QuickSpace: New Operations for the Desktop Metaphor

Dugald Ralph Hutchings, John Stasko
College of Computing, GVU Center
Georgia Institute of Technology
Atlanta, GA 30332-0280 USA
{hutch, stasko}@cc.gatech.edu

ABSTRACT
The explosion of information available to everyday users has resulted in numerous applications that allow users to access this information. Fundamental desktop operations fail to assist the user efficiently display all of the information available in these applications. We propose a number of new window and space management techniques that attempt to solve this problem.

Keywords
Window management, space management, desktop user interfaces

INTRODUCTION
The fundamental desktop operations provided by window systems (open, close, resize, reposition, iconify, and sometimes raise and lower) have remained largely unchanged since the introduction of the paradigm approximately twenty years ago. Changes in the use of the desktop have rendered this set of functions insufficient, and many different approaches (window managers) have been presented to mitigate this problem [1].

Prior developments propose to extend or expand the desktop metaphor. One extension allows users to interact with a three-dimensional space [5]. Another expands the metaphor by augmenting the fundamental properties of overlapping windows in different states [3]. Our work adopts a different approach: we introduce a set of fundamental operations that focus specifically on allowing users to quickly allocate space to a window through operations that have a direct correspondence to physical actions.

FOCUS & RESULTS
The operations that we propose strive to be a viable alternative to complex window management systems while extending the fundamental windowing operations. Specifically, we introduce functions that simultaneously (1) attempt to yield a large amount of screen space to the window on which the function is operating and (2) preserve the visible portions of all other windows on the desktop (*i.e.*, maintain the context from other windows). Additionally, the work that we present is not limited to the scope of the desktop. Indeed, our functions can be applied to multi-window applications.

Expand
The first idea that we explore is quite simple. A user selects a window, and the window subsequently consumes all available surrounding empty space. We call this process *expansion* (fig. 1, view 1). Note that "empty space" is any region on the desktop that contains no windows (the reader may find a deeper exploration of how empty space can be represented and applied to window operations in [4]). Once a border of the expanding window has reached the border of another window, or has reached the edge of the screen, expansion ceases in the direction of that border. The advantage of expansion is that with a very simple action, the user can allocate available unused space to a window without supplanting any window on the desktop.

Shove
One possible disadvantage of the expansion algorithm is that the space allocated to the expanded window is insufficient. Hence, we have developed a "more aggressive" function called *shove* that can be invoked by a user either directly or after an expansion (view 2). When a user selects a window as a shover (call the window *S*), it will grow in each direction just as it does in expansion. However, whenever *S* reaches the border of some other window on the desktop (call it *O*), *S* begins to "shove" *O* in the appropriate direction. In other words, when *S* is moving and reaches the border of *O*, *O* begins to move in the same direction. If *O* subsequently reaches the border of yet another window (call it *Y*), then *Y* also moves in the appropriate direction. This continues until the last such "shoved" window reaches the edge of the screen. This shoving action creates yet more space for *S*, while leaving the initially viewable portions of *O*, *Y*, ... still visible.

This algorithm can cause many windows to change position on the desktop. However, the advantage of this technique is that the *relative positioning* of *S* to all other windows on the desktop remains the same regardless of the final positions of other windows on the desktop. Another possible advantage is that since *S* consumes much of the desktop space, there is little search time needed by the user to find other windows. There is admittedly also a possible disadvantage to this algorithm: if a user has related two unselected windows by proximity, this relationship may be destroyed by a shoving action elsewhere on the desktop.

Friend
Both expand and shove are easily applied to a window that does not overlap any other window on the desktop. Windows commonly overlap, so we should be able to

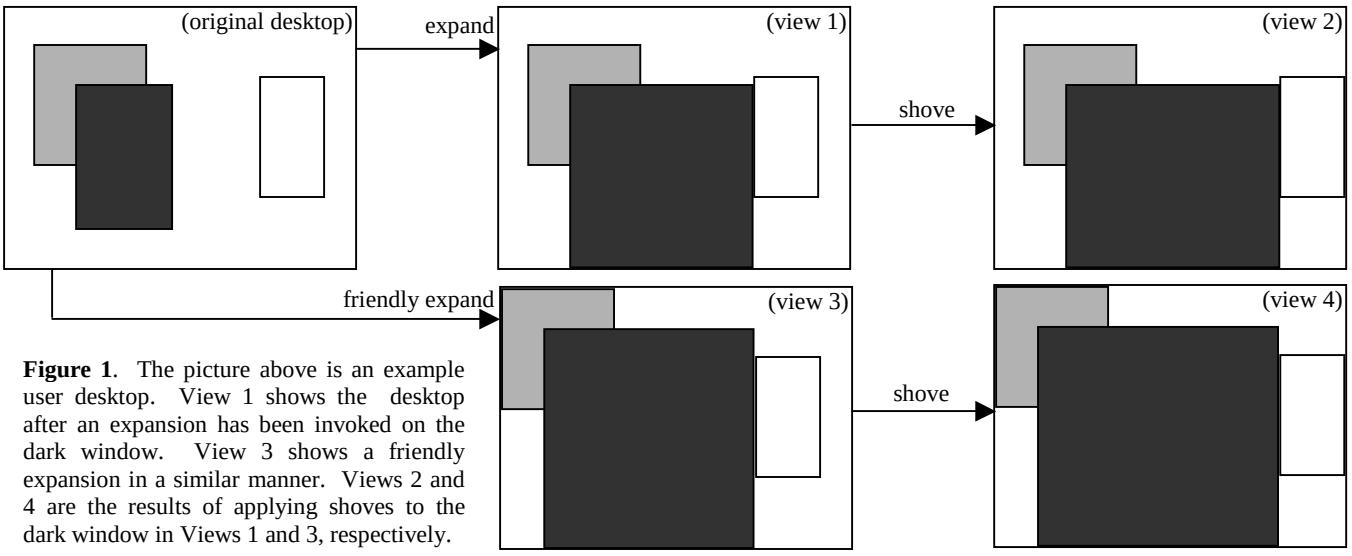

Figure 1. The picture above is an example user desktop. View 1 shows the desktop after an expansion has been invoked on the dark window. View 3 shows a friendly expansion in a similar manner. Views 2 and 4 are the results of applying shoves to the dark window in Views 1 and 3, respectively.

apply these operations to overlapping windows. Thus we define all windows *F* that overlap *W* as *friends*. Currently, users have the option to "take friends" or to "not take friends" when an expanding or shoving operation is requested. When a user selects to take friends, all friends *F* of the selected window *W* will move in the direction(s) that *W* grows and act as a virtual border for *W*. If a friend reaches a border of another (non-friend) window on the desktop, then the appropriate actions are taken based on the selected action (stopping for expansion, continuing for shoving). View 3 illustrates a friend.

Features

There are two key features of the current implementation that give the proposed operations the potential for user understanding and acceptance. The first is smooth motion between beginning and ending states, accomplished by animation. The proposed operations have an implicit sense of physical motion and reaction. Thus animating the process can be a tremendous aid in predicting the outcome of the operations. The other feature is the "undo" capability. After an operation or a sequence of operations is performed, the user has an option to return all windows to the previous locations. This process is also animated, so that users may explore operations for learning if they wish.

CONTRIBUTIONS

This work includes three advantages not specifically addressed in previous work. The first is *universality*: whether a desktop system uses an overlapping or a tiling strategy, our operations can be very useful. Another advantage is *simplicity*. The user makes exactly one selection in order to perform an operation that affects the entire desktop. Some window managers [2] have the capability of performing such operations, but require the user to define a large number of constraints each time the operation is requested. The third and final benefit is what we call *information preservation*, because the sections of windows that were visible before an operation are visible after the operation. Other systems that offer global functions often change the size or layout of other windows [6]; we simply reposition them.

FURTHER EXTENSIONS

We have a number of ideas for expanding on this work. An obvious disadvantage of these functions is that they are not very powerful when a number of large windows are present. A possible solution is the implementation of *relevant regions*: a user would define a region of a window that is most important, and when an expansion or shove is requested, the region would act as the border of the window (ignoring the actual window border). Relevant regions may also be useful for focus: if a window has one critical piece of information or contains distracting features such as animation, the user may be able to quickly hide the unimportant areas. Another possible solution is a "greedy shove" in which we sacrifice information preservation for more space by allowing a fixed percentage of the area of a window to be obscured during the operation.

REFERENCES

1. http://freshmeat.net/browse/56/
2. Badros, G.J., Nichols, J., and Borning, A. SCWM: An intelligent constraint-enabled window manager. In *Proc. AAAI Spring Symposium on Smart Graphics*, Cambridge MA, Mar. 20-22 2000.
3. Beaudouin-Lafon, M. Novel interaction techniques for overlapping windows. In *Proc. UIST '01* (Orlando, FL 2001), ACM press, 153-154
4. Bell, B.A. and Feiner, S. Dynamic space management for user interfaces. In *Proc. UIST '00* (San Diego, CA 2000), ACM Press, 239-248.
5. 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. In *ACM Trans. on Graphics*, 1986, vol. 5(3), 211-243.
6. Kandogan, E. and Shneiderman, B. Elastic windows: Evaluation of multi-window operations. In *Proc. CHI '97* (Atlanta, GA 1997), ACM Press, 250-257.