HAL
open science

The design, use, and performance of edge-scrolling techniques

Jonathan Aceituno, Sylvain Malacria, Philip Quinn, Nicolas Roussel, Andy Cockburn, Géry Casiez

To cite this version:

Jonathan Aceituno, Sylvain Malacria, Philip Quinn, Nicolas Roussel, Andy Cockburn, et al.. The design, use, and performance of edge-scrolling techniques. International Journal of Human-Computer Studies, 2017, 97, pp.58 - 76. 10.1016/j.ijhcs.2016.08.001 . hal-01624625

HAL Id: hal-01624625
https://hal.science/hal-01624625v1

Submitted on 2 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The design, use, and performance of edge-scrolling techniques

J. Aceituno[a,b,*], S. Malacria[a,c], P. Quinn[c], N. Roussel[a], A. Cockburn[c], G. Casiez[b]

[a] *Inria Lille - Nord Europe, 40 avenue Halley, 59650 Villeneuve d'Ascq, France*
[b] *Université Lille 1, Cité Scientifique, 59650 Villeneuve d'Ascq, France*
[c] *Department of Computer Science and Software Engineering, University of Canterbury, Private Bag 4800, Christchurch, New Zealand*

Abstract

Edge-scrolling techniques automatically scroll the viewport when the user points near its edge, enabling users to reach out-of-view targets during activities such as selection or drag-and-drop. Despite the prevalence of edge-scrolling techniques in desktop interfaces, there is little public research on their behaviour, use or performance. We present a conceptual framework of factors influencing their design. We then analyse 19 different desktop implementations of edge-scrolling by reverse-engineering their behaviour, and demonstrate substantial variance in their design approaches. Results of an interactive survey with 214 participants show that edge-scrolling is widely used and valued, but also that users encounter problems with control and with behavioural inconsistencies. Finally, we report results of a controlled experiment comparing four different implementations of edge-scrolling, which highlight factors from the design space that contribute to substantial differences in performance, overshooting, and perceived workload.

Keywords: edge-scrolling, scrolling, autoscroll, dragging

1. Introduction

Edge-scrolling is an interactive technique that automatically scrolls a viewport when the user points near its edge. It is typically used when other scrolling tools are unavailable

*Corresponding author
Email addresses: `join@oin.name` (J. Aceituno), `sylvain.malacria@inria.fr` (S. Malacria), `philip.quinn@canterbury.ac.nz` (P. Quinn), `nicolas.roussel@inria.fr` (N. Roussel), `andy@cosc.canterbury.ac.nz` (A. Cockburn), `gery.casiez@univ-lille1.fr` (G. Casiez)

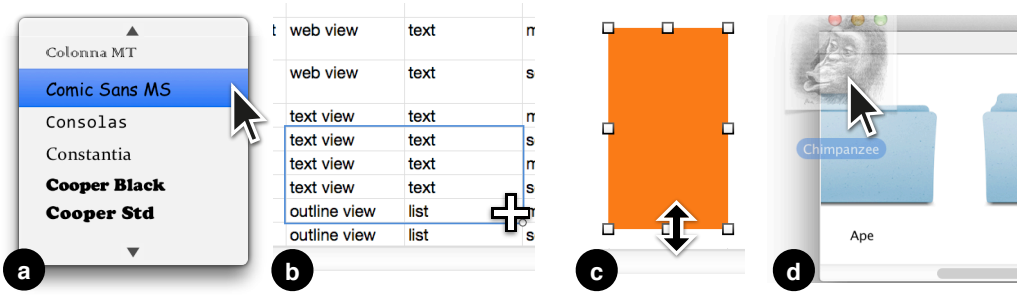

Figure 1: Situations commonly encountered in desktop applications where *edge-scrolling* can be employed to automatically scroll a viewport to complete a task: (a) selecting a font in a large menu, (b) selecting a wide range of cells in a spreadsheet, (c) resizing a shape in a drawing application, and (d) moving a file in a subdirectory.

or unsuitable, for example when the user is already engaged in a dragging action. Figure 1 shows several of these situations commonly encountered in desktop applications: (a) selecting a font in a large menu, (b) selecting a wide range of cells in a spreadsheet application, (c) resizing a shape in a drawing application, and (d) moving a file in a subdirectory. In these examples, the user has their pointer near the edge of the viewport and needs to scroll the view to complete their task; edge-scrolling assists by automatically scrolling the view at a certain speed in the desired direction.

Edge-scrolling is used daily by millions of desktop computer users, and is available in most desktop applications. However, there are substantial differences between implementations. For example, a Windows 8 user will experience different behaviours when selecting text in Internet Explorer and in Notepad: in the former, the viewport automatically scrolls while the pointer remains below its bottom edge, but the latter additionally requires continual pointer movement for scrolling to occur. Many other behaviours are displayed both within and between operating systems. These differences are near certain to have important implications for user performance, errors, and preferences, as suggested by results of prior studies into alternative techniques for general scrolling [e.g., 1, 2, 3]. Yet, despite the ubiquity of edge-scrolling use and the seemingly ad-hoc variance of the available implementations, there has been little research to understand how users interact with edge-scrolling and how it should be designed.

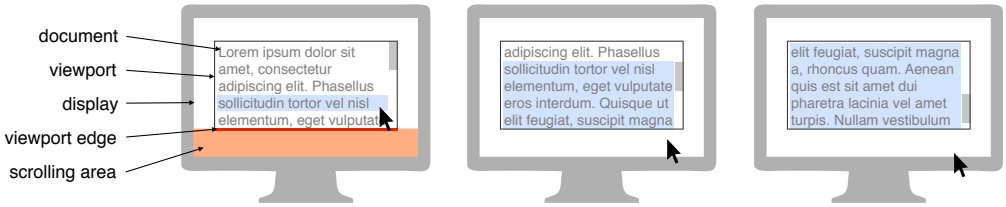

Figure 2: Edge-scrolling is used in the course of a task (here, selecting text) in order to scroll a *document* that is displayed inside a *viewport* placed on the *display*. The user activates edge-scrolling by entering a *scrolling area* near the *viewport edge* (left). Inside the scrolling area, pointing actions control the pace of scrolling (middle, right).

To advance understanding and design of edge-scrolling techniques, we present an analysis in four parts, respectively addressing the following questions: (1) what factors influence the design of edge-scrolling techniques; (2) exactly how do current implementations behave; (3) how are existing techniques used and perceived by users; and (4) how is user performance influenced by different edge-scrolling behaviours? After a review of related literature, we answer these questions by first providing a framework for understanding edge-scrolling techniques based on task requirements and the method's behavioural characteristics. We then use this framework to examine 33 existing edge-scrolling implementations, reverse-engineering their behaviour and demonstrating substantial variance in design approaches. Results of an interactive online survey, completed by 214 participants, confirm that edge-scrolling is widely used, and identifies common usability problems. We then describe a controlled experiment evaluating user performance when using various edge-scrolling designs for a text selection task, with results showing that different approaches to supporting edge-scrolling cause substantial differences in task completion time and perceived workload. Finally, from these results, we derive a series of recommendations for the design of future edge-scrolling techniques.

2. Background

When a *document* has more information than can be displayed inside a *viewport* placed on the *display*, the interface must allow scrolling so that relevant document parts can be brought into view. Various scrolling techniques and devices have received much attention

in the literature, but it seems to us that academic research has missed an opportunity to formalise understanding on an important interaction that is widely disseminated yet problematic. Edge-scrolling defines a *scrolling area* relative to the *viewport edge* (Figure 2), and when the pointer is inside the scrolling area, pointing actions control the scrolling rate.

This section reviews related literature relevant to edge-scrolling, including general scrolling techniques and transfer functions, pointing-based scrolling techniques, edge-based pointing techniques, as well as edge-scrolling disclosures.

2.1. Scrolling techniques and transfer functions

Scrolling has been a basic component of interaction since the earliest graphical interfaces [4], and a wide array of input devices and techniques have been used to support it. Widely deployed hardware devices with scrolling adaptations include the keyboard [5], touchpad [6, 7], and mouse [8, 9, 10]. Desktop and touch-enabled systems also commonly support scrolling methods that are based on pointing actions, such as scrollbar controls, middle-click rate-based scrolling [9], and touch-based flick-scrolling [11, 12]. Other forms of input for scrolling control include gaze [13], the position of a mobile device in 3D space [14], and device orientation [15].

All scrolling techniques require a *transfer function* that determines the mapping from the input signals to the resultant scrolling output, which is typically in terms of either the *absolute* position of the document, its *relative* position, or its *rate* of movement (although other variants are theoretically possible, such as document acceleration). The following subsections review work on these three primary forms of control. We use the conceptual model of transfer functions provided by Quinn et al. [3], which describes three successive transformations: *translation* (where the input signal is converted to display space units), *gain* (which modifies the amplitude of the control signal), and *persistence* (where effects, such as simulated friction, are applied across time).

Figure 3 provides an overview of relationships between literature reviewed in this section.

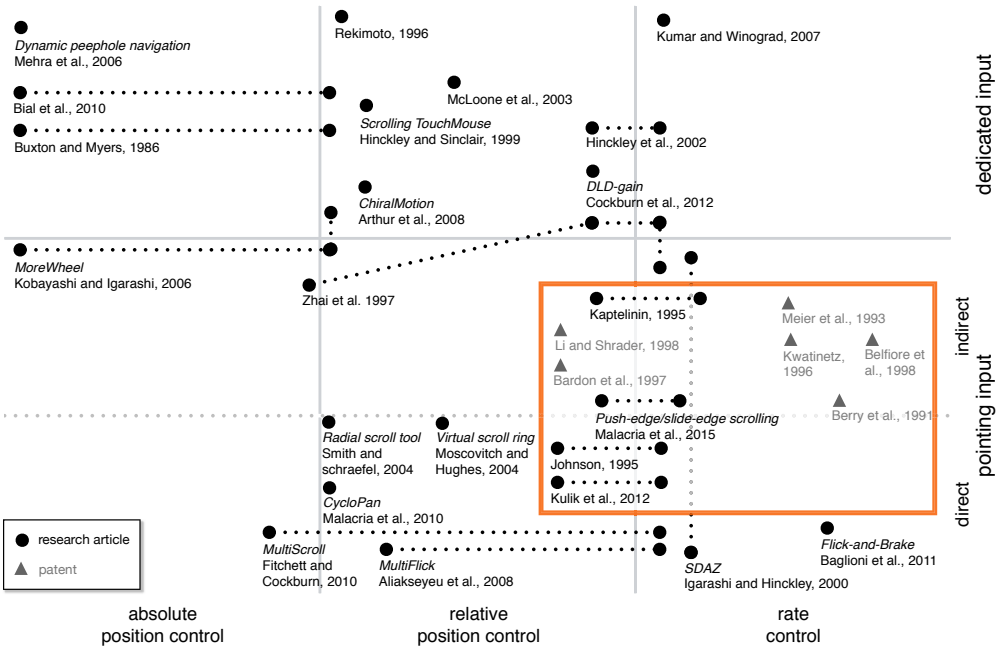

Figure 3: Overview of scrolling techniques and studies from the literature discussed in this section, by type of input (pointing or dedicated input) and transfer function (absolute/relative position control or rate control). Grey triangles are patents. The orange frame delimits edge-scrolling, a pointing-based family of scrolling techniques that is the focus of this work.

2.1.1. Absolute position control

Absolute position control functions map each point inside an area in a physical or virtual coordinate space to a single position of the viewport in document coordinates—they map an input position to a scroll position. Such functions have been used in dedicated devices [16, 7] and on touchscreens [17]. They allow users to quickly jump to remote parts of a document, but precise selections and short-distance movements are compromised [17].

Precision problems occur at the *translation* stage of the transfer function, either because the number of sampled elements of the input space does not match the number of elements in the document space, or because the transfer function itself does not consider the whole input resolution [18]. When using pointing input, this granularity problem can be solved with zooming techniques [19], or by increasing pointer precision beyond the pixel barrier [18].

2.1.2. Relative position control

Relative position control offers an articulatory direct [20] mapping of the user's movements to scrolling displacements: input movement causes document movement. Although intended scrolling movements may range from a few pixels to hundreds of lines of text [2], the physical or virtual footprint available for input is limited. Therefore, in long distance scrolling, the user must frequently disengage then re-engage the device (*clutch*) to compensate the limited input area. Repeated clutching typically compromises the user's ability to maintain a steady scrolling speed (for example, when reading) [1], can increase scrolling time [21], and requires more physical effort [2], but clutching can also be performed intentionally as part of a user strategy to optimise device movements [22].

Improvements to scrolling performance and undesired clutching exist in each of the three transformations composing the transfer function. At the *translation* stage, mapping circular motion to scrolling [23, 24, 6] removes the need to clutch, but these mappings take up two degrees of freedom in order to control one-dimensional scrolling. Instead of a circle, Malacria et al. [25] proposed using oscillatory motions along a line for 2D scrolling input, however oscillatory techniques may be harder to learn than non-oscillatory ones.

At the *gain* stage, manipulating the control-display gain reduces device footprint [26], but high gains can decrease precision unless using a non-linear transfer function that maps low input velocities to low gains and high input velocities to high gains, making both fast and precise movements possible. Non-linear transfer functions have been shown to decrease scrolling time [2] compared to a proportional gain function. Cockburn et al. [27] showed that, in addition to velocity, incorporating document length as a key input parameter substantially improved long distance scrolling performance across a variety of scrolling input devices. Finally, at the *persistence* stage, simulated inertia and friction can allow scrolling to continue autonomously while the user is clutching [12]. Inertial scrolling requires less clutching but does not necessarily improve performance compared to a non-linear transfer function [28].

2.1.3. Rate control

With rate control, user input controls the velocity of scrolling. The added indirection reduces movement compatibility [29, 30], but this can allow smooth and continuous scrolling velocities to be maintained throughout the scrolling movement [1, 2]. Examples include middle-button dragging in many desktop applications, and 'flick scrolling' on mobile devices. The *persistence* stage of rate control transfer functions may maintain the velocity over time or may gradually attenuate it (e.g., simulating friction [31]). Rate control methods require less physical effort than others [9, 28], but they can be difficult to control precisely [2]. In addition, Igarashi and Hinckley [32] remarked that beyond a certain velocity, rate-based scrolling caused disorentation, and proposed to compensate increased scrolling speed by decreasing the scale at which the document is displayed, ensuring a constant visual flow.

In summary, the existing literature on scrolling transfer functions has dealt extensively with the drawbacks and benefits of each type of control, and some works have even proposed or studied hybrid functions combining absolute and relative position control [16, 33], relative position and rate control [11], or absolute position and rate control [17].

2.1.4. Studying scrolling transfer functions

Quinn et al. [3] observed that the reproducibility of empirical studies of scrolling performance was compromised due to lack of transfer function details. They also described a method to reverse-engineer the behaviour of external scrolling devices connected through USB. Subsequent work extended the methodology to touch scrolling devices, using a robotic arm to precisely control gestural input [12].

These methods permit analysis of transfer functions associated with dedicated scrolling devices and touch-based techniques, but they do not extend to edge-scrolling techniques controlled by indirect pointing on desktop applications. Moreover, the logging strategy in these methods are not always suitable to study edge-scrolling. Quinn et al. [3] logged the behaviour of device drivers by using low-level event APIs, and Quinn et al. [12] used custom programs instrumenting specific widgets. Other methods are needed to study the viewports in closed-source applications, as edge-scrolling is implemented at the application level and is likely to vary between applications.

Indeed, there are different levels of developer guidance on how to implement edge-scrolling in custom widgets. Java's *Swing* documents how to comply with the default toolkit behaviour in the `setAutoscrolls` method of class `JComponent`[1] and in the `Autoscroll` interface[2]. In Cocoa applications, edge-scrolling is automatically present in any view inside a `NSScrollView`, but to be available during drag-and-drop it must be implemented in an ad-hoc fashion[3]. Other toolkits like Windows Presentation Foundation, Qt, or GTK+ provide no recommendation or widget-independent behaviour to developers.

Consequently, edge-scrolling requires an adapted reverse-engineering methodology based on indirect pointing input and compatible with closed-source applications (described later).

[1] `http://docs.oracle.com/javase/7/docs/api/javax/swing/JComponent.html#setAutoscrolls(boolean)`

[2] `http://docs.oracle.com/javase/7/docs/api/java/awt/dnd/Autoscroll.html`

[3] `https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NSScrollViewGuide/Articles/Scrolling.html#//apple_ref/doc/uid/TP40003463-SW3`

2.2. Issues with pointing-based scrolling

A significant portion of existing scrolling techniques use a dedicated input stream distinct from pointing input (e.g., a scrollwheel). These techniques are fundamentally different from edge-scrolling, which shares the pointing input stream and uses mode delimitation to switch between pointing and edge-scroll modes. Studies have suggested that dual-stream input eliminates costs related to sequential operation and mode switching in single-stream input [16, 34, 35]. However, it is unlikely that edge-scrolling presents these costs.

First, there is little evidence that pointing and scrolling are performed in parallel in compound tasks [16]. Based on the global precedence principle found in human motor and perceptual systems, Guiard et al. [36] argued that compound scrolling and pointing is a high-level pointing task composed of (1) a "view pointing" phase (scrolling) where the scrolling position of the document inside the viewport is controlled by the user until it intersects the target within the coordinate system of the document; followed by (2) a "cursor pointing" phase (pointing) where the user acquires the target in view through the actual pointer within the coordinate system of the display. In addition, Zhai et al. [1] showed that the opportunity of parallel interaction does not guarantee superior performance to a sequential technique if other factors, such as the transfer function, are neglected.

Second, while scrollbars remove attention and resources from the main task [9], this is not the case when switching between pointing and edge-scroll modes. Using a scrollbar requires focusing on and pointing to a small widget, but the transition between pointing and edge-scrolling is less dependent on visual feedback because it is performed by entering a large dedicated area near a viewport edge or by simply crossing the viewport edge. Moreover, taking Guiard et al.'s [36] argument further, edge-scrolling can be regarded as a direct extension of pointing actions.

2.3. Edge-based pointing

Edge-scrolling relies on the edges of the viewport for activation and control. When pointing to targets close to virtual display edges, or "edge pointing", users can perform primarily ballistic movements because the pointer remains within the target when it is stopped at the display edge. Edge pointing has been shown to bring performance benefits depending on the shape of the pointer and the direction of movement [37]. But viewport edges are seldom collinear with display edges, notably because title bars, sidebars, scrollbars, and status bars that often surround the viewport provide a small padding area when the window is maximised. Nevertheless, crossing the area dedicated to edge-scrolling can be quick when it occupies the whole screen space beyond the considered viewport edge [38].

2.4. Edge-scrolling

2.4.1. Edge-scrolling inventions

Various patents have described edge-scrolling techniques, and have claimed to resolve particular interaction problems. However, no public empirical work has validated these solutions.

Meier et al. [39] described a technique that facilitates drag-and-drop by automatically scrolling while dwelling inside an area placed along the inside edges of the viewport. Scrolling velocity increases as the pointer approaches the edge. Bardon et al. [40] disclosed a relative position-based technique in which pointer movements outside and away from the viewport cause scrolling movements in the same direction and with the same magnitude. Kwatinetz [41] commented that the high scrolling velocity found in existing rate-based edge-scrolling designs causes frequent overshooting. He proposed a method in which scrolling speed is proportional to the acceleration of the pointer.

Other patents have addressed problems associated with disambiguating the user's intention during drag-and-drop actions near the viewport edge – possibly the intended target lies within another viewport (e.g., dragging between windows), or possibly the user wants to edge-scroll the current viewport. Berry et al. [42] proposed the use of a

mouse button to achieve disambiguation, pressed before the cursor reaches the viewport edge. Li and Shrader [43] instead proposed that a keyboard modifier be used to achieve disambiguation, with the key being pressed when the user wants to drag to a different viewport; without the modifier keypress, the cursor would not be permitted to cross the window edge.

To reduce problems of unintended activation of edge-scrolling while pointing near the viewport edge, Belfiore et al. [44] proposed using a threshold on pointer velocity while the pointer is inside the scrolling area–edge-scrolling only occurs when pointer velocity is below the threshold.

2.4.2. Performance studies on edge-scrolling

The patents described above suggest that inventors and designers are aware of the need for edge-scrolling as well as the limitations of current solutions. HCI research, however, has given little attention to these issues, with a few exceptions.

In an early comparison of touchscreen scrolling techniques, Johnson [45] found that a basic edge-scrolling technique (using a fixed scrolling rate, activated by dwelling near the screen edge) was outperformed by a simple swipe-based scrolling method. Kaptelinin [46] compared four navigation techniques based on indirect pointing, including a rate-based edge-scrolling technique activated when the pointer was outside the window and where scrolling velocity was proportional to the distance to the edge. Results showed that edge-scrolling outperformed scrollbar navigation, but was outperformed by a two-dimensional absolute scrolling technique. More recently, Kulik et al. [47] examined methods for drag-based object manipulation on smartphones. They compared rate-based edge-scrolling (activated when the finger is less than 4mm away from the screen edges) to a new bimanual technique that allowed panning with the non-dominant hand in parallel with object dragging. The bi-manual method was faster than edge-scrolling.

To our knowledge, the study by Malacria et al. [28] was the only one that sought to improve desktop edge-scrolling techniques. They proposed two position-based edge-scrolling methods, *push-edge* and *slide-edge* scrolling, offering full transfer function details

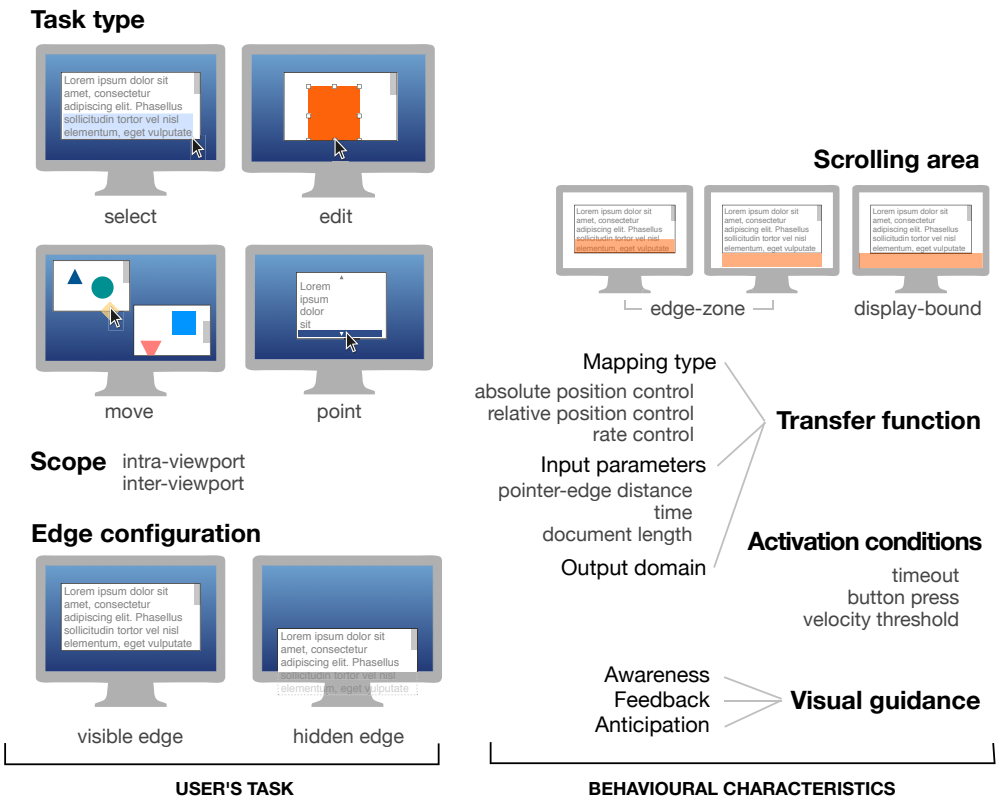

Figure 4: Designing edge-scrolling methods requires understanding two categories of considerations: the user's task (left) and the system's behavioural characteristics (right).

to facilitate replication. Baseline performance of contemporary edge-scrolling solutions was measured using a rate-based edge-scrolling technique across a wide range of scrolling distances. In text selection tasks, the position-based methods offered performance improvements of up to 13% compared to the rate-based technique.

3. The design space of edge-scrolling

This section describes a framework that encapsulates critical factors that must be considered when designing edge-scrolling techniques. The section also serves as a definition and explanation of terms used in the remainder of the paper. The two major components of the framework, described below, are summarised in Figure 4.

3.1. The user's task

Edge-scrolling techniques are used during pointing or manipulation activities where the endpoint is located outside the visible part of the document. They have particular value when the user has entered a dragging state, because doing so may preclude the use other scrolling input techniques – for example, scrollbars cannot be used until the dragging state is terminated, and mouse-wheel input is often disabled during dragging.

We consider four types of target-directed tasks that may require edge-scrolling:

Select The user selects a region of the information space by marking an initial location (e.g., by pressing the mouse button), dragging over a region within the viewport, and marking the final location of the selection (e.g., by releasing the mouse button). For example, selecting a series of cells in a spreadsheet (Figure 1b).

Edit The user modifies an object by dragging. Edge-scrolling facilitates manipulating the object beyond the content displayed in the current viewport. For example, resizing a shape in a drawing application (Figure 1c).

Move The user moves an object by dragging. For example, moving a file icon in a subdirectory (Figure 1d), or moving a paragraph within a document by drag-and-drop.

Point The user reaches a remote point in a document and selects it. For example, selecting a font in a large menu (Figure 1a).

During the execution of *Select* and *Edit* tasks, there is no ambiguity about the viewport that contains the user's intended target – the target must reside within the viewport that contained the starting point. *Move* and *Point* tasks, however, can be ambiguous – the user's target may lie within the initial viewport, or it may lie within a different viewport on the same display (for example, during drag-and-drop between windows). Designing edge-scrolling techniques therefore requires considering the merits of the technique in facilitating *intra-viewport* scrolling, as well as considering the risks that the technique

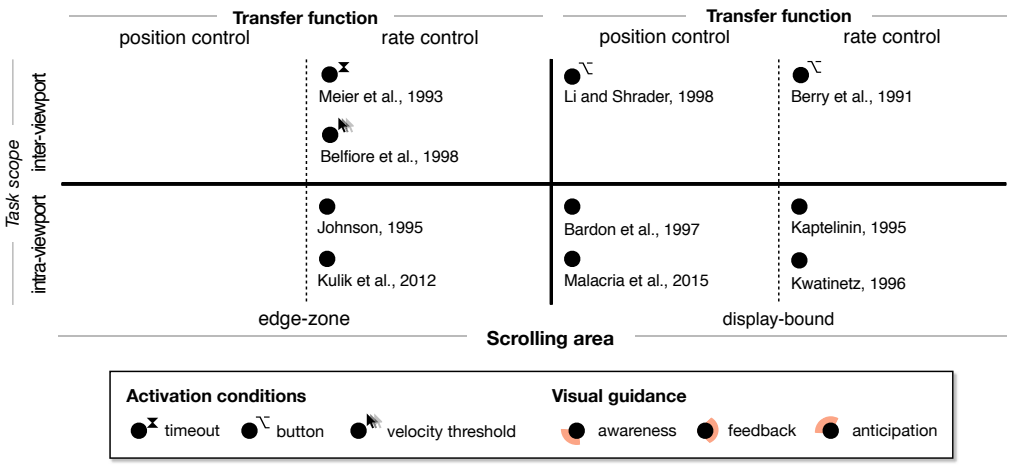

Figure 5: Existing edge-scrolling designs classified using the design space of edge-scrolling techniques from their descriptions in the literature.

may interfere with the user's legitimate *inter-viewport* activities. Furthermore, there are risks that inconsistencies between edge-scrolling techniques will confuse users or impair their performance, suggesting that there may be end-user advantages of consistency, even when task requirements permit different approaches.

The display layout also influences the nature of the user's task and their ability to effectively use different edge-scrolling techniques. When the viewport edge lies within the display (*visible edge* configuration) there is available space for a scrolling area that extends beyond the viewport, but this is not the case when the viewport is either coincident with the display edge (e.g., a maximised window) or lies outside the display (*hidden edge*).

3.2. Behavioural characteristics

The behaviour of edge-scrolling techniques is determined by four key considerations, summarised in Figure 4: *scrolling area*, *transfer function*, *activation conditions*, and *visual guidance* to the user. Prior work from the edge-scrolling literature, reviewed earlier, is classified according to these considerations in Figure 5.

3.2.1. Scrolling area

The *scrolling area* is the part of the screen, relative to the viewport, that is used to activate and control edge-scrolling. We consider two types of scrolling areas: edge-zone

and display-bound (Figure 4, top-right). An *edge-zone* area is placed near the viewport edge. It has the same width as the viewport, and has a comparatively small height. An edge-zone may be inside the viewport, outside the viewport, or a combination that need not be symmetrical. A *display-bound* area, in contrast, extends from the viewport edge (or close to it) to the adjacent display edge.

Larger scrolling areas permit a larger pointing footprint, and therefore may facilitate finer granularity of input control. However, the likelihood of accidental activation of edge-scrolling increases as the area increases. Furthermore, the ability to use a particular type of scrolling area may be influenced by task requirements (e.g., outside zones may be incompatible with the need to support inter-viewport object movement) or by the layout of viewports on the display.

3.2.2. Transfer function

Edge-scrolling techniques require a *transfer function* to determine the mapping from pointing input actions to scrolling output. Additional parameters may also be used by the transfer function, such as the length of the document [27] or the time spent in the scrolling area.

We distinguish three key *types* of edge-based scrolling transfer functions. In *absolute position control*, the vertical position of the pointer in the scrolling area is mapped to the vertical scrolling position in the document. In *relative position control*[4], vertical pointing movements are translated into vertical scrolling movements. Scrolling amplitude is either derived from the amplitude of the pointing movement, or from the vertical distance from the pointer to the edge (*pointer-edge distance*). In either case, it is possible that only movements away from the viewport are taken into account. Finally, in *rate control*, the pointer-edge distance is mapped to the velocity of automatic scrolling.

The *input domain* is the set of possible pointing input values (for instance, pointer-edge distances), and it is limited by the size of the scrolling area. Correspondingly, the *output domain* is the set of possible output values (which can be positions, displacements, or

[4]In the rest of this paper, we will also refer to relative position control as, simply, position control.

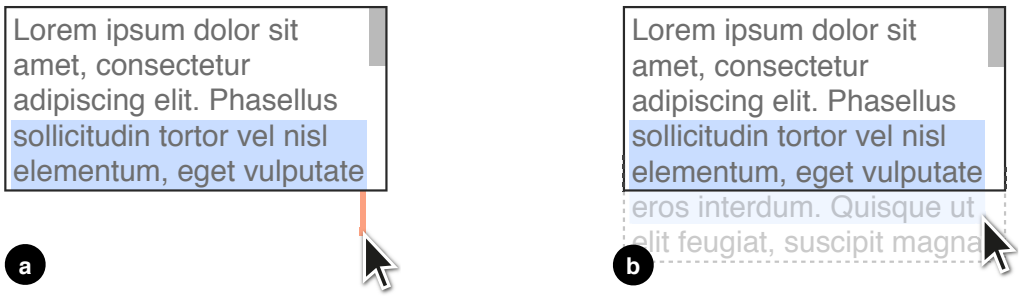

Figure 6: Edge-scrolling techniques can use visual guidance (a) to provide *feedback* on control parameters (the length of the orange line represents scrolling velocity); and (b) to support *anticipation*, by displaying parts of the document outside the viewport during edge-scroll.

velocities, depending on the type of mapping) that can be produced by the transfer function.

3.2.3. Activation conditions

The initiation of edge-scrolling requires fulfilling some *activation conditions*. The primary activation condition is that the pointer is inside the scrolling area, and this may be sufficient to initiate scrolling. Secondary activation conditions may also be used, such as the expiration of a timeout, a specific button press, or a threshold pointer velocity being attained.

3.2.4. Visual guidance

Document and scrollbar displacements provide strong visual feedback about changes in viewport position. However, additional *visual guidance* can be offered that affect *awareness*, *feedback*, and *anticipation*. First, awareness of the edge-scroll functionality can be improved with feedforward. For example, arrows in Figure 1a indicate that edge-scrolling is available at the menu's extent. Second, feedback may help users comprehend the parameters they control, such as scrolling velocity (Figure 6a). Third, the user's ability to anticipate the upcoming attainment of the target location may be assisted by visualisations (e.g., Figure 6b).

4. Analysis of existing designs

In this section we reverse-engineer the edge-scrolling behaviour of 19 common applications and GUI toolkits. We do so by monitoring the scrolling output of a viewport in response to software-simulated pointing input. We then use the design space presented in the previous section to analyse and report their high-level characteristics.

4.1. Method

Each edge-scrolling technique was examined using the following method: (1) a scrollable viewport implementing the technique was set to an initial configuration (size, location, position of the slider); (2) the viewport was manipulated using several predefined interaction patterns; (3) the scrolling response to these manipulations was observed and measured. This method facilitates exact and replicable findings.

4.1.1. Initial configuration

We used a 400 pixels[5] high viewport located at the center of a 1440x900 display, leaving 250 pixels between the bottom edge of the viewport and the bottom edge of the display. This viewport contained a large document, the content of which varied depending on the type of viewport and behaviour tested. Text and list documents used a *Lorem ipsum* filler text, typeset in *Ubuntu Mono Regular 16px*[6]. List documents used similar font and text size. Graphical documents were 100,000 pixels high. Pilot investigations showed that it was not possible to set a unique document size across all viewports due to document margins or peculiarities of typesetting algorithms, but fortunately none of the tested designs depended on the size of the document.

4.1.2. Procedure

We combined human observation and software logging of scrolling response to both human and computer-generated manipulations of each scrollable viewport. We examined

[5] In this paper, we consider *logical* pixels, which might comprise several physical ones in newer operating systems when using high-density displays. We prefer to use logical pixels instead of millimeters because edge-scrolling control is dependent on the scale of control objects on the display.

[6] We ensured that the same font size was used across all the different operating systems by using pixels instead of points as the font size unit.

downwards scrolling in response to dragging interactions the bottom edge of the viewport, but the method can be applied to any edge direction or input mode.

Dragging began at a starting point *sp*. The type of object lying under the pointer at that point determined the type of edge-scrolling task performed. For example, in a text document, dragging from a point containing selected text initiated a *move* task, but dragging from a point with no selected text initiated a *select* task.

We first used ad-hoc dragging interactions, observations, and logging to determine the following: type of scrolling area, type of transfer function mapping, and attendance to different input parameters (pointer-edge distance, time, pointer movements, font size, and viewport height). We also used the same method to determine the availability and size of the scrolling area after placing the viewport in the hidden edge configuration.

We then simulated a specific mouse input pattern with all behaviours (when possible) in order to precisely characterise the scrolling area and transfer function. The pattern (Algorithm 1) describes as many dragging sequences as there are pixels in the scrolling area. Each dragging action starts from *sp*, jumps directly to the relevant pixel, and ends after a ten second dwell. For each dragging action, software logged the scrolling position each second, allowing us to measure the effect of pointer-edge distance on scrolling velocity (for rate-based functions) or distance (for position-based ones), as well as the limits of the output domain.

4.1.3. Apparatus

We built software components to simulate mouse input and log resultant scrolling displacements for various operating systems and viewport types. To simulate mouse input on Windows, we developed a tool in C# using the Win32 API. For OS X, we developed a simulation tool in Objective-C using Quartz Event Services.

To log scrolling output, we used various methods to meet requirements of the target viewport. For standard widgets in GUI toolkits and web browsers, we wrote programs that configured the window parameters and content, recorded mouse events, and logged changes in scrolling position. To log scrolling output in standalone applications

Data:

p	the vertical position of the pointer relative to the bottom edge of the viewport
s	the vertical scrolling position of the viewport
sp	the vertical position of the starting point of the dragging interaction (task-dependent)
y_m	the top vertical position of the bottom scrolling area relative to the bottom edge of the viewport
s_{size}	the height of the bottom scrolling area

Result:

$ds_t[y]$	the recorded amplitude of total scrolling displacements on time t of a dragging interaction to point y of the scrolling area

```
y_M ← y_m + max{s_size, 250};
foreach y in {y_m, ..., y_M} do
    s ← 0;
    p ← sp;
    mouse down;
    p ← y;
    foreach t in {1, ..., 10} do
        wait 1 sec;
        ds_t[y] ← s
    mouse up;
```

Algorithm 1: Simulated mouse input pattern applied to the scrollable viewports. The pattern describes discrete successive dragging actions, one for every pixel of the scrolling area.

we used accessiblity APIs when available. On Windows, scrollable viewports implement the `ScrollPattern` control pattern using Microsoft's UI Automation API. The scrolling position in pixels can be derived using properties `VerticalScrollPercent`, `BoundingRectangle`, and `VerticalViewSize`. With OS X's Accessiblity API, scrolling position is provided by the `AXPosition` property of scrollable viewports implementing the `AXScrollArea`. In some applications, such as Photoshop, only the scrollbar accessibility object could be observed, not the viewport itself. In this case, the scrolling position y was approximated based on the normalised scrollbar value $v \in [0, 1]$, the height of the viewport h_v, and the height of the document h_d: $y \approx v \times (h_d - h_v)$.

4.2. Results

We examined the edge-scrolling behaviours of 19 common applications and GUI toolkits spread across three operating system versions (Windows 7, 8, and OS X 10.9.2). Depending on the application, the inspected viewports displayed web pages, text documents, images, spreadsheets or lists.

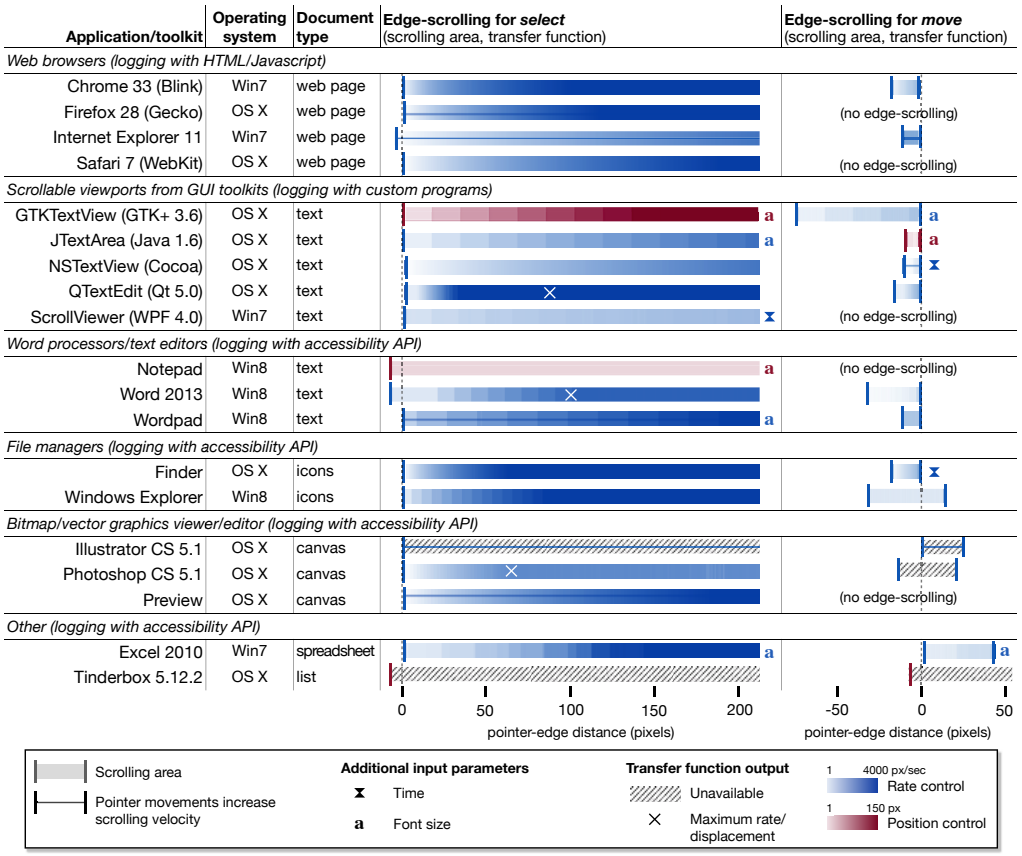

Figure 7: Overview of the 19 applications and toolkits examined across three operating systems (Windows 7, 8, and OS X 10.9.2). Bars describe scrolling areas and transfer functions of their edge-scrolling response to *select* and *move* tasks. Colour corresponds to rate control (blue) or relative position control (red), and opacity denotes either the average velocity (rate control) or the average displacement (position control) for a given pointer-edge distance. A central horizontal line in a scrolling area indicates that pointer movements generate scrolling displacements. Bars are hatched when transfer function data was not available due to incomplete or unreliable accessibility implementation in the application.

Each application was tested for *select* tasks and *move* tasks (edit tasks were not analysed as they were supported by only four applications). The combination of an application and a particular task type will be referred to as a *configuration* (e.g., GTKTextView$_{\text{select}}$). When no task type is specified (e.g., Tinderbox$*$), we refer to both edge-scrolling configurations surveyed for that application.

Overall, we surveyed 33 configurations: 19 *select* and 14 *move* (Figure 7). The following analysis focuses on each configuration's scrolling area and transfer function.

4.2.1. Scrolling area

For *select* tasks, all applications used a *display-bound* scrolling area, and for *move* tasks, all used an *edge-zone* area (with the exception of Tinderbox$*$ that uses a display-bound area).

The 20 *display-bound* areas, which extend to the display edge, varied in their starting position. In 15 configurations the area started at the viewport edge or up to 2 pixels away from it. For the 5 remaining configurations, the area started a few pixels inside the viewport (4 pixels for Internet Explorer$_{\text{select}}$, 8 pixels for Notepad$_{\text{select}}$ and Word$_{\text{select}}$, 10 pixels for Tinderbox$*$), becoming the only part of the area that is available when the viewport is maximised.

There were 13 *edge-zone* scrolling areas, which use a fixed area near the edge. Among them, 9 were positioned alongside the viewport edge or 1 pixel inside it. The observed size of these edge-zones ranged from 8 pixels (JTextArea$_{\text{move}}$) to 75 pixels (GTKTextView$_{\text{move}}$), with GTKTextView$_{\text{move}}$ using a size that is 20% of the viewport height. Windows Explorer$_{\text{move}}$ and Photoshop$_{\text{move}}$ used edge-zones that straddled the viewport edge. Interestingly, Illustrator$_{\text{move}}$ and Excel$_{\text{move}}$ used edge-zones that extended from the viewport edge to the window edge, overlapping the scrollbar and status bar.

These findings suggest that designers chose to exploit available display space (using *display-bound* areas) to maximise the input domain for *select* tasks, which are unambiguous in terms of the intended target viewport – *inter-viewport* selections are not supported. In contrast, the potential for target viewport ambiguity in *move* tasks predominantly

led designers to use fixed area *edge-zones*, despite the resultant constraints on the input domain. The only exception is Tinderbox$_{\text{move}}$, which uses a *display-bound* area and discriminates between *intra-* and *inter-* move intentions by using a pointer velocity threshold that is sampled when the pointer enters the scrolling area.

Finally, scrolling areas were typically unavailable in the *hidden edge* configuration, except for NSTextView$_{\text{select}}$, Preview$_{\text{select}}$, and Excel$_{\text{select}}$, which used fixed edge-zones that started above the display edge (50px for NSTextView$_{\text{select}}$ and Preview$_{\text{select}}$; 1px for Excel$_{\text{select}}$). Finder$_{\text{select}}$ also supported edge-scrolling in the hidden edge configuration, but did so by smoothly moving the window upward until the whole viewport was accessible again, and returning it to its original location once the drag action was complete.

4.2.2. Transfer function

Among the 33 configurations, only 5 used relative position control – GTKTextView$_{\text{select}}$, JTextArea$_{\text{move}}$, Notepad$_{\text{move}}$, and Tinderbox$_{*}$. The remaining 28 used rate control. Note that Tinderbox$_{*}$, Illustrator$_{*}$ and Photoshop$_{\text{move}}$ configurations (shown hatched in Figure 7) could not be examined due to accessibility API limitations.

Input parameters. As shown in Figure 7, most transfer functions mapped pointer-edge distance to scrolling velocity. Additional input parameters were also taken into account, including font size, time, and pointer movements.

Nine configurations used font size in their transfer function: GTKTextView$_{*}$, JTextArea$_{*}$, Notepad$_{\text{select}}$, Wordpad$_{*}$, and Excel$_{*}$. Three configurations used time as an input parameter, with velocity increasing with time in Finder$_{\text{move}}$ and NSTextView$_{\text{move}}$, and decreasing with time in ScrollViewer$_{\text{select}}$.

Eight configurations used any pointer movement to generate additional scrolling displacements in their rate control transfer function: Firefox$_{\text{select}}$, Internet Explorer$_{*}$, NSTextView$_{\text{move}}$, Wordpad$_{\text{select}}$, Illustrator$_{*}$, and Preview$_{\text{select}}$. In these configurations, pointing movements within the scroll area increase the frequency of scrolling output, giving the effect of faster scrolling. This is likely intended to work around limitations of a small input domain due to a constrained scrolling area size.

Transfer function	Task type	Mapping type	Approximate model parameters			
Constant		$f(x) = a$	**a** (y-intercept)			
Notepad	select	position	20			
JTextArea	move		17			
Windows Explorer		rate	297.58			
Wordpad			809			
Internet Explorer			1328			
Linear		$f(x) = a + bx$	**a** (y-intercept)	**b** (slope)		
NSTextView	select	rate	0.25	10		
Internet Explorer			49.96	10		
Preview			-10.36	20.26		
Safari			19.51	19.59		
Firefox			-3.4	31.67		
Finder			-53.87	56.06		
Chrome	move		1134.44	54.18		
Regular staircase		$f(x) = a + b(1 + \lfloor \frac{x+\tau}{\sigma} \rfloor)\sigma$	**a** (y-intercept)	**b** (slope)	**σ** (step size)	**τ** (x-offset)
GTKTextView	select	position	0	1	17	0
JTextArea		rate	0	10	17	0
Wordpad			521.16	16.66	18	-7
Word			0	26	10	-10
Photoshop			60	28	2	-4
Windows Explorer			0	41	6	2
GTKTextView	move		670	9.85	17	8
Excel			35	10	20	0
Nonlinear			**Comments**			
ScrollViewer	select	rate	time-based			
Excel			irregular staircase			
Chrome						
QTextEdit						
NSTextView	move		time-based			
Word			irregular staircase			
QTextEdit						
Finder			time-based			

Figure 8: The transfer functions of 28 configurations of edge-scrolling for which we could successfully measure scrolling data, organised into four classes, together with approximate models.

Finally, Windows Explorer$_{\text{move}}$, Wordpad$_{\text{move}}$, and Internet Explorer$_{\text{move}}$ all used constant transfer functions that took no input (other than activation by entering the scroll area).

Transfer function output. We arranged the transfer functions in four classes, summarised in Figure 8, depending on their scrolling output response $f(x)$ (in pixels for position control functions, in pixels/s for rate control ones) to pointer-edge distance input x (in pixels).

Constant functions (5 configurations) always produce the same output value a in response to any pointer-edge distance ($f(x) = a$), although a changed with font size for Notepad$_{\text{select}}$ and JTextArea$_{\text{select}}$[7]. Therefore, their output domain has exactly one value, offering limited control.

Linear functions map pointer-edge distance linearly to scrolling output ($f(x) = a + bx$), with slopes (b) varying from 10 to 56.06. The seven configurations with linear transfer functions used rate control.

Regular staircase functions (8 configurations) increase scrolling output in progressive steps of fixed size σ as pointer-edge distance increases – $f(x) = a + b(1 + \lfloor {}^{x+\tau}/_{\sigma} \rfloor)\sigma$, given a (y-intercept), b (slope), σ (step size), and τ (x-offset). In the 5 configurations using a text document and a regular staircase function, step size was equal to the line height (which depended on font size), causing discrete line-based movement.

Other nonlinear functions (8 configurations) were grouped into the fourth class. In QTextEdit$_*$, Excel$_{\text{select}}$, and Word$_{\text{move}}$, the transfer function increased stepwise, but step width and height were not constant. In addition, the transfer function of Excel$_{\text{select}}$ depended on font size. There were also 3 rate-based configurations where time was an input parameter (Figure 9). In NSTextView$_{\text{move}}$ and Finder$_{\text{move}}$, time spent in the scrolling area had little effect until pointer-edge distance was above 5 pixels. The change was more continuous over the whole scrolling area in ScrollViewer$_{\text{select}}$, but pointer-edge

[7] With these configurations, a was the line height, which differed between applications due to varying typesetting algorithms.

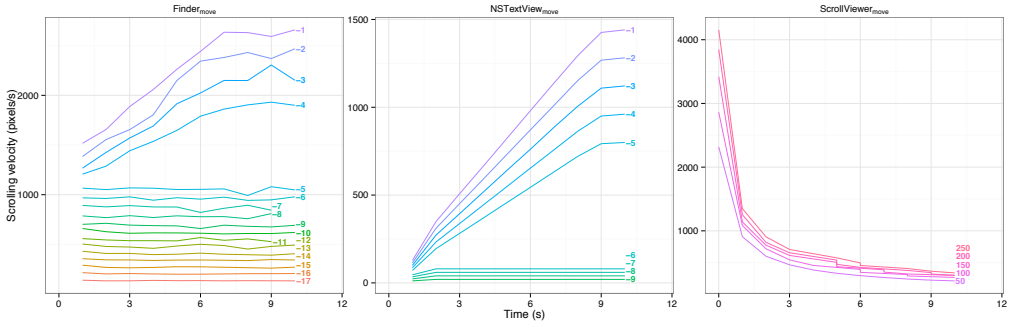

Figure 9: Scrolling velocity as a function of time at different pointer-edge distances (line labels, in pixels) for three edge-scrolling configurations that included the time spent in the scrolling area as an input parameter.

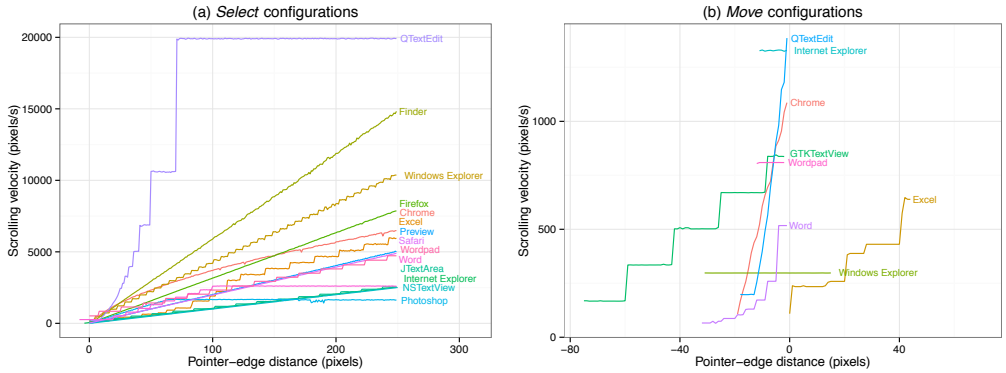

Figure 10: Scrolling velocity by pointer-edge distance, for (a) 14 *select* and (b) 8 *move* configurations, with rate-based transfer functions that did not depend on the time spent in the scrolling area.

distance had virtually no effect.

Output domain. The range of system responses that can be attained, and therefore the fidelity of control opportunity, varied with the class of transfer function: for example, regular and irregular staircase functions had sparser output domains than linear functions. We analysed the minimum and maximum scrolling velocities that could be attained with each of the rate-based transfer functions, with results summarised in Figure 10. No significant difference in minimum velocity was found (Mann-Whitney U=60, Z=1.382, p=0.178) between edge-zone (median $\tilde{x}$=112 px/s, IQR 161 px/s) and display-bound ($\tilde{x}$=60 px/s, IQR 167 px/s) scrolling area configurations. However, edge-zone configurations had

significantly lower maximum velocities ($\tilde{x}$=844 px/s, IQR 644 px/s) than display-bound approaches ($\tilde{x}$=4975 px/s, IQR 5463 px/s) – (U=166, Z=-3.849, p<0.001).

4.3. Summary of existing designs

The results demonstrate substantial variance in the design approaches to edge-scrolling, with only two areas of general conformity. First, rate-control methods are much more common than others. Second, *select* tasks typically allow larger *display-bound* scroll areas than the *edge-zone* areas typically used for *move* tasks. This observation can be largely attributed to the potential for task ambiguity in *move* tasks – the user's target may lie within the current viewport or it may lie within a different one. This ambiguity does not exist for *select* tasks, generating a somewhat simpler design space.

Other than these two general trends, the variance in design approaches is substantial, with markedly different transfer functions and input parameters leading to very different opportunities for system control. This is likely due to lack of support from the underlying operating system and toolkits, which seldom provide default behaviours or ways to adapt them to the task at hand. Reverse-engineering the behaviour of edge-scrolling viewports has exposed these substantial differences and provides a platform for reliable replication and empirical comparison.

Our approach to reverse engineering, including reliance on accessibility APIs, is subject to limitations. In particular, the use of inter-process communication in accessibility APIs, non-real-time timers for rate-based scrolling implementations, and their non-critical nature can cause imprecise measurement. Additionally, the input parameters to probe must be manually determined, and therefore it is possible to overlook a critical parameter. However, the approach permits replicable analysis without demanding access to underlying source code that is typically unavailable to researchers.

5. Interactive survey of edge-scrolling

While the previous section shows that edge-scrolling is widely supported, it is possible for users to achieve their tasks without it. For example, some applications allow mouse-

wheel, touch gesture, or keyboard scrolling input concurrently with dragging actions, or users can extend a selection range by shift-clicking. Each of these methods circumvents the need for edge-scrolling. We therefore ran an interactive online survey to determine whether users actually use edge-scrolling and to examine their perceptions of the technique.

5.1. Design

The survey focused on *select*, *move*, and *edit* tasks. To avoid priming the user's attention on edge-scrolling, the survey was designed to probe and observe text selection activities prior to introducing specific questions on edge-scrolling. Our intention was to observe how users would choose to complete tasks that *might* be completed with edge-scrolling before exposing our interest in the method. Pilot studies with ten interviewed participants indicated that our survey method was successful.

The survey featured six main parts (described below), each displayed on a separate web page. Each part had to be completed before proceeding with the next one.

Part one gave a short introduction describing the survey structure and time requirements, and asked general questions about computer use: principal operating system, web browser, pointing device, dominant hand, and time spent using a computer daily.

Part two involved an interactive guided selection task that was used to determine which scrolling technique the respondent would chose when completing a text selection task. The survey page displayed a 15-point 4932-word scrollable text viewport and a tooltip (Figure 11). The tooltip instructed the user to select the blue-coloured paragraph, and doing so required substantial vertical scrolling.

Part three asked questions about the text selection task in part two. The notion of

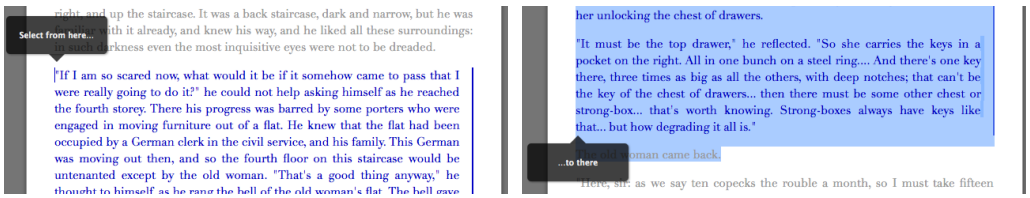

Figure 11: Text selection task for the second part of the online survey. Respondents were free to use any scrolling technique while selecting the paragraph coloured in blue.

concurrently pointing and scrolling was introduced, and questions were asked about how frequently the user encountered this need during *select*, *edit*, and *move* tasks – animations and text were displayed to clarify each task. Responses were recorded using 5-point Likert-type items, from *never* to *very often* (an 'I don't know' item was also included).

Part four asked Likert-scale questions regarding the frequency with which different methods (*edgescroll*, *wheel*, *keyboard*, and *shiftclick*) were used to complete *select*, *edit*, and *move* tasks. Again, text and animations were used to clarify the questions. The order of presentation of each technique in the survey was randomized across respondents to limit bias. An optional free-form text item asked for comments about the techniques.

Part five focused on edge-scrolling. Text and an animation described the technique, was a Yes/No question asked whether the respondent was aware of the technique before the survey. Three 7-point Likert-type items then queried perceptions of edge-scroll speed control, general experience, and frequency that edge-scrolling behaviour differs from her expectations. Free-form text comments were requested regarding edge-scroll problems, strategies to counter problems, and expectations for future designs.

Part six requested demographic information on age, gender, nationality, and occupation.

Respondents were recruited through local university mailing lists and online social networks (Facebook, Twitter, Google+). No compensation was offered for completing the survey and respondents were informed that their answers were anonymous. The survey tool was implemented as a HTML website with Javascript communicating to a home-made Node.js server using AJAX.

5.2. Results

In two months, 214 volunteers completed the survey. Most respondents were male (71%) and most reported they used a computer for more than five hours daily (84%). The median age was 26 (IQR 23-32; 46 preferred not to answer). Of the 155 respondents who specified their occupation, most were students (65), researchers (36), or engineers (25). Primary operating systems were reported to be Windows (58%), OS X (29%), and

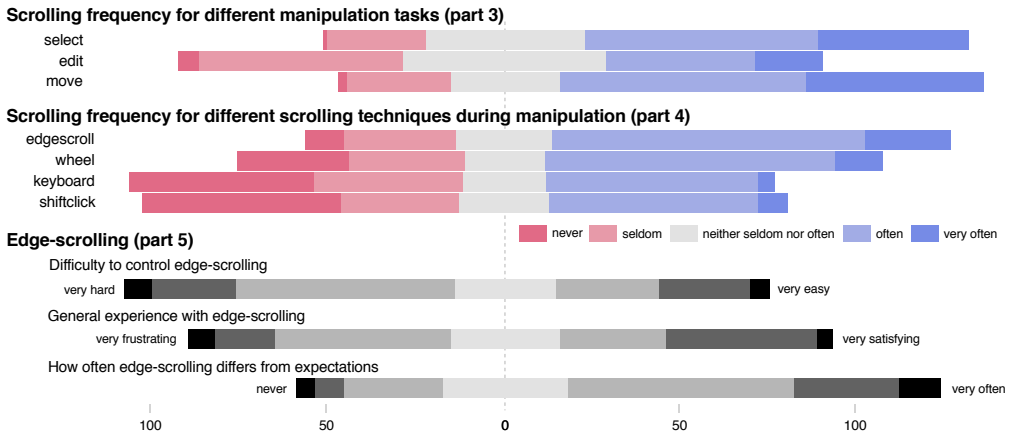

Figure 12: Responses to Likert-type items on parts 3, 4, and 5 of the survey.

Linux (13%), and primary web browsers were Chrome (47%), Firefox (34%), and Safari (13%). Almost all of the respondents operated their pointing device with their right hand (98%), and 66% were mouse users while the others used a touchpad.

5.2.1. Selection task analysis

Part two of the survey (the selection task) provides the study's primary insight into how users choose to complete text selection activities.

The survey web page logged pointing, keyboard, and scrolling events using the DOM Javascript API, recording time-stamps and high-level event descriptions, such as "the page scrolled down" or "the pointer went outside the viewport". This data permits reconstruction of the method(s) used to complete the task. When more than one method was used (7%), we retained the method that had the longest duration. We double-checked this by manually by reading through the event logs for each respondent of these 7%.

Edgescroll was the most used technique (57%). The mouse-wheel or touch gestures were used by 34%. *Shiftclick* (8%) and *keyboard* (1%) were rarely used.

This important finding indicates that edge-scrolling is a widely used method for text selection.

5.2.2. Perceived frequencies of methods and perceived limitations edge-scrolling

The third and fourth part of the survey concerned perceived frequency of use of different scrolling methods for different task types (*select*, *edit*, and *move*). The fifth part focused on more general perceptions of edge-scrolling. Responses to these survey parts are summarised in Figure 12. Key observations are that respondents found *select* and *move* tasks much more common than *edit* tasks, and that edge-scroll was perceived to be more commonly used than wheel/touchscreen gestures. Keyboard and shiftclick methods were perceived to be used infrequently.

Likert-scale responses indicated that respondents encountered difficulty controlling the viewport velocity during edge-scrolling (median response "somewhat hard", Figure 12). There is no evidence that the pointing device affects the attitude towards edge-scrolling control difficulty, as a Mann-Whitney U test showed no significant difference between mouse and touchpad users (U=4981, Z=.032, p=.975).

The general experience reported for edge-scrolling was almost equally divided between frustrated (41%) and satisfied (40%) (median response "neutral"). Again, we found no evidence that this attitude is affected by the preferred pointing device (U=5064.5, Z=.327, p=.775).

Regarding how often edge-scrolling differs from expectations, 50% of the respondents answered between "somewhat often" to "very often", while only 19% answered between "seldom" and "never"; 31 participants (15%) answered "I don't know".

5.3. Thematic analysis

To examine the comments provided by the 74 participants who gave them, we performed a thematic analysis [48], yielding three main themes concerning the context/experience of use, control/performance issues, and behavioural inconsistencies.

5.3.1. Context and experience of use

The choice of scrolling technique is heavily influenced by context of use, which includes the following factors:

- application: *"It depends on the application, operating system and how familiar I am with the application"* (P141)
- object type: *"I only use shift click when selecting multiple items in a long list"* (P127)
- input device in use: *"I use the wheel only with a mouse"* (P222)
- scrolling distance: *"for small parts of text, I prefer to use keyboard shortcuts while when I need to select entire paragraphs, I prefer to use the edge-scroll technique"* (P160)
- knowledge of endpoint location: *"[I] edge scroll often when I am not sure when to stop, and Maj + Click when I know exactly"* (P175).
- whether the user is already dragging: *"if I find out that I am already selecting I expect edge-scroll to work"* (P196).

Additionally, several respondents commented that edge-scrolling has low activation requirements compared to other techniques – e.g., *"I just click and keep dragging until I get to the bottom"* (P169). However, perceived deficiencies may prompt users to choose other methods – regarding edge-scroll, P203 commented *"the only expectation is that it will be painful!"*. Respondents also reported choosing other methods when edge-scroll response was too fast (*"when it goes too fast I use one of the keyboard based methods previously mentioned"*, P2) and too slow (*"wheel when edge-scroll isn't fast enough"*, P175). Furthermore, several participants noted that repeated inadequacies in edge-scroll encouraged them to avoid it when possible: *"it's something I often encounter but try to avoid where possible"* (P62); *"I stopped using it"* (P170); *"I try and avoid it, but sometimes it's not possible to avoid"* (P203).

5.3.2. Control and performance

The lack of control in edge-scrolling received many comments, especially imprecise velocity control. These limitations can be attributed to three causes: limitations of the

transfer function, ambiguous activation conditions, and a limited scrolling area.

Transfer function. Several comments identified inappropriate limitations on the maximum attainable scrolling velocity: *"the maximum speed is way too slow when you want to do large selection"* (P82); Others referred to the output domain permitting excessively high velocities: *"Sometimes it scrolls incredibly quickly which is difficult to control"* (P66); *"it's really annoying when it's too fast"* (P4); *"there is no way to limit the scrolling speed"* (P186). The failure to support a sufficient transition between scrolling speeds also indicates limitations in the transfer function: *"always a big jump between very slow motion (before crossing the edge) and hyper fast motion (as early as I cross the edge)"* (P20).

Activation conditions. Respondents referred to problems in activating edge-scroll mode: *"having to wiggle the mouse/pointer to get the application to realise that is what you want to do"* (P141). Furthermore, anticipating activation caused problems: *"it often takes some time for the scrolling to begin, but once it does its speed is too fast to easily control"* (P62).

Limited scrolling area. Respondents directly commented on control limitations inherited from small scrolling areas: *"there isn't enough room left to use the technique (scrolling is ridiculously slow)"* (P26); *"you have to find the edge to start the scrolling in a single pixel height line, or in a very small area. If you pass that zone, you have to come back"* (P197). Issues of display configuration were also noted as causing further control problems: *"when a window is full-screen, there's not enough room to use the technique"* (P71). This problem led some respondents to configure window placement specifically to assist edge-scrolling: *"I tend to move the active window"* (P62); *"reduce the size of the window so that it is as far as possible from the edge of the screen and enjoy greater speeds"* (P15).

The comments above indicate a variety of performance problems. Several respondents additionally identified problems with overshooting and spatial awareness.

Overshooting. For example, P92 commented on the frequent need to reverse actions after overshooting: *"I select too large a part, then I have to unselect some parts"*. While several participants noted that overshooting was caused by excessive scrolling velocities (*"too fast, too far"*, P102), one mentioned that excessive slowness caused task disengagement (*"I eventually "fall asleep" and overshoot my target"*, P81).

Spatial awareness. Problems of spatial awareness were induced by excessive scrolling speeds (*"completely losing myself in the page"*, P105), with some respondents adapting their performance to accommodate the problem (*"you have to stop regularly to check where you are in the document"*, P213).

5.3.3. Inconsistency

Many comments were directed at the lack of consistency within and between applications. Issues with the behaviour of Microsoft Excel were noted by several respondents, with some commenting it was too fast (*"too fast with Excel, I find myself at the end of the sheet without noticing"*, P2) and others stating it was too slow (*"in excel, when there are very long columns, it is often too slow"*, P74). This discrepancy might be explained by a change between versions of Excel, as noted by two respondents: *"Excel used to be very sensitive. It's not so bad any more."* (P229).

Lack of consistency between applications was also noted (*"when I copy-paste between Word files and HTML pages the speeds are generally different, even null in some editors, and it becomes frustrating"*, P175). The desire for greater consistency was explicitly stated (e.g., *"more consistency, and adapted speed"*, P2).

Lack of consistency may also contribute to a general lack of understanding or misunderstanding of edge-scrolling behaviour: P170 simply stated *"I don't understand how it works"*, and others echoed the request of P183 for *"more intuitive use"*.

5.4. Discussion

Two key findings emerge from the survey. First, results of the selection task (Section 5.2.1) confirm that most of our survey respondents (57%) chose to complete the task

using edge-scrolling. While the use of edge-scrolling might be anticipated, it is important to have empirical confirmation that users actually make use of the technique, rather than work around it using one of the many available alternative methods. Second, the respondents' comments emphasise that the reverse-engineered behavioural differences observed in Section 4 do cause interaction problems for users – respondents made many adverse comments regarding edge-scrolling control/performance and its inconsistent behaviour.

6. Effect of transfer function on edge-scrolling performance

Previous sections have exposed substantial variance in design approach to edge-scrolling, and they have confirmed that edge-scrolling is widely used and that it raises subjective concerns about its behaviour. In this section, we empirically examine and compare user performance and perceived workload when completing edge-scrolling text selection tasks using four different transfer functions that are representative of diverse approaches to edge-scrolling.

6.1. Method

The experimental task involved 1D mouse-based text selections, repeated across a large range of scrolling distances.

6.1.1. Task and stimulus

The task was to select lines of text from top to bottom, with no accuracy constraint on the horizontal axis. A document containing 21,480 lines of text typeset with *Ubuntu Mono Regular* 13px was presented inside a 400 pixel tall viewport located at the centre of the screen. The target portion of text to be selected was framed and coloured blue, starting eight lines from the bottom of the viewport, and varying in size (see Figure 13a). A shaded gradient was overlaid on the scrollbar to indicate the approximate location and size of the target selection. Selecting anywhere on a line selected the entire line. Every selection required edge-scrolling via concurrent dragging and scrolling, with other scrolling methods and devices disabled (e.g., keyboard shortcuts or scroll wheel could not be used).

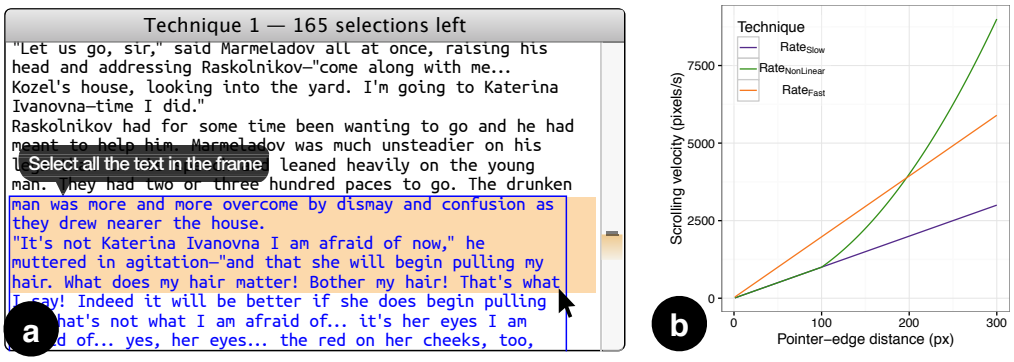

Figure 13: (a) Experimental software for the one-dimensional text selection task: participants had to use edge-scrolling in order to select the text coloured and framed in blue. (b) Scrolling velocity by pointer-edge distance for the 3 rate-based techniques used in the experiment.

To complete a trial participants had to successfully select the blue framed portion of text (Figure 13a), with no additional or missing lines. This involved positioning the pointer at the top of the target portion, pressing the mouse button to trigger the selection action, moving the pointer beyond the bottom viewport edge to trigger edge-scrolling, scrolling the viewport until they reached the bottom of the target portion, positioning the pointer over the last line of the blue portion, and releasing the mouse button to complete the selection. On completing each selection, a pop-up window prompted participants to press any key to move on to the next trial. To discourage "racing through" the study irrespective of errors, trials including an error were repeated.

6.1.2. Techniques

Participants completed the tasks with four edge-scrolling techniques that use display-bound scrolling areas. Three of the techniques were selected from the reverse-engineering study: $\text{GTKTextView}_{\text{select}}$ is a position-based method (referred to as $Position_{AnyDir}$ in the experiment); $\text{NSTextView}_{\text{select}}$ ($Rate_{Slow}$) and $\text{Safari}_{\text{select}}$ ($Rate_{Fast}$) are linear rate-based methods with very different parameter settings (Figure 8).

To study a wider range of behaviours, we also designed and tested a fourth rate-based

technique with a non-linear transfer function ($Rate_{NonLinear}$) defined as:

$$f(x) = \begin{cases} 10x, & x \leq 100\text{px} \\ 0.1x^2, & x > 100\text{px} \end{cases} \quad \text{(in pixels/s; } x \text{ is the pointer-edge distance, in pixels)}$$

Informal tests suggested that linear functions with slopes similar to $\text{Firefox}_{\text{select}}$, Windows $\text{Explorer}_{\text{select}}$, or $\text{Finder}_{\text{select}}$ provided an ample output domain, allowing to quickly scroll through long distances, but were too sensitive in the correction phase. The transfer function of the $\text{Rate}_{\text{NonLinear}}$ technique, shown in Figure 13b, is intended to facilitate precise slow scrolling as well as high velocities (up to 9000 px/s).

All four methods used a 300 pixels high scrolling area.

6.1.3. Procedure

Participants completed 3 blocks of trials with each technique. Each block consisted of 20 selections, comprising 5 selections of each of 4 text sizes, from 13 lines up to 508 lines. Participants were instructed to complete selections as quickly and accurately as possible. After each technique, participants completed a NASA-TLX worksheet.

6.1.4. Design

The experiment used a 4×3×4 within-subjects design with factors: Technique ($\text{Rate}_{\text{Slow}}$, $\text{Rate}_{\text{NonLinear}}$, $\text{Rate}_{\text{Fast}}$, $\text{Position}_{\text{AnyDir}}$), Block (levels 1-3) and Size (S_{13}, S_{58}, S_{258}, S_{508}, depending on the number of lines of text to select). The order of Technique was counterbalanced across participants. The experiment lasted approximately 40 minutes, with a total of 4×3×4×5 = 240 trials performed per participant.

6.1.5. Participants and apparatus

18 university staff and students participated, with median age 26.5 (IQR 10, 4 female).

The experimental software was written in Objective-C with the Cocoa API and ran on an Apple laptop computer running Mac OS 10.8.4, using a 22" external monitor with resolution 1680×1050 pixels, and pixel density 90 pixels per inch (PPI). The input device was a USB corded Logitech M90 mouse with resolution 1000 CPI, and was used on a varnished plywood desk. The pointing transfer function of the system was set to

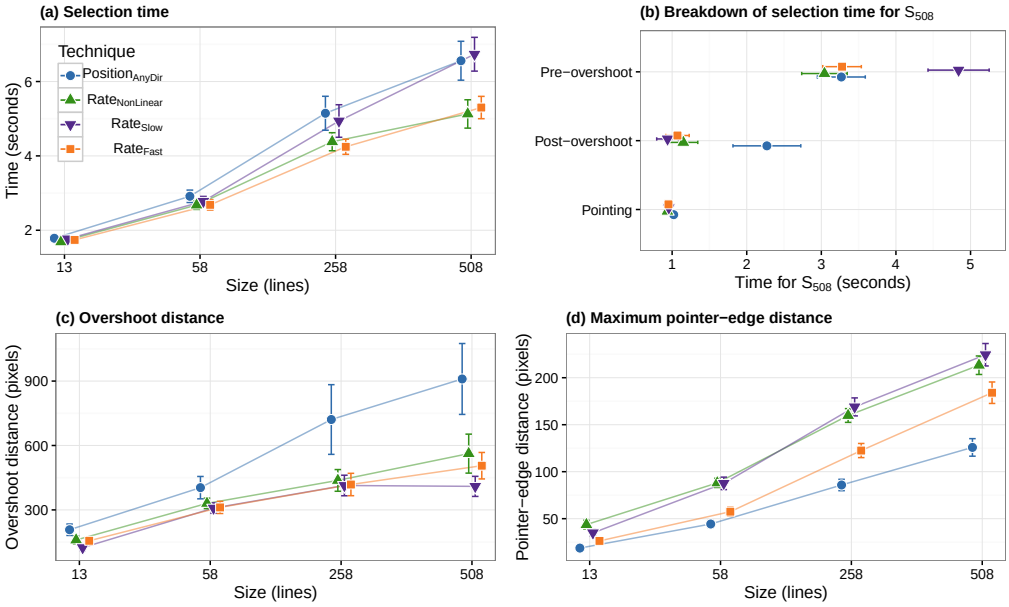

Figure 14: (a) Selection time, (b) breakdown of selection time for S_{508} into pre-overshoot and post-overshoot scrolling times and pointing time, (c) overshoot distance, and (d) maximum pointer-edge distance, by size and technique. Error bars show 95% confidence intervals.

the fourth tick of the *Tracking Speed* slider in the mouse preference pane of the System Preferences application[8].

6.2. Results

The dependent variables were selection time, overshoot distance, and maximum pointer-edge distance. We removed error trials (3.0%) in which the selection did not contain exactly the blue framed portion of text. The first repetition for each distance was also removed, as we observed that participants often failed to notice that the condition had changed. For all subsequent repeated-measures ANOVAs, we corrected the degrees of freedom using Greenhouse-Geisser estimates of sphericity when the assumption of sphericity was violated, and we used Bonferroni correction for pairwise comparisons.

[8]This system pointing transfer function can be replicated [49] using the following libpointing URI: `osx:?setting=0.6875`.

6.2.1. Selection time

Selection time is the main dependent variable and is defined as the time taken to select the target text, from the first time the pointer moved with the left mouse button pressed to the moment the left mouse button was released.

A repeated-measures ANOVA revealed a significant effect of Block on selection time ($F_{2,34}$=14.5, p<.001, η_p^2=.46). Pairwise comparisons showed a significant decrease of selection time between the first block (4.2s) and the two remaining (block 2: 3.8s, p=.004; block 3: 3.7s, p=.001), indicating a learning effect. We thus removed the first block from subsequent analyses.

We found a significant main effect of Technique ($F_{1.9,32.8}$=8.2, p=.001, η_p^2=.33), of Size ($F_{1.2,20.0}$=101.5, p<.001, η_p^2=.86), and a significant Technique×Size interaction ($F_{3.3,55.7}$=7.0, p<.001, η_p^2=.29) on selection time (Figure 14a).

Pairwise comparisons showed no significant difference in selection time between Technique for S_{13}, S_{58}, and S_{258}. However, for S_{508}, selection time was significantly lower with $\text{Rate}_{\text{NonLinear}}$ (5.1s) and $\text{Rate}_{\text{Fast}}$ (5.3s) than with $\text{Rate}_{\text{Slow}}$ (6.7s; both p<.001) and $\text{Position}_{\text{AnyDir}}$ (6.6s; resp. p=.007 and p=.028).

When selecting short portions of text, all techniques have similar performance. With long selections, however, $\text{Rate}_{\text{Fast}}$ and $\text{Rate}_{\text{NonLinear}}$ were markedly faster than $\text{Position}_{\text{AnyDir}}$ and $\text{Rate}_{\text{Slow}}$ (up to 28% faster).

$\text{Rate}_{\text{Slow}}$ and $\text{Position}_{\text{AnyDir}}$ have similar selection time for different reasons. We broke down the scrolling part of selection time around the moment the user first overshot the target line, yielding *pre-overshoot* and *post-overshoot* scrolling times (Figure 13b). Therefore, selection time can be expressed as:

$$T_{\text{selection}} = T_{\text{pre-overshoot}} + T_{\text{post-overshoot}} + T_{\text{pointing}}$$

As Figure 13b suggests, with $\text{Rate}_{\text{Slow}}$, $T_{\text{pre-overshoot}}$ was longer than with other techniques and $T_{\text{post-overshoot}}$ was similar, but the converse was true with $\text{Position}_{\text{AnyDir}}$. In other words, the low performance of $\text{Rate}_{\text{Slow}}$ in long distance selections is likely due to its

limited maximum velocity (3000 px/s, compared to 6000 px/s with $\text{Rate}_{\text{Fast}}$ and 9000 px/s with $\text{Rate}_{\text{NonLinear}}$, as shown in Figure 13c). However, the low performance of $\text{Position}_{\text{AnyDir}}$ in long distance selections could be best explained by its tendency to induce overshooting.

6.2.2. Overshoot distance

To better understand overshooting, we measured overshoot distance, the distance between the end of the target text and the farthest selection endpoint reached by the participant in a trial.

There was a significant main effect of Technique ($F_{1.2,19.9}$=7.8, p=.009, η_p^2=.32), Size ($F_{1.4,23.4}$=32.9, p<.001, η_p^2=.66), and a significant Technique×Size interaction ($F_{2.1,35.7}$=3.4, p=.041, η_p^2=.17) on overshoot distance (Figure 14c).

Pairwise comparisons showed that, for S_{13}, overshoot distance was significantly higher with $\text{Position}_{\text{AnyDir}}$ than with $\text{Rate}_{\text{Fast}}$ (208px vs. 156px; p=.043). More importantly, for S_{13}, S_{58}, and S_{508}, it was significantly higher with $\text{Position}_{\text{AnyDir}}$ than with $\text{Rate}_{\text{Slow}}$ (S_{13}: 208px vs. 124px, p<.001; S_{58}: 404px vs. 308px, p=.040; 910px vs. 410px, S_{508}: p=.020).

$\text{Position}_{\text{AnyDir}}$ is especially sensitive to overshooting, likely because pointer movements in any direction generate scrolling movements: when the target line is brought into view, users move the pointer back to the viewport, generating more scrolling than necessary. In particular, with the longest sizes, overshoot distance exceeded the height of the viewport with $\text{Position}_{\text{AnyDir}}$, compelling participants to correct their movement by scrolling backwards before pointing to the target line.

6.2.3. Maximum pointer-edge distance

To analyse the use of the scrolling area, we measured the maximum pointer-edge distance reached by the participant in a trial.

We found a significant main effect of Technique ($F_{1.9,31.8}$=41.5, p<.001, η_p^2=.71), Size ($F_{1.4,23.4}$=307.2, p<.001, η_p^2=.95), and a significant Technique×Size interaction ($F_{3.4,56.9}$=11.4, p<.001, η_p^2=.40) on maximum pointer-edge distance (Figure 14d).

Surprisingly, with the rate-based techniques, participants did not maximise their use of the scrolling area. Participants seldom used the last third of the area, especially with $\text{Rate}_{\text{Fast}}$, which produces faster scrolling velocities with sub-200px movements.

In addition, participants stayed closer to the viewport edge with $\text{Position}_{\text{AnyDir}}$ than with other techniques. Presumably, staying close to the viewport edge allowed participants to limit overshooting when going back inside the view. However, despite using only the first third of the scrolling area, participants still managed to quickly advance scrolling position by oscillating the pointer horizontally.

6.2.4. Task workload assessments

Friedman tests on the NASA TLX subscale scores showed a significant main effect of Technique on *mental demand* ($\chi^2(3)=29.3$, $p<.001$), *physical demand* ($\chi^2(3)=26.5$, $p<.001$), *performance* ($\chi^2(3)=27.1$, $p<.001$), *effort* ($\chi^2(3)=27.6$, $p<.001$), and *frustration* ($\chi^2(3)=25.2$, $p<.001$).

Pairwise comparisons showed that $\text{Position}_{\text{AnyDir}}$ was perceived as more mentally and physically demanding, as well as more effortful, than all other techniques ($p<.017$). This is likely due to the need to closely attend the behaviour to reduce overshooting.

6.2.5. Discussion

The main finding of this experiment is that there are substantial performance differences between edge-scrolling implementations.

Results suggest that mouse-based position control edge-scrolling techniques have high perceived workload, in line with the findings of Malacria et al. [28] for push-edge and slide-edge scrolling, two other position control techniques. However, unlike push-edge and slide-edge, overshooting is exacerbated when pointer movements in any direction generate scrolling movements.

Results also show that the slope of rate-based transfer functions influences performance. Of the three rate-based methods, $\text{Rate}_{\text{Fast}}$ and $\text{Rate}_{\text{NonLinear}}$ decreased selection time by up to 28% compared to $\text{Rate}_{\text{Slow}}$, which has a lower slope constant. Results did not show a performance difference between $\text{Rate}_{\text{Fast}}$ and $\text{Rate}_{\text{NonLinear}}$, despite $\text{Rate}_{\text{NonLinear}}$ being

able to produce 50% higher scrolling velocities. Further experiments are necessary to determine whether higher scrolling velocities give benefits for even longer selections than the longest examined in our study. However, note that the scrolling area was underused past 250 pixels, and the range of scrolling velocities that could be produced with the these rate-based techniques was rarely used fully.

7. General discussion

This analysis of edge-scrolling has exposed important design considerations in the user's task and in the behavioural characteristics of its implementation. We demonstrated that contemporary methods of supporting edge-scrolling vary substantially, and that these variations raise problems for users particularly in perceived inconsistency and impaired control. We also experimentally confirmed that substantial differences in performance and perceived workload are induced by the different behaviours.

This section provides general recommendations for edge-scrolling technique design, and discusses limitations in the studies and opportunities for further work.

7.1. Design recommendations

Based on the results of the previous reverse-engineering, survey, and experiment studies, a number of recommendations can be made on control, performance, as well as consistency.

Analysis of survey comments showed that all behavioural characteristics of edge-scrolling techniques were involved in users' perception of control.

R1. A small *scrolling area* can negatively impact control, both because it does not offer a large range of controllable values, and because pointing at them is difficult when their boundaries are not clearly *visually* delineated.

However, note that participants in the experiment seldom moved further than 250 pixels (70 mm).

R2. Ensure the *transfer function* permits an adequate range of output values to be attained in all possible configurations of the scrolling area.

Highly constrained transfer functions that allow only a small range of output values to be produced impede user control.

R3. *Activation conditions* based on a delay can be detrimental to the sense of control when there is no *visual guidance* to help anticipate activation.

R4. Always provide *visual feedback* on the user's location in the document during edge-scrolling to mitigate some of the spatial awareness issues reported by survey participants.

This could happen when the scrollbar is not visible, for example because it is placed behind another window or lies outside the display. In the experiment, both the current scrolling position and the target text were clearly displayed in scrollbars.

Insufficient control may distract users from their task and increase the risks of overshooting and disorientation, leading ultimately to suboptimal performance. Performance is also influenced by the transfer function.

R5. Linear rate control *transfer functions* with slopes less than 10 ($\text{NSTextView}_{\text{select}}$) may not yield optimal performance for long distances.

Experiment results suggest that users are not likely to move the pointer far enough away from the viewport edge to attain suitably high scrolling velocities. A linear rate-based transfer function with a slope constant of 20 ($\text{Safari}_{\text{select}}$; $\text{Rate}_{\text{Fast}}$ in the experiment) apparently provides a good compromise between accessibility of high scrolling velocities and fidelity of control opportunity.

R6. If deploying a position control *transfer function*, pointer movements towards the viewport should not produce scrolling displacements.

As seen in the experiment, this behaviour is likely to cause overshooting, decreasing performance and requiring more user attention. Lateral pointer movements have

been used by our participants to momentarily accelerate scrolling, but designers should be wary of diagonal movements towards the viewport, as they may cause the same problems.

Examination of existing implementations showed marked behavioural inconsistencies that were sometimes problematic, as commented by survey respondents.

R7. Unfamiliar or inconsistent *activation conditions*, *scrolling area*, or *transfer function* are noticed by users and can induce general confusion or distrust of edge-scrolling.

Consider using *visual guidance* to help users understand and control idiosyncratic behaviours.

R8. Rate-control mechanisms are far more common than position control, so consider the consistency impact before deploying a position control method.

Based on our current results, for a basic behaviour with acceptable control, performance, and external consistency, we recommend that developers implement a simple linear rate control transfer function such as that of Safari, with a slope of 20. In its current implementation, the viewport is automatically scrolled by x pixels every 50 milliseconds (x being the pointer-edge distance).

In addition, toolkit developers should provide acceptable default edge-scrolling behaviours, together with ways for individual application designers to adapt them to the task at hand if needed. In most situations, the toolkit has enough information to provide a sensible behaviour even when using uncommon input parameters (e.g., of the document length or font size). Ultimately, both users and application developers would benefit from this way of promoting consistency.

7.2. Future work

The analysis presented in this paper provides a foundation for further work on edge-scrolling. The design space identified the possibility for many edge-scrolling configurations and parameters, but the survey and empirical analysis only covered a small proportion

of these. In particular, our experiment was limited to the unambiguous intra-viewport *select* task using a display-bound scroll area.

Edge-scrolling behaviours for *move* tasks present a more challenging design space than those of *select*. This additional complexity stems from ambiguity about the target viewport – possibly the user intends to drag the object to a new location within the current viewport, but possibly the target resides in a different viewport. This ambiguity places constraints on the pragmatic extent of the scrolling area, and it also elevates the potential for accidental activation of edge-scrolling. Further work is needed to empirically examine user performance across different solutions for *move* tasks, and to understand any conflicts between behaviours for *move* and *select*.

While our studies focused on desktop applications using indirect pointing with a mouse, edge-scrolling behaviours are also used with direct touch-based input, particularly on mobile devices such as tablets and smartphones. For example, application icons can be dragged between home-screen pages, and list items can be repositioned using edge-scrolling. The relative importance of edge-scrolling design factors might well be very different from desktop applications. For instance, more tasks are expected to be intra-viewport due to the often single-windowed environment, increasing risk of small scrolling areas that compete with other interface elements for screen real-estate. Further work is needed to examine the implications of edge-scrolling in direct interaction contexts.

Finally, edge-scrolling reveals an instance of a more fundamental problem in input device research. The rate control transfer functions used by most existing designs is usually best controlled using isometric input devices [50] such as a trackpoint. However, edge-scrolling is inherently based on the pointing device, which is usually isotonic and lacks a self-centering effect, thus requiring explicit effort to reach the zero-velocity point [51]. Previous edge-scrolling research [28] already detailed this curious state of affair and showed that position control alternatives could improve edge-scrolling performance compared to rate control when using isotonic pointing devices, at the expense of physical effort. The recent availability of pressure-sensitive pointing devices in consumer hardware might

steer future research in the opposite direction and encourage the study of rate-based edge-scrolling using pressure control.

8. Conclusion

We proposed and studied a design space for edge-scrolling, a technique for navigating a viewport by pointing near its edge that has been widely implemented in desktop systems. By reverse-engineering these implementations, we exposed substantial variance in design approaches. Responses to an interactive survey demonstrated that edge-scrolling is broadly used and that users are aware of problems stemming from lack of control and perceived inconsistencies. We then reported results of a controlled experiment comparing four designs in a one-dimensional text selection task, with results showing that current solutions lead to marked differences in user performance and perceived workload. Finally, from these combined results we have derived recommendations to support future edge-scrolling designs.

[1] S. Zhai, B. A. Smith, T. Selker, Improving Browsing Performance: A study of four input devices for scrolling and pointing tasks, paper, in: Proceedings of INTERACT97: The Sixth IFIP Conference on Human-Computer Interaction, Sydney, Australia, Jul, 14–18, 1997.

[2] K. Hinckley, E. Cutrell, S. Bathiche, T. Muss, Quantitative analysis of scrolling techniques, in: Proceedings of CHI '02, ACM, ISBN 1-58113-453-3, 65–72, doi:10.1145/503376.503389, URL `http://doi.acm.org/10.1145/503376.503389`, 2002.

[3] P. Quinn, A. Cockburn, G. Casiez, N. Roussel, C. Gutwin, Exposing and understanding scrolling transfer functions, in: Proceedings of UIST '12, ACM, ISBN 978-1-4503-1580-7, 341–350, doi:10.1145/2380116.2380161, URL `http://doi.acm.org/10.1145/2380116.2380161`, 2012.

[4] I. E. Sutherland, Sketchpad: A Man-machine Graphical Communication System, in: Proceedings of the May 21-23, 1963, Spring Joint Computer Conference, AFIPS '63 (Spring), ACM, New York, NY, USA, 329–346, doi:10.1145/1461551.1461591, URL `http://doi.acm.org/10.1145/1461551.1461591`, 1963.

[5] H. McLoone, K. Hinckley, E. Cutrell, Bimanual Interaction on the Microsoft Office Keyboard, in: Proceedings of INTERACT'03, 49–56, 2003.

[6] K. W. Arthur, N. Matic, P. Ausbeck, Evaluating touch gestures for scrolling on notebook computers, in: CHI'08 Extended Abstracts on Human Factors in Computing Systems, ACM, 2943–2948, 2008.

[7] D. Bial, F. Block, H. Gellersen, A study of two-handed scrolling and selection on standard notebook computers, in: Proceedings of the 24th BCS Interaction Specialist Group Conference, British Computer Society, 355–364, 2010.

[8] K. Ohno, K.-i. Fukaya, J. Nievergelt, A five-key mouse with built-in dialog control, ACM SigChi Bulletin 17 (1) (1985) 29–34.

[9] S. Zhai, B. A. Smith, T. Selker, Dual stream input for pointing and scrolling, in: CHI'97 Extended Abstracts on Human Factors in Computing Systems, ACM, 305–306, 1997.

[10] K. Hinckley, M. Sinclair, Touch-sensing input devices, in: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press, doi:10.1145/302979.303045, URL `http://dx.doi.org/10.1145/302979.303045`, 1999.

[11] D. Aliakseyeu, P. Irani, A. Lucero, S. Subramanian, Multi-flick: An Evaluation of Flick-based Scrolling Techniques for Pen Interfaces, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08, ACM, New York, NY, USA, 1689–1698, 2008.

[12] P. Quinn, S. Malacria, A. Cockburn, Touch Scrolling Transfer Functions, in: Proceedings of UIST '13, ACM, ISBN 978-1-4503-2268-3, 61–70, doi:10.1145/2501988.2501995, URL http://doi.acm.org/10.1145/2501988.2501995, 2013.

[13] M. Kumar, T. Winograd, Gaze-enhanced scrolling techniques, in: Proceedings of UIST '07, ACM, ISBN 978-1-59593-679-0, 213–216, doi:10.1145/1294211.1294249, URL http://doi.acm.org/10.1145/1294211.1294249, 2007.

[14] S. Mehra, P. Werkhoven, M. Worring, Navigating on handheld displays: Dynamic versus static peephole navigation, ACM Transactions on Computer-Human Interaction (TOCHI) 13 (4) (2006) 448–457.

[15] J. Rekimoto, Tilting operations for small screen interfaces, in: Proceedings of the 9th annual ACM symposium on User interface software and technology, ACM Press, doi:10.1145/237091.237115, URL http://dx.doi.org/10.1145/237091.237115, 1996.

[16] W. A. S. Buxton, B. Myers, A Study in Two-handed Input, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '86, ACM, New York, NY, USA, 321–326, 1986.

[17] S. Fitchett, A. Cockburn, MultiScroll: using multitouch input to disambiguate relative and absolute mobile scroll modes, in: Proceedings of the 24th BCS Interaction Specialist Group Conference, British Computer Society, 393–402, 2010.

[18] N. Roussel, G. Casiez, J. Aceituno, D. Vogel, Giving a hand to the eyes: leveraging input accuracy for subpixel interaction, in: Proceedings of UIST '12, ACM, ISBN 978-1-4503-1580-7, 351–358, doi:10.1145/2380116.2380162, URL http://doi.acm.org/10.1145/2380116.2380162, 2012.

[19] C. Appert, J.-D. Fekete, OrthoZoom scroller: 1D multi-scale navigation, in: Proceedings of CHI '06, ACM, ISBN 1-59593-372-7, 21–30, doi:10.1145/1124772.1124776, URL http://doi.acm.org/10.1145/1124772.1124776, 2006.

[20] E. L. Hutchins, J. D. Hollan, D. A. Norman, Direct manipulation interfaces, Hum.-Comput. Interact. 1 (4) (1985) 311–338, ISSN 0737-0024, doi:10.1207/s15327051hci0104_2, URL http://dx.doi.org/10.1207/s15327051hci0104_2.

[21] J. Zhao, R. W. Soukoreff, X. Ren, R. Balakrishnan, A model of scrolling on touch-sensitive displays, International Journal of Human-Computer Studies 72 (12) (2014) 805 – 821.

[22] M. Nancel, D. Vogel, E. Lank, Clutching Is Not (Necessarily) the Enemy, in: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15, ACM, New York, NY, USA, ISBN 978-1-4503-3145-6, 4199–4202, doi:10.1145/2702123.2702134, URL http://doi.acm.org/10.1145/2702123.2702134, 2015.

[23] T. Moscovich, J. F. Hughes, Navigating documents with the virtual scroll ring, in: Proceedings of UIST '04, ACM, New York, NY, USA, ISBN 1-58113-957-8, 57–60, doi:10.1145/1029632.1029642, URL http://doi.acm.org/10.1145/1029632.1029642, 2004.

[24] G. M. Smith, m. c. schraefel, The radial scroll tool: scrolling support for stylus- or touch-based document navigation, in: Proceedings of UIST '04, ACM, ISBN 1-58113-957-8, 53–56, doi:10.1145/1029632.1029641, URL http://doi.acm.org/10.1145/1029632.1029641, 2004.

[25] S. Malacria, E. Lecolinet, Y. Guiard, Clutch-free panning and integrated pan-zoom control on touch-sensitive surfaces: the cyclostar approach, in: Proceedings of CHI '10, ACM, ISBN 978-1-60558-929-9, 2615–2624, doi:http://doi.acm.org/10.1145/1753326.1753724, URL http://doi.acm.org/10.1145/1753326.1753724, 2010.

[26] H. D. Jellinek, S. K. Card, Powermice and user performance, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 213–220, 1990.

[27] A. Cockburn, P. Quinn, C. Gutwin, S. Fitchett, Improving scrolling devices with document length dependent gain, in: Proceedings of CHI '12, ACM, ISBN 978-1-4503-1015-4, 267–276, doi:10.1145/2207676.2207714, URL http://doi.acm.org/10.1145/2207676.2207714, 2012.

[28] S. Malacria, J. Aceituno, P. Quinn, G. Casiez, A. Cockburn, N. Roussel, Push-Edge and Slide-Edge: Scrolling by Pushing Against the Viewport Edge, in: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15, ACM, New York, NY, USA, ISBN 978-1-4503-3145-6, 2773–2776, doi:10.1145/2702123.2702132, URL http://doi.acm.org/10.1145/2702123.2702132, 2015.

[29] M. S. Sanders, E. J. McCormick, Human Factors in Engineering and Design, McGraw-Hill, Inc., seventh edn., 1992.

[30] M. Beaudouin-Lafon, Instrumental interaction: an interaction model for designing post-WIMP user interfaces, in: Proceedings of the SIGCHI conference on Human Factors in Computing Systems, ACM, 446–453, 2000.

[31] M. Baglioni, S. Malacria, E. Lecolinet, Y. Guiard, Flick-and-brake: Finger Control over Inertial/Sustained Scroll Motion, in: CHI '11 Extended Abstracts on Human Factors in Computing

Systems, CHI EA '11, ACM, New York, NY, USA, ISBN 978-1-4503-0268-5, 2281–2286, doi: 10.1145/1979742.1979853, URL http://doi.acm.org/10.1145/1979742.1979853, 2011.

[32] T. Igarashi, K. Hinckley, Speed-dependent automatic zooming for browsing large documents, in: Proceedings of UIST '00, ACM, ISBN 1-58113-212-3, 139–148, doi:10.1145/354401.354435, URL http://doi.acm.org/10.1145/354401.354435, 2000.

[33] M. Kobayashi, T. Igarashi, MoreWheel: Multimode scroll-wheeling depending on the cursor location, in: Adjunct proceedings of UIST, vol. 6, 15–18, 2006.

[34] W. A. S. Buxton, There's more to interaction than meets the eye: Some issues in manual input, chap. There's More to Interaction than Meets the Eye: Some Issues in Manual Input, 319–337, 1987.

[35] A. Leganchuk, S. Zhai, W. A. S. Buxton, Manual and Cognitive Benefits of Two-handed Input: An Experimental Study, ACM Trans. Comput.-Hum. Interact. 5 (4) (1998) 326–359.

[36] Y. Guiard, M. Beaudouin-Lafon, J. Bastin, D. Pasveer, S. Zhai, View size and pointing difficulty in multi-scale navigation, in: Proceedings of the working conference on Advanced visual interfaces, ACM, 117–124, 2004.

[37] C. Appert, O. Chapuis, M. Beaudouin-Lafon, Evaluation of pointing performance on screen edges, in: Proceedings of the working conference on Advanced visual interfaces, ACM Press, doi:10.1145/1385569.1385590, URL http://dx.doi.org/10.1145/1385569.1385590, 2008.

[38] J. Accot, S. Zhai, More than dotting the i's—foundations for crossing-based interfaces, in: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, 73–80, 2002.

[39] J. R. Meier, J. W. Sullivan, P. Mercer, Intelligent scrolling, URL http://www.google.com/patents/US5196838, 1993.

[40] D. D. Bardon, R. E. Berry, S. L. Martin, S. A. Morgan, J. M. Mullay, C. A. Swearingen, Method for Auto-Scroll with Greater User Control, IBM Technical Disclosure Bulletin 40 (6) (1997) 181–182, URL http://ip.com/IPCOM/000118763.

[41] A. Kwatinetz, Scrolling contents of a window, URL http://www.google.com/patents/US5495566, 1996.

[42] R. E. Berry, S. S. Fleming, A. C. Temple, Auto-Scroll During Direct Manipulation, IBM Technical Disclosure Bulletin 33 (11) (1991) 312.

[43] S.-G. Li, T. J. L. Shrader, Scrolling a target window during a drag and drop operation, URL http://www.google.com/patents/US5740389, 1998.

[44] J. D. Belfiore, C. J. Guzak, C. E. Graham, S. M. Madigan, T. W. Trower, II, R. L. Kerr, A. M. Wyard, Auto-scrolling during a drag and drop operation, URL http://www.google.com/patents/US5726687, 1998.

[45] J. A. Johnson, A comparison of user interfaces for panning on a touch-controlled display, in: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press/Addison-Wesley Publishing Co., 218–225, 1995.

[46] V. Kaptelinin, A Comparison of Four Navigation Techniques in a 2D Browsing Task, in: Conference Companion on Human Factors in Computing Systems, CHI '95, ACM, New York, NY, USA, 282–283, 1995.

[47] A. Kulik, J. Dittrich, B. Froehlich, The Hold-and-move Gesture for Multi-touch Interfaces, in: Proceedings of MobileHCI '12, ACM, ISBN 978-1-4503-1105-2, 49–58, doi:10.1145/2371574.2371583, URL http://doi.acm.org/10.1145/2371574.2371583, 2012.

[48] V. Braun, V. Clarke, Using thematic analysis in psychology, Qualitative Research in Psychology 3 (2) (2006) 77–101, doi:10.1191/1478088706qp063oa, URL http://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa.

[49] G. Casiez, N. Roussel, No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions, in: Proceedings of UIST '11, ACM, ISBN 978-1-4503-0716-1, 603–614, doi:10.1145/2047196.2047276, URL http://doi.acm.org/10.1145/2047196.2047276, 2011.

[50] S. Zhai, Human performance in six degree of freedom input control, Ph.D. thesis, University of Toronto, 1995.

[51] W. Kim, F. Tendick, S. Ellis, L. Stark, A comparison of position and rate control for telemanipulations with consideration of manipulator system dynamics, IEEE Journal of Robotics and Automation 3 (5).

Jonathan Aceituno is a R&D engineer at Aodyo Instruments. His research interests are in the design and evaluation of desktop user interfaces and digital musical instruments. He completed his PhD in the Inria Mjolnir team, where he focused on direct and expressive interaction on the desktop.

Sylvain Malacria is a junior researcher in the Mjolnir Team at Inria. He is a computer scientist with an interest in designing novel interaction techniques that help end-users to develop skills and expertise with computer systems, and that include novel methods for selecting commands and navigating in large digital environments.

Philip Quinn is a PhD student at the University of Canterbury, Christchurch, New Zealand. His research interests include human factors measurement and modelling.

Nicolas Roussel is a senior researcher at Inria and the scientific head of the Inria Mjolnir team. His research interests are in human-computer interaction and include computer-mediated communication and groupware, engineering of interactive systems, graphical interaction, and tactile and gestural interaction.

Andy Cockburn is a full professor in the Department of Computer Science of the University of Canterbury, Christchurch, New Zealand. He is a computer scientist with an interest in modelling and empirically measuring human performance with interactive systems.

Géry Casiez is a professor of Computer Science at Université Lille 1 and a member of the Inria Mjolnir research team. His research interests include 2D and 3D interaction, haptic interaction, and the empirical evaluation of user interfaces, including associated metrics and predictive models of human performance.