Exposing and Understanding Scrolling Transfer Functions

Philip Quinn[1] **Andy Cockburn**[1] **Géry Casiez**[2,3,4] **Nicolas Roussel**[3] **Carl Gutwin**[5]
[1]University of Canterbury
Christchurch, New Zealand
philip.quinn@canterbury.ac.nz
andy@cosc.canterbury.ac.nz
[2]LIFL, [3]INRIA Lille &
[4]University of Lille
Villeneuve d'Ascq, France
gery.casiez@lifl.fr
nicolas.roussel@inria.fr
[5]University of Saskatchewan
Saskatoon, Canada
gutwin@cs.usask.ca

ABSTRACT
Scrolling is controlled through many forms of input devices, such as mouse wheels, trackpad gestures, arrow keys, and joysticks. Performance with these devices can be adjusted by introducing variable transfer functions to alter the range of expressible speed, precision, and sensitivity. However, existing transfer functions are typically "black boxes" bundled into proprietary operating systems and drivers. This presents three problems for researchers: (1) a lack of knowledge about the current state of the field; (2) a difficulty in replicating research that uses scrolling devices; and (3) a potential experimental confound when evaluating scrolling devices and techniques. These three problems are caused by gaps in researchers' knowledge about what device and movement factors are important for scrolling transfer functions, and about how existing devices and drivers use these factors. We fill these knowledge gaps with a framework of transfer function factors for scrolling, and a method for analysing proprietary transfer functions—demonstrating how state of the art commercial devices accommodate some of the human control phenomena observed in prior studies.

Author Keywords
Control-display gain; scrolling; scroll acceleration; transfer functions.

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: User interfaces – Input devices and strategies.

INTRODUCTION
Scrolling is an essential task in modern computing, and scrolling devices such as mouse wheels and trackpad gestures are ubiquitous. A fundamental element of scroll control that all techniques must address is the *transfer function* that maps the user's actions with the input device (for example, degrees of rotation, millimetres of displacement, or newtons of force) into scrolling movement of the display (typically either pixels, lines, or pages). However, there has been surprisingly little public research on scrolling transfer functions. Notable exceptions include Hinckley et al. [15] and Cockburn et al. [11], but even these studies are ambiguous about the exact functions used or tested—for example, Hinckley et al. stated "We tested the device using the manufacturer's default settings", but precisely determining the corresponding transfer function is now near impossible; Cockburn et al. explicitly acknowledged the need for further research to understand the role that the system transfer function may have played in their experiment.

The poor understanding of scrolling transfer functions creates several problems for researchers. First, existing methods are unknown because they are embedded in 'black box' driver code, making it difficult for researchers to understand the cause of performance differences between devices, or to iteratively improve on the state of the art. Second, replication of scrolling studies is frustrated by ambiguities in experimental settings—researchers lack the tools to examine, report, and replicate transfer functions. Third, researchers may inadvertently introduce confounds into experiments stemming from unknown interactions between particular transfer functions and their experimental treatment.

Casiez and Roussel [8] recently observed similar problems of "bricolage" in research treatment of pointing transfer functions. To address the problem, they developed an electronic device called *EchoMouse* to probe and inspect transfer functions. They also created a software library called *libpointing* that implemented these functions for experimental replication. They used these components to simulate human mouse control at a variety of physical input movement speeds, and to inspect the resultant system response.

Although *EchoMouse* and *libpointing* provide critical hints on how to examine scrolling transfer functions, the mapping from input actions to output effects is more complex for scrolling. While pointing transfer functions attend to two parameters (mouse velocity and user setting), scrolling functions are likely to attend to many more. Multiple parameters are necessary (or advisable) because of the paucity of scrolling input mechanics. For pointing, a modern mouse will register thousands of points per inch, and it can be moved across a two-dimensional area of several inches without clutching (physically disengaging input control in order to reposition a limb to repeat the action); in contrast, a typical scroll wheel can only be moved through five or six detents/notches across ~40° of one-dimensional rotation

UIST'12, October 7–10, 2012, Cambridge, Massachusetts, USA.

between clutching actions; and the muscle groups used to control wheel rotation (finger extensors and contractors) are likely to induce different control capabilities across scroll directions. Scrolling transfer functions, therefore, are likely to attend to input parameters that include the rate of device movement, time between clutched repetitions, scroll direction, and more. But whether they do this, and how they do it, is currently unknown.

These physical and operational characteristics of devices, and the human capabilities when operating them, have clear implications for the design of scrolling transfer functions. To help understand these issues, the following section presents a framework of the factors influencing scroll control. We then reverse engineer the scrolling transfer functions in state of the art commercial scroll drivers, and confirm that some drivers attend to many input parameters, while others are based purely on the velocity of input control.

FACTORS INFLUENCING SCROLL CONTROL

There has been extensive prior work on taxonomies that aid in understanding the design space of input devices, which we draw on to organise the physical characteristics of scrolling devices. Buxton's [4] early taxonomy organised devices by their physical properties—position, motion, or pressure—and by the number of dimensions along those properties that are sensed. Mackinlay et al. [20] and Card et al. [6, 7] expanded this into a *morphological* analysis, placing devices as points in a parametrically described design space that included the eight combinations of linear/rotary, absolute/relative, and position/force across six linear and rotational dimensions. They also composed chains of connections between the physical parameters and the semantics of an application. Buxton [5] and Hinckley and Sinclair [16] expanded this classification to include devices that operate by touch (rather than a mechanical control), and Lipscomb and Pique [19] added several dimensions of physical device characteristics (including the behaviour of the movement axes, bounds of movement, and self-zeroing behaviour).

These taxonomies can be used to classify the physical sensing properties of scrolling devices; for example, that mouse wheels are single-axis rotary controls that sense discretised changes in rotation, or that trackpads sense absolute one or two-dimensional position. They can also classify the features of the physical controls used to input these properties; for example, mouse wheels can rotate in rigid, discrete detents, or the detents can be soft and the wheel can be inertial (supplying sensor data without active user interaction). These design choices promote different methods of interacting with the device in different scrolling scenarios (for instance, rapid clutching on a discrete wheel vs. flicking and inertial one), and consequently influence the range and type of inputs that are likely to be received. However, the input parameters that are derived from these different methods of interaction are not captured by the above taxonomies.

This section presents a framework for the factors influencing scrolling behaviour and prior scrolling research. The framework is organised across considerations of input parameters, a review of the system-oriented view of scrolling, and prior studies of scrolling gain.

Input Parameters

While these taxonomies organise the physical characteristics of scrolling devices, they do not focus on how features of the manipulation can be translated into task-specific semantics.

Despite the apparent simplicity of scrolling as unidimensional translation, there exists a broad variety of devices to support it, each of which may use a multitude of input parameters for inferring the user's scrolling intention. For example, a mouse wheel senses discretised rotary motion, but the user's intention may be inferred from any combination of the following: (1) the degrees of rotation; (2) the speed of rotation; (3) the rate of change in the speed of rotation; (4) the duration of interaction; (5) the direction of interaction; and (6) the period of interaction.

In general, actions performed on a device need to be mapped from a physical manipulation to an interface command. In doing so, several *input parameters* can be considered, increasing expressivity. For example, keyboard arrow buttons are a one-dimensional discrete control, but their use may be interpreted through continuous parameters such as duration of activation or the rate of repetition. These additional input channels can be classified into three types: measures of *instantaneous action*, measures of *action duration*, and *cumulative/relative measures*, described below.

Instantaneous measures. Measures of instantaneous action are the physical properties that are sensed by a device or their derivatives from samples over time. For example, spinning a detented mouse wheel produces discrete events of rotational movement; however, sampling several events produces new input parameters of angular velocity, acceleration, and higher-order derivatives.

These measures may alter the interpretation or mapping of the original property. For example, increasing mouse wheel velocity or acceleration may be used to increase the magnitude of scrolling events generated from the input of each wheel event, or may be used as a signal to switch between scrolling modes (such as between line and page-scrolling).

Action duration. The duration that an input is maintained (or is absent) can also serve as an input to a transfer function. For example, if a key or button is held down for more than a certain duration, it may start issuing repeating events at an increasing rate. Similarly, if scrolling velocity is maintained above a certain level, then gain might increase with the assumption that the user wants to travel a long distance.

Cumulative and relative inputs. Input measures may also have a memory of prior actions to determine resultant effects. For example, Hinckley and Cutrell [17] described a scrolling transfer function that cumulatively adds gain across rapidly repeated wheel rotations in the same direction; the cumulative effect is cancelled if the user pauses too long or reverses direction. Similarly, rate-based scrolling controls scroll velocity using the relative position of the input device with respect to an anchor point set at the action's initiation.

The System's Perspective

Figure 1 depicts the conceptual transformations that form a transfer function's behaviour in converting human action

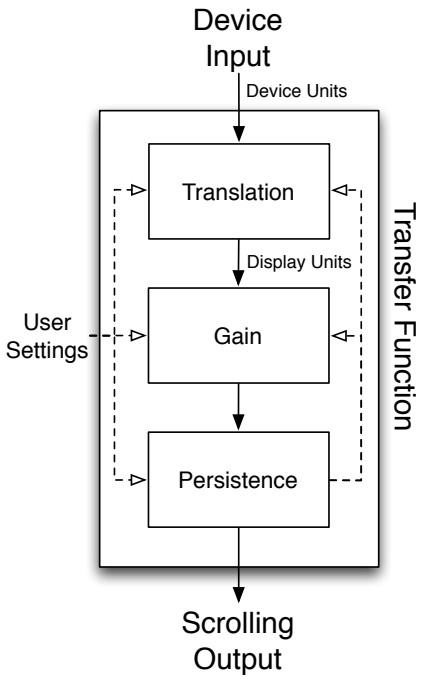


Figure 1: Depiction of the conceptual transformations occurring in a scrolling transfer function.

at the device into resultant display modification in scrolling. Not all of these transformations may be present in any transfer function: some may be absent, some may be combined, and some may be applied multiple times in different components (for example, gain applied by a driver and again by a UI toolkit or application).

Translation converts the device's physically registered events (degrees of rotation, newtons of force, millimetres of displacement, etc.) into units that are comprehensible to the system, such as pixels, lines, or pages. Users may be able to adjust this translation, either through controls on the device, or via a user interface on the system.

A *gain* function may then amplify or attenuate the control signal: for example, to allow slow precise control when the device is manipulated slowly, as well as accelerated scrolling when it is manipulated more aggressively. When gain is supported, users are commonly able to configure its setting.

Finally, a *persistence* component allows for a history of input and calculated parameters (such as input velocity and acceleration) to be preserved, or to allow for effects that are applied across time (such as cumulative effects, inertia, and simulated friction). Data from the persistence component can be used as input into the translation (e.g., allowing a switch from pixel to line scrolling if manipulation is continued for a threshold time), or into the gain (e.g., applying cumulative gain across repeated scroll wheel clutches). Finally, the user may be able to configure parameters of the persistence component (e.g., altering the degree of inertia or friction).

Most scrolling devices conform to the *Human Interface Devices* (HID) class of the USB standard [3]. The HID class provides a common, vendor-independent method for communicating interaction data from common types of devices to a computer system. Devices that implement the appropriate HID usage tables (for example, mice, keyboards, phones, and digitisers [2]) can operate without vendor-specific drivers, allowing a high degree of device/application interoperability.

HID devices report extensive descriptions of their sensing and reporting characteristics to the operating system/driver via HID *descriptors*. Of particular interest to scrolling are the wheel report range (typically 8-bit values interpreted to be between -127 and $+127$, but any size or range may be chosen by a manufacturer), the characteristics of the report (absolute/relative, wrapping/non-wrapping, linear/non-linear, etc.), and the rate at which reports are sent. The resolution and units of these reports can also be specified, but none of the devices we examined did so. System-specific extensions may also exist. For example, starting with Windows Vista, Microsoft allows devices to support horizontal scrolling and high-resolution scrolling by reporting a resolution multiplier and responding to queries from the operating system to configure it [21].

While most of the scrolling devices we examined supplied a 'Wheel' HID usage, notable exceptions to this were trackpads that used configurable gestures to enable a scrolling mode (for example, Apple's laptop trackpads and Magic Trackpad[1]). These devices transmit information about the gestures through proprietary data fields in the HID report, and rely upon manufacturer-specific drivers to interpret them and report scroll events to the operating system.

Despite the vendor-independent nature of the HID specification, drivers from device manufacturers may still play a significant role in defining the device's scrolling behaviour by attending to the input parameters discussed previously (and may be necessary to make exotic hardware that has not been anticipated by the HID usage tables useful at all—for example, trackpad scrolling gestures). Some of these features may include scrolling horizontally, independent transfer functions for each direction to match human capabilities, configurable buttons or gestures to augment or change scrolling behaviour/resolution, or different scrolling modes for each application. For example, the Logitech MX Revolution[2] features a weighted, low-fiction wheel that can have ratchets automatically engaged by the drivers as a user's scrolling behaviour changes.

Prior Studies of Scrolling Gain

In an early scroll wheel description, Gillick et al. [12, 13] described a potential transfer function that treats initial wheel events as line-scrolling, and advanced to page-scrolling once events passed a certain threshold-rate. A similar technique was recently described by Montalcini [22].

Hinckley et al. [15] describe a scroll transfer function that operates on the calculated interval between events received from the scrolling device/driver:

[1] `http://apple.com/magictrackpad/`
[2] `http://logitech.com/428/130`

$$\Delta y = K_1(1 + K_2 \Delta t)^{\alpha}$$

Where K_1, K_2, and α are constants, Δt is the interval between subsequent events, and Δy is the resulting scale factor to apply to reported magnitudes. Hinckley et al. evaluated their function when applied to a driver reporting three lines of scrolling per physical detent, one line per detent, a "standard" three lines per detent without application of the function, and an IBM ScrollPoint (isometric joystick; the configuration parameters of which are not reported) in a repeated tapping task. They found comparable or significantly better performance when using the accelerated functions.

Two further enhancements are detailed in related patents [17, and related continuity data]. One is a feature that detects changes in the scroll direction and temporarily inhibits the application of Δy—aiming to prevent amplification of overshooting errors. The other identifies rapidly repeated clutching of the wheel (in an attempt to travel a long distance) and applies cumulative gain according to the number of successive wheel flicks (N_{flicks}):

$$Z_{\text{scroll}} = \Delta y \cdot G_0 \cdot G_F \cdot N_{\text{flicks}}$$

Where G_0 is the baseline number of lines to scroll per detent, G_F is the additional amount of gain to apply per flick, and Z_{scroll} is the number of lines to scroll.

Kobayashi and Igarashi [18] explored the use of the cursor position as an input parameter to a dynamic transfer function. Their *MoreWheel* technique combines absolute and relative scrolling into the scroll wheel: dragging the mouse with the wheel depressed simulates grabbing the scroll thumb and enables absolute scrolling, while spinning the wheel produces either line or page-scrolling depending on the position of the cursor within the window (for example, line scrolling when the cursor is in the middle of the window, transitioning to page scrolling near the top or bottom edges).

Cockburn et al. [11] describe a method where two transfer functions are transitioned between based on the velocity of the scroll input to enable slow scrolling at a rate akin to Hinckley et al. [15], and rapid scrolling based on a function that utilises information about the length of the document being scrolled. The velocity of incoming scroll events is calculated, smoothed, and used to determine the proportion of each transfer function to apply:

$$g = \left[p \cdot \left(k_s - k_s \alpha^{-v}\right)\right] + \left[(1-p) \cdot \left(k_f \frac{\text{document length}}{\text{viewport size}}\right)\right]$$

Where k_s, k_f, and α are constants, v is the reported input velocity, p is the proportion of the "slow" function to apply (determined by examining the relationship of v to the user's maximum velocity), and g is the resulting scale factor to apply to reported magnitudes. An evaluation of this function using two wheel-based devices and an isometric joystick against the "additive flicking" technique of Hinckley and Cutrell [17] found it to perform significantly faster for long documents.

The above studies have explicitly examined the impact of scrolling transfer functions, but there are many more studies that have examined scrolling with imprecise and non-replicable transfer functions. This is not a criticism of the studies, but rather an unfortunate state of affairs—there has been a lack of tools supporting rigour around scrolling transfer functions. Some studies evaluate scrolling systems without mentioning the gain levels or transfer function used [e.g., 10, 14, 23]; some explicitly state the absence of acceleration, but do not state the constant translation used [e.g., 9]; and others rely on the default settings without stipulating what behaviour results [e.g., 1, 24].

REVERSE ENGINEERING CURRENT SCROLLING TRANSFER FUNCTIONS

To examine commercial scrolling transfer functions, we used a modified version of the *EchoMouse* [8]: a programmable microcontroller that allowed us to transmit scrolling events to the system in a controlled and systematic manner, emulating an ordinary scrolling device. To inspect the functions inside a particular device driver, the EchoMouse was modified to present itself as a compatible device from the appropriate manufacturer by manipulating its reported HID vendor and product identifiers. Therefore, by triggering the EchoMouse to emit scrolling events in a pre-defined pattern, and inspecting the scrolling events received by a user application, we can examine how the original events have been transformed.

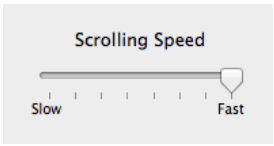
We tested the scrolling drivers found in Apple Mac OS X 10.7.3, Microsoft Windows 7 (SP1), Microsoft IntelliPoint (8.20.468 on Windows), Logitech SetPoint (driver 5.33.14 on Windows), and Logitech Control Center (3.5.1-23 on Mac OS X)—these represent some of the most popular operating systems and device manufacturers. With each driver, we impersonated the characteristics of several representative devices that they supported to gather data (testing low and high-resolution devices, although no differences between devices was found). The Mac OS X and Windows 7 drivers represent generic drivers that are used by the operating system when no vendor-specific drivers are available. We did not test the Mac OS X version of Microsoft's IntelliPoint driver as it conflicted with our EchoMouse control software, nor did we test an X11 environment as pilot testing showed that it (xorg 1.11.4-2; Fedora 16) does not implement scroll acceleration (the interpretation of each count is left to individual UI toolkits or applications).

This section presents an analysis of the publicly available information about these functions, followed by the testing methodology that we used to gather data about their embedded transfer functions.

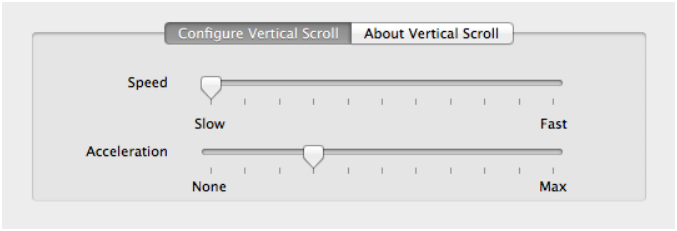
Analysis of Existing Transfer Functions

Apple Mac OS X. Apple release several of their low-level input processing frameworks under an open source licence, including those for HID devices. Within the `IOHIDFamily` framework[3] (version 368.20, corresponding to Mac OS X 10.7.3 was examined for this study), the `IOHIDFamily/IOHIDPointing.cpp` and `IOHIDSystem/IO-`

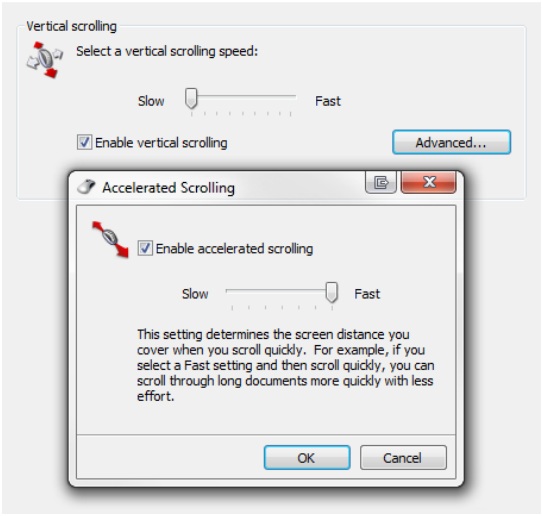
[3] `http://opensource.apple.com/source/IOHIDFamily/`



(a) Mac OS X.



(c) Logitech Control Center (Mac OS X).



(b) Microsoft IntelliPoint (Windows).

Figure 2: User interfaces for configuring scroll control.

`HIPointing.cpp` source files contain much of the code pertinent to pointing (and by extension, scrolling) devices.

Drivers can supply an encoded table of acceleration lines (slopes m and intercepts b) to be applied at different input magnitudes; these lines are scaled based on the user's scrolling speed setting (detailed below). When a scroll event is received with magnitude y, it is added to a smoothing window of the last eight events to avoid rapid changes in gain. The average time delta between events in the smoothing window $\overline{\Delta t}$ and average unaccelerated magnitudes $\overline{y}$ is then used to calculate a threshold:

$$l = \left[(K_a \cdot \overline{\Delta t}^2) - (K_b \cdot \overline{\Delta t}) + K_c\right] \cdot r \cdot \overline{y}$$

Where K_a, K_b, and K_c are constants, and r is an input rate multiplier (1 by default).[4] An acceleration line appropriate for an input magnitude larger than l is selected from the table, and applied:

$$y' = y \cdot \frac{b + (l \cdot m)}{|y|}$$

The control exposed to users for this function is a slider in the system preferences to manipulate "Scrolling Speed" with eight intervals from "Slow" to "Fast" (shown in Figure 2(a)), and a corresponding API (`IOHID[Get/Set]ScrollAcceleration()`, where the notches on the slider are mapped to the API values {0, 0.12, 0.31, 0.5, 0.69, 0.88, 1, 1.7}). An interesting feature of this control is that a negative value (which can only be selected via the API) completely disables scroll acceleration for generic devices, or engages a page-scrolling mode for an Apple trackpad.

Drivers can report scrolling events in units of either lines or pixels, with an automatic conversion by the system between them of 10 pixels per line. We report output in pixels to match the reports given by the system to user applications.

[4]Code also exists for scaling these functions with the screen resolution, but the calculations are currently fixed.

It should be stressed that the behaviour described above is the *default* that is applied should no better drivers match a connected device. Manufacturers are free to use and adjust the described behaviour in part, or as a whole. For example, Apple's closed-source driver for their laptop trackpads supplies a scrolling acceleration table that can be decoded with the source code provided, but that alone does not guarantee that it will be applied in the manner described above.

Microsoft Windows. Windows provides scrolling information to applications via `WM_MOUSEWHEEL` messages with a parameter indicating the distance the wheel has been rotated in units of `WHEEL_DELTA`. These values are intended to be scaled by the user setting `SPI_GETWHEELSCROLLLINES`, indicating how many lines to scroll per unit of `WHEEL_DELTA` (or, a special value indicating that each unit should be interpreted as a page scroll). The interpretation of "lines" or "pages" is left to the application receiving the message. On all current systems, `WHEEL_DELTA` is set to 120, which allows high-resolution devices to indicate scrolling of fractional lines.[5] We report the output from drivers running under Windows in "lines" (i.e. units of `WHEEL_DELTA`).

Microsoft IntelliPoint. Microsoft's IntelliPoint drivers for their branded devices presents two controls for the user to configure the scrolling transfer function (shown in Figure 2(b)). The first controls `SPI_GETWHEELSCROLLLINES` in the range [1, 40]; the second is a seven-interval slider to control accelerated scrolling from "slow" to "fast" (with an option to disable it entirely).

Logitech. Logitech produces two driver packages for their devices: *SetPoint* for Windows, and *Control Center* for Mac OS X. The configuration options and range of supported devices differs between these packages; in particular, *SetPoint* provides options to configure `SPI_GETWHEELSCROLLLINES` for line or page scrolling but with no options for acceleration,

[5]`http://msdn.microsoft.com/library/ms997498`

while *Control Center* allows customisation of the scrolling "speed" (from "slow" to "fast") and "acceleration" (from "none" to "max"), as shown in Figure 2(c) (both of these are continuous sliders, but were tested at the marked intervals).

Testing Methodology

Scrolling events from the EchoMouse have values in the range -127 to $+127$. We observed that reports from devices were typically either -1 (scroll down) or $+1$ (scroll up) with wider values used when the physical manipulation of the device exceeded its HID input report rate (typically 100~125Hz, but high-end devices may report at rates up to 1000Hz); we emulated this behaviour.

A potential issue when impersonating other devices is matching their input resolution. While the USB HID specification allows devices to specify the resolution and physical units of their input, none of the device we tested did so. For instance, a Microsoft Wheel Mouse Optical[6] sends 18 scroll counts per complete revolution of its wheel (20° per detent), while a Logitech MX500[7] sends 24 (15° per detent), but their reports are indistinguishable to a generic driver (similar issues exist for trackpads that transmit events corresponding to millimetres of displacement, or other types of physical control). Because we tested a range of devices with different input resolutions, we report our input velocity in "counts" per unit of time, where one count corresponds to one scroll event of magnitude -1 or $+1$ (issues surrounding device resolution are discussed later).

As we are interested in the various input parameters that transfer functions may attend to (and not only how they operate under levels of velocity), we performed four mechanised tests of each possible configuration of driver and device:

- *Constant velocity:* emulating a constant speed of device operation for five seconds, and measuring the resultant output scrolling velocity as an average over that period.
- *Maintained velocity:* emulating a constant speed of device operation for five seconds, and measuring the resultant output scrolling velocity for each event.
- *Clutching:* we emulated clutching actions, manipulating the speed of device operation, the duration of clutches, and the time between successive clutches.
- *Direction changes:* we emulated direction changes (alternating between scrolling up and scrolling down) while maintaining a constant speed of device operation.

These tests were repeated for each configuration option presented to users (as described above), and across a range of possible user control input rates. Custom software monitored the system's response using the low-level event reporting APIs provided by each operating system (free from potential manipulation by higher-level frameworks or toolkits).

RESULTS AND ANALYSIS

The results of our analyses are summarised in Table 1. We found that neither Microsoft Windows 7, nor Logitech's SetPoint drivers provide any scroll acceleration (the gain is always constant). Due to the large number of configurations tested and the many commonalities discovered between them, the following subsections present a survey of the most salient and interesting behaviour characteristics and parameters attended to (a complete spreadsheet of the acceleration tables collected is also available[8]). Following the main results, we summarise device-specific issues in the analysis.

	Velocity	Direction	Duration	Clutching
Apple Mac OS X	●	◐	○	◐
Microsoft Windows 7	○	○	○	○
Microsoft IntelliPoint	●	●	○	●
Logitech SetPoint	○	○	○	○
Logitech Control Center	●	◐	●	○

Table 1: Summary of the tested drivers' attendance to tested input features—○: no attendance, ●: attendance, ◐: partial attendance (details in text).

Gain with Respect to Velocity

How the different systems alter gain across input velocity is shown in Figures 3(a), (b), and (c) for Mac OS X, Microsoft IntelliPoint, and Logitech Control Center, respectively. The multiple lines in each figure show different levels of user setting for scrolling "speed" (Mac OS X, Figure 2(a)) or scroll "acceleration" (Microsoft IntelliPoint and Logitech Control Center, Figures 2(b) and (c)). The solid and dashed lines in Figure 3(b) differentiate between scrolling direction (up and down); the other drivers respond to both directions equally.

The maximum attainable scale factors range from ~14× with Mac OS X, to ~18× with Logitech, and ~21× with IntelliPoint. The key differences between the three curve shapes is that Logitech's curves for high acceleration show a dramatic drop in gain after peaking at 18× at ~28 counts/s. This is a result of a falling (but still positive) gradient in the output velocity curve; however the rationale for this design choice is unknown. Both Mac OS X and Logitech allow input to be attenuated (with a gain of less than 1) at low input speeds, increasing the expressivity of devices with poor resolution.

Direction as an Input

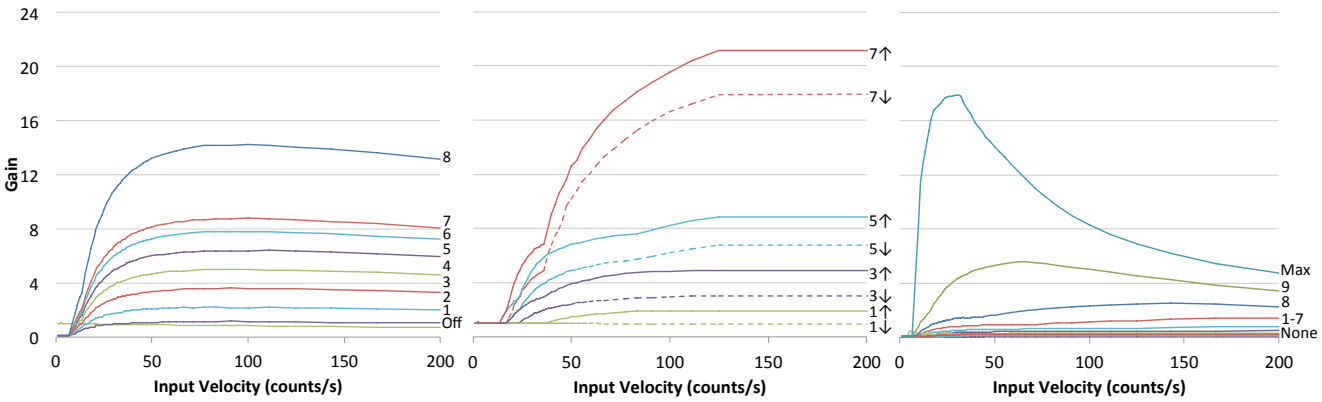
Figure 3(b) shows that Microsoft IntelliPoint drivers apply differing gain levels across each scrolling direction. This is probably applied to compensate for the differing maximum input velocities attainable in the two directions (Cockburn et al. [11] showed marked differences between maximum scroll wheel rotation speeds upwards and downwards).

Mac OS X and Logitech drivers do not vary gain across direction, but directional changes do momentarily "reset" gain, as shown for Mac OS X in Figure 6(a). This is due to a reset of the smoothing window upon direction change, resulting in reduced gain until the window is re-filled. The same task with Microsoft's IntelliPoint drivers is shown in Figure 6(b), where we observed a very brief drop in gain and the application of different levels of gain for each direction.

[6] `http://microsoft.com/hardware/en-nz/d/wheel-mouse-optical`

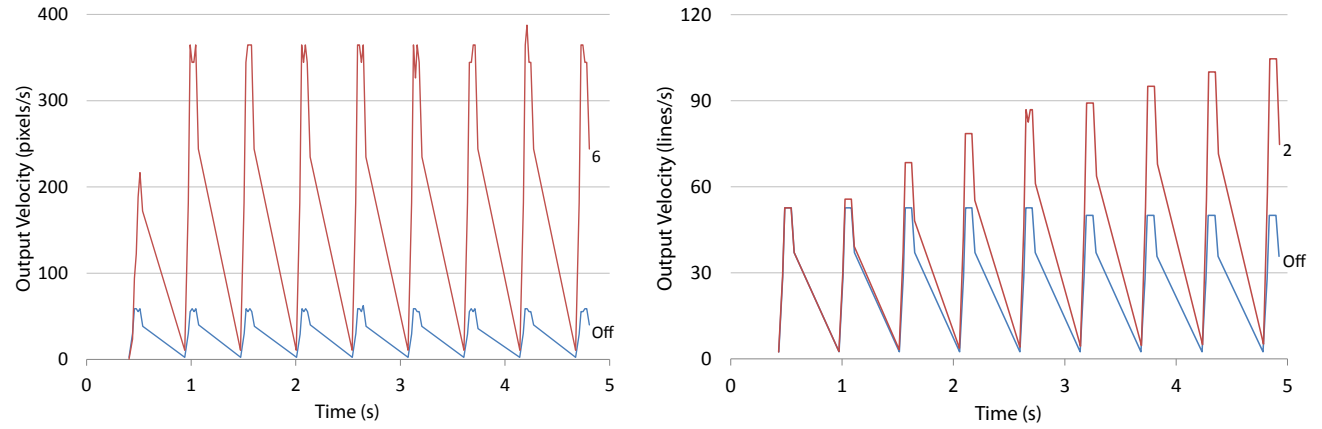
[7] `http://logitech.com/428/910`

[8] `http://cortex.p.gen.nz/research/scrolling/`



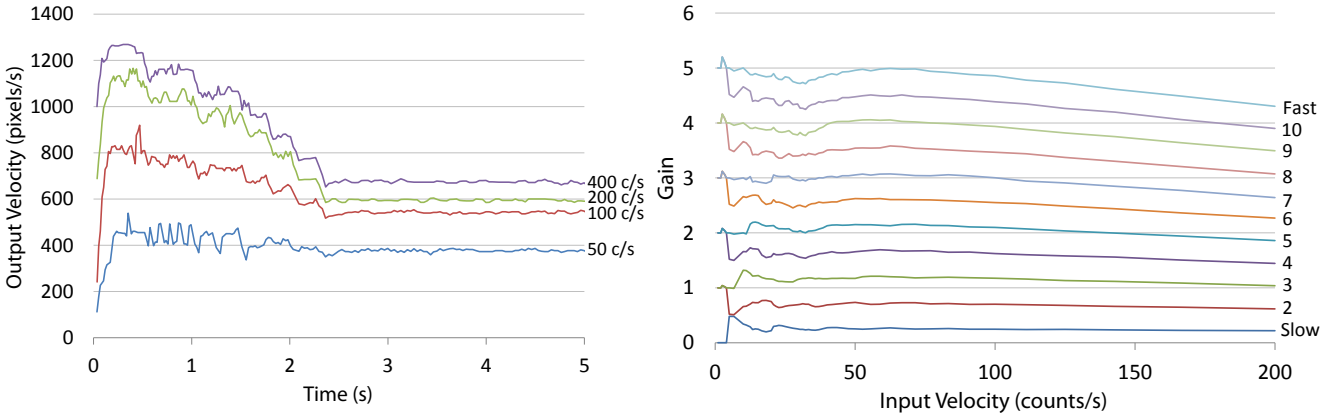
(a) Mac OS X (Generic Device). (b) Microsoft IntelliPoint: up (solid) and down (dashed). (c) Logitech Control Center.

Figure 3: Gain scale factors across input velocity (counts per second) with Mac OS X, Microsoft IntelliPoint (under Windows 7), and Logitech drivers under Mac OS X. Gain is measured as the level of amplification in the system's base unit (pixels per count for Mac OS X and Logitech; lines per count for Microsoft IntelliPoint), and is plotted at varying levels of each driver's respective UI sliders for acceleration.



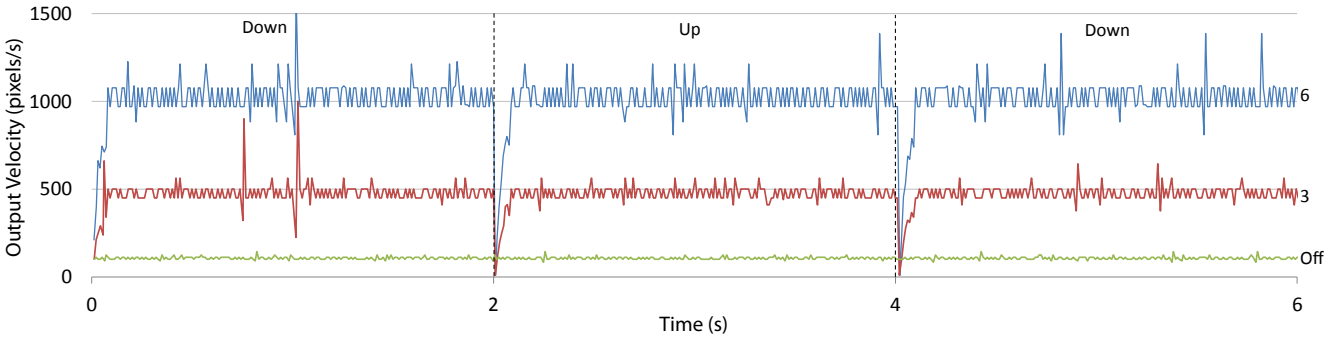
(a) Mac OS X (Generic Device). (b) Microsoft IntelliPoint.

Figure 4: Output velocity response to repeated clutching.

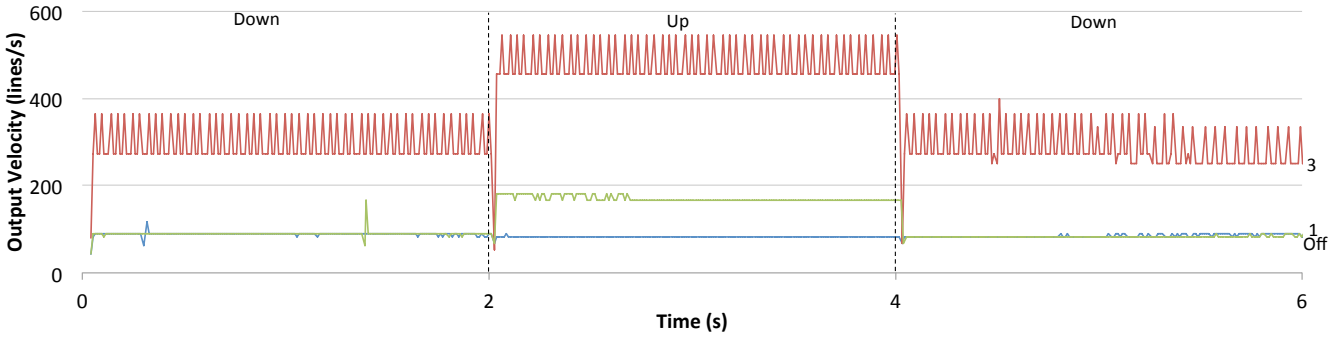


(a) Output velocity over time as a constant input velocity is maintained. (b) Gain across input velocity at levels of the "speed" slider shown in Figure 2(c).

Figure 5: Logitech's Control Center: Attendance to duration and the control of the "speed" slider.



(a) Mac OS X (Generic Device).



(b) Microsoft IntelliPoint.

Figure 6: Output velocity over time as a constant velocity is maintained but input direction is switched at the dashed lines (2 and 4s). Three curves—two levels of acceleration and one with acceleration disabled—are shown.

Clutching as an Input

Figure 4 shows how Mac OS X and Microsoft IntelliPoint drivers respond to clutching of the scroll wheel (Logitech's response is not shown as it is similar to Mac OS X). The main finding here is that IntelliPoint cumulatively adds gain across successive clutches when acceleration is turned on (similar to the technique described by Hinckley and Cutrell [17]). Mac OS X (Figure 4(a)) and Logitech drivers do not vary their response (the reduced gain for the first impulse is due to an empty smoothing window), however, comments in the Mac OS X driver source code[9] indicate the timeout value for resetting the smoothing window was chosen specifically to avoid doing so between clutches.

Duration as an Input

Figure 5(a) shows that Logitech's driver attends to scroll duration while Mac OS X and Microsoft IntelliPoint do not (not shown). When stimulated with a constant velocity scroll rate, Logitech's output velocity diminishes over approximately 2.5s. This time-based fall-off is particularly marked at high input velocities, and it is therefore likely designed to enhance user performance with Logitech's free-spinning inertial scroll wheels that readily allow high input speeds.

Other Features

It is interesting and potentially important that the speed and acceleration parameters in Logitech's Control Center interact with one another, and that setting acceleration to 'None' does not disable acceleration: Figure 5(b) shows the gain observed at various settings of the speed slider with the acceleration slider set to "None". The noise at low speeds is puzzling, and the lack of constant gain suggests that Logitech Control Center should be used with caution in research experiments.

[9] `IOHIDSystem/IOHIPointing.cpp`, lines 555–559

DISCUSSION

We have presented a framework for understanding the factors influencing the transformation of human action with scrolling devices (particularly scroll wheels) into resultant scrolling output. We have also reverse engineered the scrolling transfer functions from the drivers of popular manufacturers, with results demonstrating substantial variation in both the factors attended to and the manner in which they do so. Key observations include the fact that Microsoft's IntelliPoint drivers apply different levels of gain to each scrolling directions (presumably to accommodate differences in human mechanics), that they also apply cumulative gain across repeated clutching actions, and that Logitech's drivers on Mac OS X apply variable gain even when user settings stipulate that acceleration should be turned off.

The framework and findings have several implications for research that aims to develop new transfer functions or use scrolling devices in experimental conditions, and there are many avenues for further work.

Facilitating Rigour in Scrolling Studies

Scrolling researchers are typically interested in either new input devices [e.g., 24], transfer functions [e.g., 11, 15], or new interactive techniques [e.g., 18]. Scrolling devices are

also used in experiments where scrolling is not the focus of an investigation, but as an interaction tool.

In all cases, comparative evaluations are normally conducted to measure performance over the state of the art, and the choices made in the implementation and administration of experimental treatments must be made with the awareness and knowledge of the underlying transfer functions. As experimental software typically operates on-top of existing drivers (rather than replacing them), understanding the interaction between the transfer function of the driver and that of the experimental condition is critical in answering the question of whether the treatment is causing any observed difference, or whether it may be attributed to the interacting transfer functions.

What should researchers do to maximise rigour and facilitate replication? We make three recommendations. First, a constant level of gain (i.e. scroll acceleration is disabled) should be used as a baseline in experiments where gain is not intended to be a factor. Given the complex nature of the gain functions observed in our results, their variation across drivers, and their potential volatility across different versions from the same manufacturer, replicating non-constant gain settings across experiments may be extremely difficult. Note that using constant gain also means that there should be no adaptive *translation*, such as a transition from line to page-based scrolling units reported by Gillick et al. [12, 13] (we are unaware of commercial drivers that do so, however some Logitech mice have a mechanical switch that the driver can activate to transition the wheel from detented to free-spinning when a threshold scroll velocity is exceeded—altering the possible range and behaviour of a user's input).

Second, user settings for disabling acceleration should be treated with suspicion, and researchers should check carefully whether acceleration is actually disabled. Ideally, an inspection similar to that described in this paper should be conducted, but otherwise, researchers should avoid drivers that are known to exhibit non-constant velocity scale factors (e.g., Logitech's drivers under Mac OS X, as reported here). Consequently, when gain is disabled, how this was achieved (user settings, API calls, etc.) should be reported.

The third recommendation is to report details of the transfer function, described next.

Reporting Scrolling Transfer Functions

In reporting a transfer function, there are two components that deserve attention: the *translation* and the *gain*.

The translation concerns the device and display resolutions, the level of action required to generate a scroll event on the device, and the magnitude of those events (in display units). The device resolution considers the number of events reported per unit of physical action: for example, Logitech's MX500 mouse reports 24 events per complete wheel rotation, while Microsoft's Wheel Mouse Optical reports 18 (i.e. a complete revolution of the MX500 is equivalent to 1.33 revolutions of the Wheel Mouse Optical). The level of action required to generate a scroll event considers the lower-bounds of physical action generating scroll events: in a scroll wheel, this would be the minimum wheel rotation and resistance to generate a scroll event, but for trackpad scroll gestures it could concern the minimum velocity of movement, or the minimum total displacement. Finally, the event magnitude considers the number of pixels, lines, or pages that the minimum scroll event moves (which may be fractional with high resolution devices [e.g., 21]).

Where a constant gain is used, the scale factor between the event magnitudes from the translation component should be reported. Where non-constant gain is used, the gain function(s) should be described using formulas, figures, and/or tables with the mapping between the output of the translation component to the final scroll behaviour. In both cases, the mechanics of the device and input/output resolutions should be reported.

Limitations and Further Work

Most of the gain functions analysed in this paper were reverse engineered without source code. It is therefore possible that our descriptions of the functions are incomplete because we failed to probe a salient input parameter: for example, we did not probe for attendance to acceleration or jerk (the derivative of acceleration). Similarly, we analysed the data that is sent from the operating system to user applications, and did not consider possible manipulation of that data by applications or the frameworks/libraries they are built upon. These higher-level systems have access to information about the information space being navigated (for example, the document length [11]) that may be used to augment scrolling behaviour. We have, however, presented a framework for understanding that such parameters could influence behaviour, and a method for inspecting their impact if required.

Our attempt to reverse engineer the behaviour of Microsoft's IntelliPoint driver under Mac OS X failed because the driver prevented us from communicating with the EchoMouse. There are two hardware solutions to this problem: either EchoMouse could be engineered to store a pre-programmed set of signals to be emulated, or it could be designed to support a second USB input (one for sending signals to EchoMouse, and the other for sending messages to the driver).

None of the devices we tested supplied information about their physical units or resolution of input. Our analysis therefore used "counts" rather than physical units (such as degrees). Similarly, to our knowledge, drivers currently do not consider the display resolution when calculating gain (e.g., pixel size, pixel density, and/or scaling factors); but as higher resolution devices and displays become available, there are opportunities for investigating how transfer functions can be better designed to adapt to different input and output resolutions—for example, the interaction between user performance and input resolution (both device resolution, and human capabilities), and similarly for the output resolution (adapting to different display configurations).

There are also other types of scrolling hardware that have not been examined here, most notably trackpads and touch mice (devices that feature a touch-sensitive surface). Some of these devices feature transfer functions that enable features such as simulated momentum and friction when scrolling.

CONCLUSIONS

Scrolling is an elemental interface control, and system transfer functions are fundamental in determining their behaviour. Yet despite their importance, scrolling transfer functions have received little research attention. This paper examined how scrolling transfer functions work and the input parameters they attend to. We described a method to reverse engineer the state of the art in scrolling transfer functions, and we used the method to expose how systems vary with the input parameters they attend to and in their processing of these parameters. As well as providing a firmer foundation for research into improving scrolling transfer functions, the paper's findings also suggest that when evaluating scrolling techniques, researchers should be cautious about potential interactions between the system transfer function and experimental treatment. The method proposed allows system transfer functions to be precisely recorded, aiding experimental replication. In further work, we will examine gesture-based transfer functions on touchscreens and trackpads, and compare user performance with different functions.

ACKNOWLEDGEMENTS

This work was supported by a Royal Society of New Zealand Marsden Grant 10-UOC-020.

REFERENCES

1. Arthur, K. W., Matic, N., and Ausbeck, P. Evaluating touch gestures for scrolling on notebook computers. In *Proc. CHI EA '08*, ACM (New York, NY, USA, 2008), 2943–2948.
2. Bates, B. M., Dezmelyk, R., Ingman, R., Lieb, R., McGowan, S., Ray, K., Schumacher, S., Sherman, N. C., Stern, D., van Flander, M., and Zimmerman, R. HID usage tables, Version 1.12. USB Implementer's Forum (2004).
3. Bergman, M., Peuranch, T., Schmidt, T., McGowan, S., Crowe, J., Dezmelyk, R., Zimmerman, R., van Flandern, M., Nathan, B., Davis, M., and Rayhawk, J. Device class definition for human interface devices (HID), Version 1.11. USB Implementer's Forum (June 2001).
4. Buxton, W. Lexical and pragmatic considerations of input structures. *SIGGRAPH Computer Graphics 17*, 1 (January 1983), 31–37.
5. Buxton, W. Touch, gesture, and marking. In *Human-Computer Interaction: Toward the Year 2000*, R. M. Baecker, J. Grudin, W. Buxton, and S. Greenberg, Eds. Morgan Kaufmann Publishers, San Francisco, 1995, ch. 7, 469–482.
6. Card, S. K., Mackinlay, J. D., and Robertson, G. G. The design space of input devices. In *Proc. CHI '90*, ACM (New York, NY, USA, 1990), 117–124.
7. Card, S. K., Mackinlay, J. D., and Robertson, G. G. A morphological analysis of the design space of input devices. *ACM Transactions on Information Systems 9*, 2 (Apr. 1991), 99–122.
8. Casiez, G., and Roussel, N. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proc. UIST '11*, ACM (New York, NY, USA, 2011), 603–614.
9. Chipman, L. E., Bederson, B. B., and Golbeck, J. A. Slidebar: analysis of a linear input device. *Behaviour & Information Technology 23*, 1 (2004), 1–9.
10. Cockburn, A., and Gutwin, C. A predictive model of human performance with scrolling and hierarchial lists. *Human-Computer Interaction 24*, 3 (2009), 273–314.
11. Cockburn, A., Quinn, P., Gutwin, C., and Fitchett, S. Improving scrolling devices with document length dependent gain. In *Proc. CHI '12*, ACM (New York, NY, USA, 2012), 267–276.
12. Gillick, W. G., and Lam, C. C., U.S. Patent No. 5530455. U.S. Patent and Trademark Office (Washington, DC, 1996).
13. Gillick, W. G., and Rosenberg, R. A., U.S. Patent No. 5446481. U.S. Patent and Trademark Office (Washington, DC, August 1995).
14. Gutwin, C., and Cockburn, A. Improving list revisitation with listmaps. In *Proc. AVI '06*, ACM Press (New York, NY, USA, 2006), 396–403.
15. Hinckley, K., Cutrell, E., Bathiche, S., and Muss, T. Quantitative analysis of scrolling techniques. In *Proc. CHI '02*, ACM (New York, NY, USA, 2002), 65–72.
16. Hinckley, K., and Sinclair, M. Touch-sensing input devices. In *Proc. CHI '99*, ACM (New York, NY, USA, 1999), 223–230.
17. Hinckley, K. P., and Cutrell, E. B., U.S. Patent No. 7173637. U.S. Patent and Trademark Office (Washington, DC, 2007).
18. Kobayashi, M., and Igarashi, T. MoreWheel: Multimode scroll-wheeling depending on the cursor location. In *UIST 2006 Adjunct Proceedings: Demonstrations* (2006), 57–58.
19. Lipscomb, J. S., and Pique, M. E. Analog input device physical characteristics. *SIGCHI Bulletin 25*, 3 (July 1993), 40–45.
20. Mackinlay, J., Card, S. K., and Robertson, G. G. A semantic analysis of the design space of input devices. *Human-Computer Interaction 5*, 2–3 (1990), 145–190.
21. Microsoft Corporation. Enhanced wheel support in windows. Tech. rep., 2010.
22. Montalcini, A. L., U.S. Patent No. 7661072. U.S. Patent and Trademark Office (Washington, DC, 2010).
23. Wherry, E. Scroll ring performance evaluation. In *CHI EA '03*, ACM (New York, NY, USA, 2003), 758–759.
24. Zhai, S., Smith, B. A., and Selker, T. Improving browsing performance: A study of four input devices for scrolling and pointing tasks. In *Proc. INTERACT '97*, Chapman & Hall, Ltd. (London, UK, 1997), 286–293.