Glimpse: a Novel Input Model for Multi-level Devices

Clifton Forlines, Chia Shen
Mitsubishi Electric Research Labs (MERL)
201 Broadway, Cambridge MA 02139 USA
{forlines, shen}@merl.com

Bill Buxton
Buxton Design
888 Queen St. East, Toronto
Ontario Canada, M4M 1J3
bill@billbuxton.com

ABSTRACT
We describe a technique that supports the previewing of navigation, exploration, and editing operations by providing convenient *Undo* for unsuccessful and/or undesirable actions on multi-level input devices such as touch screens and pen-based computers. By adding a *Glimpse* state to traditional three-state pressure sensitive input devices, users are able to preview the effects of their editing without committing to them. From this *Glimpse* state, users can undo their action as easily as they can commit to it, making *Glimpse* most appropriate for systems in which the user is likely to try out many variations of an edit before finding the right one. Exploration is encouraged as the cumbersome returning to a menu or keyboard to issue an *Undo* command is eliminated. *Glimpse* has the added benefits that the negative effects of inconsistencies in the *Undo* feature within an application are reduced.

Author Keywords
Pressure Sensitive Input, Undo, Direct Manipulation, Three-State Input, Touch Screens, Stylus, Navigation

ACM Classification Keywords
H.5.2.h Information interfaces and presentation (e.g., HCI): User Interfaces - Input devices and strategies

H.5.2.i Information interfaces and presentation (e.g., HCI): User Interfaces – Interaction Styles

INTRODUCTION
Undo is a critical feature in many computer applications as it frees the user from the fear of experimenting with changes to the application's state. Creativity is enhanced when users are able to easily retract their changes if the result is unsatisfactory. People in creative industries will often state that the quality of their end product is a direct function of how many variations they *tried out and threw away* during the development of their product. Because of the frequency with which *Undo* is used, even small improvements to this feature can have a large positive effect. Additionally, many editing operations that take place with a GUI are irreversible because of the inconsistent implementation of the *Undo* feature by software developers.

CHI 2005, APRIL 2–7, 2005, PORTLAND, OREGON, USA.
ACM 1-59593-002-7/05/0004.

We propose a system-wide method of providing a *Glimpse* of the results of any operation completed with multi-level input devices. By multi-level, we mean that the input device is capable of sensing at least two levels of input (e.g. a stylus that senses light and heavy pressure or a mouse with a two-state button [5]) in addition to providing positional feedback. This positional feedback can be on-screen, in the case of an on-screen mouse pointer that tracks the movement of the mouse, or implicit, in the case of a finger or stylus with which the user operates directly on the display surface with graphical elements that are positioned directly underneath the input device. Our method has the added benefit that opting out of an action is as easy to perform as committing to it, making *Glimpse* most appropriate for systems in which the user is likely to try out many variations of an edit before deciding on any particular one.

BACKGROUND PART 1: PRESSURE SENSITIVE INPUT
Figure 1 shows Buxton's three-state model for stylus input [2]. In this model, light pressure input results in the

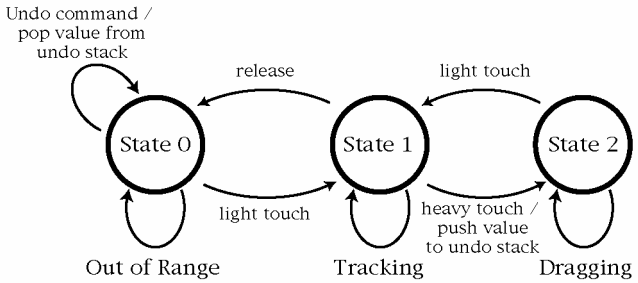

Figure 1. Buxton's three-state model for pressure sensitive input. Dragging an object results in its value being pushed onto the system undo stack. Restoring this saved value occurs after a separate command.

"tracking" of the input device (similar to moving the mouse while its button is up). While "tracking", a graphical pointer follows the movement of the input device on screen. Heavy pressure input results in "dragging" operations (similar to moving the mouse while its button is down). While indirect input devices like mice and trackballs would be impossible to use without State 1's "tracking" feedback (imagine using a mouse without the onscreen mouse pointer to tell you where you are), direct input devices like stylus and touch screens are less reliant on this positional feedback. "Tracking" the tip of a stylus or the user's finger with the virtual pointer is redundant (although it may improve

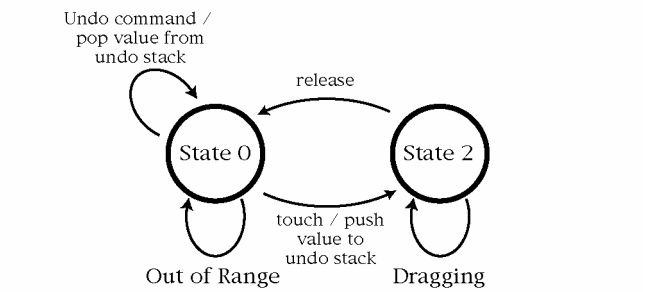

Figure 2. Buxton's touch sensitive direct input device state transition diagram without State 1 "tracking" feedback. Any touch that affects the value of an object will push that value to the undo stack. Restoring the saved value occurs after a separate command.

accuracy at the expense of speed) as the physical pen or finger provides clear feedback as to where the interaction in State 2 would begin were heavy input applied [2]. In other words, the pointing device itself (be it finger or stylus) becomes the tracking pointer.

The redundancy of State 1 is shown in Figure 2, which depicts non-pressure sensitive touch screen input. In this example, all contact with the screen results in State 2 interaction, and "Tracking" is done completely on the part of the user without the system's knowledge. The widespread use of and effectiveness of non-pressure sensitive touch screens illustrates the redundancy mentioned above of providing State 1 "Tracking" feedback for direct input devices. Thus, for direct input devices that do sense pressure, designers are free to experiment with mapping pressure to other characteristics of the interaction.

Ramos, et al. [4] described a continuous pressure-sensing stylus to manipulate multi-state objects. They mapped continuous pressure to visual properties of the pointer, e.g., moving the pointer down a list of menu selections as pressure increases, or to change the appearance of objects, e.g., making objects larger and smaller based on pressure. While this work provides a good exploration of the design space for pressure sensitive widgets, no recommendations are made for implementing pressure sensitivity in a system-wide manner.

Zeleznik, et al. [5] describe a set of interaction techniques enabled by adding a two-state button to a mouse. They refer to this type of device as a pop-through mouse that replaces the simple mouse button with a tactile pushbutton similar to the focus/shutter-release button used in many cameras. The authors advocate for a consistent functionality for this two-level input, and present several options such as tying sequential operations to the two-states or mapping the two-states to fine and coarse control of an object.

BACKGROUND PART 2: PROBLEMS WITH UNDO

In both Figures 1 and 2, when the system enters State 2, the current value of the property being edited is pushed onto the system's undo stack. Executing an *Undo* command (popping a value off of the stack and applying it to the previously edited object) is an entirely separate process from the multi-state input. The *Undo* feature is often provided through a menu command, through a toolbar button, or through a keyboard command. A user who is manipulating an object in an application with a mouse, stylus, or touch screen must move their pointer, pen, or finger away from the object being edited and traverse to the application's menu to issue the command. Alternatively, they may switch input devices in order to issue a keyboard command (assuming that a keyboard is present). After issuing the command, the user returns to the previously edited object to first make sure that the *undo* operation has successfully completed, and second to continue working with the object. A small amount of inconvenience in issuing the *undo* command is multiplied by high frequency of use. Therefore, even small improvements to the issuing of an *undo* command are desirable.

Undo is also inconsistently enabled by software developers. In general, the editing of the content of a document is an undoable command while the editing of the meta-data of a document is not. For example, most paint applications will allow a user to undo the result of a brush stroke or the changing of a color of an element in the application, but few will allow users to undo the result of picking a new color to paint with. Similarly, an operating system GUI may allow a user to undo the action of deleting a file, but not the action of moving a window across the screen. Text editing applications allow users to undo the results of editing a portion of text, but do not allow users to undo the results of scrolling to a different part of the document, making it difficult to return to a previous location in the text. The current color of a pen, the location of a window, and the positional value of a scroll bar are examples of values that are lost when edited.

Therefore, there is a need for a method for editing all objects that allows users to make, save and undo changes while in continuous control of the object without having to relocate the pointer to different locations on the display during an *undo* operation. Users will benefit from receiving a glimpse of the results of their actions.

THREE-STATE INPUT WITH AUTOMATIC UNDO

The technique we propose provides a method for editing objects with a multi-level input device such as a pressure sensitive stylus, pressure sensitive touch screen, or pop-through mouse. We have used both a TabletPC and a touch sensitive DiamondTouch [3] surface as our pressure sensitive input device. A TabletPC senses at least 8-bits of pressure for any contact with the tablet surface. The DiamondTouch device senses 8-bits of signal strength when a user's fingers or hands are in contact with the tabletop. No touch, light touch, and heavy touch are each associated with a range of these values. Any multi-state input device that also provides tracking (explicitly, as in the case of the pop-through mouse's on-screen pointer, or implicitly, as in the case of a stylus or finger) can exploit this technique.

As shown in Figure 3, our method replaces Figure 1's State 1 with a new state, which we call *Glimpse*. When an object

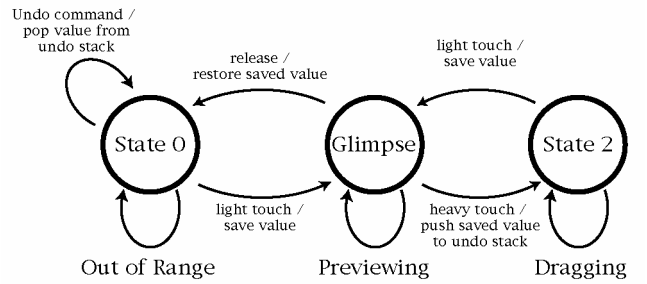

Figure 3. Glimpse enabled transition diagram for pressure sensitive direct input devices.

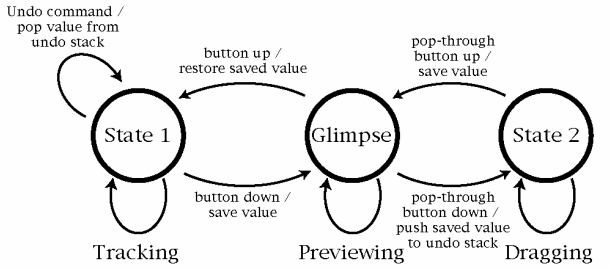

Figure 4. Glimpse enabled transition diagram for multi-level indirect input devices, such as a pop-through mouse [5].

is selected for editing through light pressure input, the system enters the *Glimpse* state and the current value of the property being edited is saved to memory separate from the system's undo stack. This light pressure input indicates intent to edit the selected object. While the user continues to manipulate the object using light pressure input, the system responds by previewing the results of their action.

When editing is finished, the user can either reject or accept the edit by performing one of two actions. If the user lifts their finger or stylus (or otherwise releases the input), the system returns to State 0 and the edit is automatically 'undone' by retrieving the saved state. When possible we animate this undo graphically so that the action is as clear to the user as possible. If the user increases the pressure of their input past a certain threshold, the system enters State 2 and the previewed changes to the edited object becomes the object's current state. In this transition, the previously saved values of the object are pushed onto the system's undo stack. While the user remains in State 2, changes to the object are saved as they occur. Reentering the *Glimpse* state from State 2 again stores the current value of the object being edited to memory. The *Glimpse* state previews the further change of this value, which can again be confirmed by reentering State 2.

It is important to note that our method does not replace the traditional use of the system's undo stack, but rather augments it. After finishing an interaction and while in State 0, a user is free to undo the most recent change in the traditional manner by using a menu or keyboard command. Our method stands in contrast to traditional systems that require the user to save the changes, and place the pointer on a menu command, an "*undo*" button, or to let go of the mouse and issue a keyboard command in order to return the object to the original state before the changes.

Figure 4 shows a *Glimpse* enabled indirect input device with multi-level input, such as a pop-through mouse. An on-screen pointer tracks the movement of the input device, and two-levels of input allow for previewing and confirmation of change.

EXAMPLES

Glimpse can be applied to well-known point-based editing and layout operations, such as changing the position, orientation, and scale of graphical objects, as well as meta-transactions on objects, such as changing selections and repositioning windows. Some less obvious examples of how *Glimpse* could be used are described in this section.

Pan and Zoom Interface

When navigating through a dataset using a pan and zoom interface, one often wants to temporarily zoom-in in order to take a more detailed look at some portion of the data before returning to the current zoom level. Using a traditional interface, zoom-in and zoom-out are separate commands (and may require the user to traverse to a tool pallet in order to switch tools). Furthermore, if zooming does not occur in fixed increments, inaccuracies in the operation of the zoom tool can make the task of returning to an exact zoom level difficult if not impossible. Similarly, for drag-to-pan movement around a dataset, retracing one's path in order to return to a previous location can be very difficult. It is a combination of these two difficulties that cause many users to complain that they become "lost" in the dataset when using a pan-and-zoom interface.

Our technique would enable users to preview different magnification levels with light touch input before choosing to remain at the new level with a heavy touch or to return to a previous level by releasing. Similarly a user may click and drag using light input to pan to other portions of the document, easily able to return to their previous position. Taking a temporary glimpse at details that are too small to see clearly (in the case of zooming) or off-screen (in the case of panning) becomes a single touch operation.

Navigation in a 3D World

The disorienting effects of panning and zooming around a two-dimensional space are multiplied when navigating through a 3D virtual environment. A common operation in a 3D modeling or animation application is the temporary repositioning of the virtual camera to "get a good look at" an object in the scene. Many applications, recognizing the special nature of this type of camera movement, create separate undo stacks for camera operations and editing operations on the 3D model. A typical sequence of commands is: 1) move or turn part of the model, 2) move the camera to another point of view to see if the part's new position is satisfactory for different viewpoints, 3) return the camera to the previous view, and 4) continue adjusting the model. While many techniques have been employed to address this type of sequence, including displaying multiple camera views and allowing for the quick return to common viewpoints (like orthographic and ¾ view), our technique

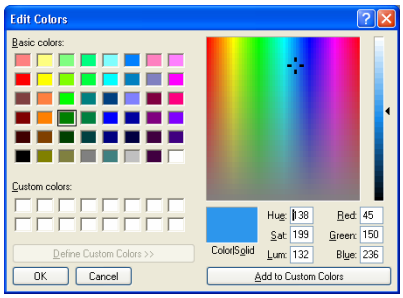

Figure 5. When choosing a custom color, there is no way to undo the repositioning of the color cursor within the color space. The user must explicitly save the current selection or remember the numerical values if they wish to experiment with other colors and then return to this selection.

would allow for the quick inspection of an object from several points of view without the fear of loosing one's position in space.

Color Selection

Most image editing and illustration applications allow the user to undo the results of a brush stroke or the changing of the color of an element in an illustration; however, few (if any) allow the user to undo the changing of the currently selected color in a color pallet. Figure 5 shows a typical color selection tool in which several properties of the color are exposed. When picking a color, users run into trouble if they are relatively happy with the selection, but want to continue searching for a better choice. Once they change the selected position in the HSB color space, they cannot return to the previously selected color without either remembering the exact numerical values or without having previously saved the color to the custom pallet.

Using our technique, the user could use light touch input to preview the effects of the color change while retaining the ability to return the tool to the previously selected color. Heavy pressure input would confirm the change in color for the tool.

Volume Control

Unlabeled slider bars such as a volume control make the task of accurately returning to a previous value difficult if not impossible. Using our technique, a user could preview new volume levels with light pressure input while still being able to quickly and accurately return to a previous volume.

Window Control

Window management has occupied a great deal of users' attention since overlapping windows were first introduced. One common operation is the temporary moving of a window to reveal what is behind it. Recent versions of Microsoft's Windows OS go so far as to include an easily accessible "show desktop" button in the interface so that users can quickly access items on their desktop. Similarly, users may minimize a window, view the items behind the windows, traverse to the task bar, and then restore the window. Beaudouin-Lafon introduced the idea of *peeling back windows* [1], with which a user could grab a corner region of an overlapping window to temporarily peel it back in order to view and select windows underneath it. Using our technique, windows could be moved or resized temporarily with first level input, so that the user can glimpse at the contents below it, and returned to their previous position by simply lifting the input device. With both techniques, a user is able to check under the foreground window without permanently reorganizing the display device.

Scroll Bar

While working on this paper, the authors commonly scrolled to the end of this document in order to glance at the paper's references. Similarly, when editing code, a programmer often uses a scroll bar to take a quick look at the details of another method or the definition of a variable. Our technique would enable a user to scroll to and view another portion of a long document before returning to the exact location they were previously editing. While various applications employ a wide array of methods for jumping around within a document, we are not aware of any that provide the accuracy and ease of use of our technique. Users would be able to glimpse at other portions of a document before returning to their previous location using only the scroll bar, never needing to find or traverse to another tool.

CONCLUSION

We have presented a model for multi-level input devices that aims to support both creativity in design and exploration of data. For many tasks, actions are as often, if not more often, undone as accepted; therefore, we have promoted *undo* to a first class operation. Future work must include the inclusion of Glimpse into design and data exploration applications so that experts in these fields can judge its effectiveness.

REFERENCES

1. Beaudouin-Lafon, M. Novel interaction techniques for overlapping windows, *Proc. UIST 2001*, ACM Press (2001), pp. 153-154.
2. Buxton, W. A Three-State Model of Graphical Input, In D. Diaper et al. (Eds), *Human-Computer Interaction - INTERACT '90*. Amsterdam: Elsevier Science Publishers B.V. (North-Holland), 449-456.
3. Dietz, P., and Leigh, D. DiamondTouch: a multi-user touch technology, *Proc. UIST 2001*. ACM Press (2001), pp. 219–226.
4. Ramos, et al. Pressure widgets, *Proceedings of the 2004 Conference on Human Factors in Computing Systems*, pp. 487-494, 2004.
5. Zeleznik, et al. Pop through Mouse Button Interactions, *Proc. UIST 2001*, ACM Press (2001), pp. 195-196.