Following Phase 2, participants engaged in a brief interview about the snip tool and their experiences during the experiment.

3.2 Hypothesis

Informally stated, our main hypothesis is that participants will respond to statements in the snipped sets significantly faster than they will respond to statements in the regular sets. More formally, let $A(W, p)$ represent the time needed for participant number p to answer all of the questions for each set of windows $W \in \{G2, G4, G6\}$ and let n represent the number of participants, yielding the following inequality as our hypothesis.

$$\frac{\sum_{p=1}^{n} A(W_r, p)}{n} >> \frac{\sum_{p=1}^{n} A(W_s, p)}{n}$$

If this hypothesis holds, *i.e.*, if the average amount of time to answer all of the questions is smaller for snipped windows than for regular windows, then for each window group W we can calculate a number R_W that represents the expected average savings in referencing a window from W_s as compared to referencing a window from W_r. Further, we can then find the smallest number N_W such that $R_W N_W > S(W_r)$ where $S(W_r)$ is the expected average time needed to snip the windows in W_r. N_W represents the number of references that a user needs to make to W before snipping becomes worthwhile. In other words, given that a user makes k references to windows in W, the user can expect to save $(k - N_W) \cdot R_W$ amount of time as compared to not snipping the windows.

3.3 Experimental Results

Thirteen participants enrolled in the study but one participant ultimately declined to participate, thus we collected data from twelve participants. For each participant p and for each set of windows W, we calculated the time needed to answer all of the questions in the regular set of windows W_r and the snipped set of windows W_s, which again are denoted $A(W_r, p)$ and $A(W_s, p)$ respectively. We then ran Student's paired-samples, one-tailed t-test on those total times. In Table 1, we report the means of the collected data, the associated standard deviations, and also the p-values returned by the t-tests. Participants answered questions in the snipped sets significantly faster than in the regular sets, with time improvements approximately ranging between 15% and 30%. Note that the average error rate over 12 questions was below 1 for all 6 sets of windows. There were no significant differences in error rate between the snipped sets of windows and the regular sets or windows.

Table 1. *t*-test results for each of the three main window groups.

t	$G2_r$	$G2_s$	$G4_r$	$G4_s$	$G6_r$	$G6_s$
$\bar{x}$ (sec)	92.1	65.4	92.9	62.9	79.2	66.3
σ	25.5	20.4	22.0	16.3	18.2	20.7
p	< 0.01		< 0.01		< 0.01	

Thus our main hypothesis holds; people answered questions more quickly for the snipped sets of windows than for the regular windows. Thus we can now calculate R_W, *i.e.*, the expected average savings in referencing a snipped window from W_s as compared to referencing a regular window from W_r for each $W \in \{G2, G4, G6\}$. For each participant p, we calculated $\Delta(W, p) = A(W_r, p) - A(W_s, p)$, which represents how much time participant p saved in the snipped set relative to the corresponding regular set. We then calculate the mean of these $\Delta(W, p)$ values, *i.e.* the average time savings across all participants for the whole set of windows, and divide by 12 (the number of questions in the set) to arrive at R_W:

$$R_{G2} = 2.22 \text{ sec}$$
$$R_{G4} = 2.51 \text{ sec}$$
$$R_{G6} = 1.07 \text{ sec}$$

Having calculated R_W, we would like to calculate N_W, which is the number of references that a user needs to make to the windows in W before snipping becomes worthwhile. In other words, we desire to find N_W such that $N_W \cdot R_W = S(W_r)$, where $S(W_r)$ is the total time required to snip all of the windows in W_r. This means that N_W is subject to the following equation.[2]

$$N_W = \left\lceil \frac{S(W_r)}{R_W} \right\rceil$$

As previously mentioned, each participant snipped each of the seven unique types of windows twice, for a total of 14 snips per participant. The mean time needed to perform a snip operation was 3.34 seconds with a standard deviation of 0.95. In order to create a more conservative estimate of snipping time, we constructed a confidence interval around the mean subject to the standard deviation and $\alpha = 0.01$, yielding a conservative mean of 3.53 seconds. Calculating $S(W_r)$ is thus accomplished by multiplying this conservative mean by the number of windows in W_r, providing the following values for N_W.

$$N_{G2} = 4 \text{ references}$$
$$N_{G4} = 6 \text{ references}$$
$$N_{G6} = 20 \text{ references}$$

[2] We are not suggesting that users will consciously make a decision whether to snip based on this calculation. Rather, we are providing the analysis to demonstrate how low the overhead of snipping tends to be should users decide to snip.

In other words, to overcome the overhead cost of snipping all of the windows in the groups of 2, 4, and 6 windows, the user must make 4, 6, and 20 references to windows in the set, respectively. This is a respective average of 2, 1.5, and 3 references per window. Once this overhead cost is paid, the users can expect a time savings of R_W per reference.

It is interesting to note that in our data analysis we do not consider the time needed to arrange the windows on the "reference monitor." In this study, we pre-arranged these windows for our users so that each window always had a piece visible, regardless of the *z*-order at any time. As the number of windows increases (especially above four windows, where a "four corners" strategy no longer works), the difference in arrangement time between the snipped windows and the regular windows becomes non-trivial. As a result, if we included arrangement time in the calculation of N_W, then we would expect the values to remain stable or decrease, particularly for N_{G6}.

We observed one additional interesting behavior. Seventy-five percent of the participants would move the mouse from the left monitor to the right monitor for every question in the regular sets of windows, even when the window containing the answer to the question was already on top. They would not move the mouse to the right monitor for the snipped windows. When mouse movement was unnecessary in the regular sets, there was nevertheless an expectation of locating the needed window and bringing it to the top of the *z*-order, causing unnecessary navigation time.

3.4 Interview Results

All participants concurred that regardless of the number of windows displayed on the right-hand screen, the snipped window sets were never overwhelming and they felt that they answered questions faster when windows were snipped. Participants varied however on how much faster they felt they could answer. Three participants felt twice as fast, with one even saying that he thought snipping allowed him to be exponentially faster as the number of windows linearly increased. Three participants indicated that they felt "just a little" faster with the snip operation and three others indicated that they had no idea how much faster they were. Other responses included 10% faster, 25% faster, and "a few seconds" faster.

All but one of the participants felt that the mechanics of the snip operation made sense, with one participant indicating that clicking a button to begin a snip was awkward; he would have preferred a keyboard shortcut. Half of the participants indicated that they would use snip for their everyday interactions, whereas the other half said that they "might use it" or "would use it depending on the circumstances of the day." Participants overwhelmingly indicated that they could use snipping for reference materials and notes or for email updates. Other responses included generally "Web browsing," calendars, sports scores, and traffic information. Interestingly, a number of participants indicated situations where they definitely would *not* use snipping: three said they would never use it for news and one said she would never use it for instant messages.

There were a few other comments of note. One participant requested that snipped windows also be able to be designated as always-on-top, especially when referencing information and using it elsewhere. This would be a technically simple capability to

add to the currently implemented version of snip. There were two potentially conflicting opinions about virtual desktops. One participant indicated that she would like to have a virtual desktop of snipped windows to make it easy to bring reference information to the fore in a single action when multiple monitors are equally engaged in interaction (which is like Apple Macintosh OSX Dashboard though the participants did not explicitly mention this application). Another user indicated that he would avoid snipping because virtual desktops already allowed him to avoid overlapping windows. It is unclear how snip would interact with virtual desktop users, though currently snip operates correctly regardless of any virtual desktop system being run by a user.

3.5 Discussion

The results of this study show promise for snip: they complement the space efficiency gain showed in a previous field study [9] with an indication of a strong time-efficiency gain that a user can realize. The reader may rebut that sometimes a user might snip the wrong region of a window and will have to unsnip to get at information and possibly resnip back to the original size. Clearly this will have an impact on the time-efficiency increase to be expected by the user, but we do not currently know how often this situation arises. However, unsnip and resnip should cost about as much as an original snip. Devising studies to uncover the frequency of unsnipping, motivations for unsnipping and resnipping, and determining the cost of unsnipping and resnipping are obvious next steps in this line of research. Other variations to the presented study including allowing users the ability to move and resize the regular windows, which has the benefit of "personalized space" and might improve users' spatial memory of different windows, but also had the drawback of the validity in comparing times needed to answer questions about the windows (every user is looking at a different visual configuration). On the other hand, we could incorporate this window management time into the total time needed to answer a question. We could also provide more challenging sets of overlapping windows by, for example, having windows that occlude all others once selected.

Our laboratory-based evaluation results further show promise for several region-based tools that have recently appeared in various HCI venues. WinCuts is a window management tool very similar to snip but has the following two major differences: (1) WinCuts provides live *copies* of a region the source window instead of operating on the window itself, but (2) due to technical limitations of the Windows XP system, the copies are not interactive [18]. However, in the scenario of simply referencing information from a copy, the results of this snip study also apply to WinCuts. Metisse is a system that provides users with a variety of tools to manipulate windows as if the windows were images and Metisse runs on any X-based platform, including the Macintosh OS X system [4]. A demonstration of the capabilities of Metisse shows how WinCuts and snip can be implemented on the system, though Metisse improves on WinCuts by allowing interaction in either the source window or in the live copy [17]. Our results again help show the potential utility of Metisse and as a result extend past the Windows XP window manager to other systems.

4 Related Work

Controlled studies of window management tools appear infrequently in the literature. Two of the more recent systems to have been evaluated in laboratory studies are the TaskGallery [14] and Elastic Windows [13]. Both studies are similar to the snip study since the tools were compared to standard window manager systems and operations. However, both Task Gallery and Elastic Windows were tested on single-monitor systems and both concerned window grouping functionality rather than operating on individual windows. Another difference is that the snip study reflects observed multiple-monitor window management behavior whereas the other two studies did not collect field data prior to the design of the controlled study.

Other studies recorded data from use of an interface in the field but did not include complementary controlled studies. One prominent study is that of the Rooms system [8] (notably the design of the Rooms tool was motivated by field research [3] just as snip was motivated by field research [7, 9, 10, 11]). The Rooms study demonstrated the utility of what are today referred to as virtual desktops. More recently the GroupBar was evaluated over a 1-week period [15]. Both Rooms and GroupBar also relate to window grouping behavior rather than direct window operations like slight rotation and peeling [1], which, as mentioned, have not been evaluated in the field or in the lab.

Another line of research that is related to the snip tool is development of interfaces dedicated specifically to peripheral awareness (as opposed to snip, which operates on any window). This body of research is considerably large so we have selected a few representative examples. For example there is Sideshow, which provides small views onto many pieces of information in a single application. Sideshow is limited to certain types of information and further must be tethered to the edge of a monitor instead of anywhere in the display areas [2]. The InfoCanvas is an artistic rendering of important information such as news, stock prices, airline ticket prices, incoming email messages, or any piece of dynamic information from an Internet source [16]. This tool is specifically designed for use on a second monitor (or as a standalone, separate display) but occupies the entirety of that display space. Macintosh OS X includes the Dashboard and its widgets that provide small tools and views of peripheral information like weather and news. The Dashboard cannot bet set to be always visible and appears only when the user summons it as opposed to snipped windows, which can be persistently displayed (http://www.apple.com/macosx/features/dashboard/).

5 Conclusion and Future Work

We have demonstrated that the snip tool, designed to reduce the space needed to display information from a window, resulted in the ability to reference that information 15% to 30% more quickly than without the tool. Further, the overhead cost of performing a snip is rather low, especially for 2 or 4 windows of information. This suggests that the snip tool is valuable for multiple-monitor users. Other recently developed region-based tools should also exhibit similar time-efficiency gains for their users. The general result is intuitive (finding relevant information in a smaller space

without the need for interaction should be faster) but the specific result demonstrating the magnitude of the difference is valuable in characterizing potential productivity benefits for users of multiple-monitor systems. Further the fact that the study was based on observed behavior of the tool among multiple-monitor users should provide a realistic estimate of real-world impact.

While our primary motivation for developing snip was to assist multiple-monitor users take better advantage of the extra space provided in a common context of use, *i.e.*, for displaying reference information, follow-up experiments could assess the effects of window snipping in physically large display environments, high-resolution displays, or in small display environments. In addition to broadening the display environments, we could also narrow the potential types of windows to specific classes of users or applications to see if any of the effects become magnified in the context of certain tasks.

Finally, we would like to explore an "anti-snip" tool that would allow a user to show everything in a window *except* for the selected region. This would allow users with the specific intent of hiding a piece of information (as opposed to showing it) a tool to potentially satisfy that need. A variation on both snip and anti-snip would be to automatically make the window full-sized and fully visible when active and then automatically re-snip (or re-anti-snip) the window when the user places input focus elsewhere. This variation could be useful for windows like instant messages so that the text of the conversation is only visible when that conversation retains input focus.

Acknowledgements. Thanks to the reviewers of this paper for providing additional points to add to the discussion sections. Thanks to the National Science Foundation for supporting this work under grant IIS-0414667.

References

1. Beaudouin-Lafon, M. Novel interaction techniques for overlapping windows. *Proc. UIST 2001*, ACM Press, 153–154.
2. Cadiz, J. J., Venolia, G., Jancke, G., and Gupta, A. Designing and deploying an information awareness interface. *Proc. CSCW 2002*, ACM Press, 314–323.
3. Card, S. K., Pavel, M., and Farrell, J. E. Window-based computer dialogues. *Proc. INTERACT 1984*, 239–243.
4. Chapuis, O. and Roussel, N. Metisse is not a 3D desktop! *Proc. UIST 2005*, ACM Press, 13–22.
5. Colvin, J., Tobler, N., and Anderson, J. A. Productivity and multi-screen displays. *Rocky Mountain Comm. Review 2:1 2004*, Dept. Comm. Univ. Utah, 31–53.
6. Czerwinski, M., Smith, G., Regan, T., Meyers, B., Robertson, G. and Starkweather, G. Toward characterizing the productivity benefits of very large displays. *Proc. INTERACT 2003*, IOS Press, 9–16.
7. Grudin, J. Partitioning digital worlds: focal and peripheral awareness in multiple-monitor use. *Proc. CHI 2001*, ACM Press, 458–465.
8. Henderson, D. A. Jr. and Card, S. K. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Trans. on Graphics 5:3 1986*, ACM Press, 211–243.

9. Hutchings, D. R. Making multiple monitors more manageable. Doctoral Dissertation, Georgia Institute of Technology, Atlanta, GA, USA, August 2006.
10. Hutchings, D. R., Smith, G., Meyers, B., Czerwinski, M., and Robertson, G. Display space usage and window management operation comparisons between single monitor and multiple-monitor users. *Proc. Advanced Visual Interfaces 2004*, ACM Press, 32–39.
11. Hutchings, D. R. and Stasko, J. Revisiting display space management: understanding current practice to inform next-generation design. *Proc. Graphics Interface 2004*, Canadian Human-Computer Communications Society, 127–134.
12. Hutchings, D. R. and Stasko, J. Shrinking operations for expanding display space. *Proc. AVI 2004*, ACM Press, 350–353.
13. Kandogan, E. and Shneiderman, B. Elastic windows: Evaluation of multi-window operations. *Proc. CHI 1997*, ACM Press, 250–257.
14. Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Risden, K., Thiel, D., and Gorokhovsky, V. The Task Gallery: A 3D window manager. *Proc. CHI 2000*, ACM Press, 494–501.
15. Smith, G., Baudisch, P., Robertson, G., Czerwinski, M., Meyers, B. Robbins, D., and Andrews, D. GroupBar: The Taskbar evolved. *Proc. Australian Comp. Human Interaction Conf. (OZCHI) 2003*, 34–43.
16. Stasko, J., Miller, T., Pousman, Z., Plaue, C., and Ullah, O. Personalized peripheral information awareness through information art. *Proc. UbiComp 2004*, Springer, 18–35.
17. Stuerzlinger, W., Chapuis, O., Phillips, D., Roussel, N. User interface façades: towards fully adaptable user interfaces. *Proc. UIST 2006, ACM Press*, 309–381.
18. Tan, D. S., Meyers, B., and Czerwinski, M. WinCuts: manipulating arbitrary window regions for more effective use of screen space. *CHI 2004 Extended Abstracts*, ACM Press, 1525–1528.