Push-Edge and Slide-Edge: Scrolling by Pushing Against the Viewport Edge

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

Push-Edge and Slide-Edge: Scrolling by Pushing Against the Viewport Edge

Sylvain Malacria[1,2], **Jonathan Aceituno**[2,3], **Philip Quinn**[4], **Géry Casiez**[2,3], **Andy Cockburn**[4], **Nicolas Roussel**[2]

[1]University College London, United Kingdom, [2]Inria Lille, France
[3]University of Lille, France, [4]University of Canterbury, New Zealand

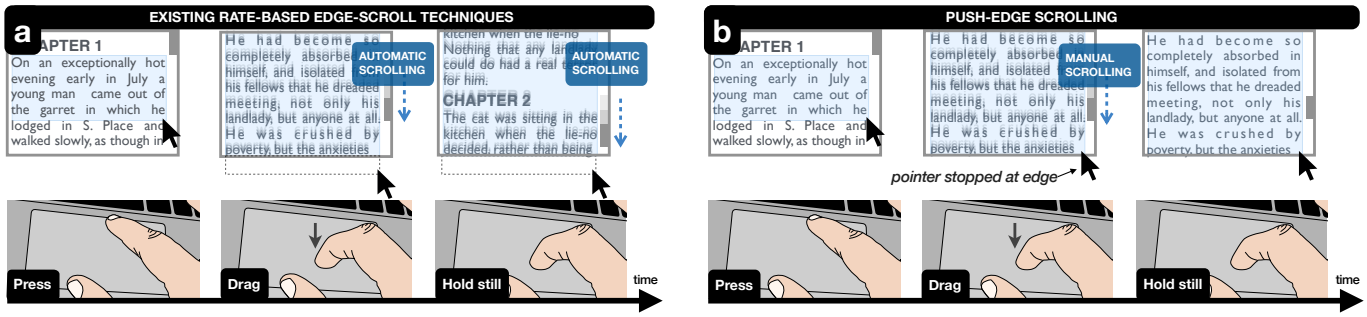

Figure 1. When selecting text with a touchpad, downward movements after crossing the viewport edge will (a) change the rate of automatic scrolling with existing techniques (rate control); or (b) manually scroll the document, stopping the pointer at the edge with *push-edge scrolling* (position control).

ABSTRACT

Edge-scrolling allows users to scroll a viewport while simultaneously dragging near or beyond a window's edge. Common implementations rely on rate control, mapping the distance between the pointer and the edge of the viewport to the scrolling velocity. While ubiquitous in operating systems, edge-scrolling has received little attention, even though previous works suggest that (1) rate control may be suboptimal for isotonic pointing devices like mice and trackpads and (2) space beyond the window's edge might be scarce, limiting scrolling control. To address these problems, we developed Push-edge scrolling (and Slide-edge scrolling, its inertial variant), two novel position-based techniques that allow scrolling by 'pushing' against the viewport edge. A controlled experiment shows that our techniques reduce overshoots and offer performance improvements by up to 13% over traditional edge-scrolling.

Author Keywords

Edge-scroll; rate control; position control; inertia; autoscroll.

ACM Classification

H.5.2 [**User Interfaces**] Interaction styles.

INTRODUCTION

Scrolling while dragging an object is a common interaction problem when working within the confines of a viewport. Dragging is achieved by moving an input device in a special state (e.g., holding down a mouse button) – which allows actions such as moving file icons and selecting regions of text. Dragging tasks are straightforward when the start and end locations are both contained within the viewport, but many activities require scrolling to bring the end location into view – for example, selecting several paragraphs of text, or manipulating data in a large spreadsheet. The dragging state precludes using scrollbars or other on-screen navigation tools, and device-specific mechanisms, like the mouse wheel, are not always responsive during dragging.

To allow scrolling while dragging, interfaces support *edge-scroll* (also called *autoscroll*) that triggers a scrolling motion when a dragged pointer enters a *control area* near the edge of the viewport. For example, in Figure 1a the user first presses the trackpad button with the thumb, then uses another finger to drag the pointer downwards and select a text region. When the pointer enters the control area (typically the region between the window border and the screen edge), the document starts scrolling until the pointer leaves this area or the user exits the dragging mode. The most common approach to control edge scroll is rate-based: inside the control area, scrolling velocity is based on the pointer's distance from the viewport edge [2].

While edge-scroll is commonplace and designs have been disclosed [7, 8], we are unaware of any empirical work, which is surprising because current techniques suffer from two major limitations. First, prior research on pointing and scrolling suggests that rate control may be inappropriate for edge-scrolling [11, 12]. Second, the size of the control area is inconsistent and possibly limited, for instance when the window is maximised or with small displays, offering suboptimal control.

Based on these limitations, we introduce *push-edge scrolling* (Figure 1b) and its inertial variant, *slide-edge scrolling*. These

CHI 2015, April 18 - 23 2015, Seoul, Republic of Korea

ACM 978-1-4503-3145-6/15/04...$15.00
http://dx.doi.org/10.1145/2702123.2702132

novel *position*-based techniques block the pointer as it reaches the edge of the viewport, and transform subsequent pointing device movements that would otherwise move the pointer into scrolling displacements. Experimental results demonstrate that *push-edge* and *slide-edge scrolling* outperform rate-based techniques in a text selection task.

LIMITATIONS IN CURRENT EDGE-SCROLL TECHNIQUES

Edge-scrolling should allow users to quickly scroll to close or distant target locations, but there are two main reasons to question the adequacy of current designs with respect to these goals: rate control may not be superior to position control with common input devices, and window placement can limit scrolling control because it constrains the control area.

Mouse and Trackpad Rate Control

Prior research suggests that rate-based methods for scrolling control are most desirable when the user's primary task is reading because they facilitate a smooth and continuous scrolling speed [12], or when input is provided through an isometric input device such as a trackpoint [11]. In contrast, isotonic devices, such as mouse or trackpads, are less suited to rate control because they lack a self-centring effect, requiring fine coordination to control movement and to explicitly get back to zero point to stop it [6]. In summary, there is support for exploring position-based alternatives to rate-based edge-scroll. Such a mechanism for normal scrolling was patented [4], scrolling a view depending on the distance between the cursor and an activation point. However, the distance-based nature of its transfer function makes it hard to precisely control scrolling velocity. Moreover, scrolling has to be explicitly activated by pressing a button, unlike edge-scrolling which is as a direct extension of the dragging action.

Control Area Size and Location

Current edge-scroll methods rely on the availability of a control area outside the viewport. Scrolling begins when a dragged pointer enters this area, and the scroll velocity increases as the pointer moves away from the viewport's edge. Therefore, the actual size of the control area limits the maximum scrolling velocity in rate control. If the control area lies inside the viewport, it has to be small enough to avoid unintended scrolling, at the expense of the granularity of control. A common alternative is to expand the control area from the viewport edge so that it takes up the whole display space beyond. However, available screen space is likely scarce when maximised windows or small displays, making the control area smaller. Either way, a small control area is likely to make rate-based edge scrolling hard to control for the user.

When using position control, it is possible to overcome this problem by blocking the pointer when it crosses the edge [1] so that every subsequent movement will scroll the view instead of moving the pointer, effectively having a zero-sized control area that does not impede scrolling control.

PUSH-EDGE AND SLIDE-EDGE SCROLLING

Push-edge scrolling is designed to overcome the above limitations and permit rapid and accurate scrolling while a dragging state is maintained. It operates by 'capturing' the cursor at the viewport edge, and mapping subsequent device signals that would otherwise move the cursor as input to a scrolling transfer function (Figure 1b) – further movement away from the viewport results in scrolling displacements of the view, while the cursor remains captured in a fixed position at the edge. The captured cursor is released either by moving it back towards the viewport, by releasing the dragging state, or when scrolling reached the end of the document.

However, the range of a pointing movement – from a pixel to a screen – is much narrower than the range of a scrolling movement – from a few lines to hundreds of pages [3, 5]. Thus, using raw pointing device displacements only might not scale to long distance scrolling. Consequently, *push-edge scrolling* converts motor-space device displacements d (in metres) into display-space scrolling displacements D (in pixels) by multiplying d with a unitless gain produced by a *sigmoid* transfer function [10] and converting the result in pixels using the pixel density of the display. The transfer function transforms the device velocity v (in metres per second) using a linear interpolation between constants g_{min} and g_{max} in the interval (v_1, v_2), and is clamped elsewhere.

Since the physical range of motion is different between devices, we used the results of a pilot study to calibrate the transfer function differently for a trackpad ($g_{min} = 1$, $g_{max} = 15$, $v_1 = 0.2$ m/s, $v_2 = 0.5$ m/s) and mouse ($g_{min} = 1$, $g_{max} = 10$, $v_1 = 0.1$ m/s, $v_2 = 0.5$ m/s).

Because the transfer function is not rate-based, it requires repeatedly lifting and repositioning the mouse or finger (*clutching*) to scroll very long distances. To reduce clutching-induced effort, we also designed *slide-edge scrolling*, a variant of push-edge that applies inertia to continue scrolling after the user stops operating the pointing device based on a simulation of residual momentum [9]. When the raw device velocity at liftoff v_l exceeds 0.15 m/s, friction progressively decreases velocity V over time according to the function $V(t) = V_l \times e^{-4t}$ – where t is the time elapsed since liftoff, in seconds.

EXPERIMENT

We conducted an experiment to compare performance, amount of control, and perceived workload between push-edge (Push), slide-edge (Slide), and a traditional rate-based edge-scroll (Rate) using a mouse and a trackpad in a one-dimensional text selection task. The primary hypothesis (**H1**) was that selection time would be lower for Push and Slide than for Rate; secondary hypotheses are: (**H2**) inertia (Slide) reduces selection time, (**H3**) Push and Slide offer better control and result in less overshoots than *Rate*, and (**H4**) inertia (Slide) reduces physical effort in long selections.

Method

The experiment was conducted on a 13″ Apple MacBook Pro Retina running OS X 10.9.4, with display resolution set to *Best* (1280×800 px, 116 ppi). Input was provided through the laptop's trackpad and a Logitech M90 optical mouse on a plywood desk. The pointing transfer function was set to the fourth tick of the *Tracking Speed* slider in the respective touchpad and mouse preference panels. Raw device input was

used for Push and Slide. Experimental software was written in Objective-C with the Cocoa API. Raw device input was monitored using the I/O Kit and Apple's private multitouch API. The Quartz Event Services API was used to lock the pointer. The software implemented push-edge and slide-edge scrolling, and it used Apple's default rate-based edge-scroll method for the *Rate* condition. It is defined in the `autoscroll:` method of the `NSClipView` class – which scrolls the viewport p pixels every 100 ms, where p is the pointer-to-edge distance.

Procedure, task, and design

Participants were instructed to perform a sequence of top-to-bottom text selections as quickly and accurately as possible. The window was displayed at the centre of the screen, 400 pixels tall, containing 21,480 lines of text typeset with *Ubuntu Mono Regular* 13px (Figure 2). For each trial, participants had to select a section of text framed and coloured blue – starting eight lines from the bottom of the viewport, and varying in size. Supposing that edge-scroll is mainly target-directed, we overlaid a gradient on the scrollbar as a hint of the number of lines participants had to select. As the task was one-dimensional, selecting anywhere on a line selected the entire line. Every selection required concurrently dragging and scrolling using the requested device and edge-scroll technique. Other scrolling methods and devices were disabled.

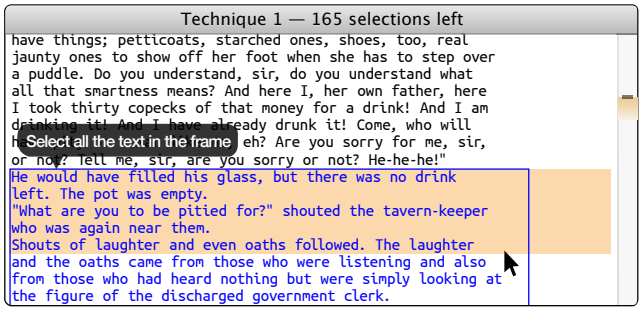

Figure 2. Experimental software for the text selection task.

To perform a selection, participants had to move the pointer to the top of the blue text, press the pointing device button to start the selection (and enter a dragging mode), and scroll downwards until they reached the bottom of the blue text. Once the pointer was positioned over the last line, they could release the button to complete the selection. If their selection did not exactly capture the blue text, an error was recorded, and the trial was repeated.

The experiment used a $2 \times 3 \times 3 \times 4$ within-subjects design for the factors: *device* (mouse and trackpad), *technique* (Push, Slide, and Rate), *block* (1-3, with the first block serving as opportunity for learning the new methods), and *size* (shortest: 15; short: 45; long: 135; longest: 405 lines of text to be selected). The order of *technique* and *device* was counterbalanced. *Sizes* were presented from the shortest to the longest, one for each *block*, with 5 consecutive repetitions for a given size within a block – for a total of 360 trials per participant. Participants completed NASA-TLX worksheets after each technique. The experiment lasted ~45 minutes.

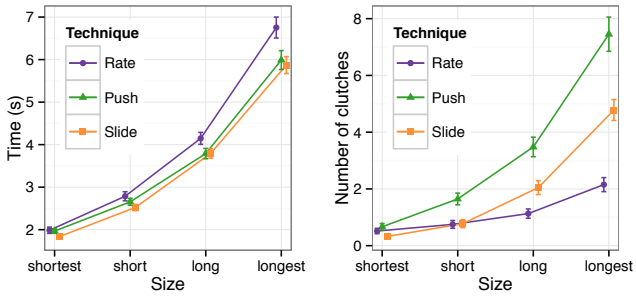

Figure 3. Mean trial time (left) and number of trackpad clutches (right) by size and technique (error bars denote 95% CI).

Participants

Twenty-four university staff and students (five female) participated in the experiment (mean age 30.4, $\sigma = 7.0$). Most used a trackpad as their primary pointing device (58%).

Results

Results for each of the dependent variables (*trial time*, *overshoot distance*, and *number of clutches*) are presented below. Error rate was not significantly different (p=.313) between Push (6.0%), Slide (5.7%), and Rate (6.9%). We excluded error trials from analyses. We also excluded the first repetition of each size, as changes were not immediately noticed.

Trial time

Trial time is the total text selection time, from the first mouse movement after the button was pressed, to the button release.

Repeated-measures ANOVA[1] revealed a significant effect of *block* on trial time ($F_{2,46} = 28.0$, $p < .001$, $\eta_p^2 = .55$; block 1: 4.0s, 2: 3.7s, 3: 3.6s), with a significant *technique*×*block* interaction ($F_{4,92} = 3.0$, p=.023, $\eta_p^2 = .11$) due to a learning of the new technique behaviour during the first block. As we are concerned with user performance after familiarization, the remaining analysis discards the first block.

There were significant main effects of *device* ($F_{1,23} = 28.7$, $p < .001$, $\eta_p^2 = .56$), *technique* ($F_{2,46} = 6.9$, p=.002, $\eta_p^2 = .23$), and *size* ($F_{1.1,25.6} = 360.8$, $p < .001$, $\eta_p^2 = .94$), and a significant *technique*×*size* interaction ($F_{2.4,55.3} = 4.2$, p=.014, $\eta_p^2 = .16$) on trial time. Post-hoc tests revealed that trial time was significantly lower for Slide than for Push (p=.009) or Rate (p=.006) with the shortest size, and than for Rate with the short size ($p < .004$). Trial time was also found significantly higher with Rate than with both Push and Slide for the long (respectively, p=.014 and p=.004) and longest (respectively, p=.029 and p=.003) sizes.

Importantly, Push and Slide consistently outperformed Rate by up to 13% in the longest size (Figure 3, left), supporting **H1**. There was no significant *device*×*technique* interaction (p=.405), suggesting that the benefits of Push are consistent across devices. Finally, inertia had no effect on performance as post-hoc tests revealed no significant difference between Push and Slide (p=.405), leading to rejection of **H2**.

[1] Greenhouse-Geisser corrections to the degrees of freedom were applied when sphericity was violated. Pairwise comparisons used Bonferroni correction.

Overshoot distance

Overshoot distance was measured as the maximum scroll distance beyond the target bounds that was reached during the trial. As expected, ANOVA confirmed that overshoot distance increases with *size* ($F_{1.4,31.0} = 106.1$, $p < .001$, $\eta_p^2 = .82$). There was also a significant effect of *Technique* ($F_{2,46} = 66.7$, $p < .001$, $\eta_p^2 = .74$), with Rate having the largest overshoot distance (369 px) and Push the least (262 px; Slide: 279 px); supporting **H3**. The increased overshooting with Rate likely stems from the need for the user to return the cursor to a location inside the window before scrolling stops after they notice the target has come into view.

Physical effort: clutching and perceived workload

We analysed the number of clutches used on the trackpad, assuming that frequent clutching indicates high physical workload. ANOVA showed significant main effects of *technique* ($F_{2,46} = 47.3$, $p < .001$, $\eta_p^2 = .67$) and *size* ($F_{1.3,30.7} = 172.4$, $p < .001$, $\eta_p^2 = .88$), as well as a significant *technique*×*size* interaction ($F_{2.2,50.2} = 28.2$, $p < .001$, $\eta_p^2 = .55$). As Rate allows continuous scrolling to be maintained without movement, it is unsurprising that it has significantly less clutching than Push. Pairwise comparisons between Slide and Push confirmed that the addition of inertia decreases clutching (Figure 3, right).

NASA-TLX responses showed *physical demand* to be lower with Rate than Push or Slide, for both the mouse (Friedman $\chi^2(2) = 17.2$, $p < .001$) and trackpad ($\chi^2(2) = 11.9$, p=.003). We also found that for the mouse, Slide had significantly higher perceived *effort* than Rate ($\chi^2(2) = 7.7$, p=.022), which probably stems from the difficulty of lifting the mouse while maintaining a high tangential velocity. Therefore, clutching and subjective data only partially support **H4**.

DISCUSSION

Our key finding is that push-edge and slide-edge scrolling both improved text selection time over a rate-based edge-scroll technique. The performance benefits were consistent for both trackpad and mouse pointing devices. Workload measures suggest that push-edge scrolling is best combined with mouse input, and slide-edge scrolling best with a trackpad.

It is likely that the actual performance benefits for push-edge and slide-edge scrolling over rate-based methods are understated with our results. Our experiment intentionally used a relatively small window centered on the screen, which is a 'best case' scenario for traditional techniques because of the large control area outside the window. When a window extends to the edge of the display (e.g., when maximised) the control area is compromised or unavailable, substantially impairing traditional methods – but there is no reason to anticipate any performance detriment in this case with our techniques.

The target of an edge-scrolling action while selecting text is consequently located in the scrolled viewport. However, there are situations where the scope of a dragging action may span multiple viewports (e.g. dragging a file between two windows) where moving the pointer beyond the viewport edge could be interpreted either as the continuation of the dragging action or as a request to activate edge-scroll. Current systems use various methods to distinguish these actions, such as waiting for a pointer to dwell over an interior control area in order to start scrolling. The relative merits of each approach, as well as their applicability to our techniques are left for future work.

Finally, one potential usability issue of our techniques is that they temporarily remove the user's direct control of the cursor as it is 'captured' by the window edge. The immediate feedback of seeing the viewport scroll should mitigate the user's surprise, but not if it reaches its terminus. To help visually communicate the captured cursor state, the techniques could apply a visual resistance effect to the viewport.

CONCLUSION

Edge-scroll is extensively used to enable scrolling while a concurrent dragging state is maintained. Most existing designs use rate control, where cursor position into a control area around the window edge controls scrolling velocity. We showed the potential performance limitations of current techniques and designed push-edge and slide-edge scrolling to work around them by allowing to scroll by "pushing against the viewport edge". Experimental results demonstrate that push-edge and slide-edge scrolling significantly improve performance.

ACKNOWLEDGMENTS

This work was supported by ANR (TurboTouch, ANR-14-CE24-0009). Thanks to UCLIC members, F. Chevalier, G. Bailly, T. Pietrzak and A. Goguey for their helpful comments.

REFERENCES

1. Accot, J., and Zhai, S. More than dotting the i's — foundations for crossing-based interfaces. In *Proc. CHI '02*.
2. Apple Computer Inc. *Macintosh human interface guidelines*. Addison-Wesley Publishing, nov 1992.
3. Cockburn, A., Quinn, P., Gutwin, C., and Fitchett, S. Improving scrolling devices with document length dependent gain. In *Proc. CHI'12*.
4. Hinckley, K., and Bathiche, S. Positional scrolling, 2006. US Patent 7,071,919.
5. Hinckley, K., Cutrell, E., Bathiche, S., and Muss, T. Quantitative analysis of scrolling techniques. In *Proc. CHI'02*.
6. Kim, W., Tendick, F., Ellis, S., and Stark, L. A comparison of position and rate control for telemanipulations with consideration of manipulator system dynamics. *IEEE Journal of Robotics and Automation 3*, 5 (1987).
7. Kwatinetz, A. Scrolling contents of a window, US. Patent 5495566 (1996).
8. Li, S., and Shrader, T. Scrolling a target window during a drag and drop operation, US. Patent 5740389 (1998).
9. Quinn, P., Malacria, S., and Cockburn, A. Touch scrolling transfer functions. In *Proc. of UIST '13*.
10. Roussel, N., Casiez, G., Aceituno, J., and Vogel, D. Giving a hand to the eyes: Leveraging input accuracy for subpixel interaction. In *Proc. of UIST '12*.
11. Zhai, S. *Human performance in six degree of freedom input control*. PhD thesis, University of Toronto, 1995.
12. Zhai, S., Smith, B. A., and Selker, T. Improving browsing performance: A study of four input devices for scrolling and pointing tasks. In *Proc. INTERACT'97*.