

# Shrinking Window Operations for Expanding Display Space

Dugald Ralph Hutchings, John Stasko  
GVU Center/College of Computing  
Georgia Institute of Technology  
Atlanta, GA 30332 USA  
{hutch, stasko}@cc.gatech.edu

## ABSTRACT

Recent research and technology advances indicate that multiple monitor systems are likely to become commonplace in the near future. An important property of such systems is that the physical separation of the display prompts users to place windows entirely within monitors, and thus does not fully alleviate the problem of managing windows on smaller monitors. Another finding about multiple monitor systems is that an additional monitor often holds windows that help the user maintain awareness rather than support interaction with information, but that multiple monitor users tend not to have many more windows visible than their single-monitor counterparts. We therefore present a window shrinking operation that specifically intends to help users display a window's relevant information. The operation should help to create smaller windows to manage, helping the "small monitor management" problem and targeting use of awareness windows on multiple monitor systems.

## Categories and Subject Descriptors

D.4.9 [Operating Systems]: Systems Programs and Utilities – *window managers*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *GUI, windowing systems*.

## General Terms

Management, Human Factors

## Keywords

multiple monitors, window operations, relevant regions

## 1. INTRODUCTION

For the past twenty-five years, the "desktop" of a personal computer has remained largely unchanged while the number of tasks that a computer supports has risen dramatically. For example, one can now complete a research paper without once leaving the computer, because all documents can be created and edited with a word processor and an image manipulator (for figures), all supporting resources can be found and viewed through a web browser and various file viewers, all communications can occur through email, instant messages, and in-document comments, and **all** of these applications can easily be running simultaneously. Many systems have been proposed to help users manage multi-window tasks, and a few techniques have been introduced to more

carefully control individual windows, but the user actions of *add*, *delete*, *move*, and *resize* remain as the standard window operations.

While the set of fundamental window operations has remained stable, there is a growing trend toward larger amounts of screen space. Recent research indicates that multiple monitor systems are likely to become commonplace within the next few years, and researchers have already identified potential usability problems of existing interfaces when used on multiple monitor systems. One issue is that users now have an opportunity to see more information in more windows, but at the cost of managing a larger amount of screen space. This "display space management" can be split into two sub-issues: *across-task management*, in which interfaces focus on helping users switch among different tasks, and *within-task management*, in which interfaces focus on the display of the one, two, or more windows that constitute a task. Systems such as Rooms [4], Elastic Windows [13], and the Task Gallery [18] have targeted the former on single-monitor systems and systems such as Kimura [15] and Scalable Fabric [19] target this problem for users of larger display spaces. The fundamental set of window operations generally aid more in the latter issue, and the trend toward multiple monitor systems has prompted us to reconsider whether the standard window operations could be improved or expanded to better help users manage the space.

For example, if input is only directed at one window at a time, then probably only one monitor is actively receiving user input at one time, leaving the other monitors to simply display windows. We consider a window as two parts: (1) *information*, which is of principal interest to the user and (2) *interaction components*, which aid the user in manipulating the information. When a window is visible but is not receiving input from the user, the interaction components are using display space that could be better used for the display of information. Shrinking a window that will be used to simply display information can be undesirable since it often results in the realignment of both information and interaction components. This realignment can disorient the user and usually favors the display of the interaction components rather than the information. We propose a shrinking operation that displays only a window's relevant content, where "relevant" is flexibly defined by users (Section 3). We have developed a prototype of the operations using Java and Swing to demonstrate how the proposed operations would work in an actual window manager by using rectangular images as windows. Before explaining these operations though, we discuss related work.

## 2. RELATED WORK

Work related to our proposed operations falls into two broad categories: studies of multiple monitor & large display usage and systems and techniques for managing display space.

## 2.1 Multiple Monitors and Larger Displays

Field work by Grudin [8] demonstrates how physical separation of the display space prompts users to place windows entirely within monitors and shows that one use of an additional monitor is to hold windows that help the user maintain awareness, whether to display information relevant to the current task or to keep channels of communication visible (such as email windows). Lab work by Czerwinski *et. al.* [5] also indicates that a physical separation as small as 8 mm can prompt users to avoid windows that straddle monitors. Both papers indicate task completion efficiency gains for multiple monitor users, whether actual [5] or perceived [8].

Some large display systems include alternative window shrinking operations. Flatland [17] is a large electronic whiteboard system that uses a tiled windowing system. When a user moves a window to the side or corner, the window automatically scales down in size. The Interactive Mural [9] is a large, high-resolution system built to aid brainstorming. It uses a pre-defined region that scales windows that enter the region. In both cases scaling acts more as an iconification operation than a resizing operation because they are intended to move unused windows out of the way and may scale windows so small as to be unreadable. The Interactive Mural also allows users to copy window regions as static images.

## 2.2 Display Space Management

There has been considerable work on window management systems. Myers gives a good overview of the general topic as well as descriptions of many window management systems built prior to 1988 [16]. He comments that the term *active window*, which means the window receiving user input, is a poor term because windows can serve important awareness functions when they are not receiving user input, such as when one window displays an outline of a paper while the user actually writes the paper in another window.

Some recent systems and techniques have looked at ways to better exploit empty display space. Our QuickSpace operations focus on using empty space to grow one window while maintaining the visible contents of other windows, aiming to help users maintain awareness primarily on single-monitor systems [12]. Non-overlapping Dragging helps keep windows visible by automatically moving recently occluded windows to empty space [3]. Both of these systems could be enhanced by shrinking operations that assist in creating empty display space. Beaudouin-Lafon presents some operations that aid in grouping windows through a tabbed interface in addition to window-specific operations that allow users to see pieces of more windows by slightly rotating them like physical pieces of paper [2]. Some systems such as CIWM [6] automate every window management operation, deciding the position, size, visibility, and depth order for every window, and deciding when to automatically close windows.<sup>1</sup> These systems have generally not been adopted since inferring desired window placement, size, and indeed existence can cause users considerable confusion and frustration [6]. SCWM balances automated window management with user control by allowing the user to group and interact with windows by setting constraints [1].

Unfortunately, very few window and space management systems and techniques have been formally evaluated, but there are some

<sup>1</sup> CIWM also appears to be the first window manager designed specifically for a multiple monitor system, albeit specialized for defense applications.

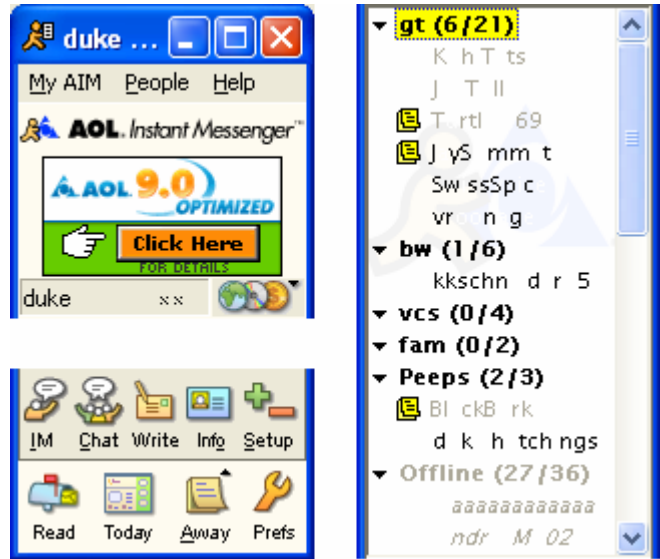


Figure 1. On the left are the top and bottom sections of an instant messenger client. On the right is the middle section, which is the only useful part when it is not active.

cases of more general evaluation. One particular example is Gaylin's study of overlapping window manager users [7]. He videotaped 9 users for 22 minutes (each) in their natural work environments to assess what window operations they used the most. By doing so, he developed reasonable benchmark tasks for testing future window manager designs. In closely related work, Hutchings *et. al.* [10] compare window operation and display space use between single- and multiple-monitor users. Several of the findings from that work suggest that multiple-monitor users tend to view several windows without interacting with them, some of which are related to a current task and other which are related to other tasks. Furthermore, they find that multiple monitor users with smaller amounts of absolute pixel space tend not to have many more windows visible than their single-monitor counterparts.

The technique that we present in this paper is based on the ideas of many pieces of related work, but also can be applied to our recent study of display space management practices that are independent of window manager and number of displays [11]. In that study, we identified two important classes of the way people display windows: (1) *near maximizers*, who complete a lot of manual resizing and moving to keep desktop icons visible, and (2) *careful coordinators*, who generally keep many pieces of windows visible for a variety of purposes. Both groups indicated many situations in which they manage windows not necessarily to make some information visible, but to hide interface components or information in other windows for purposes such as hiding distracting components and partially obscuring sensitive or private information. In the next section, we propose a new shrinking concept and operation borne both of our analysis of multiple monitor users and our understanding of window management.

## 3. SHRINKING WINDOWS

*Resize* is the standard operation for either growing or shrinking a window. In the case of shrinking, we argue that sometimes the operation does not match a user's desire to place the window's information (as opposed to interaction components) in a smaller

amount of space. This is especially the case when a window will display information that the user will track while working in a different task or is useful for the task at hand, but will not often actively receive input (in other words, awareness windows). We use the following example to illustrate why resize often fails; many other window applications exhibit similar behavior.

Figure 1 (previous page) is an instant messaging client. From top to bottom, the client has a title bar, menu, advertisement, label indicating the owner of the client, the buddy list (listing the online contacts), and two groups of buttons. The most useful information in the client is the buddy list. Most of the time, users are not directly interacting with the window but rather glancing at it, so the only part of the window that needs to be visible is the buddy list. However, when a user shrinks the window, the only part that gets smaller is exactly that region; the interaction components remain on-screen. Indeed, most applications strive to keep all of the interaction components visible so users can clearly see that with which they can interact. Shrinking a window often does nothing but shrink the displayed information, which does not necessarily match user intentions for viewing, but not interacting, with the window.

Many applications are created with redundant interface components to give users a variety of ways to accomplish the same action. For example, again looking at Figure 1, everything that can be accomplished through the buttons at the bottom can also be accomplished through the menu at the top. All of the instant message-specific actions can further be accomplished by right-clicking on buddies' names in the buddy list. So if shrinking the window kept more of the buddies visible and less of the other interaction components visible, the user could both interact with the window in a shrunken state and have more space available for other windows when the window did not receive user input.

Participants in our aforementioned study [11] indicated trouble in effectively using display space because often they want to use just the information in "secondary windows" to aid the interaction in some "primary window," but shrinking the secondary windows

just hides desired information. As people move to systems with more than one monitor, it is likely that more windows will act as secondary windows since interaction happens on only one monitor at a time and one of the noted uses of other monitors is to either display information relevant to the current task or to monitor information that cuts across tasks (instant messaging and email, for example) [8]. To tackle the problem with the resize operation, we propose shrinking operations to support relevant region-based shrinking.

### 3.1 Relevant Region-based Shrinking

The *relevant region* of a window is a rectangular sub-region of that window, as defined by the user to be the important part of the window to see. Just as the window is not aware of its underlying application components, the application is not aware of the relevant region. Thus, relevant regions can span UI boundaries or can be only parts of a particular piece of displayed information. For example, in Figure 1, a relevant region might be the entirety of the buddy list component, or if the users do not often use the scroll bar, the relevant region could be just the part of the buddy list without the scroll bar. If the user has several buddy lists under different aliases, the relevant region might include the label at the top of the buddy list so that the user can differentiate among the different lists.

Once defined, the user can opt to display only the relevant region (in other words hide the remainder of the window) by using the *snip* operation, and can regain the original window by using the *unsnip* operation. If windows are considered as images, snipping can be considered as a clipping operation. Snip is thus a shrinking operation that more closely matches the situation of diminishing a window to see a particular piece of information rather than to interact with it, which the standard resize does not accomplish. When unsnipped, a window retains its relevant region and uses a light visual indicator to remind the user where he or she has placed the region. A snipped window can be moved around by the user, but cannot be resized in the traditional way without first being unsnipped.

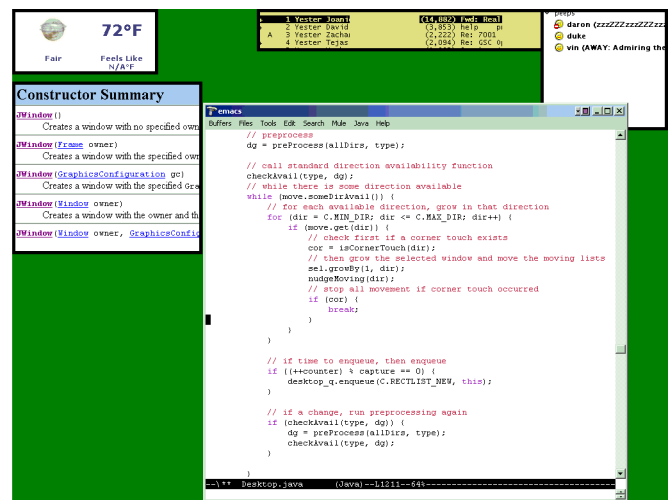
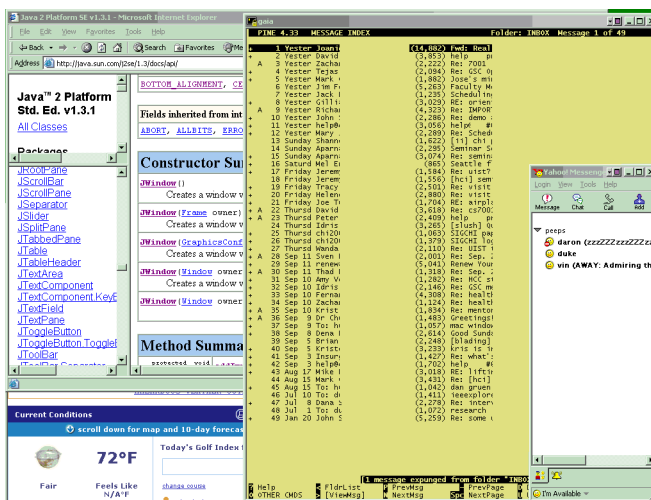


Figure 2. On the left, the user has some code documentation, an email client, an instant messaging client, and the weather in secondary windows, but has no room to place the primary code editing window. On the right is the result of the user snipping the secondary windows, allowing her to monitor new email, see who is online, use the relevant code documentation, and track the weather while having a large amount of space in which to edit code. The result is an information-dense, visually sparse display.

### 3.2 Other Usage Scenarios

Figure 2 illustrates an example of relevant region-based shrinking in action. The user has been awaiting arrival of email for some information about a piece of computer code that she needs to write and is additionally waiting for an instant message from a remote colleague to discuss some aspect of the code. She is also gathering documentation about a component she needs in writing the code, and since her office has no window, she also wants to keep an eye on the weather. All of these windows have left her with no good space in which to put the actual code editing window (left-hand side image). But, by defining relevant regions and snipping each window (either individually or through a global snip function that snips all windows at once), she has dramatically increased the amount of empty space (right-hand side image). This creates an information-dense but visually sparse display space in which a code editing window has been easily placed.

Another example of awareness is display space used while crafting a document on a dual-monitor system. The monitor to the left can have many snipped windows containing information from references used in the paper and the monitor to the right can have a document editor and a document viewer for the paper. Without relevant region snipping, the left screen could only contain one or two resources at one time because the user would have to display entire windows. Snipping allows quick access to many resources, with the ability to gain more context by simply unsnipping any window. Window snipping could be viewed as superior to using physical resources in that contents never have to be “put away” or “bookmarked” to find information because the relevant parts are always glanceable and the context is always nearby.

### 4. DISCUSSION

We must consider the feasibility and usability of relevant regions and the corresponding snip and unsnippet operations. We envision the simplest of interfaces: a simple mouse-click and drag to draw a rectangle around the relevant area of the window, and a button in the border of the relevant region to snip and unsnippet the window. To disambiguate between input destined for the window, we may have to force the user to hold a special keyboard button when drawing the region, or use a hot-key to indicate that a user is about to draw a relevant region. We claim that if the operation is found to be valuable, then users will gladly pay this additional cost; the apparent popularity of the Exposé hot-key operations for switching among windows is some evidence supporting our claim [14]. Furthermore we find that the operation is technically feasible, given systems such as Ametista [20] that demonstrate how windows can be treated as dynamic images.

Of course a critical piece of future work will involve the evaluation of our proposed operations in a real windowing environment. Claims about an evaluation of a system that is designed to aid with task and space management should be made very carefully. Even for simple space management, quantitative analysis is difficult and perhaps unfruitful (because managing space is less about how much space is used and more about how well space is used). We can envision many simulations of a task that involves several related subtasks (such as some of the examples already mentioned in this paper, including the document creation task), but constructing a viable simulation of the management of several disparate tasks is indeed quite difficult. Consequently, a longitudinal study is likely to be more appropriate for our operations, because we could assess how the operations have influenced users’ day-to-day behaviors. Other researchers

have also argued for this type of evaluation for their space management systems (e.g., Rooms [4]).

### 5. REFERENCES

- [1] Badros, G.J., Nichols, J., and Borning, A. SCWM: An intelligent constraint-enabled window manager. *Proc. AAAI Spring 2000 Symp. on Smart Graphics*, AAAI press, 76-83.
- [2] Beaudouin-Lafon, M. Novel interaction techniques for overlapping windows. *Proc. UIST 2001*, ACM press, 153-154.
- [3] Bell, B.A. and Feiner, S. Dynamic space management for user interfaces. *Proc. UIST 2000*, ACM Press, 239-248.
- [4] Card, S. K., and Henderson, A. A multiple, virtual-workspace interface to support user task switching. *Proc. CHI 1987*, ACM Press, 53-59.
- [5] 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.
- [6] Funke, D. J., Neal, J. G, and Paul, R. D. An approach to intelligent automated window management. *Int. J. of Man-Machine Studies* 38, (1993), 949-983.
- [7] Gaylin, K. How are windows used? Some notes on creating empirically-based windowing benchmark task. *Proc. CHI 1986*, ACM Press, 96-100.
- [8] Grudin, J. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. *Proc. CHI 2001*, ACM Press, 458-465.
- [9] Guimbretière, F., Stone, M., and Winograd, T. Fluid interaction with high-resolution wall-size displays. *Proc. UIST 2001*, ACM Press, 21-30.
- [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. AVI 2004*, ACM Press, (these proceedings).
- [11] Hutchings, D. R. and Stasko, J. Revisiting display space management: Understanding current practice to inform next-generation design. *Proc. Graphics Interface 2004*, (to appear).
- [12] Hutchings, D. R. and Stasko, J. QuickSpace: New operations for the desktop metaphor. *CHI Extended Abstracts 2002*, ACM Press, 802-803.
- [13] Kandogan, E. and Shneiderman, B. Elastic windows: Evaluation of multi-window operations. *Proc. CHI 1997*, ACM Press, 250-257.
- [14] Macintosh OS X Exposé hot key operations. <http://www.apple.com/macosx/panther/expose.html>.
- [15] MacIntyre, B., Mynatt, E. D., Volda, S., Hansen, K. M., Tullio, J., and Corso, G. M. Support for multitasking and background awareness using interactive peripheral displays. *Proc. UIST 2001*, ACM Press, 41-50.
- [16] Myers, B. Window interfaces: A taxonomy of window manager user interfaces. *IEEE Computer Graphics and Applications* 8, 5 (1988), 65-84.
- [17] Mynatt, E. D., Igarashi, T., Edwards, W. K., and LaMarca, A. Flatland: New dimensions in office whiteboards. *Proc. CHI 1999* ACM Press, 346-353.
- [18] Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Ridsen, K., Thiel, D., and Gorokhovskiy, V. The Task Gallery: A 3D window manager. *Proc. CHI 2000*, ACM Press, 494-501.
- [19] Robertson, G., Horvitz, E., Czerwinski, M., Hutchings, D. Baudisch, P., Meyers, B., Robbins, D., and Smith, G. Scalable Fabric: A flexible representation for task management. *Proc. AVI 2004*, ACM Press, (these proceedings).
- [20] Roussel, N. Ametista: a mini-toolkit for exploring new window management techniques. *Proc. Latin American Conf. on HCI 2003*, ACM Press, 117-124.