Content-Aware Scrolling

Edward W. Ishak *Steven K. Feiner*
Columbia University, Department of Computer Science
New York, NY 10027
{ishak, feiner}@cs.columbia.edu

ABSTRACT

Scrolling is used to navigate large information spaces on small screens, but is often too restrictive or cumbersome to use for particular types of content, such as multi-page, multi-column documents. To address this problem, we introduce *content-aware scrolling* (CAS), an approach that takes into account various characteristics of document content to determine scrolling direction, speed, and zoom. We also present the *CAS widget*, which supports scrolling through a content-aware path using traditional scrolling methods, demonstrating the advantages of making a traditional technique content-aware.

ACM Classification: H.5.2 [**Information interfaces and presentation**]: User Interfaces—*Graphical user interfaces (GUI), Interaction styles, Screen design, Windowing systems*

General terms: Design, Human Factors

Keywords: Scrolling, interaction, navigation, path traversal, content-awareness

INTRODUCTION

Scrollbars are conventionally used to navigate large documents on small screens. A viewport, which shows the visible portion of the entire scrollable document, is accompanied by two scrollbars, allowing users to push visible content out of view, replacing it with off-screen content. When using scrollbars, scrolling along either axis can be done independently, but usually not simultaneously.

Some applications, such as Adobe Reader [2] and Google Local [6], allow users to pan within the viewport. Panning, typically performed with a mouse-down and drag, pushes visible content out of the viewport in any direction, while pulling previously hidden content into view. This works well for short distances, but is limited in that the longest dragging distance is equal to the diagonal of the rectangular viewport. Some systems allow *vector scrolling*, where a mouse or joystick is used to define a vector, indicating the direction and speed of scrolling [17]. However, this requires users to steer precisely in the desired scrolling direction, which is often difficult to do when the natural reading direction for the content is not axis-aligned (e.g., consider a multi-column, multi-page text document).

In this paper, we introduce *content-aware scrolling* (CAS), which varies the direction, speed, and zoom during scrolling, based on document content properties, as shown in

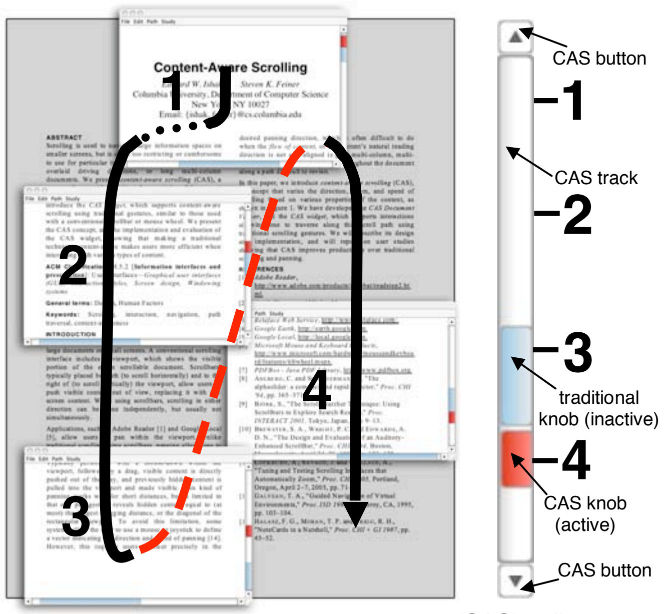

Figure 1: Content-aware scrolling (CAS) allows scrolling in document reading direction using CAS widget (enlarged on right). Reading direction is shown roughly as overlaid arrow, where black dots indicate small unimportant region (traversed with different scroll distance mapping) and red dashes indicate large unimportant region (traversed with animation). CAS window snapshots 1–4 on left correspond to track locations on right, and are traversed in a single scroll. Light blue knob indicates traditional knob location (inactive during CAS) for current CAS knob location (currently at position 4).

Figure 1. We then describe the design and implementation of a document viewer that uses the *CAS widget*, allowing the user to scroll along any path using traditional scrolling gestures.

RELATED WORK

Augmenting and Replacing the Scrollbar

Some systems have augmented the scrollbar to provide additional information about off-screen content. For example, the bookmark scrollbar [19] provides bookmarks adjacent to the scrollbar, such that when the user drags within the vicinity of a bookmark, it snaps to the nearest one. The ScrollSearcher [11] indicates within the scrollbar the results of a document search. The auditory-enhanced scrollbar [13] uses non-speech sounds to help identify off-screen locations.

Others have developed alternative techniques intended to outperform scrollbars, such as the Alphaslider [9], LensBar [20], and FineSlider [21], which allow for quick visualization through a large list of data items. Some researchers have developed gestures that support scrolling in one [22, 24] or two [5, 17] dimensions, or have developed [17] and analyzed [15] systems that automatically zoom while scrolling in two dimensions. However, none of these systems allow the user to revisit a previously traversed path without requiring them to duplicate the same (possibly complex) set of scrolling gestures.

Hardware Interaction Devices

Hardware interaction devices have been created to scroll across documents in both one [29] and two dimensions [3, 7]. However, unlike desktop and laptop computers, many mobile devices (e.g., cell phones and PDAs) do not easily accommodate additional hardware. Furthermore, these interaction devices are not designed to allow the user to easily revisit a previously traversed path.

Navigating Hypertext and 3D Environments

Some hypertext systems, such as Scripted Documents [28], allow authors to create conditional and programmable paths within and between online documents. There have also been many systems developed to traverse a 3D path [18, 26, 27]. CAS builds upon the idea of these 3D path traversal systems, following Galyean's "river analogy" [16], which proposes that a user is like a boat floating down a river, being steered by the water current, but still able to veer slightly off the path using the rudder. In CAS, the default path can be determined automatically, based on the content of the document, or manually by the user. To support user control while scrolling, we have applied this approach to an existing widget, the scrollbar, such that no extra screen space is used and almost no additional training is required to use it.

CONTENT-AWARE SCROLLING

CAS is built upon the concept of *content-awareness*, an approach that takes into account various characteristics of a document's content to determine how the user interacts with it. In other words, if the system knows something about the data, it can help the user interact with it more effectively. In CAS, when the user scrolls, content is not treated as a single undifferentiated layer of information. Instead, various properties are taken into account to identify important regions in which to vary the scroll direction, speed, and zoom. This is a special case of what Smith and Taivalsaari call "generalized scrolling" [25].

Varying Direction

CAS is useful in navigating to off-screen content, provided that the CAS path is consistent with the user's intended scrolling direction, which can make the difference between a pleasant and a frustrating user experience. We define the *flow of content* as the path through which a user wishes to view the content at a given time. For example, this can be the reading direction of a text document, or the path through the results of a text search, as shown in Figure 2. Adobe Reader automatically infers the reading direction of documents for both its "Read Out Loud" feature and third-party screen readers to speak the text, as an accessibility feature. We use this same reading direction.

The "reflow" feature of Adobe Reader takes a different approach. To accommodate smaller displays, reflowing reformats the content so that the user only needs to scroll vertically. CAS takes a different approach by also recognizing the importance of the content's intended layout, thus allowing a user to view the content unmodified while only needing a single scrollbar to traverse it, and potentially giving CAS an advantage over reflow in spatial recognition tasks.

Adobe Reader also supports reading "article threads," allowing a user to start reading on one page and continue on a different page later in the document by pressing a key or clicking within the document; this is a form of hypertext that is limited to links within a single document. Furthermore, when reading an article, the page view zooms discontinuously, such that the line width of the content being read fills the screen width, which may force a zoom level that is inadequate for reading when the viewport is too narrow. StyleCam [14] supports continuous 3D interaction, allowing a user to control the speed and direction with which they traverse a manually authored camera path. In contrast, CAS affords manual control of automatically-generated task-specific 2D document paths. Furthermore, unlike article threads, CAS automatically extracts the reading path, making it effective on documents that were not authored with content-awareness in mind.

Identifying Important Regions. CAS identifies important regions of a document to create a scrolling distance mapping between scrollbar and document pixels. For a reading task (e.g., as shown in Figure 1), an important region is the area whose width spans the left edge of the first character to the right edge of the last character on a line and whose height is the height of that line. Following the flow of content, these text-line–sized important regions coalesce into contiguous stacks of rectangles with possibly varying widths and heights. Since a path is continuous, the unimportant regions along the path are those parts of the document that lie between the rectangles as encountered sequentially along the path. For example, when the next line of text is in a different column or on a different page, the distance along the path that lies within unimportant regions can be as large as thousands of pixels.

Varying Speed

We vary the scrolling speed based on the locations of important regions, in the spirit of the adaptive control/display

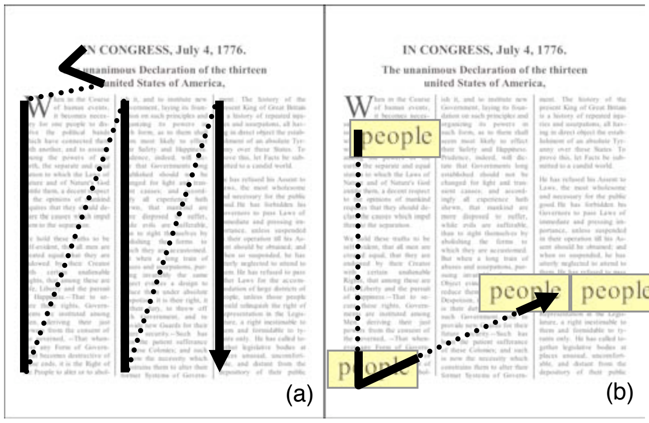
IN CONGRESS, July 4, 1776.
The unanimous Declaration of the thirteen united States of America,
people
people
people
people
(a)
(b)

Figure 2: Path defined by flow of content can vary, depending on task, as shown in (a) reading task, and (b) text search task for "people" in same page. Solid and dotted black parts of path indicate important and unimportant regions, respectively. (Arrows and popouts are included for purposes of illustration only.)

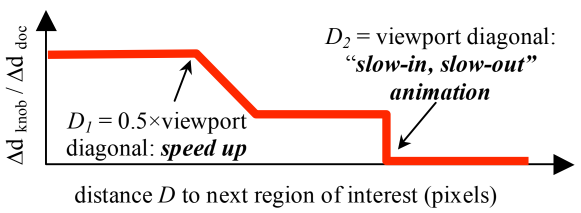

Figure 3: Variation of scrolling distance mapping ($\Delta d_{knob}/\Delta d_{doc}$) depends on proximity of nearby content in relation to size of viewport. If nearest content on path is substantially far away, viewer performs a "slow-in, slow-out" animation to next region of interest.

ratio of [12]. The pixel distance that the CAS widget's knob is dragged (Δd_{knob}) is not uniformly mapped to the pixel distance the document is scrolled (Δd_{doc}), but rather, is a function of the distance D to the next important region relative to the size of the viewport, as shown in Figure 3. For example, consider traversing a search results path within a textual document, as shown in Figure 2. The continuous path traversed by the CAS widget may contain many (possibly large) unimportant regions (i.e., ones that have no search results). Therefore, we change the scroll distance mapping through these regions (shown as dotted parts of the path), despite a constant physical scrolling gesture, decreasing the control/display ratio. For *very* large distances (those larger than the viewport's diagonal), the viewport automatically flies to the next important region using a "slow-in, slow-out" animated change in speed.

Varying Zoom

We vary the zoom level of the document based on the size of and distance between important regions. The zoom level is initially set at 200%; however, it may be reduced slightly if the width of an important region (e.g., a paragraph of text) cannot fit within the current viewport. Unlike Adobe Reader's article threads, we do not force fit if the zoom level results in illegible content (we have informally found that 150% is an acceptable lower bound for legibility of most 10pt fonts on our 800×600 5" mobile display). For very large distances between important regions, in combination with the variation in speed described above, we also use a "slow-in, slow-out" animation change in zoom, similar to navigation in Pad++ [10]. Igarashi and Hinckley [17] showed how varying the zoom level based on the user's scrolling speed can help maintain a constant visual flow, even at high scrolling speeds. However, zooming may not be suitable for some content, such as text, that is illegible at low zoom values. Therefore, we only zoom through unimportant regions.

CAS WIDGET

The CAS widget looks and feels like a traditional scrollbar. Consider, for example, how the CAS widget behaves when its scroll path is mapped to the reading direction of a text document, as shown in Figure 1. Dragging its knob along its track scrolls the document continuously along the reading path. Clicking the arrow buttons at each end advances the document incrementally along the path. Clicking outside the knob on the track also advances the document along the path one "page" at a time, such that the new window that is mapped to the viewport slightly overlaps the old window, providing visual continuity. Mouse wheel scrolling is also supported, which helps users maintain precise control when traversing long CAS paths, especially on small displays.

APPLICATION

We have built an application that uses the CAS widget to support content-aware scrolling for different types of content. The *CAS Document Viewer* application is written using the Java 5.0 SDK for cross-platform operation, and supports both text-based PDF documents and JPG images. The scrolling mode is initially set to "normal," meaning the CAS widget behaves like a traditional scrollbar. Conventional mouse-down-and-drag panning and vector scrolling are also supported.

A CAS path is constructed automatically by the system or manually by the user, depending on the type of content, as described below. To scroll along that path, the user must first change the *scrolling mode* from "normal" to "CAS," through a menu option, a toolbar button, or temporary depression of the "alt" modifier key. Upon enabling CAS mode, if the current viewport does not show a part of the document on the path, the viewer automatically pans the document to the nearest path point. The vertical CAS widget in this mode now traverses the custom path. If the viewport is too narrow for a particular point along the path, the horizontal CAS widget allows left-right scrolling spanning only the width of the content on the path. The document orientation does not change along the path.

A user may wish to venture off the path temporarily, eventually returning to her last path location. We allow the user to anchor her current path location by depressing the Shift key. Extending the work on the bookmark scrollbar [19], this creates an implicit bookmark, allowing the user to pan and scroll freely until Shift is released. The viewer then automatically springs back to the anchored location.

PDF Viewer

Upon opening a PDF document, we use the PDF text extraction tool PDFBox [8] to automatically extract each character, word, and line of text, along with their pixel locations. A *reading path* is then automatically constructed through the beginning of each line, aligned along the left edge of the viewport.

An integrated search function finds and highlights all occurrences of a specified string. When searching, the CAS widget follows the search path for that string until the search function is cancelled or a new path is explicitly chosen. A *search path* starts at the first string occurrence (centered within the viewport), then pans directly to the second occurrence, then to the third, and so on. If two consecutive occurrences are located sufficiently far apart, scrolling is sped up and/or animated between those two points, as described earlier.

Image Viewer

Our viewer also supports viewing photographs. Selecting the menu item "View Faces" finds faces using the Betaface face and eye detection web service [4]. A Hamiltonian path is then automatically constructed through the center of each face, creating a *faces path* that visits each face in the photograph exactly once.

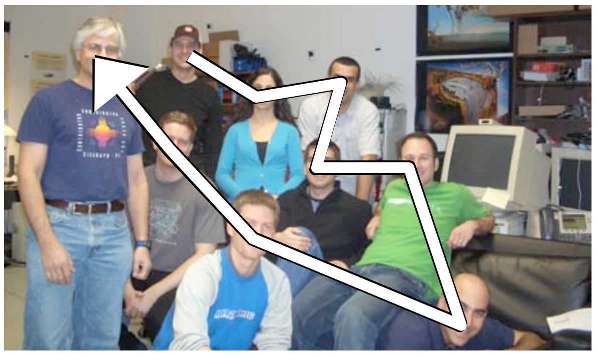

Figure 4: "Find Faces" option in Image Viewer creates a Hamiltonian path through each face of a photograph. Scrolling with CAS widget follows this path (as shown by the arrow).

Path Creation and Persistence

A CAS path consists of a sequence of nodes, each with a location, zoom level, and dimensions of the content at that point in the path. For content types whose paths are created automatically (e.g., textual PDF documents), persistence is not an issue, since the path is recreated every time the document is opened. However, we also allow paths to be created manually by the user using a simple path-editing program (similar to CyberCoaster [23]) invoked from a menu. For user-created paths to be persistent, we require the document to be in JPEG image format (if not, we convert it), and then store this path as metadata using Adobe XMP [1]. If a file's content has changed since the last path was created, the user must explicitly edit the path or create a new one using our path editor. If a document contains multiple paths (e.g., both reading and search paths), all paths are stored within the file's metadata.

USER FEEDBACK AND FUTURE WORK

We have asked colleagues to informally try our CAS document viewer to peruse PDF documents on both laptop and handheld displays. Users say that they appreciate the ability to read a document on a small display in its original format using a one-dimensional gesture, such as a mouse wheel scroll, or a single scrollbar arrow press. We note that no one complained about stimulus-response incompatibility (e.g., when dragging the knob down, the viewport can pan up or sideways). We plan on improving CAS to recognize additional types of content and search types (e.g., traversing headings), and use spline paths. We are also designing a user study to determine whether CAS has advantages over traditional scrolling in reading and search tasks.

ACKNOWLEDGEMENTS

This research is funded in part by NSF Grant IIS-0121239, Office of Naval Research Contract N00014-04-1-0005, and a gift from Microsoft Research.

REFERENCES

[1] *Adobe Extensible Metadata Platform (XMP)*, http://www.adobe.com/products/xmp/.
[2] *Adobe Reader*, http://www.adobe.com/products/acrobat/readstep2.html.
[3] *Apple Computer / Mighty Mouse*, http://www.apple.com/mightymouse/.
[4] *Betaface Web Service*, http://www.betaface.com/.
[5] *Google Earth*, http://earth.google.com.
[6] *Google Local*, http://local.google.com.
[7] *Microsoft Mouse and Keyboard Products*, http://www.microsoft.com/hardware/mouseandkeyboard/features/tiltwheel.mspx.
[8] *PDFBox - Java PDF Library*, http://www.pdfbox.org.
[9] Ahlberg, C. and Shneiderman, B., "The alphaslider: a compact and rapid selector," *Proc. CHI '94*, pp. 365–371.
[10] Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D. and Furnas, G., "Pad++: A Zoomable Graphical Sketchpad For Exploring Alternate Interface Physics," *Journal of Visual Languages and Computing* (1996)*, 7*, pp. 3–31.
[11] Björk, S., "The ScrollSearcher Technique: Using Scrollbars to Explore Search Results," *Proc. INTERACT 2001,* Tokyo, Japan, July 9–13.
[12] Blanch, R., Guiard, Y. and Beaudouin-Lafon, M., "Semantic Pointing: Improving Target Acquisition with Control-Display Ratio Adaptation," *Proc. CHI 2004*, pp. 519—526.
[13] Brewster, S. A., Wright, P. C. and Edwards, A. D. N., "The Design and Evaluation of an Auditory-Enhanced ScrollBar," *Proc. CHI 1994,* Boston, Massachusetts, April 24–28, 1994, pp. 173–179.
[14] Burtnyk, N., Khan, A., Fitzmaurice, G., Balakrishnan, R. and Kurtenbach, G., "StyleCam: Interactive Stylized 3D Navigation using Integrated Spatial & Temporal Controls," *Proc. UIST 2002*.
[15] Cockburn, A., Savage, J. and Wallace, A., "Tuning and Testing Scrolling Interfaces that Automatically Zoom," *Proc. CHI 2005,* Portland, Oregon, April 2–7, 2005, pp. 71–80.
[16] Galyean, T. A., "Guided Navigation of Virtual Environments," *Proc. I3D 1995,* Monterey, CA, 1995, pp. 103–104.
[17] Igarashi, T. and Hinckley, K., "Speed-dependent Automatic Zooming for Browsing Large Documents," *Proc. UIST 2000,* San Diego, CA, 2000, pp. 139–148.
[18] Khan, A., Komalo, B., Stam, J., Fitzmaurice, G. and Kurtenbach, G., "HoverCam: Interactive 3D Navigation for Proximal Object Inspection," *Proc. I3D 2005,* Washington, D.C., April 3–6, 2005, pp. 73–80.
[19] Laakso, S. A., Laakso, K.-P. and Saura, A. J., "Improved Scroll Bars," *Proc. CHI 2000*, April 1–6, 2000, pp. 97–98.
[20] Masui, T., "LensBar - Visualization for Browsing and Filtering Large Lists of Data," *Proc. InfoVis '98*, 1998, pp. 113–120.
[21] Masui, T., Kashiwagi, K. and Borden, G. R., "Elastic graphical interfaces for precise data manipulation," *Proc. CHI '95 Conference Companion*, 1995, pp. 143–144.
[22] Moscovich, T. and Hughes, J. F., "Navigating Documents with the Virtual Scroll Ring," *Proc. UIST 2004,* Santa Fe, NM, October 24–27, 2004, pp. 57–60.
[23] Satou, T., Kojima, H., Akutso, A. and Tonomura, Y., "CyberCoaster: Polygonal Line Shaped Slider Interface to Spatio-Temporal Media," *Proc. ACM Multimedia,* Orlando, FL, October 1999, p. 202.
[24] Smith, G. M. and schraefel, m. c., "The Radial Scroll Tool: Scrolling Support for Stylus- or Touch-Based Document Navigation," *Proc. UIST 2004,* Santa Fe, NM, October 24–27, 2004, pp. 53–56.
[25] Smith, R. and Taivalsaari, A., "Generalized and stationary scrolling," *Proc. UIST 1999,* Asheville, NC, pp. 1–9.
[26] Tan, D. S., Robertson, G. G. and Czerwinski, M., "Exploring 3D Navigation: Combining Speed-coupled Flying with Orbiting," *Proc. CHI 2001,* Seattle, WA, pp. 418–425.
[27] Zeleznik, R. and Forsberg, A., "UniCam - 2D Gestural Camera Controls for 3D Environments," *Proc. I3D 1999,* Atlanta, GA, pp. 169–173.
[28] Zellweger, P. T., "Scripted Documents: A Hypermedia Path Mechanism," *Proc. Hypertext 1989*, November 1989.
[29] Zhai, S., Smith, B. A. and Selker, T., "Improving browser performance: A study of four input devices for scrolling and pointing tasks," *Proc. INTERACT '97*, pp. 286–292.