An Insidious Haptic Invasion: Adding Force Feedback to the X Desktop

Timothy Miller and Robert Zeleznik
Department of Computer Science
Box 1910
Brown University
Providence, RI 02912
(401) 863-7653
tsm@cs.brown.edu, bcz@cs.brown.edu

ABSTRACT

This paper describes preliminary work in a project to add force feedback to user interface elements of the X Window System in an attempt to add true "feel" to the window system's "look and feel". Additions include adding ridges around icons and menu items to aid interaction, alignment guides for moving windows, and other enhancements to window manipulation. The motivation for this system is the observation that people naturally have many skills for and intuitions about a very rich environment of interaction forces in the non-computer world; however, these skills are largely unused in computer applications. We expect that haptic modifications to conventional graphical user interfaces, such as those we present, can lead to gains in performance, intuition, learnability, and enjoyment of the interface. This paper describes details of the implementation of the haptic window system elements, in addition to higher-level haptic design principles and informal observations of users of the system.

KEYWORDS: force feedback, haptic user interface, graphical user interface

INTRODUCTION

Current graphical user interfaces (GUIs) involve the sense of touch only in what amounts to an accidental manner: haptic feedback comes entirely from the basic physical properties of input devices and does not change with the state of the human-computer interaction (as anyone knows who has used a computer that has crashed). Haptic feedback in these GUIs is limited to the feel of mouse buttons as they are pressed, the feel of keys on the keyboard, and the friction and proprioception involved in moving the mouse. Although unrelated to the state of the interface, these forms of *accidental* haptic feedback often correlate with the user's input and, further, prove useful during interaction. For example, Barrett and Krueger [1] found that both touch typists and casual users had significantly better performance with and more positive feelings toward a standard keyboard than a flat keyboard that did not provide any haptic feedback as keys were pressed. In the context of the virtual environment that user interfaces attempt to provide, however, accidental haptic feedback is impoverished when contrasted with the rich haptic feedback generally present in real-world interactions. Thus, adding explicit control of haptic feedback can radically change an interface and add true feel to the interface's "look and feel". Moreover, since people develop many skills that rely on the rich haptic feedback of everyday contexts, it seems reasonable that similar skills might be utilized in 2D graphical user interfaces and might well enhance performance, intuition, and enjoyment.

UIST '98. San Francisco, CA

This paper describes the initial stages of a project for haptic augmentation of standard GUI elements of the X Window System [10]. The device used to provide force feedback is a 1.0-workspace PHANToM [4] with encoder gimbal, made by SensAble Technologies, Inc. This device (see Figure 1) is a 6-degree-of-freedom (DOF) position input/3-DOF force output device with a stylus grip; the switch mounted on the stylus is mapped by this program to X's button 1.

PREVIOUS WORK

Traditional GUIs have long attempted to provide what might be considered workarounds for their lack of interaction-dependent haptic feedback. These workarounds typically rely on snapping unconstrained mouse input to the constrained range needed by an input technique (*e.g.*, manipulating the tab of a scrollbar or drawing a constrained line in a drafting application). However, this snapping technique is not completely satisfactory: during a scrolling operation, for example, drifting off the scrollbar reduces the correspondence of the interaction as the angle between the user's direction of motion and the scrollbar's orientation varies. Force feedback can ameliorate this problem by physically constraining the user's input to valid positions.

Rosenberg and Brave [6, 8] used a 2-DOF input/2-DOF output force-feedback joystick to enhance GUIs for people with neuromotor disabilities, noting that the techniques have more general potential applicability; Rosenberg [7] announced the extension of these techniques to numerous other GUI elements for nondisabled users, using a mouselike 2-DOF out-

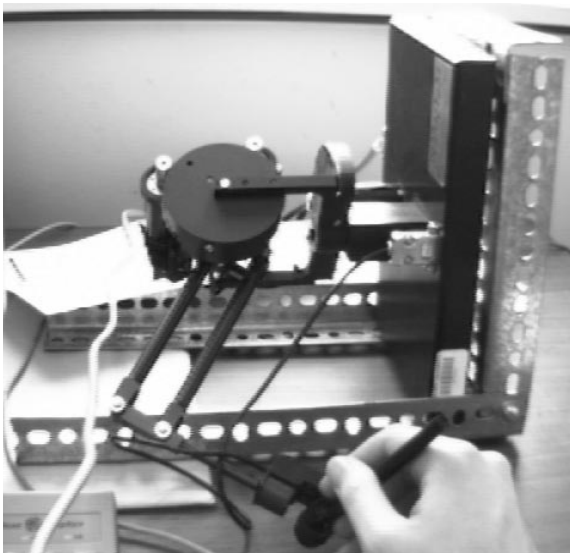

Figure 1: The 1.0-workspace PHANToM force feedback device in its tilted home (see Implementation Details for why it's tilted).

put device. Munch and Stangenberg [5] used a modified mouse with vibrotactile and braking feedback to add predictive target acquisition aids to a 2D GUI. The PHANToM used in this project provides different DOFs from both of these and is in addition held by the fingers in a pen-style grip rather than by the whole hand like a joystick or mouse, thus lending itself to different interface choices.

HAPTIFIED X

Guiding Principles

Literally haptifying the 2D GUI in the sense of adding forces according to the gradient of the pixel intensities might produce advantages, but we do not believe it would take full advantage of haptics. To exploit haptics further we identified three principles for designing haptic interfaces:

Reduction of Errors Through Guidance We want to use haptics to reduce two types of user movement errors. The first type relates to large-scale human motor-control errors, such as the natural drift occurring when users attempt to draw a straight line. An example of this in 2D GUIs is the tendency to drift off menus or hierarchical menu items. The second kind of error is smaller-scale noise in the user's input, which is magnified by small workspaces. This error is particularly important because users of our system tend to work in the small area defined by their fine-motor, fingertip interactions. To address these errors, haptic user interface elements are needed in both targeting and basic motion control (*e.g.*, friction).

Force As Feedback In general, we provide force as feedback based on, but not controlling, the user's input. Thus force feedback is frequently used to indicate, but not preclude, an impending transition, say to warn the user when sliding between two menu items. In addition, even when assisting the user in a motor control task, say to move along a straight line, we provide only force feedback that is directly proportional to the input forces applied by the user. In this latter example, the difference between force feedback based on the user's input and force control of the user's position is subtle and more a matter of degree than of kind. In either case, our principle, restated more specifically, is that forces should be *user-inspired*: the forces output should be scaled by the force of the user's input, generally along some direction other than that influenced by the output force. For instance, both friction and ridges in the surface exert a lateral force that depends on the user's perpendicular force.

Overridable Guidance Previous systems have implemented the idea that guiding haptic constraints can be overridden by forcibly "popping through" them; we expand this principle to include the ability to sidestep a constraint by going around it in the third dimension. Thus the user can move around a constraint known to be irrelevant without having to struggle through it.

Implemented Techniques

So far, only some of the many potential haptic GUI elements and techniques have been implemented. The intent of this paper is not to describe a final organized symphony of techniques, but rather to give some examples within a loose overall structure to point the way towards that symphony.

Workspace The basic haptic workspace is a shallow box $50\ \mathrm{mm} \times 40\ \mathrm{mm} \times 2\ \mathrm{mm}$ whose longest dimensions correspond to the 1280×1024 screen and are horizontal; the tip of the stylus is constrained to remain inside the box. That $2\ \mathrm{mm}$ of vertical distance may seem small, but it actually feels quite large when using the interface. Nearly all the forces in the implemented techniques occur through interaction with the surfaces of the box, which may have slight changes in their geometry (ridges added along windows, for instance). The bottom of the box, the *floor*, is the primary interaction surface; the side walls currently merely prevent the stylus from getting out of range. The upper surface, the *ceiling*, has a mirror image of all the interaction features of the floor and in addition has one special interaction described below for raising windows.

Icons Icons[1] are slightly dimpled, making a slight ridge around their edges to aid targeting. When an icon is being dragged to a target, the target icon has a deeper dimple (thus providing feedback about the potential valid targets as well as making them easier to target). In early evaluations, however, users often moved so fast that they would start to coarticulate their button presses and releases with the ends of their movements, so that, for instance, they clicked slightly before reaching an icon and then jammed up against the ridge around the icon.

To improve interface behavior here, a check is made for the user running into the ridge of an icon less than a tenth of a second after clicking (or releasing) the button, provided that the cursor is moving towards the icon at the time of the click or release. From informal observations, this check appears to catch nearly all occurrences of this kind of coarticulation,

[1]The icon haptification was done in an earlier pilot project and has not yet been integrated with the rest of the system.

and does not not trigger erroneously when the user wants to drop an icon near the target or intends to click before reaching an icon. Negative viscosity in the direction of the target was also implemented, but, seemed to have no significant effect. (Nor do many people seem to notice—presumably because the implementation is gentle enough to offset only viscosity naturally present in the device itself, the user's body, etc. Levels high enough to be noticeable feel disruptive and destabilizing.)

Windows Most of the haptic elements implemented so far have concerned window manipulation. From our observation, many people like their windows constrained to be entirely on the screen when dragging them, but occasionally want to override this constraint. To facilitate this, very small ridges were added to the surface when dragging a window that prevent dragging the window off the screen as long as the stylus is in contact with the floor, but do not interfere with motion if contact is not maintained. This lets the user switch smoothly between constrained and unconstrained dragging without having to distinguish them by the use of different combinations of modifier keys or the like.

Our informal observation also indicates that some people want to align their windows to be just adjacent to each other without necessarily being tiled rigidly, although they frequently want to break such alignment constraints. To support this alignment, small ridges were again used that could be "popped" through by applying a sufficient lateral force. In addition, since people often align windows in this way within groups with little regard to relationships between groups, collision checking is done only within the same "layer" of windows (this has no relation to visual layering on the screen; see the implementation details section for more information).

An earlier pilot program constrained the cursor and corresponding stylus tip to remain inside the program's window, and thus gave many users the intuition that they ought to be able to move the window by pushing against the side walls. This has been implemented in the present project by slightly dimpling client windows, leaving a small ridge on the edge that when pushed sideways moves the window. There are currently significant implementation constraints on this technique; see the implementation details section.

Placing ridges around the client windows made it somewhat harder to select the window manager's decoration border in order to raise an obscured window, particularly annoying if the titlebar is obscured or is far away from the cursor. (Our window manager setup uses a keyboard-input-follows-mouse-focus policy and requires windows to be raised explicitly, not just whenever the user clicks in them.) This problem was addressed by letting the user raise the window under the cursor by pulling up. The force sensation of raising a window is like pulling up on an inverted physical button. However, if the cursor is over a window that cannot be raised, users simply feel the hard surface of the ceiling when they pull up. Thus, the haptic feedback matches the possibility of performing the window-raise operation.

A final haptic technique was added to windows to let them be moved across the screen merely by pressing much harder than usual against the floor over the window, as if sliding a piece of paper across a desk. Making the force required to move the window too large results in discomfort and intermittent loss of the window, yet making it too small results in accidental moves of the window; the current setting is a compromise. The non-computer world has a similar ambiguity: the force required to slide a sheet of paper is quite similar to the force required to write on the paper. Thus this might be a good opportunity to apply two-handed input, just as in the non-computer world where the non-dominant hand is typically used to steady a sheet of paper for writing.

Menus Force feedback was added to menus by putting ridges between adjacent menu items. The intent was to make it easier to stop at a given item without overshoot, and, in hierarchical menus, to make it easier to remain on a given item while sliding onto its submenu. Unfortunately, the initial implementation of the ridges seems too "sticky" and menu selection is considerably harder: the user must exert significant force to overcome the ridge and winds up overshooting even more. This problem can probably be fixed, perhaps by making the bottoms of the ridges rounded or by allowing "tunneling" between menu items, particularly since Rosenberg and Brave [8] report increased performance with their menu haptification. Once the user gets to the right menu item, however, staying on it to select a hierarchical submenu does informally seem to be easier with our technique.

Observations

In preliminary refinement and informal evaluations of the drag-and-drop interface on a repetitive task of simply dragging icons to a target one after another, one author's speed seemed about double that when using the mouse. Others in our lab less familiar with the interface or device seemed to approach that level of speed with modest practice, on the order of 30 minutes. Clearly, these reports are not the whole story: they concern only performance, not comfort, usability, intuitiveness, naturalness, or other aspects; they concern only a very artificial task, not taking into account all the other tasks involved in normal use of a computer, some of which (such as typing) will be unaffected by this interface; and they also need to be confirmed with a user study. However, these reports do seem to indicate that this may be a promising avenue for exploration.

A radical change in the character of the interface is that, unlike traditional GUIs with which users visually attend to the pointer and target during targeting [2, 3], users of our system can instead visually attend to "what they are doing"[2], combining approximate spatial memory of their target with fine tuning from the haptic clues. It also appears that the usual deceleration time in targeting motions can be greatly

[2] "What the user is doing", in the sense of the interesting parts of the interaction, may not be the same as the target of a motion. Consider dragging a number of icons to a target that presents no information other than targeting feedback. Although [2, 3] do not address this case, [2] does report that lack of visual feedback significantly impairs targeting tasks, suggesting that some other kind of feedback would be necessary if the users were to attend to the icons to be dragged instead. From our own informal observations, attending to the next icon to be dragged yielded a significant fraction of the improvement in using the PHANToM over the mouse, while trying this strategy with the mouse impaired performance due to errors.

reduced, since users can anticipate the stopping force of the ridge around a target and maintain muscle tension so that that force will be sufficient to stop their motion.

The ergonomics of the device and setup for this application have elicited some negative comment, generally because of its prototype nature. Some users experience some cramping after moderate use; this is probably due in part to the lack of proper counterbalancing (see the implementation details) and in part to the unnaturally stretched arm positioned necessary to reach the device. The former can probably be remedied simply with mechanical expertise, but the latter seems to require more radical redesign of the physical desktop, as simply moving the PHANToM closer to the edge of the desk would leave no space to rest the hand. It is also difficult currently to grasp the stylus as close to the tip as one would like for good control; this is partly because the stylus button is too far back, as well as being a little big and clumsy, and partly because the encoder housing gets in the way. Both of those drawbacks should be simple for the manufacturer to resolve.[3] The current setup is also designed for right-handed people; the PHANToM can certainly be flipped around to become left-hand oriented, although this is something of a chore with the current mounting.

IMPLEMENTATION DETAILS

Architecture and X Interface

This system was implemented on an SGI O2 with an R10000 processor, using SensAble Technologies' GHOST haptic toolkit. The X pointer was controlled through the (fairly standard) XInput extension; SGI provides an addition to X that allows new input devices to be made available through that extension without having to recompile the X server.

The primary principle guiding this part of the implementation was to run the same applications with and without haptics with a completely seamless transition, ideally without the programs even needing to check whether haptics is running or not. To that end, all of the haptics-specific communication from X clients to the haptics process is done by the clients setting X properties on their windows. The one part of the current implementation that doesn't meet our ideal is that the window manager must check whether the haptics process is running to know whether to constrain windows to the screen (in the absence of haptics) or not when doing a constrained move. The only things other than pure simulation of a standard mouse's behavior that the haptics process must currently send back to X are indications of when the user is pressing down hard enough to move a window, and when the user has pulled up to raise a window. These are both currently communicated by pretending that the device has two extra buttons, one for each user action.

Since the PHANToM only has one button mounted on its stylus, while three would be preferable use with typical X applications, the haptics process uses the XInput extension to open the mouse device and get its middle and right button states to forward them as if they were the PHANToM's. Since other styli, such as many made for use with tablets, come with multiple buttons, this would not seem difficult to fix in the PHANToM's hardware. The XTEST extension was used to make the haptics process immune to X server grabs, thus preventing deadlock arising when the haptics process needs information from the server to release a button, and certain interface techniques wait for a button release before ungrabbing the server.

To facilitate rapid changes and prototyping of the client interaction with the haptics process, the first client haptification was of gwm, the Generic Window Manager (available by anonymous ftp from `ftp.x.org` in `contrib/window_managers/gwm`), because it is programmable in a variant of Lisp and thus does not require recompiling to make changes to its interface. As a result, only slight `C`-code modifications were necessary, primarily to allow setting of X properties on the window manager's windows rather than just the client windows. This made it possible to communicate the layering of windows for interwindow collision checking simply by having the window manager set a property on its top-level window indicating which layer the window should be in. As a result, the window manager can easily have user preferences determine which layer a given client window should go in.

Haptic Details

In an earlier pilot program, it was discovered that trying to use the stylus as a pen with the PHANToM in its standard orientation (rotated $90°$ clockwise from Figure 1) made the armature interfere with the stylus and user's hand often resulting in accidental button clicks. The PHANToM was therefore mounted in the makeshift structure shown in Figure 1. Unfortunately, the encoders at the tip of the stylus are not counterbalanced against gravity, so that the users feel a force attempting to twist the stylus out of their hands. Some experiments with makeshift counterbalances seem to indicate that this can be overcome, but the attempts were truly makeshift and fell apart fast.[4] In order to encourage a stylus grip more closely approximating that of a normal pen, the haptic workspace was set to have the floor as close to the physical desktop as possible without encoder interference, and four stacked mousepads were provided as a handrest whose top then is at about the level of the workspace floor. That floor is currently horizontal, although it seems likely that some tilt toward the user would be better.

A number of force parameters must be set in the implementation. For most of these, our initial guess worked out well and has not been adjusted. Closer attention has been paid to a few, however: the basic friction of the surface was set to try to approximate the friction between pencil and paper while maximizing controllability of pointer movement, and the resulting static and dynamic friction coefficients were both 0.15. The basic surface spring constant is set to $0.8\,\mathrm{N/mm}$, and the user must exert at least $3.2\,\mathrm{N}$ downward to slide windows.

Since the PHANToM cannot be constrained mechanically to remain exactly on the surface of an object, GHOST uses a virtual surface contact point (SCP) to calculate the force that

[3] In fact, the newest version of the PHANToM, shown at SIGGRAPH'98, resolves both these drawbacks and the counterbalancing problem.

[4] Again, the newst version of the PHANToM appears to resolve this issue.

should be applied from interaction with the virtual surface. This SCP is, roughly, the closest point on the surface of the object to the PHANToM, except that it is not allowed to pop through the surface and thus depends on the motion history. To implement friction, an additional SCP, the *stiction* point, is maintained that tends to stay in the same place on the object's surface but "slides" across the surface if the tangential force exerted by the user exceeds the force of static friction. (This discussion is based on the description in [9], which is presumed to be the technique employed in GHOST because their behaviors match and because at least two of the authors iare involved with with SensAble.)

However, the tangential force appears to be computed in GHOST based on the SCP reported by the object, rather than on the user's position; this means that when the user comes to a concave-outward corner that "traps" the SCP, the stiction point remains some distance away from the corner, no matter how hard the user pushes. Since the stiction point is the SCP value reported by GHOST to the user interface, the user cannot move the cursor all the way to the edge of the screen in the presence of friction. The effect of this is small in terms of physical movement, but becomes very noticeable with the large magnification between the haptic workspace and the screen size of this application. It also gets worse as the user grows more frustrated and thus presses the stylus down harder, increasing the force of static friction. To work around this problem, the program implements its own friction in the object's SCP computation routine by controlling the SCP it reports and moving it based on a pseudo-SCP that is the projection of the user's current point onto the line between what the current SCP would be without friction and the previous stiction point. (GHOST's friction constants are set to zero.) If the stiction SCP and current frictionless SCP do not share any face in common, the reported SCP is simply set to the frictionless SCP.

GHOST's implementation of surface contact damping appears to damp velocity in all directions whenever the stylus is in contact with the surface; the surfaces thus feel as though they have more friction than they should, which is particularly bad for low friction surfaces. To work around that, the program implements its own damping in a gstEffect that damps only the component of velocity perpendicular to the surface.

Menu items, icons, and windows all had the ridges around them implemented by dimpling each user interface element into the surface of the floor, as shown in Figure 2. The sides of the ridges always sloped in at a $45°$ angle; the sloped slides of all elements except drag-and-drop targets were one pixel wide, while drag-and-drop targets had 12.5-pixel-wide sides. The abrupt disappearance of the dimples, for instance when a menu is removed, means that the PHANToM is suddenly under the surface and as a result displays no forces; to avoid this, the system offsets the workspace by the depth of the dimple and gradually restores it to its original position at a rate of $1\,\mathrm{mm/s}$. (This is implemented with a gstDynamic subclass so that GHOST maintains the proper SCP positions.)

The infinitesimal ridges used to constrain windows onscreen

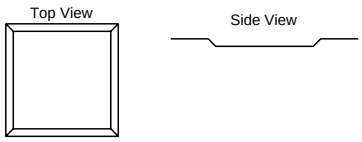

Figure 2: Geometry of the haptic dimple used in menu items, icons, and windows.

and for interwindow collision were implemented by having the desktop object prevent its reported SCP from moving outside of the appropriate region; this feels like a little ridge as long as the user is on the surface, but disappears as soon as the stylus is lifted off it. If the user exerts more than $2\,\mathrm{N}$ sideways against a ridge, the SCP pops through that constraint by temporarily suspending it.

The ridges placed around windows that allow them to be moved by pushing are currently active only for windows that are completely unobscured on the screen (currently determined by watching X VisibilityNotify events). In the future, we expect to track the layering of all the windows so that forces can be generated for all visible window borders (and for pushing from the outside too). The technique of pushing down into the floor to slide windows is available for partially visible windows as well.

The "click" the usef feels when pulling up to raise a window is implemented by having the program compute its own normal force (setting GHOST's parameter to zero) in a gstEffect subclass working in conjunction with the desktop object. This allows the program to simulate a standard button model with an initial springy area, dead band, and hard stop. Both the initial springy area and hard stop have the same spring constant as the rest of the ceiling, while the dead band currently has a constant restoring force of $0.5\,\mathrm{N}$. The click feedback occurs only when the pointer is over a window that can be raised, which is determined by checking if the pointer is contained in any mapped, non-override-redirect top-level window but not contained in any mapped, override-redirect top-level window.

CONCLUSIONS AND FUTURE WORK

Adding force feedback to 2D GUIs does seem to be a promising direction to explore. In particular, using a third DOF in this study offers control advantages over 2-DOF devices. The gain seems to be significant when users change their strategies to direct visual attention to the area of interest for their next activity. We explored this issue in the case where users had a general idea of where to move, and where haptic feedback was sufficient to guide them the rest of the way to the target. It seems reasonable that this gain results from the additional bandwidth introduced by the haptic channel. More careful and systematic studies in this area are clearly needed.

All the user-inspired forces tried in this project were either vertical displacement or displacement of the stylus tip into the simulated surface. There may be other ways in which forces can be inspired by the user's actions, but it certainly seems to work well to have vertical displacement control engagement of objects, constraints, and guides, with displacement into the simulated surface controlling the intensity of

feedback forces. Unfortunately, this conclusion seems not to be completely general, as the attempt with menu items didn't work well; some other principles may well be identified in future work, perhaps relating how smoothly the surface slope would change or to how closely objects or features are packed.

There are a number of techniques that would be interesting to prototype in this system. One is to have buttons with sequences of pressure levels, like the focus-hold or light-meter feature of the shutter button on many cameras. A possible mapping of operations is to show a preview of the main operation (by temporarily inserting text for a paste operation or bringing up a transparent window for a find operation) for light pressure, undoing the preview by releasing and confirmiing it by pressing harder. Other possible operations to map include simple help labels, more extensive help, and changing settings. It would also be nice to implement more extensively the idea of preventing clicking where clicking is not accepted.

It may be fruitful to prototype haptic user interface ideas using a general device like the PHANToM, and then see whether specific techniques can be extracted and made to work with other devices. The PHANToM does not have the best possible ergonomics for this application, but it is the best existing device the authors are aware of and this project may be regarded as prototyping for future more ergonomic devices. The PHANToM is also expensive, and if the multiple-pressure-level idea is viable, it could presumably be implemented using an ordinary mouse whose buttons have been modified to have multiple pressure stops (perhaps with a simple clutching mechanism to provide limited feedback). Another alternative is to reduce the cost of the PHANToM itself; SensAble has said that PHANToMs with smaller workspaces can be made less expensively than the current ones, and the results reported here indicate that a very much smaller workspace is viable for this application.

Other future work includes user studies, integrating the earlier drag-and-drop program elements with the current X version, smoothing dimples in some way (perhaps via force shading) to address the sticking problem with the current menus, adding force feedback to other simple UI elements such as scrollbars, URL targets in web browsers, etc., and constructing a toolkit of UI techniques so that haptic user interface elements could be assembled and prototyped from those techniques rather than implementing them from scratch.

ACKNOWLEDGMENTS

Thanks to Lee Markosian, Katrina Avery, John Hughes, and Joseph LaViola for reading this paper and making suggestions. Thanks also to the helpful suggestions of the anonymous reviewers. This work is supported in part by the NSF Graphics and Visualization Center, Alias/Wavefront, Advanced Networks and Services, Autodesk, Microsoft, Sun Microsystems, Silicon Graphics, Inc., and TACO.

REFERENCES

1. BARRETT, J., AND KRUEGER, H. Performance effects of reduced proprioceptive feedback on touch typists and casual users in a typing task. *Behaviour and Information Technology 13*, 6 (November–December 1994), 373–381.

2. ELLIOTT, D., LYONS, J., AND DYSON, K. Rescaling an acquired discrete aiming movement: Specific or general motor learning? *Human Movement Science 16*, 1 (February 1997), 81–96.

3. KENNEDY, A., AND BACCINO, T. The effects of screen refresh rate on editing operations using a computer mouse pointing device. *The Quarterly Journal of Experimental Psychology 48A*, 1 (1995), 55–71.

4. MASSIE, T. H. Initial haptic explorations with the phantom: Virtual touch through pointer interaction. Master's thesis, Massachusetts Institute of Technology, February 1996.

5. MÜNCH, S., AND STANGENBERG, M. Intelligent control for haptic displays. *Computer Graphics Forum 15*, 3 (September 1996), C217–C226. Proceedings of EUROGRAPHICS'96.

6. ROSENBERG, L., AND BRAVE, S. Using force feedback to enhance human performance in graphical user interfaces. In *CHI 96* (April 1996), ACM SIGCHI, pp. 291–292.

7. ROSENBERG, L. B. FEELit mouse: Adding a realistic sense of FEEL to the computing experience. `http://www.force-feedback.com/feelit/white-paper.html`, October 1997.

8. ROSENBERG, L. B., AND BRAVE, S. The use of force feedback to enhance graphical user interfaces. In *Stereoscopic Displays and Virtual Reality Systems III* (1996), M. T. Bolas, S. S. Fisher, and J. O. Merritt, Eds., pp. 243–248. proc SPIE 2653.

9. SALISBURY, K., BROCK, D., MASSIE, T., SWARUP, N., AND ZILES, C. Haptic rendering: Programming touch interaction with virtual objects. In *1995 Symposium on Interactive 3D Graphics* (1995), ACM SIGGRAPH, pp. 123–130.

10. SCHEIFLER, R. W., AND GETTYS, J. *X Window System*, third ed. Digital Press, Burlington, MA, 1992.