WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space

Desney S. Tan, Brian Meyers, Mary Czerwinski
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
desney@cs.cmu.edu, {brianme, marycz}@microsoft.com

ABSTRACT
Each window on our computer desktop provides a view into some information. Although users can currently manipulate multiple windows, we assert that being able to spatially arrange smaller regions of these windows could help users perform certain tasks more efficiently. In this paper, we describe a novel interaction technique that allows users to replicate arbitrary regions of existing windows into independent windows called WinCuts. Each WinCut is a live view of a region of the source window with which users can interact. We also present an extension that allows users to share WinCuts across multiple devices. Next, we classify the set of tasks for which WinCuts may be useful, both in single as well as multiple device scenarios. We present high level implementation details so that other researchers can replicate this work. And finally, we discuss future work that we will pursue in extending these ideas.

Categories and Subject Descriptors: H.5.2 [Information Interfaces and Presentation]: User Interfaces - graphical user interfaces, screen design, windowing systems; H.5.3 [Group and Organizational Interfaces]: Computer supported cooperative work.
General Terms: Human Factors, Performance.
Keywords: Windows, information management, spatial layout, screen space, regions, interaction technique, collaboration.

INTRODUCTION

An increasing number of tasks require that users coordinate and operate on information from multiple sources. Each source of information is typically contained within a window, the fundamental unit at which users can currently easily manipulate information. Oftentimes, users benefit from simultaneously viewing relevant information that exists within different windows. Additionally, the spatial layout of this information may be crucial to effective task performance as it helps users not only to establish spatial relationships but also to visually compare contents.

Unfortunately, even with the emergence of large displays, there is usually not enough screen space to view all windows simultaneously. Even when this is possible, having entire windows visible can introduce so much extraneous space in between relevant information that spatial location becomes less helpful. Alternatively, users can adjust windows so that they contain only the relevant information, and then lay them out. However, this is an extremely tedious task as it involves multiple iterations of resizing windows and scrolling content, especially in applications such as web browsers that recalculate the layout of information based on the size of the window.

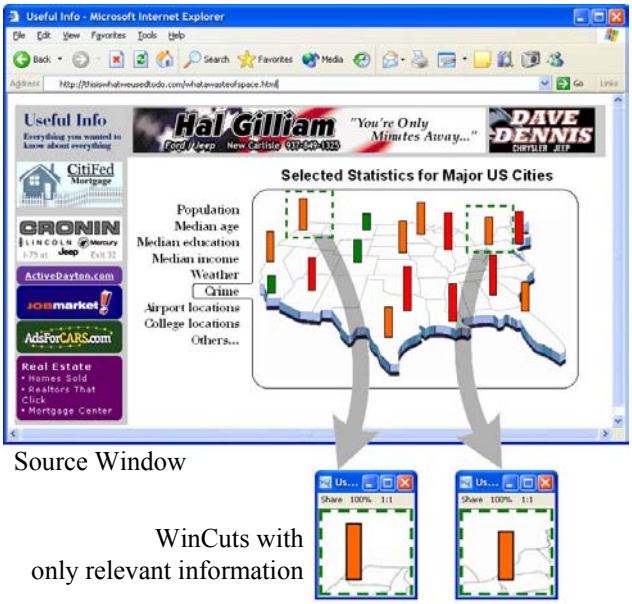

Figure 1: User makes two WinCuts to compare statistics between Seattle and Pittsburgh. The user could further WinCut the menu to control the different charts in the existing two WinCuts without using the source window. Using WinCuts for the task requires less screen space and effort.

Our work has focused on allowing users to easily specify and organize relevant regions of information contained within multiple windows so that they can make more effective use of screen space to perform their tasks more efficiently. In this paper, we describe a novel interaction technique that allows users to replicate arbitrary regions of existing windows into independent windows called WinCuts. Each WinCut provides continuous visual updates of the region as well as input redirection mechanisms that allow users to interact with content. We further describe how the ability to share WinCuts across devices may extend the utility of this system to multiple users engaged in co-

located collaboration as well as single users working with multiple devices. We provide a non-exhaustive classification of the set of tasks that we believe might benefit from the use of WinCuts. Next, we discuss high-level implementation details for our current prototype. Finally, we discuss future directions that we will pursue with this work.

RELATED WORK

There exists a large body of research exploring window management systems, which allow users to arrange multiple windows on the screen (for history and review, see Myers [5]). Within this body of work, many researchers have compared the cost-benefit tradeoffs that such systems impose. For example, Bly et al. compared tiled window systems, which automatically determine the size and location of all windows such that each window is always completely visible, to overlapping window systems, which allow the user to control size, location, as well as the overlap or visibility of windows [1]. They found that tiled systems were optimal when the system picked arrangements that conformed to the relevant contents of windows. However when it did not, overlapping systems were far superior, even though the user had to exert additional effort to explicitly manage windows. These results suggest that the ideal system is one which requires users to exert the least amount of effort but ensures that information is laid out in a manner that best supports the task at hand. Wickens et al. further augment these findings with their proximity compatibility principle, which holds that the more two information sources are used within the same task, the closer they should be displayed on the screen [9].

More recent work such as the Adaptive Window Manager [7], Elastic Windows [3], and Hutchings' operations for display space management [2] has explored different mechanisms for more efficient windows management. Unfortunately, since these schemes continue to treat windows as the fundamental unit of information, it is still difficult to lay out smaller chunks of relevant information contained within multiple windows.

In separate work, researchers have explored the benefits of viewing and operating on information and applications across multiple devices. This is useful both for individual users working on multiple devices, as well as for groups of users, each with personal devices, working together. Like much of the work on window management systems, most of this work allows users to share entire screens or regions of the screen [6] or, more recently, entire windows and applications (for review, see Li & Li [4]). To our knowledge, WinCuts is the first piece of work that allows users to explicitly manage and share smaller regions of windows.

WINCUTS INTERACTION TECHNIQUE

To create a new WinCut, users hold down a keyboard modifier combination, control-"accent grave", which brings up a semi-transparent tint over the entire desktop. They then click and drag the mouse over a region of a window to specify a rectangular region of interest (ROI). They can redefine this region as many times as they like. When they are satisfied with the ROI, they release the keyboard keys. The tint disappears and a new WinCut appears on top of the source window, slightly offset from the location of the ROI. The source window is unaffected. The WinCut is differentiated from regular windows by a green dotted line around the content region of the WinCut (see Figure 1). Users may make as many WinCuts as they wish, either from a single source window, or from multiple windows.

Each WinCut is a separate window and can be managed much like a regular window. It shows up in the Windows taskbar and can be minimized, restored, moved, and closed. Unlike other windows, however, maximizing or resizing a WinCut preserves the relevant information that is shown and instead rescales the content within the WinCut. This allows the user to make the information fill as little or as much space as they would like. For convenience, we have provided menu functions that allow a user to return the content to its original size or to constrain its aspect ratio.

WinCuts contain live representations of the content that appears within the ROI on the source window. In other words, the user can not only view updating content from the source window through the WinCut, but can also directly interact with content in it, just as they would in the original window. Since this view is tethered to the source window and not a region of the screen, users can move and even hide the source window without affecting the WinCut.

We have augmented this basic interaction with the ability to share WinCuts across multiple machines. Running the WinCuts system allows a user to send and receive WinCuts from other machines running the system. After creating a WinCut, a user can click on the "Share" button in the menu bar of a WinCut. A dialog box pops up for the user to specify the machine with which the WinCut should be shared. Doing this causes the current WinCut to appear on the destination machine. Once this is done, the user can interact with any local window and have the relevant WinCut update on the destination machine. However, aside from minimizing, restoring, resizing, moving, and closing the WinCut, users cannot interact directly with content on the remote WinCut. We do not perform input redirection on remote WinCuts because of the complication it introduces when multiple input streams collide. A red dotted line around the content indicates that the WinCut is read-only on the remote machine. This system works with multiple machines simultaneously sending and receiving WinCuts.

In order to provide a simple mechanism to manage WinCuts on remote machines, we couple WinCuts with a separate program we call Visitor. Visitor redirects the input stream over the network so that a user can use a mouse and keyboard connected to one computer to control the input on another computer. When running Visitor, a user simply moves the cursor off an edge of the local screen to take control of the cursor on a remote screen. In this way, users

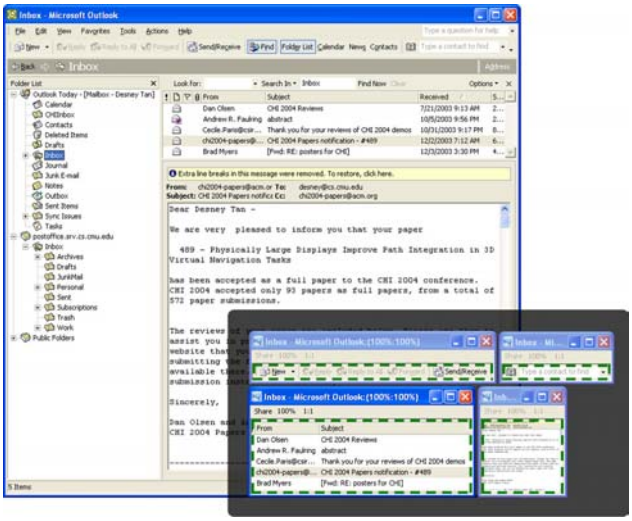

Figure 2: (*Behind*) Original full-sized E-mail client window. (*Front inset*) WinCuts used to construct a functional e-mail interface that takes up less than one third of the screen space.

can easily use the mouse connected to the local computer to manage WinCuts on the remote screen. Multiple users can take turns doing this.

WINCUTS FOR SINGLE MACHINE TASKS

Our main motivation for initially creating WinCuts was to provide users with tools necessary for effective **spatial organization of information with limited screen space**. At its core, WinCuts provides a lightweight mechanism for a user to specify relevant regions of information contained in various windows and then use standard windows management techniques to organize and lay these out.

However, in addition to content layout, we quickly found that WinCuts were also very useful for **monitoring or peripheral awareness** tasks. In these tasks, users are trying to keep abreast of updating information using the least amount of screen space possible. Using WinCuts, users can now specify the ROI, scale it to an appropriate size, and move it to an appropriate location on the screen.

Perhaps the most unexpected use of the system has been as a **rapid interface prototyping** tool. In fact, we can recreate entire interfaces by making WinCuts of various regions and scaling or rearranging them appropriately. Using this technique, we have explored how various interfaces would function if laid out differently. We have also explored several focus-in-context and fisheye view ideas by creating multiple adjacent WinCuts and scaling them to different degrees. Without WinCuts, most of these ideas would have taken much longer to build and explore.

Figure 2 illustrates one scenario in which WinCuts is useful. In the inset, the user has spatially reorganized regions of the original e-mail client window shown behind to reduce the screen space used by this task. The user has reconstructed relevant interface components, including parts of a toolbar and the "find a contact" combo box. They have also chosen to display only two columns of the inbox, the sender and the subject. Finally, they have scaled the message pane down to get an idea of what is contained within messages. The user can interact both with the inbox to change the message in the message pane as well as with the various interface components. They can also hit the 100% button on the message WinCut to read it at full size.

REMOTE WINCUTS FOR MULTIPLE MACHINE TASKS

In small informal groups, users often come together with various pieces of information contained on their personal devices, such as their laptops. In current **co-located collaboration** scenarios, users have to view the contents of one machine at a time, for example when sharing a common projector. Alternatively, to view all the material together, users could either print out relevant material or combine it on a single machine. This is not optimal, especially when the material is dynamic and cannot be determined beforehand, requires live editing, or is interspersed within private information that the author does not wish to share.

Using WinCuts, users can easily exchange updating views of relevant information, which owners can continue to edit on their respective source machines. Furthermore, if there is a shared display available, such as a projector, multiple users can send their WinCuts to this shared visual space so that everyone has a consistent view of the shared information (see Figure 3). Using Visitor, users can manage WinCuts on the shared machine.

As currently implemented, WinCuts allows users to **annex display space** offered by various devices on their desktops. For example, users can send WinCuts of peripheral tasks onto the screen of their laptop rather than taking up valuable screen space on their main machine. As we explore input redirection on remote machines, we expect that this will become an even more compelling scenario, as users can use WinCuts to take advantage of specialized input capabilities on various devices, such as pen input on the tablet PC. Also, this might be useful in placing particular parts of interfaces on a remote machine and interacting from afar, for example, using the laptop as a **remote control** for the media player running on the desktop machine.

HIGH-LEVEL IMPLEMENTATION DETAILS

We have implemented WinCuts as a standalone application in Windows XP using Microsoft Visual C++ .NET and util-

Figure 3: Users share information from their respective laptops by sending WinCuts to the projected display.

izing the Win32 Graphics Device Interface (GDI) API. When a user specifies a desired WinCut, we first calculate the coordinates of the ROI within the source window. Next, we create a device context, into which we periodically force the entire source window to render, using the `printwindow` API call. From this device context, we perform a `stretchblt` to scale and copy the ROI into our WinCut. The reason we first do the `printwindow` is to ensure that occluded parts of the window that do not normally render are properly captured. In order to ensure that content remains relatively fresh, we refresh the image once every second for each WinCut. We are exploring schemes that dynamically update when necessary rather than being based on a timer. This would allow us to work at more interactive rates, but is technically difficult as it requires knowing when each individual application has repainted any part of its window. To improve performance, subsequent WinCuts that depend upon the same source window reuse the appropriate device context that we have already created.

In order to redirect input, we currently activate the source window and bring it to the front when the cursor first moves into the WinCut content region. We then programmatically move the actual cursor to the corresponding spot on the source window. In order to simulate interaction with the WinCut, we draw a copy of the cursor on the corresponding piece of content in the WinCut. Hence, while all interaction actually happens directly on the source window, the user has all the feedback of interacting on the WinCut. One caveat to this approach is that source windows coming to the front can sometimes occlude information that is relevant to the task. We currently get around this by sending the source window to an extra display device that is not visible to the user. Another caveat is that since we do not explicitly handle them, popup menus and other windows that appear based on the location of the actual cursor appear on the source window and may not be seen on the WinCut. Solving these problems remains future work.

For remote WinCuts, we currently open peer-to-peer socket connections and send images of the `printwindow` device context, compressed as Portable Network Graphics (PNG), to the destination machine. We also send the corresponding coordinates so that appropriate WinCuts can be made on the destination machine. Sending subsequent WinCuts then, is as easy as sending additional coordinates along with the name of the device context.

FUTURE WORK

Because of how easy and useful it is to create WinCuts, users usually end up with many more WinCuts than they had windows. Unfortunately, most windows management systems do not scale well to a large number of windows, some of which might be logically associated with others. This has motivated us to explore methods that allow users to easily manage groups of windows as well as to perform simple operations on these groups [7]. We hope that these systems will prove useful even when the user is not using WinCuts, but just has a large number of windows open.

Another area we are exploring is the utility of tying the ROI to the underlying information rather than window regions. In the current system, when a user scrolls the source window, multiple WinCuts may be affected since they are defined only by the geometric region of the window and not the semantic content. While this is useful in many scenarios, we realize that tethering WinCuts to actual content might provide additional utility.

Finally, we plan to conduct formal evaluations to measure the usability of the interaction model surrounding WinCuts. We will also perform field studies deploying WinCuts to users involved in information work and group meetings as well as controlled studies to closely examine the usefulness of WinCuts in particular settings. We are especially interested in how WinCuts might affect productivity as well as social interaction in co-located collaborative work.

ACKNOWLEDGEMENTS

We are grateful to George Robertson, Patrick Baudisch, Greg Smith, Duke Hutchings, and Judy Olson for their insightful comments and discussion of this work.

REFERENCES

1. Bly, S.A., Rosenberg, J.K. (1986). A comparison of tiled and overlapping windows. *CHI 1986*, 101-106.
2. Hutchings, D.G., Stasko, J. (2002). New operations for display space management and window management. *Georgia Institute of Technology Technical Report GIT-GVU-02-18*.
3. Kandogan, E., Schneiderman, B. (1997). Elastic windows: Evaluation of multi-window operations. *CHI 1997*, 250-257.
4. Li, D., Li, R. (2002). Transparent sharing and interoperation of heterogeneous single-user applications. *CSCW 2002*, 246-255.
5. Myers, B.A. (1988). A taxonomy of window manager user interfaces. *IEEE Computer Graphics & Applications*, 8(5), 65-84.
6. Richardson, T., Quentin, S., Wood, K.R., Hopper, A. (1998). Virtual Network Computing. *IEEE Internet Computing*, 2(1), 33-38.
7. Smith, G., Baudisch, P., Robertson, G.G., Czerwinski, M., Meyers, B., Robbins, D., Horvitz, E., Andrews, D.. (2003). Groupbar: The taskbar evolved. *OZCHI 2003*.
8. Stille, S., Minocha, S., Ernst, R. An adaptive window management system. *INTERACT 1997*, 67-68.
9. Wickens, C.D., Carswell, C.M. (1995). The proximity compatibility principle: Its psychological foundation and relevance to display design. *Human Factors*, 37, 473-494.