Content-Aware Interaction in User Interfaces

Edward Waguih Ishak

Advisor
Steven Feiner

Defense Committee
John Kender
Julia Hirschberg
Eric Bier
Bay-Wei Chang

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2007

Abstract

Content-Aware Interaction in User Interfaces

Edward Waguih Ishak

Whether interacting with a mobile phone or a wall-sized display, users often encounter screen space limitations that prevent the simultaneous display of all visual objects required to complete a task. Depending on the individual user and their current task, screen space limitations result from several factors, including insufficient physical space and inadequate pixel resolution. A number of techniques have been created to address these limitations, including overlapping semi-transparent windows, scrolling, and layout management. However, these techniques are typically designed to be generic and do not consider the properties of the content to which they are applied. In this thesis, we present an approach to making user interfaces content-aware by augmenting existing interaction techniques. By *content-aware*, we mean that they take into account various physical and semantic attributes of the content, such as size, location and type.

We designed, implemented, and evaluated content-aware versions of three existing user interface techniques: content-aware transparency, content-aware scrolling, and content-aware layout. C*ontent-aware transparency* applied to overlapping windows makes it possible for users to interact with otherwise hidden content by rendering important regions of windows opaque, and unimportant regions transparent, thus keeping overlaid contents legible and distinguishable at all times. Furthermore, based on properties of the

overlapping material, appropriate image-processing filters are applied to obstructed content to help disambiguate the overlapping content. *Content-aware scrolling* allows a user to scroll along a system- or user-defined path within a document using conventional scrolling interactions, varying scrolling speed depending on content location. *Content-aware layout* lays out windows containing content relevant to that of the currently focused screen area, positioned relative to that area.

We present quantitative user performance data gathered from formal experiments, as well as qualitative questionnaire feedback to show that interaction with content-aware techniques can provide an effective advantage over techniques that are not content-aware. We also describe a testbed environment, called CASTLE, for browsing and searching textual notes, which explores how all three content-aware techniques can be coordinated.

Table of Contents

List of Figures

Acknowledgements

First and foremost, I thank God for everything.

I thank my parents, Waguih and Ragaa, and my brother, Andrew, for strongly encouraging me to discover my limits.

I thank my wife, Irina, for her never-ending selfless support and her seemingly infinite patience.

I thank my advisor, Steven Feiner, for his continued guidance throughout this dissertation. He has made me more of a perfectionist than I ever could have imagined.

I thank my dissertation committee members for their helpful comments and suggestions concerning this research: John Kender, Julia Hirschberg, Eric Bier, and Bay-Wei Chang.

I thank all of my colleagues in the Computer Graphics and User Interfaces lab that have provided insight to my research and endless thought-provoking conversations: Blaine Bell, Hrvoje Benko, Gábor Blaskó, Sinem Güven, Drexel Hallaway, Steven Henderson, Simon Lok, Ohan Oda, Alex Olwal, and Sean White.

This research is funded in part by NSF Grant IIS-0121239, Office of Naval Research Contract N00014-04-1-0005, IBM Research, NSF Grant EIA-02-02063, and gifts from Microsoft Research and Mitsubishi Electronic Research Labs. Parts of this research are part of a project being conducted under the IBM Open Collaborative Research program.

Dedication

To my parents, Waguih and Ragaa

Chapter 1

Introduction

1.1. Problem Statement

Current computer processors make it possible for users to work on multiple tasks simultaneously. However, as the number of tasks increases, there is a corresponding decrease in the amount of available screen space dedicated to each task. Screen space limitations result from the space required for each task's screen objects (e.g., windows, palettes, and icons), combined with size and resolution constraints imposed by the physical display devices on a user interface. Even if a user is working on just a single task, there may not be enough space for all objects associated with that task to be simultaneously visible, forcing the user (or in a semi-automated layout environment, the user together with the system) to choose a subset for display. As a result, some objects

will become either partially or completely obstructed by other visual objects, may be iconified or moved off-screen, or may be scaled down in size so that their contents are rendered too small to be legible or are cropped by the edges of a window. The frequent need to eliminate content from display has given rise to a number of common user interface techniques for making hidden content visible, including uniform semi-transparency (using alpha blending), conventional scrolling, and window layout.

1.1.1. Semi-transparency Using Alpha Blending

Rendering overlaid objects as semi-transparent (through the use of alpha blending [Porter1984]) makes it possible to view at least partially the otherwise obstructed content that lies beneath them. Semi-transparency works well when overlapping contents are familiar and distinguishable, such that their alpha-blended appearance maintains their individual legibility and makes clear which objects contain them. A problem arises when overlapping contents are either unfamiliar or too similar in appearance, such that their alpha-blended appearance is either illegible or indistinguishable, as shown in Figure 1.1. This often prevents the perception of transparency, making it difficult for the user to perceive the overlapping contents as two distinct layers.

With the use of semi-transparency, visible content may not always be directly accessible, as there may be obstructing semi-transparent objects in the way. Since each pixel may represent the blended representation of many overlapping objects, including the background, interacting with any object underneath the topmost object typically forces one to move or resize overlaid objects to ensure that the target content is not obstructed.

Figure 1.1: A screenshot of a window rendered with nVidia's nView uniform semi-transparency feature. The alpha-blended appearance of the overlaid and obstructed content produces several lines of illegible overlapping text.

1.1.2. Conventional Scrolling and Panning

Traditional scrollbars are conventionally used to navigate large information spaces on smaller screens. A conventional scrolling interface includes a viewport, which contains the visible portion of the entire scrollable content. Scrollbars, typically placed beneath (to scroll horizontally) and to the right of (to scroll vertically) the viewport, allow users to push visible content out of view, replacing it with off-screen content. When using scrollbars, scrolling in either direction can be done independently, but usually not simultaneously.

Many types of content, such as maps, tables, and multi-column text documents, require navigating in two dimensions. Applications, such as Adobe Reader and Google Local, allow users to pull hidden content into view from any direction. Typically performed with a mouse-down and drag within the viewport, visible content is directly pushed out of the

way, and previously hidden content is pulled into the viewport and made visible. This kind of scrolling is also referred to as *panning*, and it works well for short distances, but is limited in that one pan gesture reveals hidden content equal to, at most, the longest dragging distance, or the diagonal of the rectangular viewport. To avoid this limitation, some systems allow the user to use a mouse or joystick to define a vector indicating the direction and speed of panning [Zhai1997]. However, this requires users to steer precisely along a particular path, which is often difficult to do when the destination within the document is not axis-aligned with the current position. Consider, for example, reading a multi-column, multi-page document on a mobile display, as shown in Figure 1.2. After reading the left column, a user must interrupt her reading task and navigate to the top of the next column to continue reading.

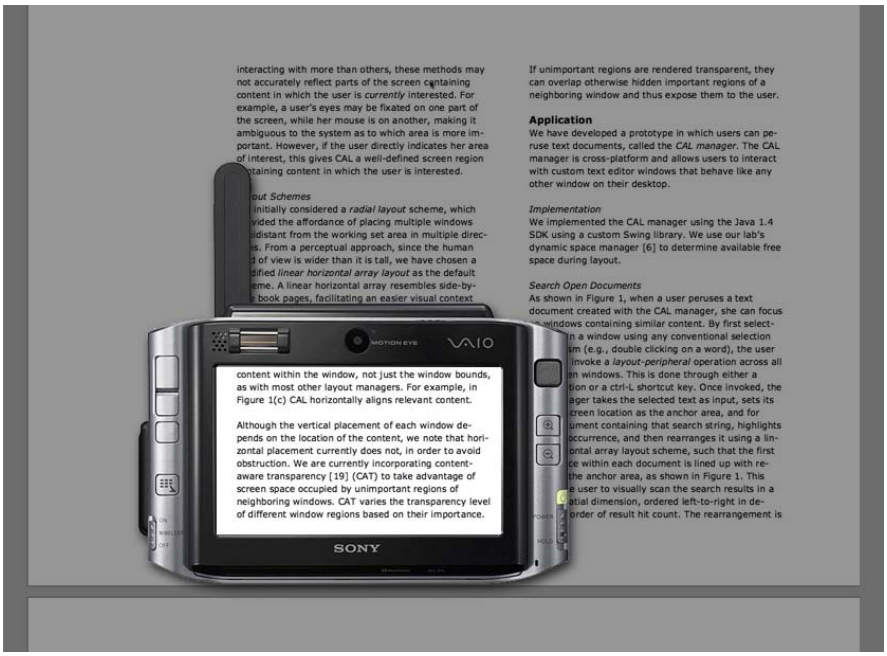

Figure 1.2: Navigating along a non-linear path can be difficult, especially when the user must steer precisely in the desired direction. Reading a multi-column, multi-page document on small displays often requires an interruption when navigating between columns and pages.

1.1.3. Window Layout

Some window managers can automatically lay out all of a user's non-minimized windows, such that every window is unobstructed. For example, since version 3.1, Microsoft Windows has allowed a user to automatically tile or cascade each window using a least-recently-used order. Apple's Exposé [2] allows users to interact with any open window by scaling, tiling, and neatly rearranging all open windows (or all open windows of a particular application) at the press of a key, thus allowing a user to click on the scaled, tiled version of the window with which she wishes to interact. Upon selection, Exposé then restores each window to its original size, location, and *z*-order, with the exception of the selected window, which moves to the front. However, tiling/cascading often resizes windows in addition to rearranging their positions (as shown in Figure 1.3), which may be undesirable by the user. Further, neither tiling/cascading nor Exposé allow a user to choose a subset of windows to rearrange based on their contents (e.g., windows containing content that matches a search query).

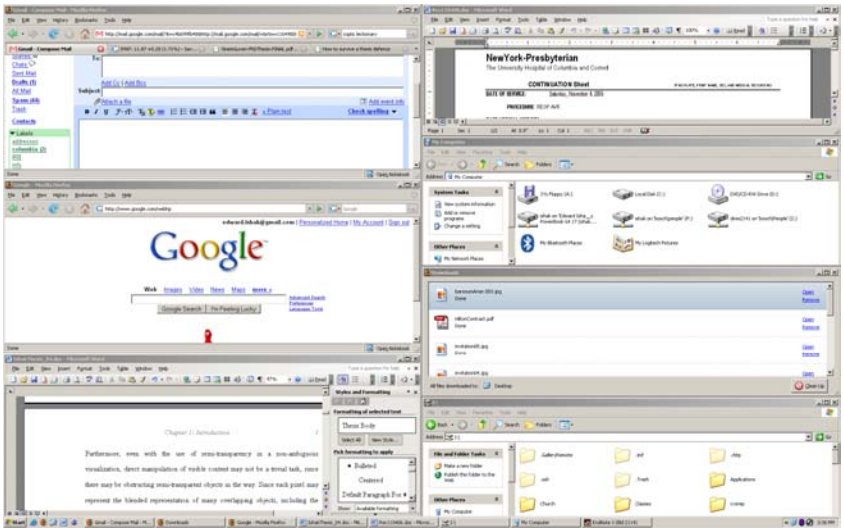

Figure 1.3: Screenshot of a desktop in which all open windows are tiled by the system. While this makes every window visible, their system-defined aspect ratios and locations can be undesirable.

1.2. Goal and Approach

The goal of this thesis is to show how selected traditional interaction techniques can be modified to significantly improve interaction with documents, as well as to improve the overall user experience, by making those techniques content-aware. We use the term *content-aware* to refer to a technique that takes into account characteristics of a document's content to determine how the user interacts with it.

We redefine the functionality of these existing interaction techniques based on various attributes of the content with which the user interacts. For example, consider conventional scrolling, which can be loosely defined as the act of sliding a document underneath the window of an insufficiently large display. This requires a user to control specifically where and how fast they are going, or more precisely, to provide two types of transfer input: position (i.e., zero order) and rate control (i.e., first order) [Zhai1997]. However, a document is normally treated as a single, flat layer of information, such that both position and rate control are uniform across the entire document, irrespective of its content. Furthermore, the input devices and widgets that are commonly used, while allowing a user to easily provide restricted position and rate control (e.g., strict up-and-down scrolling using the vertical scrollbar restricts position control to changes in y only), often make it difficult to navigate a complex 2D path precisely within the document. The user input required to traverse such a path precisely can widely vary, depending on the content being viewed at the time.

Our approach considers the document's content when determining how to perform a particular task effectively. For example, in the case of scrolling, when reading a

document, we will show that the knowledge of various properties of the content, such as its location and order with respect to other content within the document, can allow the system to translate user input into the exact position and rate control inputs necessary to complete the task effectively. Content-aware scrolling maps the position control of a 1D input device into the position control of a possibly complicated 2D spatial control task, and allows a constant rate control of that device to automatically vary the scrolling speed, such that a user spends less time in unimportant parts (e.g., transitions between columns and pages) of the document.

We have also designed and run a set of user studies that show that our content-aware interaction techniques can help users interact with documents more effectively than the original versions, and that users prefer the content-aware techniques.

1.3. Contributions

This thesis makes the following contributions:

- *Development and evaluation of content-aware transparency*. *Content-aware transparency* (CAT), allows a user to interact with otherwise hidden content by varying the levels of transparency within different regions of a window. In our implementation, we render *important* regions opaque and *unimportant* regions transparent, with a smooth opaque-to-transparent gradient in between. Based on properties of the overlapping material, various image-processing filters are applied to obstructed content to help disambiguate the overlapping material. We designed, implemented, and evaluated a user interface that employs CAT. Our

user study showed that participants were more effective with the use of CAT and also preferred user interfaces that employed CAT over ones that did not. We also developed a set of *CAT interaction techniques* that allow users to unambiguously interact with objects rendered with CAT: pop-through, focus filter, and mouse-over pie menu. The *pop-through* technique allows a user to directly manipulate an obstructed object. The *focus filter* technique allows a user to temporarily restore obstructed image-processed content to its unfiltered form. The *mouse-over pie menu* technique allows a user to select an object to interact with from a pie menu of all objects that are currently under the mouse cursor's position.

- *Development and evaluation of content-aware scrolling*. *Content-aware scrolling* (CAS) allows a user to scroll along a document path defined by the user or system, varying the direction, speed, and zoom of scrolling depending on the document's content and the task at hand. We designed, implemented, and evaluated a user interface that employs CAS. Our CAS Document Viewer automatically extracts the reading path and search paths within text PDF documents, as well as the faces path within photographs containing people's faces, and allows one to traverse these paths using traditional scrolling gestures (e.g., using the mouse scroll wheel). Our user study showed that participants greatly prefer using CAS to peruse unfamiliar documents. CAS also significantly outperformed both traditional and vector (i.e., free) scrolling in short distance navigation tasks.

- *Development of content-aware layout*. *Content-aware layout* (CAL) takes into consideration the contents of windows on a user's desktop to determine if and where they should be placed on the screen by applying constraints to content within the windows, rather than to the windows' bounds. We developed a testbed application to demonstrate how CAL could be useful when perusing text documents. When a user selects text within a window, CAL rearranges other windows containing that text, horizontally aligning the search results of those windows with the selected text. Similarly, a user can perform a search across all open windows, such that search results are horizontally aligned in the center of the screen. Portions of windows not containing search results are used as available screen space, such that windows can overlap without obstructing any search result in a neighboring window.

- *Development and informal evaluation of a content-aware user interface combining CAT, CAS, and CAL*. CASTLE (Content-Aware Scrolling, Transparency, and Layout Environment) incorporates coordinated implementations of CAL, CAT, and CAS to help users visualize and interact with related information across multiple windows. Using CAL, similar content across multiple windows is horizontally aligned on the screen. Content that would otherwise be obscured can be seen through unimportant regions of an overlapping window using CAT. Users can scroll through search results within and between neighboring windows by simply using the mouse scroll wheel. Transitions within windows are performed using CAS, while transitions between windows are performed with CAL. In an informal study, physicians used CASTLE to peruse

patient status notes and reported that they are able to make otherwise important inferences much more easily and quickly than with their current system.

1.3.1. Content-Aware Transparency

Many user interface scenarios contain overlapping windows. For example, maximized windows or full-screen applications utilize every pixel on the display, thereby obscuring all other content underneath them. Chapter 3 describes the development of *content-aware transparency* (CAT) [Ishak2004], a user interface technique that selectively varies the level of transparency within different regions of a window depending on a combination of overlapping content properties, as shown in Figure 1.4. To demonstrate our ideas, we implemented a CAT testbed application in which users can visualize text, thumbnails and full-size images within overlapping 2D windows. To maintain the legibility of overlaid content, our testbed uses application semantics to identify important window regions to be rendered opaque, and unimportant regions to be rendered transparent, with smooth opaque-to-transparent gradients in between them (see Section 3.4.2.1). This helps reduce interference resulting from rendering overlapping contents uniformly semi-transparent, which can leave overlaid contents illegible. Furthermore, our testbed uses the high-level type, estimated spatial frequency characterization, and colors of the content to determine whether image-processing filters should be applied to underlying content, and which kind should be applied, to help users correctly identify the window that contains the content (see Section 3.4.2.2).

We have evaluated two different CAT implementations (see Section 3.6), both rendered with gradient transparency values varying from 0% (opaque) to 50% (semi-transparent)

with one incorporating the additional use of desaturation and blur filters, against three other transparency techniques: traditional semi-transparency at 25%, at 50% (i.e., more transparent than at 25%), and at 50% combined with blurring of the underlying contents. In a task where participants were asked to identify whether a highlighted icon was in the top or bottom of two overlapping windows (where the icon was always positioned in the intersection of the overlap), we found that participants were able to correctly identify the containing window as quickly and accurately with our CAT implementations as with windows rendered with a 25% uniform semi-transparency value. Both of our CAT implementations, as well as semi-transparency at 25% significantly outperforming windows rendered with a 50% uniform semi-transparency value (both with and without blurring). This suggests that our CAT implementation affords the overall performance and accuracy benefits of less transparent windows, but with the added "see-through" advantages (within unimportant regions of windows) of more transparent windows.

In the same study, the same participants were able to search for text within a window overlaid onto similar content (e.g., text onto text of the same font, size, style and color) significantly faster when using our CAT implementation than with the other techniques. A questionnaire given to participants in the search study also indicated that, overall, they preferred using CAT and rated it easier to use, more satisfying, and more intuitive than the other techniques.

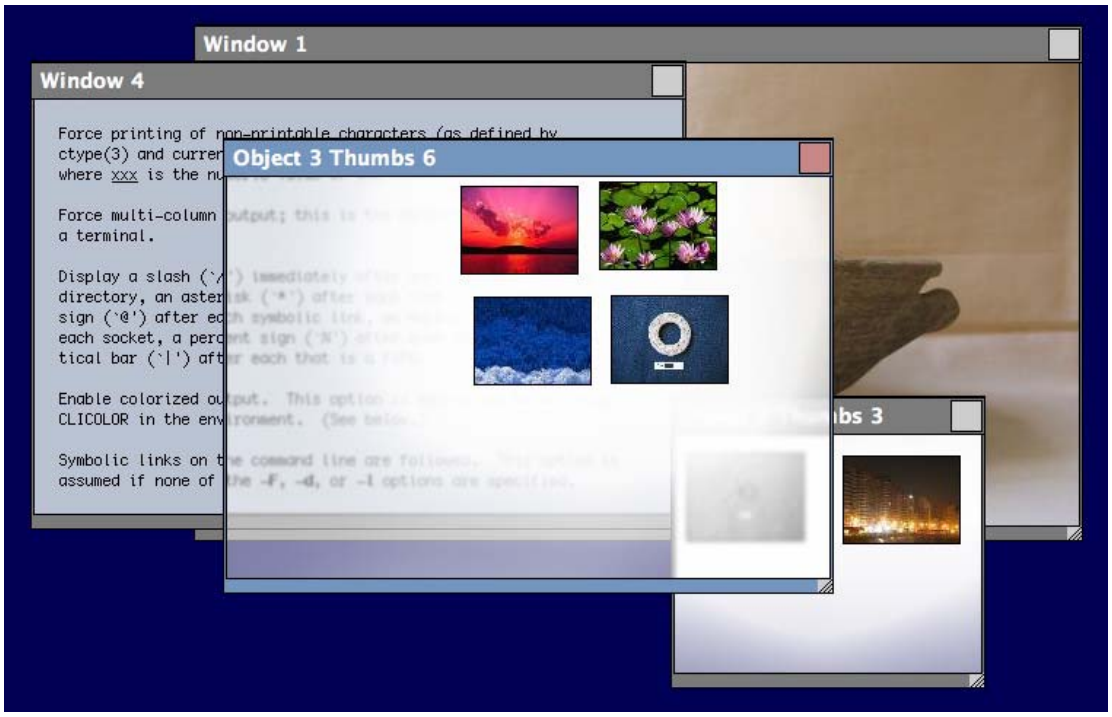

Figure 1.4: Four overlapping windows rendered with our implementation of content-aware transparency. Levels of transparency are varied in the overlaid window based on the location of content. Image-processing filters are applied to underlying content based on the combination of overlapping content types.

We have also developed three interaction techniques for use with our CAT implementation (see Section 3.7). The *pop-through* technique allows a user to interact with objects seen through unimportant regions of overlaid windows. Surpassing a pressure threshold on a touchpad, or using a left-mouse down with a 500ms delay, an obstructed window directly below an overlaid window at the current mouse position "pops through" and is available for immediate interaction. The *focus filter* allows a user to temporarily restore filtered content to its original unfiltered form. This allows visualization of obscured content without first bringing it to focus or moving or resizing any window. The *mouse-over pie menu* presents a user with a list of window thumbnails representing the windows that contain a user-specified pixel location. By hovering one's

mouse over a particular thumbnail in that list, that window is immediately placed on top of all other windows and becomes focused for interaction.

1.3.2. Content-Aware Scrolling

When a user reads a multi-column, multi-page document within a smaller viewport, they are often interrupted by the need to scroll or pan between columns and pages. Chapter 4 describes the development of *content-aware scrolling* (CAS) [Ishak2006], a technique that varies the speed, direction, and zoom of scrolling based on the properties of the content through which a user scrolls. For example, increasing the scrolling speed (or with the use of animations) within column and page transitions, users can spend less time in those unimportant regions when reading the document. In our CAS implementation, users can scroll along a system- or user-defined path using traditional scrolling gestures.

To explore this concept, we developed a CAS document viewer that, upon opening a text-based PDF document, automatically extracts lines of text along with their locations and dimensions, and constructs a reading path, as roughly shown overlaid onto the first page of a document in Figure 1.5. This path is continuous and connects different parts of the document. To allow users to easily peruse the document, we map this path to the CAS widget, which looks and feels like a traditional scrollbar, allowing one to use the mouse wheel or a secondary knob on the scrollbar track to continuously navigate along this path. For short regions along the path that serve as transitions between columns and pages (i.e., unimportant regions), we change the *scroll-document distance mapping*, a control-display ratio of the scroll knob's distance to the document's distance, such that users can more quickly scroll through these regions (see Section 4.3.2), as shown as black dots in Figure

1.5. We automatically transition the viewer between longer unimportant regions using a slow-in, slow-out animation to reduce the time and effort normally required to precisely scroll between those regions, as shown as red dashes in Figure 1.4.

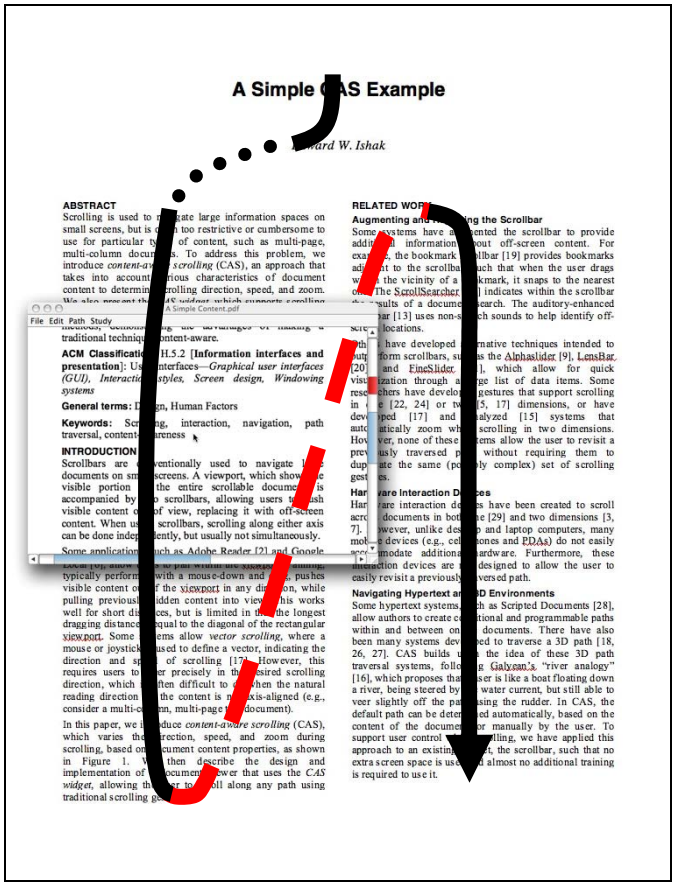

Figure 1.5: Content-aware scrolling allows scrolling in the document reading path using a 1D control input device, such as a mouse scroll wheel. The reading path is shown roughly as the overlaid arrow, where black dots indicate a small unimportant region (traversed with a different scrolling distance mapping) and red dashes indicate a large unimportant region (traversed with an animation).

An integrated search function constructs a search path that directly passes through each search result and maps this path to the scrollbar until the search is cancelled. As with the reading path, while traversing the search path, the scroll-document distance mapping can change depending on the relative distances between search results, or for substantially larger distances, the path is traversed using a slow-in, slow-out animation. Our viewer

also supports the viewing of photographs, automatically creating a Hamiltonian path through the faces of people in the photograph, which can be traversed like any other CAS path.

We designed and conducted a user study of our CAS implementation (see Section 4.6) by asking participants to both slowly peruse and quickly navigate through multi-column, multi-page text documents. We asked them to use each of three techniques: our CAS implementation, traditional scrolling, and vector (or free) scrolling, which allows the user to define a direction and speed of scrolling using the mouse cursor. The study participants greatly preferred our CAS implementation when perusing a document at a self-controlled pace, mentioning that the automatic transitions provided a richer user experience. In addition, our CAS implementation outperformed the other techniques when used to quickly navigate to nearby offscreen parts of the document.

1.3.3. Content-Aware Layout

Commercial window managers can automatically rearrange windows on a user's desktop, but do so without taking into consideration the content within the window. Chapter 5 describes the development of content-aware layout (CAL) [Ishak2007], a user interface technique that determines if and how each window on a user's desktop should be placed, depending on its content. Our testbed implementation of CAL uses scaling and animation techniques similar to Apple's Exposé, such that layout operations are invoked on the press of a key. Each window's state can be easily restored to its original size and position.

When reading a document, a user may wish to view related contextual information in other windows. When a user selects any text within a window and invokes the *layout-peripheral* command, CAL identifies all other open windows containing the selected text and peripherally rearranges them, such that every occurrence of that string is highlighted and the first occurrence in each window is horizontally aligned with the selected text in the focused window, as shown in Figure 1.6. By rearranging and highlighting the important parts of windows across a single horizontal axis, a user can quickly view all windows containing content similar to that selected in the focused window with a simple left-to-right scan.

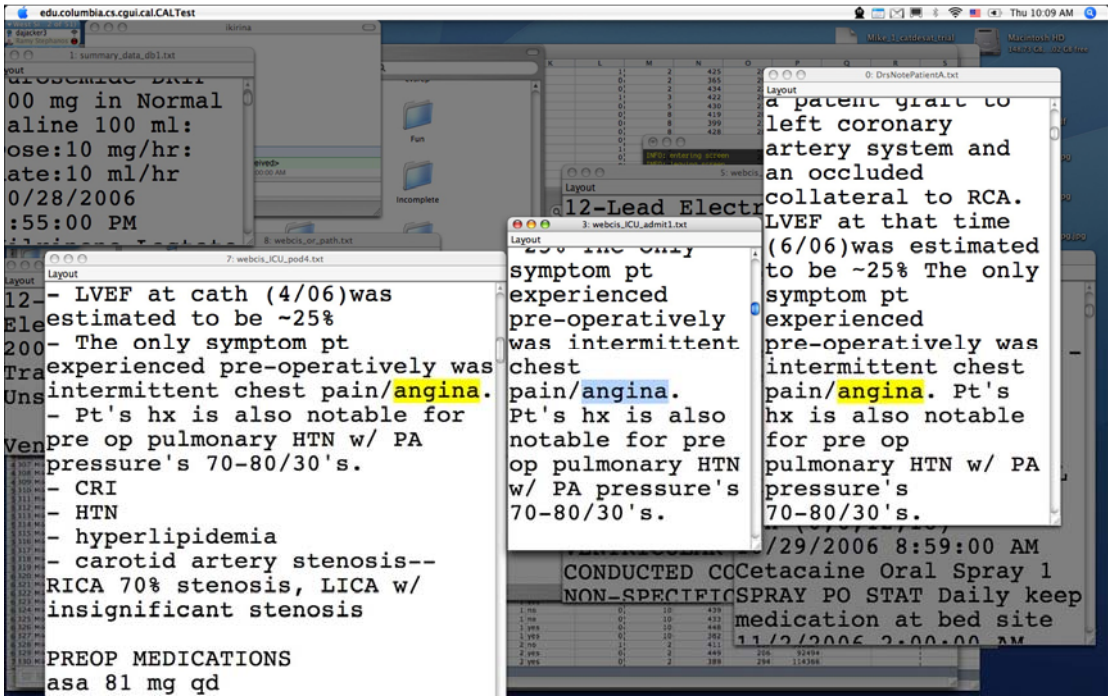

Figure 1.6: A user selects the text "angina" and invokes a layout-peripheral operation to search for it in all other open windows. CAL rearranges only those windows containing the search term, such that all the search terms are highlighted and horizontally aligned with the selected text.

A user can also perform a search without first selecting text by invoking a *search-and-layout* operation, which allows a user to first enter a search query. Upon pressing the "enter" key, windows that contain content matching the search query are rearranged, such that all the search terms are lined up horizontally in the center of the screen.

CAL not only considers the window contents to decide which windows to rearrange, but also uses a constraint solver that applies constraints to the bounds of selected portions of the windows' contents. (In contrast, other work on constraint-based window management has applied constraints only to the bounds of entire windows, rather than to their content.) CAL tries to constrain each window to be fully unobstructed. However, in case this is not possible, it takes advantage of *unimportant regions* of windows (i.e., those that do not contain search results) to allow neighboring windows to overlap each other without obstructing *important regions* (i.e., those that do contain search results).

1.3.4. Content-Aware User Interface

In Chapter 5, we also explore how our set of content-aware techniques can complement each other with the development of CASTLE (Content-Aware Scrolling, Transparency, and Layout Environment), a content-aware user interface that employs CAL, CAT, and CAS to allow users to visualize and interact effectively with information spread across multiple windows. CASTLE rearranges windows using CAL, horizontally aligning similar content within multiple windows with respect to the content location, rather than to the window bounds.

To allow more content to be visible, CASTLE incorporates CAT, so that unimportant regions of a window can be seen through transparent overlaid regions of an overlapping window, as shown in Figure 1.7. It maintains the legibility of the overlaid important content (e.g., search results) by varying the transparency within those regions (i.e., important content is rendered opaque and unimportant content is rendered transparent, with a smooth gradient between them), while applying a Gaussian blur to underlying content to reduce interference.

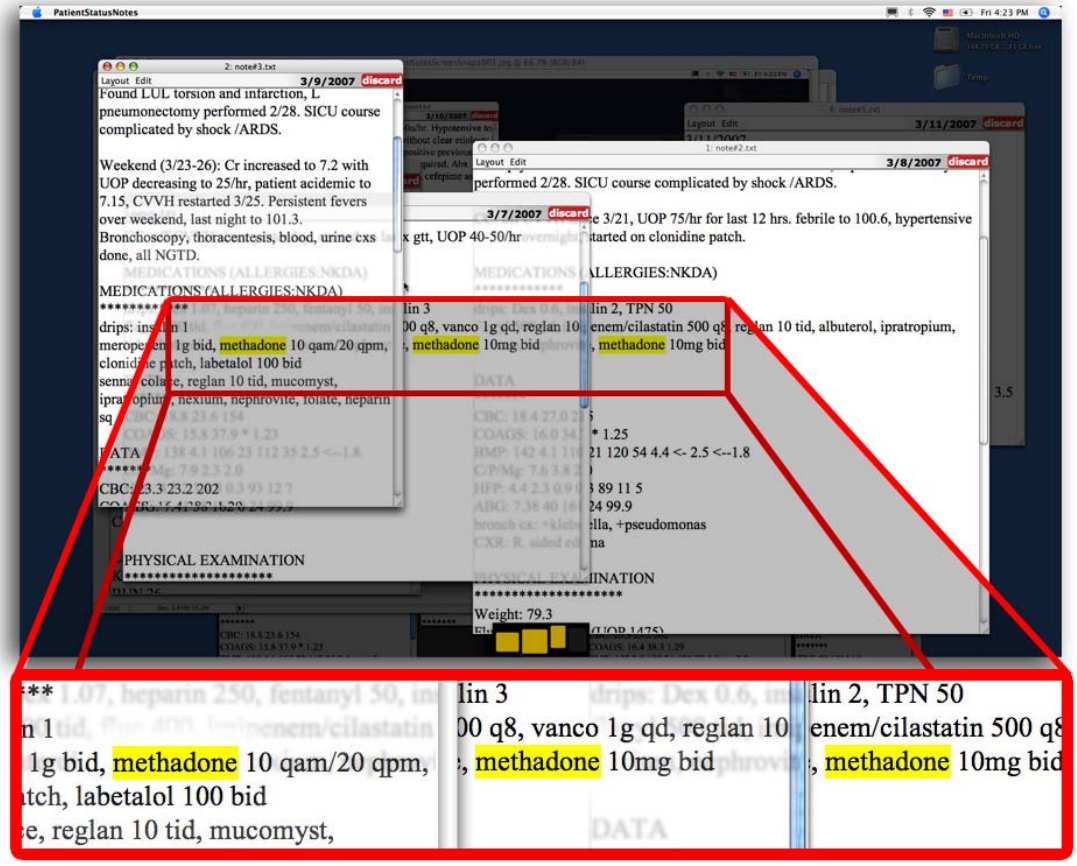

Figure 1.7: CASTLE allows a user to view and interact with search results across multiple windows. Overlaid window regions can expose otherwise obstructed unimportant content using CAT. Using the mouse wheel, users can scroll through all the search results within a document (using CAS) and across documents by rearranging the windows using CAL.

CASTLE also makes use of CAS to allow a user to scroll through all the search results of the rearranged windows using the mouse scroll wheel. By first placing the mouse cursor over one of the windows, the user rotates the mouse wheel to scroll through each search result within that window. As this happens, CASTLE uses CAS to transition between each result. When the last result of that window is reached, on a subsequent downward scroll, CASTLE uses CAL to rearrange the windows, such that the next window (or, on a subsequent upward scroll, the previous window) is placed in the current window's location. Upon reaching the last result of the last window, CASTLE cycles back to the first result of the first window. This makes it possible to use the mouse wheel—a 1D rotational input device—to continuously navigate through every search result of every open window, a task that could normally require manual window resizing and repositioning operations and a separate search operation for each window.

We have asked two physicians with whom we are collaborating at New York Presbyterian Hospital, to use CASTLE to peruse daily patient status notes, possibly written by them or by other doctors on previous days. Both with and without the use of CASTLE, they were both asked to search for a particular section within each dated note to report on a patient's status over time. Using CASTLE, notes are searched and rearranged in chronological order (dates extracted from the note text) such that patient progress over time is easily observed. Physician feedback has shown that they can make certain inferences easier and quicker with CASTLE, which uses content-aware interaction techniques, than with tools that do not.

Chapter 2

Content-Aware Defined

In this chapter, we will define what it means for an interaction technique to be content-aware, and explain why we selected particular interaction techniques and how we modified them to make them content-aware.

2.1. Initial Motivation

The motivation behind the concept of content-aware modifications to traditional techniques stemmed from development of what we originally called “free-space transparency” [Ishak2003]. Using relatively large and wide displays for software development of various software projects, we were writing code that produced user interface scenarios where lines of code varied widely in length, sometimes producing

large regions of white (i.e., “negative”) space in between regions containing text, as shown in Figure 2.1. Making the entire window semi-transparent seemed like a viable solution, but ultimately the text was important, and therefore it needed to remain legible and uncompromised. However, it seemed potentially beneficial to make use of available “parts” of windows to visualize additional information. By rendering these “free” window regions semi-transparent, we could visualize content beneath them without compromising the appearance of the code in the occupied regions. However, the application of transparency had to be “smarter” than the traditional approach of applying it uniformly to the whole window—it had to know where the important content lived within the window, in order for transparency to be applied selectively. This was the rationale behind our development of free-space transparency [Ishak2003], a precursor to content-aware transparency (described in Chapter 3).

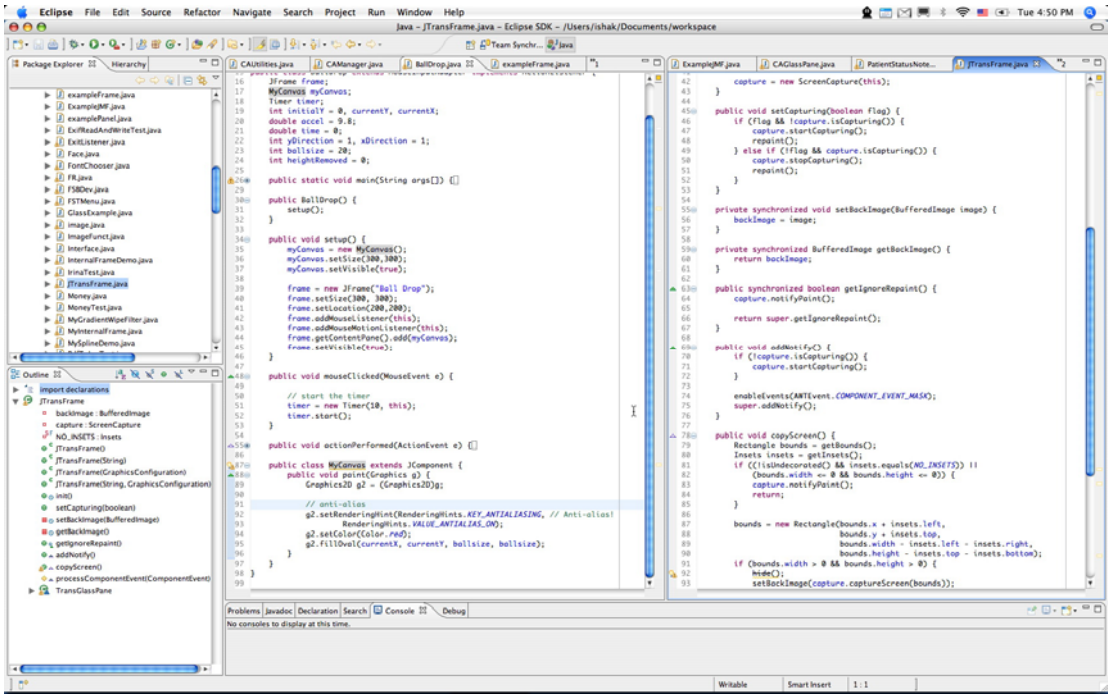

Figure 2.1: Writing and editing source code often produces substantial white space due to varying line lengths.

2.2. "Not Content-Aware" by Example

Before defining what it means for an interaction technique to be content-aware, it helps to understand what it means for a technique ***not*** to be content-aware. By looking at examples of existing techniques that are not content-aware, and by recognizing their drawbacks, we can determine how to significantly improve them.

We begin by taking a closer look at a specific example of a technique that is not content-aware—*uniform semi-transparency*. Semi-transparency (using alpha-blending) is available as a feature in almost every commercial window manager, and is typically applied uniformly to all pixels of an overlaid graphical object to allow a user to "see through" it. Conventionally, a graphical window or palette is made semi-transparent by assigning the same alpha component to each pixel making up that window, as shown in Figure 2.2a. It takes each pixel's color and uniformly alpha-blends it with the contents of the frame buffer behind it, thus giving the appearance that an overlaid window transmits light from whatever is below (i.e., that the overlaid window is transparent). This technique, as just described in its conventional form, *does not take into account any attributes of the content* within the containing window. Given this property, we do not qualify it as being content-aware.

Many generic techniques, such as uniform semi-transparency, can be beneficial to user interaction, but can often produce unwanted visual results. Our goal in this thesis is to show that these drawbacks can be avoided if a technique knows something about the data to which it is applied. For example, the illegibility of overlaid window contents (resulting from interference with the contents of the frame buffer behind it) is a common side effect

of traditional semi-transparency as it is generally used in commercial systems today. However, certain physical characteristics of the overlapping contents, such as their relative locations or spatial frequencies, seem to deterministically reveal when these drawbacks will occur. To alleviate the illegibility problem just described, one approach is to uniformly modify the alpha value of the overlaid layer depending on what kind of content lies beneath it. Due to the high spatial frequency of the content, lowering the alpha value of the overlaid window (i.e., making it more opaque) produces more legible overlaid content, although it results in more illegible underlying content (e.g., Figure 2.2b), which is arguably a desirable approach in this situation. On the other hand, raising the alpha value of the overlaid window (i.e., making it more transparent) results in both illegible overlaid and underlying contents (e.g., Figure 2.2c).

Unfortunately, depending on the overlapping contents, there may be no single uniform alpha value that makes both the entirety of the overlaid window contents and parts of the underlying window contents sufficiently legible. However, this may be possible by varying the transparency across the overlaid window, and this variance may depend on the content within the window.

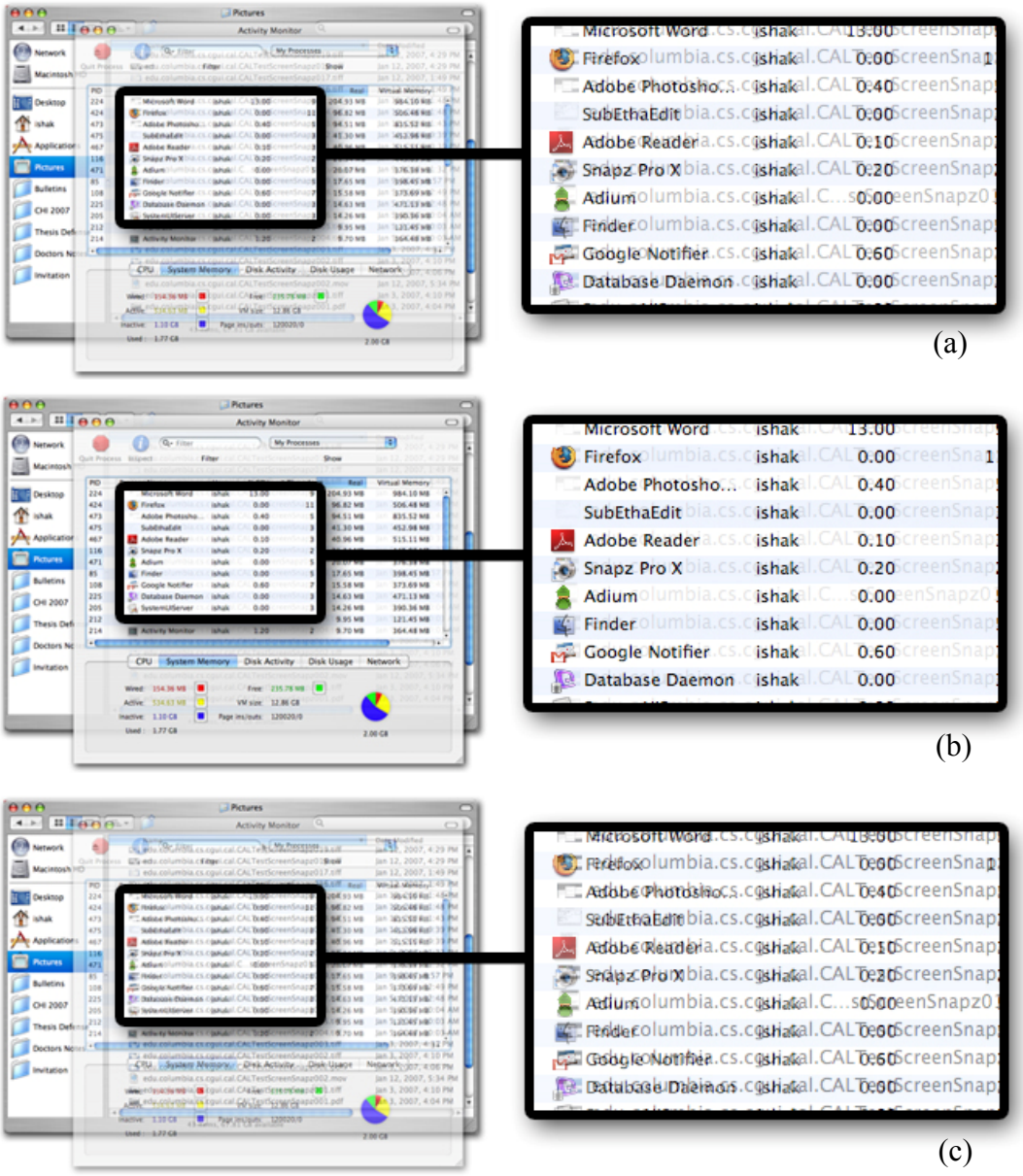

Figure 2.2: (a) 25% transparency level applied to the overlaid window (i.e., alpha-blending 75% of the overlaid window's color, handling each color channel separately, with 25% of the color of the contents of the frame buffer behind it) produces overlaid content that is difficult to read. (b) Decreasing the transparency of the overlaid window increases the legibility of the overlaid content, but makes the underlying content more illegible. (c) Increasing the transparency of the overlaid window decreases legibility of both the overlaid and underlying content.

2.3. Content-Aware Defined

Modifying the alpha value based on the type of overlapping contents is an example of what we call a *content-aware modification*, or more generally, modifying the behavior of

a technique depending on various properties of the content to which it is applied. For example, as we will show in Chapter 3, our content-aware modification to transparency varies the alpha value of an overlaid window depending on the *location* of overlaid content, amongst other properties. This allows parts of the overlaid window that do not contain content to be rendered more transparent than those that do, thus allowing otherwise obstructed material to be more legible.

2.4. An Approach to Making a Technique Content-Aware

In general, a content-aware modification to an interaction technique is meant to reduce the effects of particular drawbacks that negatively impact user performance when completing a task. For example, when *searching* for text within an overlaid window, the time it takes to search may be negatively impacted by uniform semi-transparency applied to that window (as compared to when the window is rendered fully opaque). This is because a sufficiently high transparency may allow content that lies beneath the window (e.g., text) to possibly interfere with the overlaid text, thus making it less legible and making it more difficult or impossible, to perform the search. Semi-transparency is useful in some cases, for example, to monitor dynamic underlying content, such as a progress bar for an internet download, that would otherwise be fully obstructed by an opaque overlaid window. However, this technique in its conventional form does not take into account whether an object lies beneath, or what potentially interfering qualities that object might have. For example, consider semi-transparency applied to the same text window, but now overlaid on top of a uniform white background. This may not negatively impact the user's search performance due to minimal interference with the monochrome background. Furthermore, even if underlying textual content is placed

beneath parts of the overlaid window that are *not important* to the user at a particular time, this also may not negatively impact the user's search performance. In other words, the negative impact on user performance when completing a task probably correlates to some deficiency in how important content is displayed.

2.4.1. Being Importance-Aware

For a technique to be made content-aware, it also needs to be made *importance-aware*; that is, it must be aware of what is important to the user at a particular time, given the current task. In a graphical user interface, physical regions of the screen, which we call *important regions*, should be identified, such that properties of the content within these regions help identify *what* causes the particular drawbacks, and *how* to correctly display the content to reduce the effects of those drawbacks.

The entity (or entities) responsible for identifying these important regions can vary. For example, the *rendering engine* could automatically determine the location of white space, or some other textual patterns, and identify those regions to be unimportant. Alternatively, the *user* can explicitly identify important regions by selecting text with her mouse or keyboard, for example. Another possibility would be to have the *application* report what is important based on the task, while also taking input from the rendering engine or the user (or both). Our specific content-aware modifications take the third approach by having the application report to the interaction technique important regions in the form of a set of rectangles of arbitrary widths and heights that have various properties that describe the contents they contain. In our implementations, each of these regions is assigned a binary value of importance (i.e., important or not).

2.4.2. Being Content-Aware

In our content-aware modifications, once important regions have been determined, properties of these regions (and of the content within them) that contribute to the drawbacks are identified. For example, the technique could take into account the size, location, order of, and distance between important regions. The technique could also take into account *visual properties* of the content within these regions, such as its color or spatial frequency spectrum. It could also consider *semantic properties* of the content within these regions, such as the high-level type or the search result score of the content within these regions (e.g., during a search, how well the text within the important region matched the search query).

2.4.3. The Display Model: Degree of Interest

We then loosely follow an existing importance-aware *display model* that was introduced by George Furnas's Generalized Fisheye Views [Furnas1986] to determine how to correctly display the content. He discussed that, given data to display and a certain task, local detail and global context could be provided based on a *degree of interest* function, which assigns a number to each point in an abstract structure defining how interested the user is in seeing that point. In our approach, the important regions' content properties essentially assign numbers to points along a *display* or *interaction* domain. In other words, these properties determine *what*, *when*, or *where* to display the content, such that the identified drawbacks are alleviated.

This dissertation will demonstrate a modification to three different techniques, each across a unique display or interaction domain, as described below:

- Our content-aware modification to transparency demonstrates how the legibility of overlapping material can be improved by varying the *display* of content across the ***appearance domain*** (in our implementation, we mean the color and alpha properties of pixels that make up that content) by taking into account its semantic and visual characteristics. This includes the location and size of important regions, as well as the combinations of colors, spatial frequency spectra, and high level types of the overlapping contents within those regions. In other words, we vary the transparency of (and possibly apply image-processing filters to) overlapping material to maintain its legibility and reduce interference.

- Our content-aware modification to scrolling demonstrates how the performance of reading and search tasks within text documents can be improved by varying the *display* and the *interaction* of content across the ***temporal domain***, based on the size, location, order of, and distance between important regions. For example, when reading a document, given a uniform scrolling gesture, we vary the speed at which offscreen content approaches the viewport and becomes visible. This allows important regions (e.g., parts of the document containing text) to remain visible within the viewport for a longer time than unimportant regions (e.g., parts of the document not containing text).

- Our content-aware modification to layout demonstrates how the user experience can be improved when searching across and comparing multiple documents by varying the *display* of content across the ***spatial domain***, based on the location and size of important regions. For example, when selecting text within a window,

other open windows that also contain that text are repositioned on the screen, such that the parts of those windows that contain that text are aligned horizontally with the user's selection. This provides an easier context switch between the selected text and the search results.

2.5. Existing Content-Aware Techniques

According to our definition, content-aware 2D interaction techniques exist in both conventional everyday uses, as well as in experimental systems. Consider, for example, conventional scrolling using the mouse wheel. Most systems have their default scroll wheel settings to advance "one line" per notch advancement (which can be changed in most systems). When scrolling through an image, "one line" may be undefined, but when scrolling through a text document, the viewer must know something about the content to correctly advance one line of text. This kind of interaction has to be content-aware on such a level that it knows the height of a line, as well as the line spacing, to correctly scroll through the document.

Another example is the Table Lens [Rao1994]: a table-viewing application that uses a fisheye view [Furnas1986] to display tabular information. Unlike traditional spreadsheet applications that require manual resizing of table cells, the Table Lens uses importance levels of cells to determine its size, thus distorting its spatial layout. This allows many more cells to be displayed thus improving interaction.

In this dissertation, we modify existing 2D interaction techniques and make them content-aware. We will show that such a modification can significantly improve the functionality of existing techniques, improving user performance and user experience.

2.6. Content-Aware vs. Context-Aware

The use of the phrase "content-aware" is uncommon when describing 2D user interface interaction. A more commonly used adjective is ***context****-aware*. Therefore, it is important to understand the differences between these two terms within the "context" of this dissertation.

Context-aware can refer to, amongst other things, the user scenario or task, as well as the system capabilities that surround the interaction. In essence, it is often defined not as the variables that *directly* involve the user's interaction with content, but rather as the variables that complement this interaction. On the other hand, *content-aware*, by our definition, refers to having knowledge of the characteristics of the content itself, such as properties that describe, for example, its physical appearance. In other words, the *content* could be placed in a different *context* and its inherent characteristics may not change. More simply put, context-aware interaction incorporates information *surrounding* the conditions under which interaction with the content is occurring, whereas content-aware interaction incorporates information *about* the content itself.

We do not (nor do we want to) claim that context-aware and content-aware knowledge fall into two mutually exclusive categories. While having knowledge of certain characteristics (e.g., the colors of the content) arguably makes one content-aware, and

having knowledge of certain environment information (e.g., the physical location of the computer display on which the content is viewed) arguably makes one context-aware, it is those more ambiguously-categorized properties (e.g., size or position relative to other content, or to the screen) that cross the boundary between the two categories.

Throughout this dissertation, we do not exclude such properties when defining what it means for a technique to be content-aware and when considering which properties would be useful in modifying the technique, but rather exploit them in any way to help improve interaction. The overall goal of this dissertation is to make interaction techniques more effective and to help users complete a task more efficiently (e.g., more quickly and/or accurately), as well as to improve the user experience. Our claim is that by being more aware of the content to which the technique is applied, this can be achieved, even if it requires making a technique context-aware in addition to content-aware. Furthermore, although we may refer to having knowledge of various content characteristics as being content-aware, we fully acknowledge that others may classify it as being context-aware.

Chapter 3

Content-Aware Transparency (CAT)

To view or interact with the content of an obscured window in conventional window managers, users are often forced to resize or move the obscuring window. Some users try to bring the obscured window to the top through the use of “alt-tab” or Apple’s Exposé. Some window managers incorporate traditional alpha-blending to allow users to see through obscuring objects to visualize what lies beneath them [NVIDIA]. However, as we discuss later, overlapping contents often interfere with each other, rendering them illegible, or make it difficult to identify to which window they belong.

In this chapter, we present *content-aware transparency* (CAT) [Ishak2004], an approach that selectively applies transparency to overlapping windows based on their content. CAT

also modifies the appearance of content to help reduce the interference caused from overlapping contents. To demonstrate and evaluate our approach, we have designed and implemented a version of CAT that allows a user to make efficient use of screen space by rendering unimportant window regions transparent and important window regions opaque, with a smooth gradient between them. Furthermore, our implementation takes into account characteristics of the obscured and overlaid content and applies appropriate image-processing filters and gradients to further reduce content ambiguity, as shown in Figure 3.1. This attempts to guarantee that the important content of overlaid windows will be readable at all times, while simultaneously exposing hidden content beneath the unimportant regions.

We have evaluated our CAT implementation against other transparency techniques: uniform alpha-blending with alpha set to 25% (i.e., 75% opaque) and 50%, both with and without blurring of underlying contents. We found that in studies asking participants to either quickly identify in which overlapping window a target icon appeared, or to quickly search for specified text in an overlaid window, CAT afforded the performance benefits of the less transparent 25% alpha with the "see-through" benefits of the more transparent 50% alpha. Also, six out of ten participants preferred CAT over any of the other techniques.

We have also developed a set of interaction techniques for use with CAT. The *pop-though* technique allows a user to interact with content beneath unimportant regions of an obscuring window without moving or resizing it. The *focus filter* is a "magic lens" [Bier1993] that allows a user to temporarily transform a filtered portion of obscured

content to its original unfiltered form, clarifying the exposed content beneath an obscuring window. Finally, since any pixel may render information from multiple windows, we allow users to determine the window with which they will interact by using a *mouse-over pie menu*.

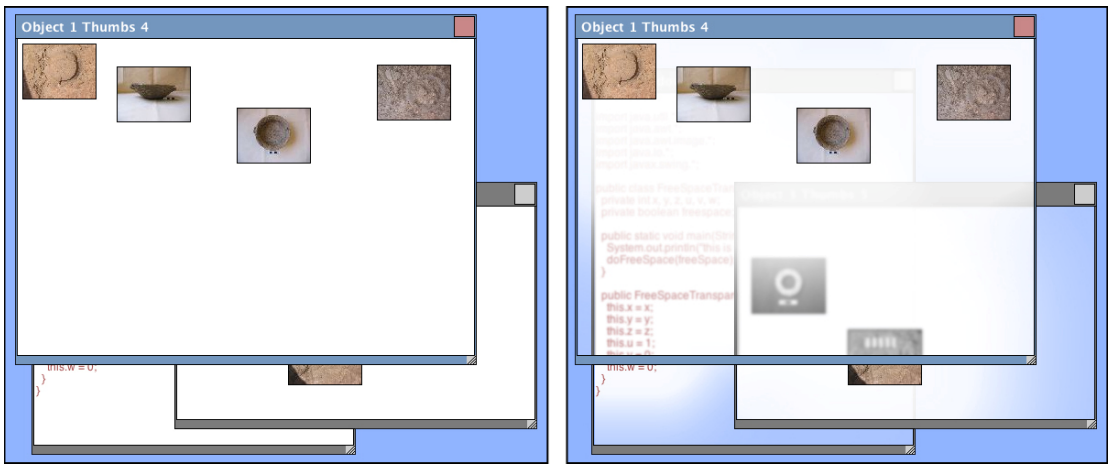

Figure 3.1: Windows rendered (left) *without*, and (right) *with* content-aware transparency, exposing blurred hidden content underneath.

3.1. Related Work

To increase the amount of simultaneously visible content of overlapping windows, some systems have tried rendering the entire obscuring window semi-transparently [Colby1991, Harrison1995a, Lieberman1994]. This traditional use of semi-transparency (accomplished by uniformly alpha blending the window with the contents of the frame buffer behind it), allows a user to visualize content that is behind an obstructing window, but often makes it difficult to determine visually which content belongs to which window, and in extreme cases can render the overlapping content illegible. Multiblending [Baudisch2004a] addresses this issue by allowing the use of different blending functions for different visual features, such as luminance and high spatial frequency. It modifies the

blending process to enable the preservation of more relevant features—if necessary at the expense of reducing other, less relevant features. Although multiblending preserves the visibility of both background and foreground windows containing familiar contents, users may have difficulty understanding unfamiliar overlapping contents. Users may also have difficulty with similar appearing overlapping contents, especially text. Our CAT implementation takes a different approach by selectively rendering all important regions opaque. By preventing obscured content from showing through those important regions, CAT increases the legibility of even unfamiliar overlapping information.

In Macintosh OS X, a text-only terminal window can have an opaque text color rendered on a semi-transparent background color. The Macintosh Command-Tab menu uses the same approach, rendering opaque icons on a transparent background. Although this approach may produce more legible overlaid content than does uniform semi-transparency, ambiguity can still arise when obscured pixels are blended with background pixels that lie in between the pixels of overlaid content, especially when the obscured pixels underneath are from content of the same type as the overlaid region above (e.g., text, in the case of the Macintosh terminal window, as shown in Figure 3.2). Harrison et al. [Harrison2001, Harrison1995a] introduced an anti-interference effect for the borders of opaque text of transparent overlaid windows, thus reducing the interference with obscured pixels. Paley [Paley2003] similarly shows how character outlining, combined with hue and font variation, motion, and antialiasing can contribute to improve overall readability of overlaid text. CAT generalizes Harrison et al.'s approach by applying an anti-interference opaque-to-transparent gradient around groups of objects, such that window regions, whether or not they contain content, are correctly perceived as part of

the same window, making the content more legible, unambiguous, and aesthetically pleasing, as explained in Section 3.4.2.1.

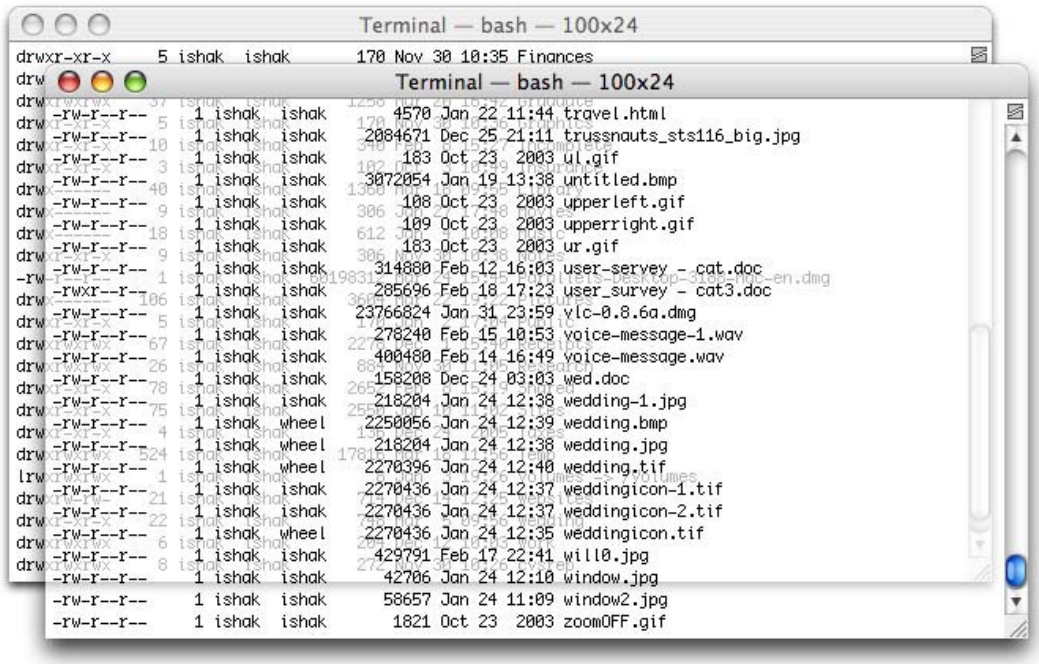

Figure 3.2: A Macintosh terminal window, rendered with an opaque foreground on a 40% transparent background, overlaid on a second terminal window with similar content. Ambiguity arises when obscured pixels are blended with background pixels that lie in between the pixels of overlaid content.

3.2. A Perceptual Approach to Transparency

We first summarize a perceptual model of transparency developed from research in psychophysics and psychology, to establish a psychophysical basis for our approach to designing an implementation of CAT. This helps us provide a foundation for our design, as well as show how our CAT implementation has distinct advantages over traditional alpha-blending implementations to help users understand when and how to perceptually discriminate among overlapping layers. More specifically, we will discuss the kinds of modifications that can be made to one or both of the overlapping layers to retain each

layer's individual legibility, as well as to reduce ambiguity in identifying to which container the content in each layer belongs.

3.2.1. The Episcotister Model and X-junctions: When to Scission

What is arguably the most influential theory of the perception of transparency was introduced by Fabio Metelli [Metelli1974], who based his theory on an *episcotister*, or a flat disc that has a sector (with relative area α) removed. If one rotates this disc at a sufficient speed on top of a bipartite surface with two halves **A** and **B**, with reflectances *a* and *b*, respectively, the disc appears to be semi-transparent, as shown in Figure 3.3.

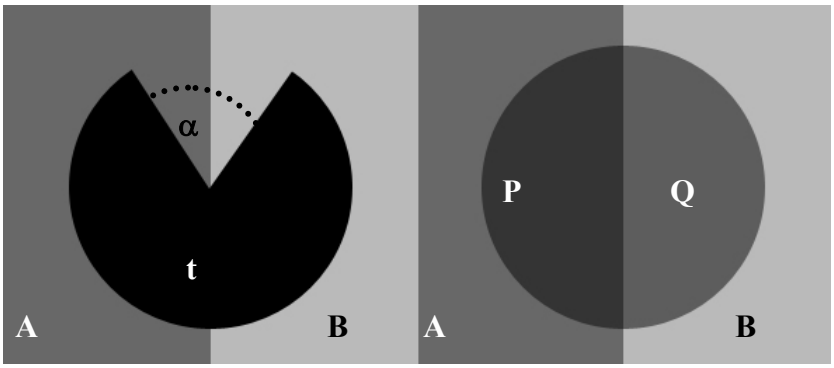

Figure 3.3 (based on a Figure in [Singh2002]): An episcotister (left) with an open sector with relative area α and reflectance t is placed in front of a background with two halves, each containing a distinct brightness value. When the episcotister is rotated sufficiently fast (right), it leads one to perceive that it is transparent. Metelli argued that the overlaid transparent region must preserve contrast polarity. In other words, if region A is darker than region B, then region P must be darker than region Q.

Metelli used this model to formulate many equations to describe perceptual transparency—more specifically, *when* and *how* to scission (i.e., partition) overlapping layers. He argued that if region **A** is darker than region **B**, then region **P** must be darker than region **Q**, and vice versa. In other words, regions **P** and **Q** must preserve *contrast polarity* relative to regions **A** and **B**. Further, the difference in luminance between regions

A and **B** must be greater than the difference in luminance between regions **P** and **Q**. This is known as the *magnitude constraint*. For many years, these constraints were used to help determine *when* an observer begins to perceive transparent overlapping objects, and *how* to scission the layers.

In the context of when to perceive transparency, Metelli's contrast polarity constraint was further extended to help classify the relation of the four luminance values at *X-junctions*, which occur whenever the boundary of an overlaid semi-transparent object crosses a contour in the background, as shown in Figure 3.4 (image and caption taken from [Fleming2005]). Beck and Ivry [Beck1988] showed how the order of local luminance values at an X-junction can help determine whether or not an observer will perceive transparency amongst two overlapping objects with the assumption that the two layers are homogeneous in transmittance (i.e., opacity) and reflectance. By drawing line segments between the four adjacent areas at the X-junction in order of increasing luminance, only one of three configurations can be drawn: a Z (also referred to as S), C, or criss-cross configuration.

As shown in Figure 3.4, Beck and Ivry explain that, in a Z (or S) configuration, either of the two objects will be perceived as transparent, while in a C configuration, only the rotated object will be perceived as transparent. Finally, in the criss-cross configuration, neither object will be perceived as transparent, all with some exceptions. One exception, for example is noted by Beck et al. [Beck1984] when they observed that strong figural groupings can allow overlapping objects to be perceived as transparent even though they may violate contrast polarity constraints.

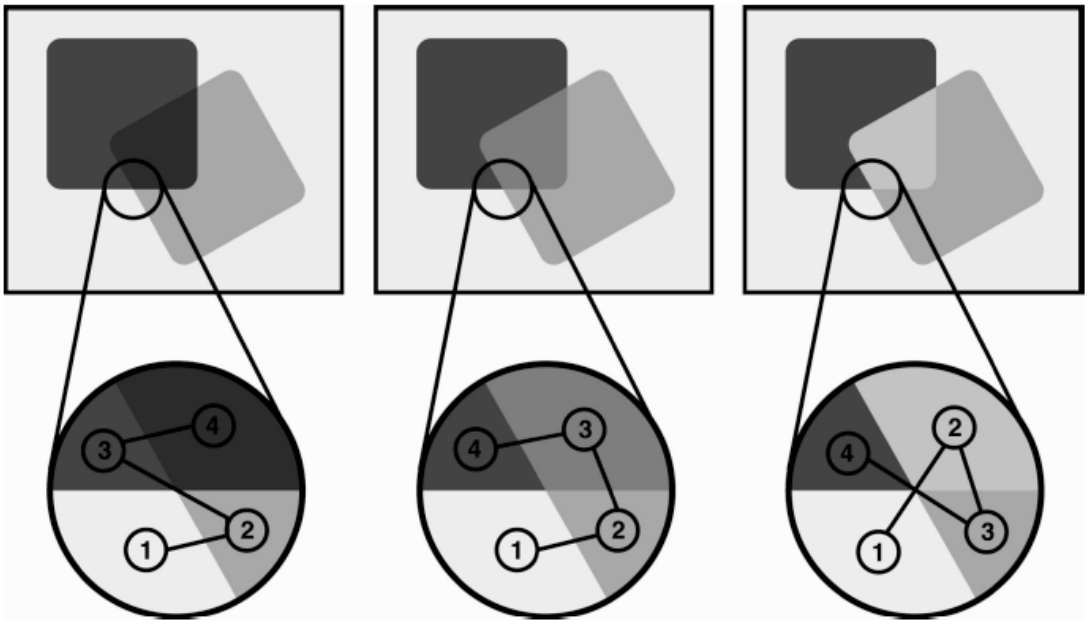

Figure 3.4: X-junctions occur where two contours intersect in the image. The ordinal relations between the intensities that form the junction determine which percepts are possible. The S-shaped ordering leads to a bistable percept in which either square can be seen as a transparent filter; the C-shaped ordering means that only the tilted square can be seen as transparent; the criss-cross ordering is inconsistent with either of the squares appearing transparent. (Image and caption taken from [Fleming2005].)

However, the model of perceptual transparency based on these constraints made the assumption that the X-junctions consisted of four distinct intensities. In many of today's user interfaces, users are rarely presented with a scenario where either the transparent overlapping layer, or the background, is homogeneous in transmittance and reflectance. Rather, screen objects are often textured or contain a range of intensities and colors. In the next section, we summarize an extension to Metelli's model that is generalized for heterogeneous textures.

3.2.2. Correcting the Episcotister Model

Singh and Anderson [Singh2002] showed how Metelli's episcotister model failed to explain why an observer sometimes failed to perceptually scission an overlaid semi-

transparent layer from a background layer, even though the magnitude constraint held. They showed that both (1) the difference in luminance between overlaid regions must be lowered in adjoining regions (i.e., Metelli's magnitude constraint), as well as (2) the difference in luminance divided by twice the mean luminance (also known as the Michelson contrast) must also be lowered if transparency is to be perceived.

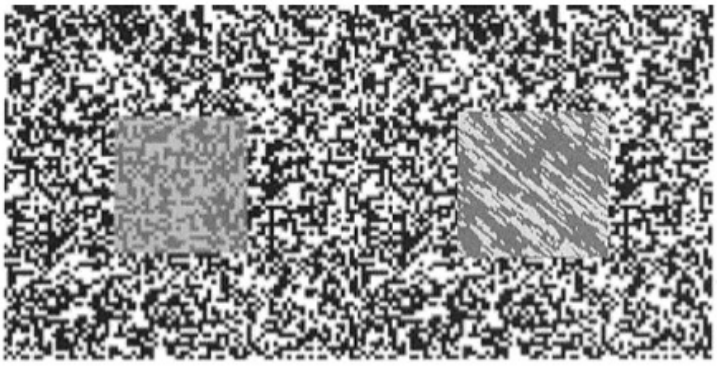

Figure 3.5: The texture in the low-contrast image region must be continuous with the texture in the high-contrast image region for a user to perceive transparency, as shown on the left. Transparency is not perceived when the texture in the low-contrast image region is discontinuous with the texture in the high-contrast image region, as shown on the right. (Image taken and caption adapted from [Singh2002].)

Their studies also found that the perception of transparency is more reliable when there is a "continuous" texture (or range of luminance values), rather than the two distinct constant values of Metelli's model, seen across the border of the transparent overlaid surface. This continuity can only vary in the contrast and mean luminance across that border, as shown in Figure 3.5. However, a caveat: the above analysis is based on the assumption that the transparent layer does not modify or augment the background texture seen through that transparent layer. In other words, a transparent layer with light-scattering properties (e.g., blurring what is seen through it) in addition to light-transmitting properties, can also allow one to perceive transparency without the constraint

of a continuous texture across the transparent layer's contours, as mentioned above. This leads us to believe that the use of appropriate image-processing filters that produce light-scattering effects applied to the underlying content may help users disambiguate overlapping window contents. We show how we selectively apply these filters in Section 3.4.2.1.

Singh and Anderson's modification also allows us to understand the effect of *unbalanced transparency*, which occurs when the transparent layer is inhomogeneous in either transmittance or reflectance (or both). They were able to show that their model holds for "locally balanced" transparency—when the changes in transmittance and reflectance are smooth and gradual. This gives us confidence that our use of opaque-to-transparent gradients between important and unimportant window regions does not hinder the perception of an overlaid transparent object, as long as it is smooth and gradual, thus leading one to perceive continuity in the transparent layer. These gradients are discussed in greater detail in Section 3.4.2.1.

3.2.3. How to Scission Transparent Layers

As stated earlier, Metelli adequately attributed luminance relationships between overlapping layers to correctly determine *when* a user perceives transparency. However these same relationships could not hold when observers were asked to determine *how* to scission, or to identify different levels of transparency. Singh and Anderson performed several experiments to determine how an observer computes the transparency of an overlaid layer and found that the visual system does not use a strict luminance contrast, but again, a Michelson contrast to assign transparency. This helped explain why users

perceived a rotating black episcotister as more transparent than a white one, when a sector of the same size was removed from each. This is because the black episcotister generates a higher Michelson contrast than the white one, although the two produce an identical luminance contrast.

Akerstrom and Todd [Akerstrom1988] showed that color differences between overlapping layers facilitate the segregation of overlapping objects. This implies that, given illegible overlapping objects of similar color, modifying the color of one of the layers can help the user scission the two layers, possibly making each layer more legible. Although Akerstrom and Todd used only primary colors in their experiments, this prompts us to investigate the application of a color-change filter to obscured content to help disambiguate overlapping material in various user interface scenarios. We investigate the use of these filters in Section 3.4.2.1, showing how a desaturation (color removal) filter can aid users in determining how to scission.

Akerstrom and Todd also investigated the use of orientation to help scission overlapping transparent layers. They found that there was no evidence of layer segregation distinguished only by the orientation of the objects in each layer. However, their test cases used overlapping textures containing random-line stereograms that were not typical of user interface objects. Although we do not use orientation in our implementation of CAT, we will not rule out its potential use, but rather discuss possible ways in which orientation in combination with other types of modifications, such as animation, can help users understand how to scission overlapping layers. We discuss this in Section 6.2, where we talk about alternatives to our implementation of CAT.

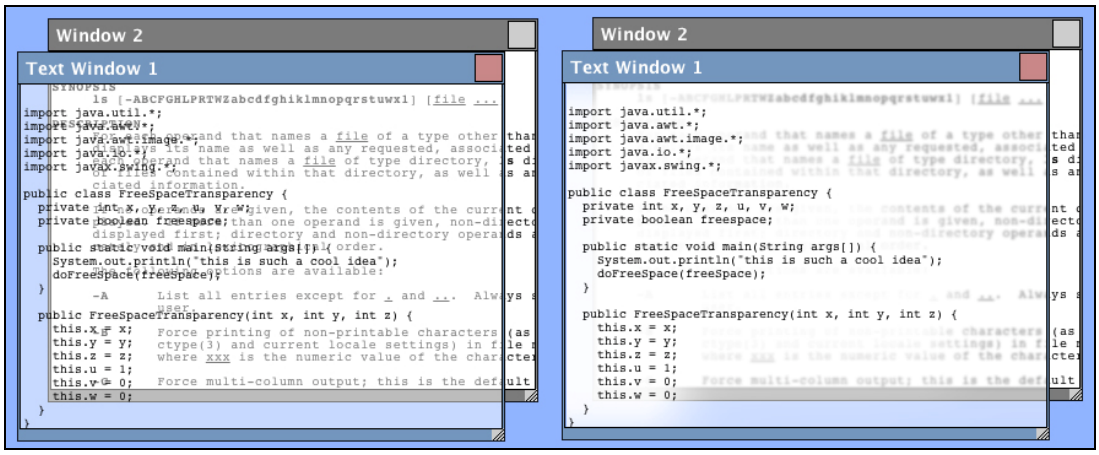

Figure 3.6: Window rendered (left) with opaque text on uniformly transparent background (alpha = 0.5), and (right) using CAT (alpha varying from 0.5 to 1.0), where background pixels of important regions are rendered opaque, producing more legible overlaid content.

3.3. CAT Overview

Our CAT implementation allows a window's application to inform the rendering engine of the unimportant regions, as described in Section 3.4. It evaluates and compares the characteristics of window content, such as the high-level type (e.g., text, images, icons, or a combination), colors, and spatial frequency of both the obscuring and hidden window content. Based on the combinations of these characteristics, image-processing filters, such as Gaussian blur [Baudisch2004a, Beryl2006] or desaturation, alone or in combination, are applied to the content exposed through the unimportant regions of the obscuring window. Additionally, gradients are applied between the opaque and transparent regions. This is an example of how a content-aware technique varies the display of content across the *appearance domain*, such that the *color and transparency* of the pixels making up the content are modified to allow important regions to maintain their legibility. An example of overlapping text is shown in Figure 3.6.

3.4. CAT Implementation

A key issue in implementing CAT, discussed below, is to identify the important and unimportant regions of a window. This determines which regions of that window are to be rendered opaque and which are to be rendered transparent. When rendering the gradient between these regions, it is desirable that the opaque-to-transparent transition be made smooth and visually appealing, since an abrupt boundary could imply a separation of the opaque and transparent sections, leading the user to believe that one window is actually divided into multiple objects. We render no pixel 100% transparent, since it would completely expose the content underneath, potentially misleading the user into believing that this content was associated with the overlaid window. We have found that a 50% transparency value works well. We have also found that rendering important regions with a 10% transparency (90% opacity) value allows for some visibility through overlaid content, with little risk of content ambiguity; this supports earlier evaluations of usable and efficient transparent user interfaces, such as an experiment that employed a variant of the Stroop effect [Stroop1935], in which subjects viewed color names through a transparent colored patch [Harrison1995a, Harrison1995b].

Since users often interact with window decoration (title bar, menus, border, scroll bars), we consider this important content. Therefore, our CAT implementation renders the pixels that make up these regions at 90% opacity, in the same way that it treats important regions of the window body. In the case of opaque windows, keeping the window decoration at 90% opacity allows users to disambiguate window boundaries more easily. We have found that it is not necessary to incorporate gradients connecting the nearly opaque pixels making up the window decoration to nearby unimportant regions.

3.4.1. Important vs. Unimportant Regions

We have considered several approaches to determining important and unimportant regions in a window. In one approach, the rendering engine can classify window regions containing only a particular background color or texture as unimportant. Alternatively, the user might explicitly identify unimportant regions manually with a mouse or touchpad, or automatically by using an eye tracker to detect window regions on which their gaze does not dwell. Regions containing empty space (i.e., ones devoid of text, icons, and images) could also be automatically classified as unimportant, which is what we do in our implementation. However, since some applications utilize empty space (e.g., displaying page margins in a document), we allow the window's application to notify the rendering engine of these unimportant regions. This may require the application designer to provide a bitmap of alpha values, or specify a set of geometric bounds with characterizations of their contents, as in our testbed application.

3.4.2. Classifying Content

A CAT window takes into account characteristics of both its own content and that of the windows it obscures to determine a gradient and transparent filter combination that will promote efficient use of screen space, and unambiguous visualization of the overlapping data.

3.4.2.1. Opaque-to-Transparent Gradients

Our CAT implementation uses opaque-to-transparent gradients following a Gaussian falloff between important (i.e., opaque) and unimportant (i.e., semi-transparent) regions. We also experimented with linear gradients, and found that a Gaussian-shaped gradient

provides a more aesthetically pleasing transition. Using a 2D isosurface equation, for each pixel (p_x, p_y), a scalar value s is computed as a weighted sum (s_{sum}) of N intermediate scalars s_i, each computed relative to one of N objects with centroid (X_i, Y_i) in the window, added to the weighted scalar maximum, s_{max}, for that pixel. Adding a weighted s_{max} allows a pixel closer to content than another pixel with an equal s_{sum} value to have a higher overall scalar value. We tested many different weights, and found that the value of 0.3 for β seemed to produce the most aesthetically pleasing results. Objects can vary from an icon to a block of text and can have arbitrary dimensions. The scalar value s for each pixel is computed as:

$$s = \beta s_{max} + (1-\beta) s_{sum},$$

$$\text{where } s_{sum} = \sum_{i=1}^{N} \frac{s_i}{d} \text{ and } s_i = e^{v\left((p_x - X_i)^2 + (p_y - Y_i)^2\right)}.$$

We experimented with the exponent value v and have found that –5 works reasonably well, although all real numbers we have tried, ranging from –3 to –7, produce acceptable results. The variable d is the desired gradient distance from opaque to transparent pixels. Pixels containing scalar values of 1.0 or above are rendered opaque, 0.0 is rendered transparent, and intermediate values are rendered with a proportional opacity. The amount of otherwise hidden content revealed through overlaid windows depends on the combination of the d value and how small and widely interspersed the unimportant regions are. For example, a large d value with many small, interspersed regions will reveal little hidden content.

Overlaid Content			Obscured Content			Filter(s) Applied to Obscured Content
Type	Color	Freq.	Type	Color	Freq.	
text	C_X	–	text	C_Y	–	Blur (r=3)
text	C_X	–	text	C_X	–	Desaturation, Blur (r=5)
not icons	–	–	NC	–	high	Desaturation, Blur (r=5)
not icons	–	–	NC	–	low	Blur (r=3)
icons	–	–	icons	–	–	Desaturation, Blur (r=5)
icons	–	–	not icons	–	high	Desaturation, Blur (r=5)
icons	–	–	not icons	–	low	Blur (r=3)

Table 3.1: The image-processing filters applied to obscured content, based on characteristics of both the overlaid and obscured contents. C_X and C_Y represent arbitrary colors. "–" signifies that it is not considered. The "Freq." column specifies whether the content has predominantly low or high spatial frequencies.

3.4.2.2. Content-Dependent Transparency Filters

Using traditional, unconditional whole-window blending, certain window layout arrangements make it difficult for the user to correctly associate overlaid content with the window in which it resides. In Section 3.2.2, we discussed how an overlaid transparent layer with light-scattering properties can help an observer perceptually scission it from an underlying layer. In our implementation, we evaluate overlapping window contents to apply the most suitable filters to allow correct scission of the layers. Our goal is to afford correct perceptual scission, while modifying the underlying content as little as possible (or not at all). For example, in comparing overlaid versus obscured contents, if the types are both text or both image, we apply a light Gaussian blur (radius=3) for content disambiguation. The rationale for such a non-aggressive modification stems from the continuous textures normally inherent in images and text, as also discussed in Section 3.2.2, which helps one to perceive and separate overlapping transparent layers more

easily than with homogeneous layers. When both types are icons (or thumbnail images), we apply a desaturation followed by a more aggressive Gaussian blur (radius=5) to disambiguate more than with the lighter Gaussian blur alone. This is because icons (or thumbnails) can have arbitrary shapes and locations within both the overlaid and obscured windows, and often lack a continuous texture, sometimes making it difficult to associate the icons with the window in which they reside. The application of a desaturation and a Gaussian blur afford a greater disparity in the appearances of overlaid and underlying content, which we feel helps in content disambiguation. Table 3.1 illustrates the conditions under which we apply different image-processing filters to hidden content.

3.5. Demo Application

To test our ideas, we have developed a Java application that allows users to visualize archaeological data within traditional 2D rectangular windows, as shown in Figure 3.7. Users can view images, thumbnails, and text pertaining to objects excavated from a dig site. Regardless of content type, every window can be moved and resized. Icon windows allow adding, deleting, and moving thumbnails and icons to dynamically create and destroy important regions. Using a control panel interface, users can specify whether to render the windows using CAT, traditional uniform semi-transparency, or no transparency at all. To improve performance, image-processing techniques are not applied to content while windows are being resized or moved.

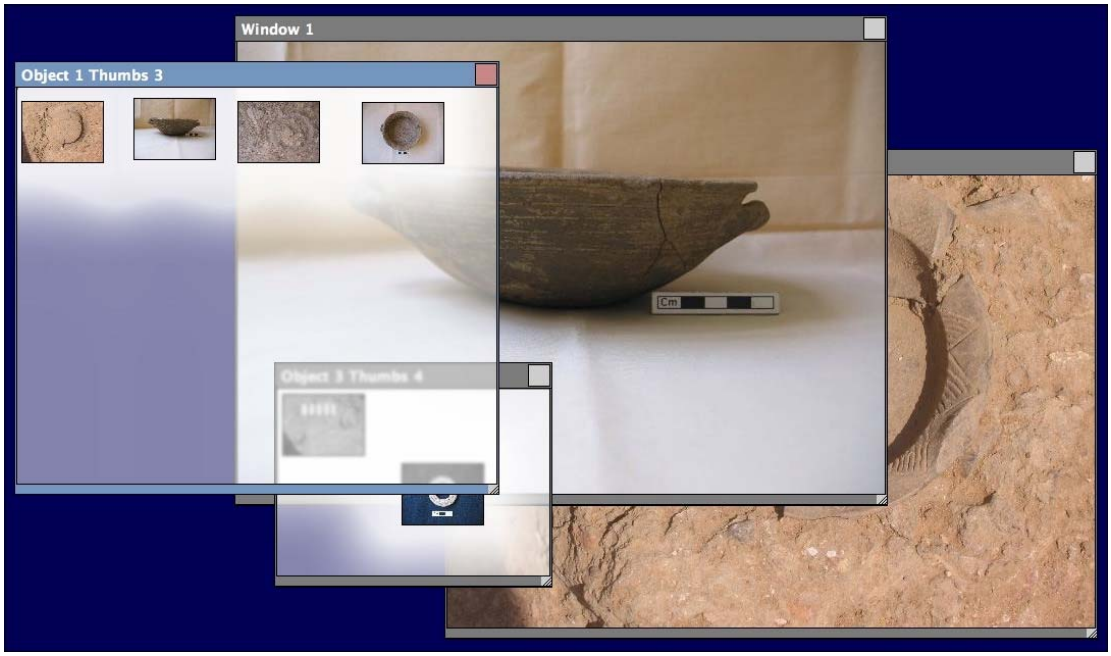

Figure 3.7: A screenshot of the CAT demo application, in which a user can peruse through thumbnail and high-resolution images of archaeological data.

3.6. CAT User Study

To measure the effectiveness of our CAT implementation, we performed experiments to see if users were more effective and if they preferred to use transparent user interfaces that employ CAT as compared to those that do not. Our goal was not to prove that our CAT implementation outperformed all others in every user interface scenario, but rather to support our intuition that, depending on the content, the application of non-uniform transparency could provide an advantage over uniform transparency. We also wished to determine if the decisions we made about when and what image-processing filters to apply to underlying content (again, given a particular user interface scenario) were effective and if they also provided an added performance benefit as compared to without their application (in those same scenarios).

Ten paid participants (8 male, 2 female), ages 20–36, were recruited by email to undergraduate and graduate Computer Science student mailing lists at Columbia University. Every participant was a frequent computer user, and most did not have experience with transparency in user interfaces. Those with corrected vision wore their glasses or contacts, as they would when normally using a computer. We asked the participants to perform two tasks within one 45-minute session: a container identification task and a search task. Both tasks involved two overlapping transparent windows. The container identification task was always performed first and required participants to identify whether a highlighted icon appeared in the top (near) or in the bottom (far) window. The search task required participants to identify if the top window contained a particular icon. Both were to be completed using four techniques, described below.

3.6.1. ALPHA-50 Technique

The ALPHA-50 technique involves rendering each pixel with a 50% transparency level. This is done by alpha-blending 50% of its color (each color channel separately) with 50% of the color of the frame buffer behind it. This kind of traditional alpha-blending is a popular technique used in existing commercial windowing systems, and therefore, we felt was an adequate baseline technique. Although some commercial systems allow a user to specify a custom transparency level for the background (e.g., the Mac OS X terminal window), in most systems, such as NVIDIA's nView and Microsoft Word for Mac palettes, a 50% transparency is used as the default value.

3.6.2. ALPHA-50+B Technique

The ALPHA-50+BLUR technique is the same as ALPHA-50, but has an added light scattering effect with the application of a Gaussian blur filter (radius=3) to underlying pixels seen through the overlaid window.

3.6.3. ALPHA-25 Technique

The ALPHA-25 technique renders each pixel with at a 25% transparency level. This is done by alpha-blending 75% of its color (each color channel separately) with 25% of the color of the frame buffer behind it. This produces a less "see-through" effect as compared to the ALPHA-50 technique, but still allows a user to perceive content that lies beneath overlaid windows.

3.6.4. ALPHA-X+BD Technique

The ALPHA-X+BD technique takes each pixel making up a window and renders it using CAT by alpha-blending a variable percentage of its color (but uniformly across each color channel) with the color of the frame buffer behind it. This percentage is determined by an exponential function of the distance between the pixel and important content within the window, as described in Section 3.4.2.1. Percentage values between 50% (for pixels far away from content) and 100% (for pixels very close to content) were used. Light scattering effects in the form of a Gaussian blur (radius=3) and a desaturation filter were applied to any underlying pixel seen through the overlaid window.

3.6.5. ALPHA-X Technique

After five participants successfully completed the study using the four techniques described earlier, we decided to add a fifth technique, called ALPHA-X: a technique similar to ALPHA-X+BD in its use of CAT, except that it does not apply any image processing filters to the underlying content. The reason for the addition of this technique was to understand if the variation of transparency within a window alone produces a performance benefit over the ALPHA-50 and ALPHA-25 techniques, which use uniform transparency values. The next day, two of the five participants returned to complete the container-identification task using the ALPHA-X technique. For the five remaining participants, the order in which the five techniques were used was counterbalanced. This allowed seven of the ten total participants to use all five techniques.

3.6.6. Experimental Setup

The experiments were performed on a PC running Windows XP Professional using a single Dell UltraSharp 2407WFP 24" LCD display running at 1920×1200 pixel resolution (@ 60Hz). A standard English keyboard was used for input.

3.6.7. Container-Identification Task

We first asked participants to perform a container-identification task with repeated trials. For each trial, a participant was presented with a screenshot of two overlapping windows. They were asked to identify the appropriate window that contained a "target" icon placed amongst other icons of identical appearance, all randomly positioned within the overlapped portion of the two windows (the two windows always overlapped substantially). This icon appeared in either the top (near) or bottom (far) window, and

always appeared in each trial, almost fully unobstructed. The target was pointed to by a distinct red arrow and was given the distinct name “target” written beneath it in a black 13pt Lucida Grande sans-serif font. All other icons had distinct names starting with “icon” followed by an integer (e.g., “icon 5”). A screenshot of an actual trial using the ALPHA-25 technique is shown in Figure 3.8.

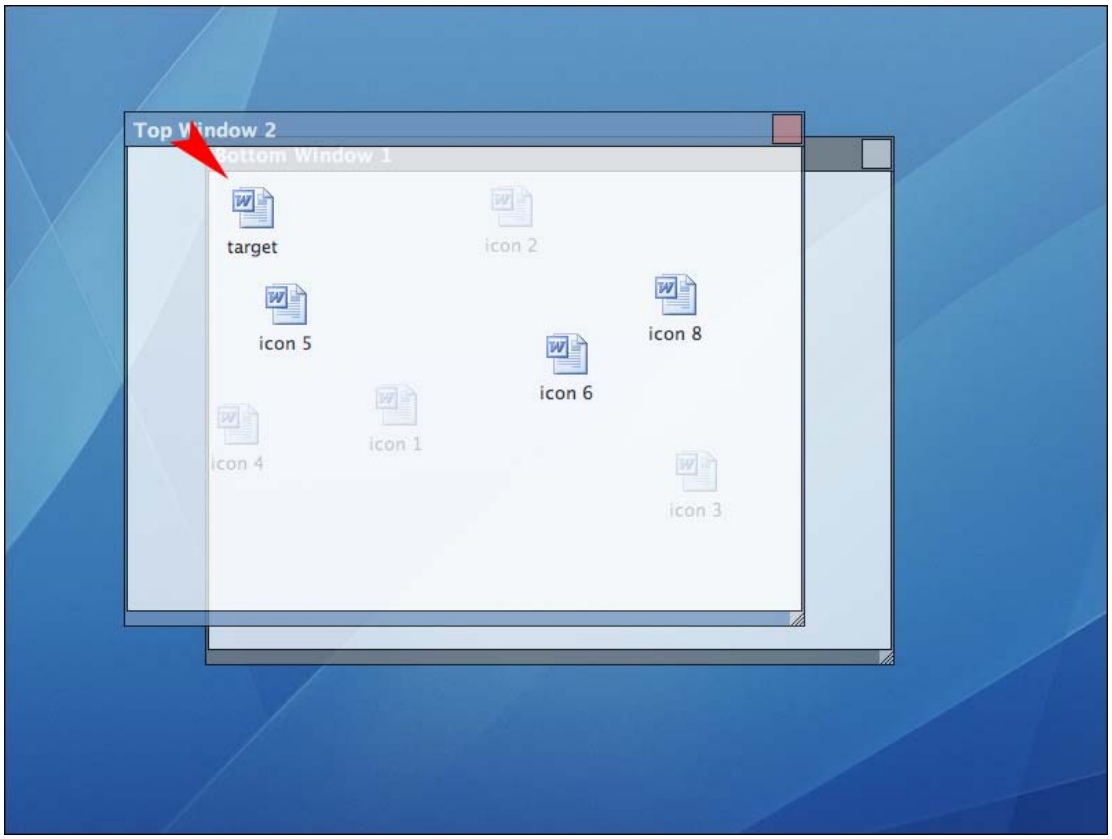

Figure 3.8: A screenshot of a trial used in the container-identification task using the ALPHA-25 technique. Participants had to decide whether the highlighted icon (pointed to by a red arrow) was in the top or bottom window. The red arrow was visible in each trial.

Each participant was verbally instructed to indicate within which window they felt the highlighted icon appeared by pressing the up-arrow key on the keyboard to signify the top window, and the down-arrow key to signify the bottom window. They were verbally instructed to perform each trial as fast as possible, while maintaining correctness. After

each trial, a confirmation screen appeared indicating that the participant had given a correct (green check) or incorrect (red "X") response. The participant would then press the spacebar key to proceed to the next trial.

Figure 3.9: Screenshot snippets showing a side-by-side comparison of the visual appearance of icons use in trials of the container-identification task for each of the five techniques. For each pair of icons, the left one shows how an icon in the bottom window would appear through the top window. The right one shows how an icon in the top window would appear.

Figure 3.10: Low spatial frequency icons (three on the left) and high spatial frequency icons (three on the right) used in the container-identification task.

3.6.7.1. Procedure

A within-subjects, repeated measures design was performed using the five techniques described earlier (Figure 3.9 demonstrates the differences in visual appearance of icons in the top and bottom windows using each of the five techniques) and two types of icons: those with predominantly low spatial frequencies and those with predominantly high spatial frequencies, as shown in Figure 3.10. Each participant completed 400 trials (5 techniques × 2 frequency levels × 40 repetitions) and the order in which the techniques were used was counterbalanced across all participants. Frequency levels were randomized within each technique and were not consistent across techniques.

3.6.7.2. Hypotheses

We formulated the following hypotheses before running the container-identification experiment:

(H1) Due to the similarity in appearance of overlapping contents, the ALPHA-50 technique will be the *slowest* in identifying within which of two overlapping windows an icon appeared.

(H2) Due to the similarity in appearance of overlapping contents, the ALPHA-50 technique will be the *most erroneous* in identifying within which of two overlapping windows an icon appeared.

(H3) Due to a large discrepancy in the appearance of overlapping contents, the ALPHA-X+BD technique will be the *fastest* in identifying within which of two overlapping windows an icon appeared.

(H4) Due to a large discrepancy in the appearance of overlapping contents, the ALPHA-X+BD technique will be the *least erroneous* in identifying within which of two overlapping windows an icon appeared.

(H5) Regardless of technique, trials will be completed faster and less erroneously with icons containing high spatial frequency content versus icons containing low spatial frequency content.

In summary, when performing a container-identification task using the five techniques, we presumed that ALPHA-50 would be the worst performer, ALPHA-X+BD would be the best performer, and regardless of technique used, interacting with low spatial frequency icons would be more difficult than with high spatial frequency icons.

3.6.7.3. Results

We analyzed the results, measuring completion time, success rate, and subjective preference. Only successful trials were included in the completion-time analysis. To compensate for right-skewing frequent to human response time, we used median values for our completion-time analysis. This also helps reduce the significance of outliers in the data. We used $\alpha = 0.05$ to denote statistical significance.

Completion-time analysis: We performed a 4 (Technique) × 2 (Spatial Frequency) repeated measures ANOVA on median completion times for successful trials using the 10 participants as the random variable. (Since three of the participants did not attempt ALPHA-X, we excluded this technique from this analysis.) Spatial frequency ($F_{(1,9)}$ = 11.215, $p < 0.01$) and Technique ($F_{(3,27)}$ = 4.821, $p < 0.01$) each had a significant main effect on completion time, while the interaction between Technique and Spatial Frequency did not ($F_{(3,27)}$ = 2.142, $p = 0.118$).

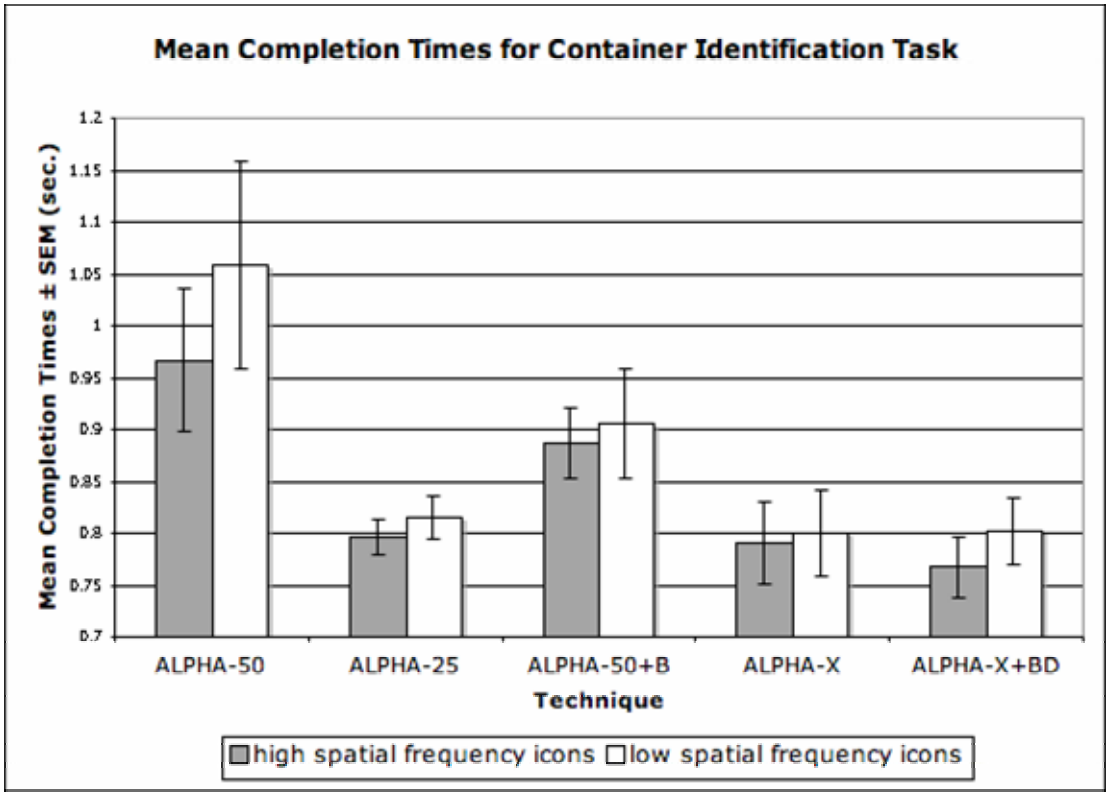

Figure 3.11: Mean completion times for the container-identification task, showing that the ALPHA-50 technique performed significantly worse than all other techniques.

Figure 3.11 shows mean completion times for the container-identification task for all five techniques broken down by the spatial frequency level (classified as either "high" or "low") of the icons used in the task. (Note that completion times for only 7 of the 10

participants were used for the ALPHA-X technique.) The ALPHA-25, ALPHA-X, and ALPHA-X+BD techniques all performed comparably well, followed by the ALPHA-50+B technique, and then, finally the ALPHA-50 technique. Performing a paired-sample t-test between completion times of the ALPHA-50 and ALPHA-50+B techniques shows a significant main effect ($t_{19} = 1.792$, $p < 0.05$), thus confirming H1. Performing paired-sample t-tests on the completion times between each pair of the top three techniques showed no significant differences, thus failing to prove H3. However, it is interesting to note that ALPHA-25 and ALPHA-X performed comparably, even though ALPHA-X applies a higher transparency than ALPHA-25 to certain parts of the window (those parts containing no content). This leads us to believe that, using ALPHA-X, one can take advantage of the performance of the less-transparent ALPHA-25, but still maintain the high transparency of ALPHA-50, thus getting the best of both worlds.

Finally, a paired-sample t-test on completion times, comparing trials using icons of high spatial frequency vs. those using icons of low spatial frequency (across all techniques), showed that participants performed significantly faster with icons of higher spatial frequency ($t_{46} = 3.75$, $p < 0.001$), partially confirming H5.

Error-rate analysis: We performed a 4 (Technique) × 2 (Spatial Frequency) repeated measures ANOVA on error rates, using the 10 participants as the random variable. (Again, since three of the participants did not attempt the ALPHA-X technique, we excluded it from this particular analysis). Technique had a significant main effect ($F_{(3,27)} = 3.591$, $p < 0.05$) and Spatial frequency had a borderline main effect ($F_{(1,9)} = 4.817$, $p =$

0.056) on error rate, while the interaction between them ($F_{(3,27)}$ = 0.637, p = 0.598) did not.

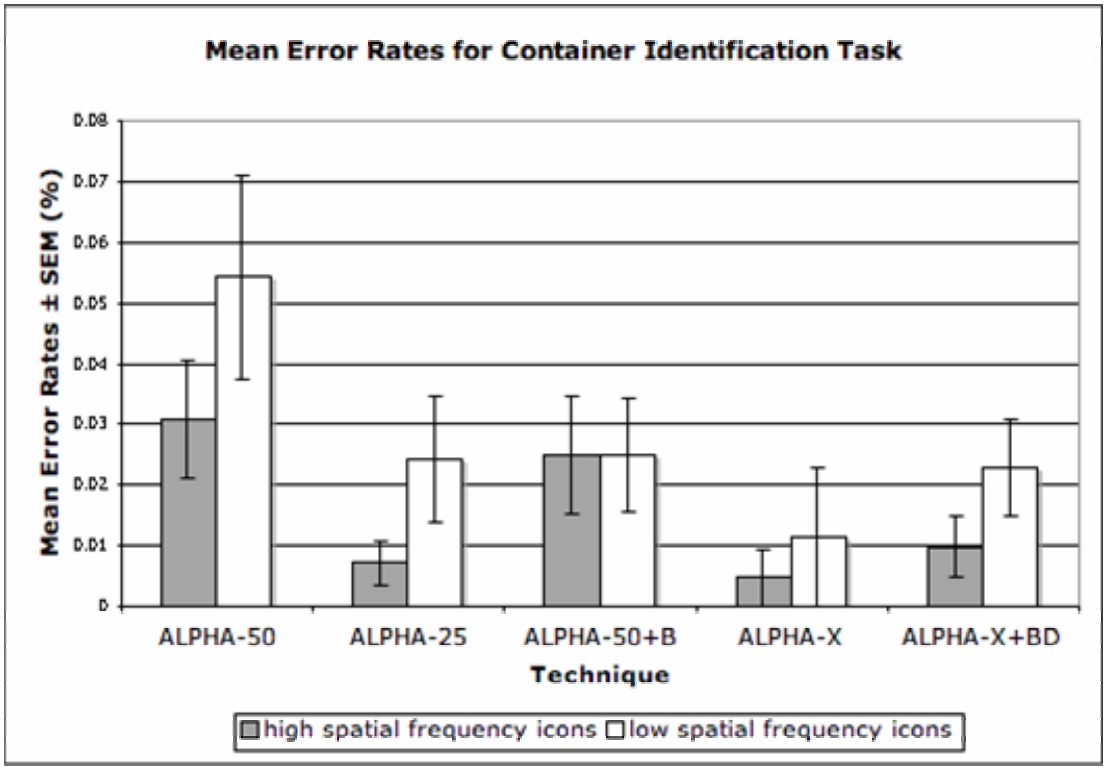

Figure 3.12: Mean error rates for the container-identification task, showing that ALPHA-50 proved to be the most erroneous technique. However, they could not show which technique was the least erroneous.

Performing a paired sample t-test between the two most erroneous techniques, ALPHA-50 and ALPHA-50+B, showed little significant effect on error rate (t_{19} = 1.440, p = 0.083), thus failing to confirm H2. As well, we see from Figure 3.12 that ALPHA-X allowed participants to be less erroneous than ALPHA-X+BD (thus failing to confirm H4), and surprisingly, the difference in error rate between them was borderline significant (t_{13} = 1.764, p = 0.051). Although difficult to prove, we speculate that this may have occurred because two of the seven participants had the benefit of using ALPHA-X after performing the container-identification task with the other four techniques, since the

ALPHA-X technique was added after these two had finished the experiment. The extra session was performed on a different day with the same verbal instructions given; however, the added experience and slightly different experimental conditions (e.g., different time of day) may have contributed to the lower, although borderline significant, mean error rates. Another possible explanation stems from the use of the desaturation and blur filters in the ALPHA-X+BD technique. We speculate that since the appearance of underlying content using the ALPHA-X+BD technique differed from the overlaid content across several channels (i.e., *color* due to the desaturation, *luminance* due to the semi-transparent overlaid background, and *contrast* due to the Gaussian blur), the underlying content was perceptually more difficult to scission from the overlaid content, as compared to with the use of the ALPHA-X technique, which modified the appearance of the underlying content across only one channel (i.e., *luminance* due to the semi-transparent overlaid background). This is because we speculated that participants determined if an icon was in the top or bottom window based on its relative difference in appearance from other icons. Given the semi-transparent white background of the overlaid window acting as a filter applied to anything seen through it, participants may have been looking only for an increase in luminance (due to alpha-blending the color of the icons' pixels with white) to identify an icon as being contained in the bottom window. However, given a desaturated and blurred target icon (again, given a semi-transparent white background of the overlaid window), although its appearance greatly differs from its unmodified version, the removal of color and contrast made it more difficult to compare relative luminance.

Finally, performing a paired-sample t-test on the error rates, comparing the use of high spatial frequency icons versus low spatial frequency icons across all techniques, showed a significant difference ($t_{46} = 2.253$, $p < 0.015$), thus fully confirming H5.

After the completion of the container identification task, we considered whether error rates may have been affected by participants possibly having trouble correctly mapping input keys to window stack order (i.e., using the up- and down-arrow keys to indicate the top and bottom windows in depth, respectively). Perhaps, since trials were to be completed as fast as possible, participants may have mistakenly misused the up- and down-arrow keys to indicate the top and bottom windows along the *y*-axis. However, an analysis of the data showed that unsuccessful trials included a comparable number of occurrences of pressing the up-arrow when the top window was lower than the bottom window along the *y*-axis, and pressing the down-arrow when the top window was higher than the bottom window along the *y*-axis.

Qualitative Evaluation Analysis: We asked each participant to fill out a hand-written questionnaire rating each technique for the container-identification task on a Likert scale of 1–5 (5, being the best). Figure 3.13 shows the average ratings for each technique, evaluating its ease of use, satisfaction, and intuitiveness. The ALPHA-50 technique was rated the worst, followed by the ALPHA-50+B technique. Comments about ALPHA-50+B included “this technique suffered the same problems as the ALPHA-50” and “difficult to separate what is in the foreground and what is in the background” indicating that the blurring effect did not facilitate an easier disambiguation between overlaid and underlying contents. Six out of ten participants ranked either ALPHA-X or ALPHA-

X+BD easier, more satisfying and more intuitive than the rest. Comments about these included “the easiest one”, “great technique”, and “this was probably the best, but only because it was not transparent around the icon,” indicating that the opaque background facilitated an easier disambiguation between overlaid and underlying contents. However, one participant did say that the ALPHA-X+BD technique was “the most difficult and confusing technique, feels too much information tries to approach the eye, and there aren’t many clues to separate them.” Interestingly, this same participant performed the container-identification task the fastest with this technique and this was the only technique that was completed by this particular participant with no errors.

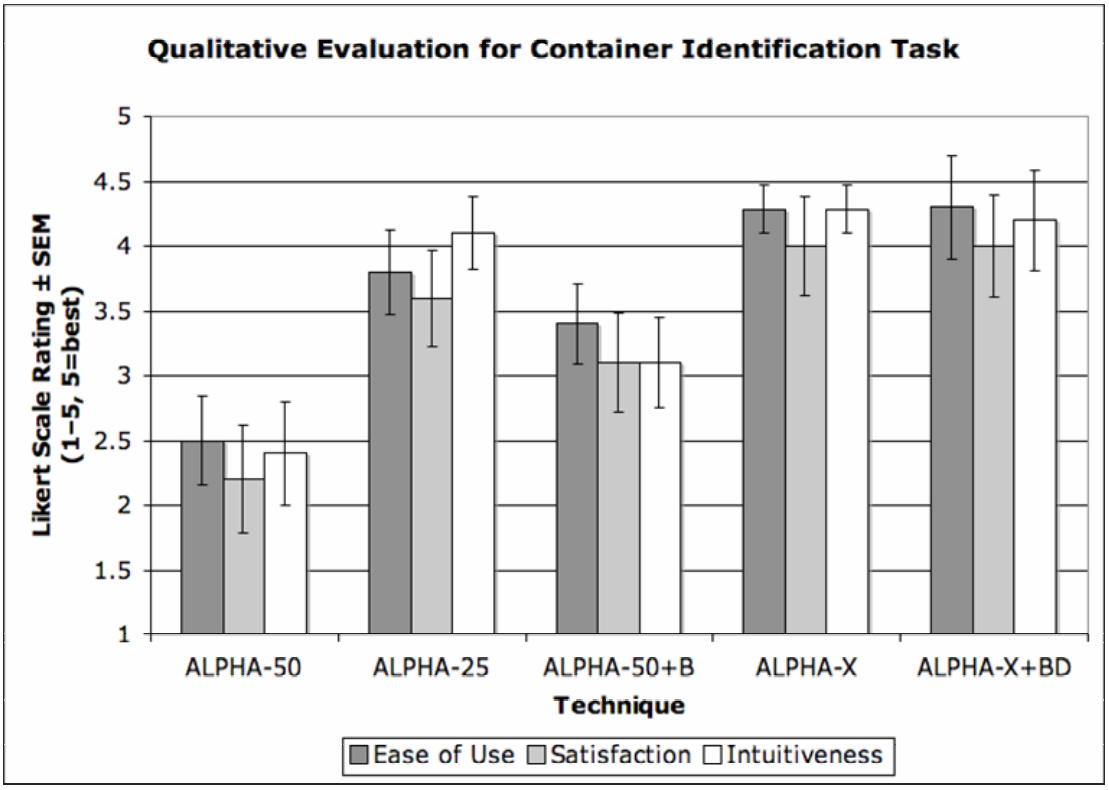

Figure 3.13: Results from the questionnaire for the container-identification task show that the ALPHA-50 technique was rated the most difficult, most frustrating, and most confusing.

3.6.8. Search Task

Immediately after the container-identification task, we asked the same participants to perform a search task using a similar experimental setup. Again, for each trial, the participant was presented with a screenshot of two overlapping windows. The top (near) window contained several icons of identical appearance (from those shown in Figure 3.14), all randomly positioned within the overlapped portion of the two windows (the two windows always overlapped). The bottom (far) window contained various types of content, including icons of identical appearance to those in the top window, high spatial frequency images (from those shown in Figure 3.15), and plain text of different sizes. A screenshot of an actual trial using the ALPHA-25 technique is shown in Figure 3.16.

Figure 3.14: The icons used in the search task. For each trial, multiple icons of identical appearance were used.

Figure 3.15: The predominantly high spatial frequency images used in the search task.

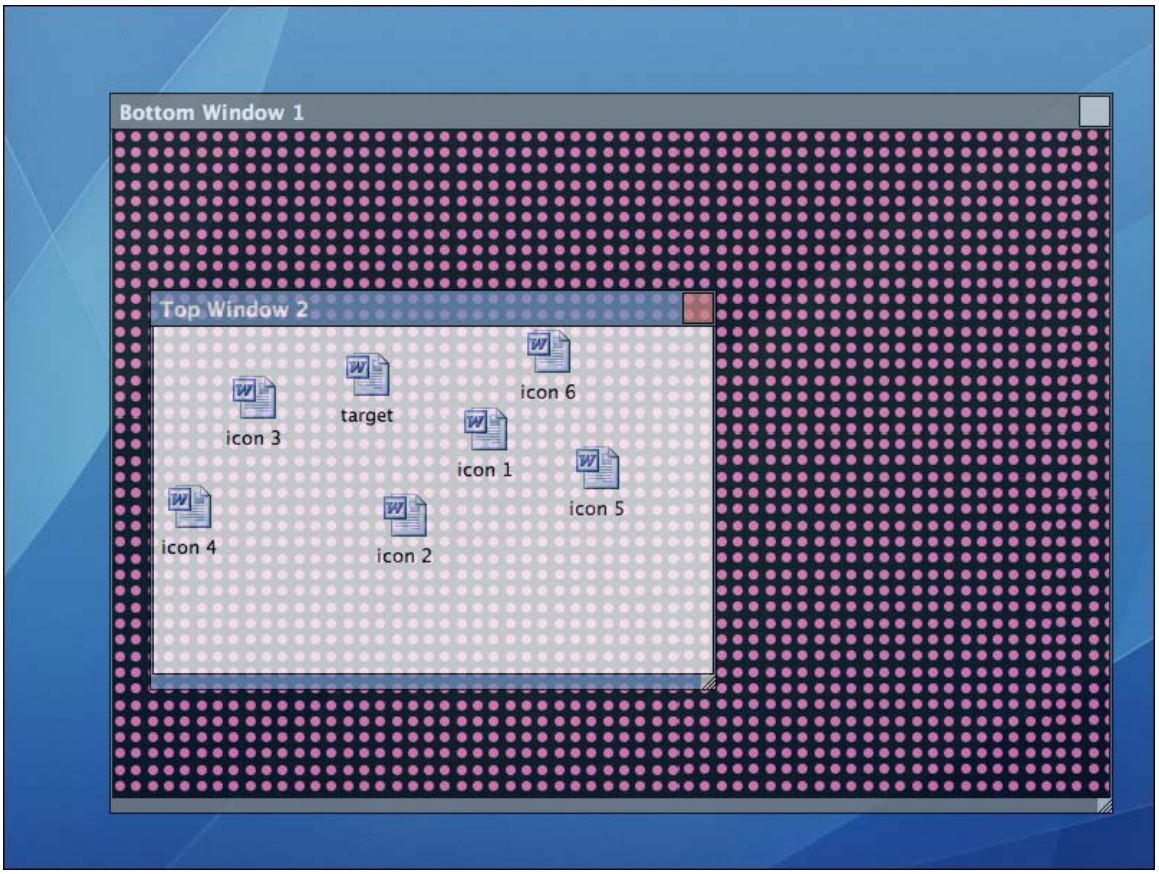

Figure 3.16: A screenshot of a trial used in the search task using the ALPHA-25 technique. Participants had to decide whether an icon named "target" appeared in the top window (which it does in this trial).

The participants were verbally instructed to identify if the top window contained an icon with the label "target" positioned beneath it and to perform this identification as fast as they could, while maintaining correctness. All other icons had distinct names starting with "icon" followed by an integer (e.g., "icon 5"). All names appeared in a black 13pt Lucida Grande sans-serif font. The participant was to indicate if the target appeared in the top window by pressing the up-arrow key on the keyboard to signify "yes", and the down-arrow key to signify "no." After each trial, a confirmation screen appeared, indicating that the participant had given a correct (green check) or incorrect (red "X") response. The participant would then press the spacebar key to proceed to the next trial.

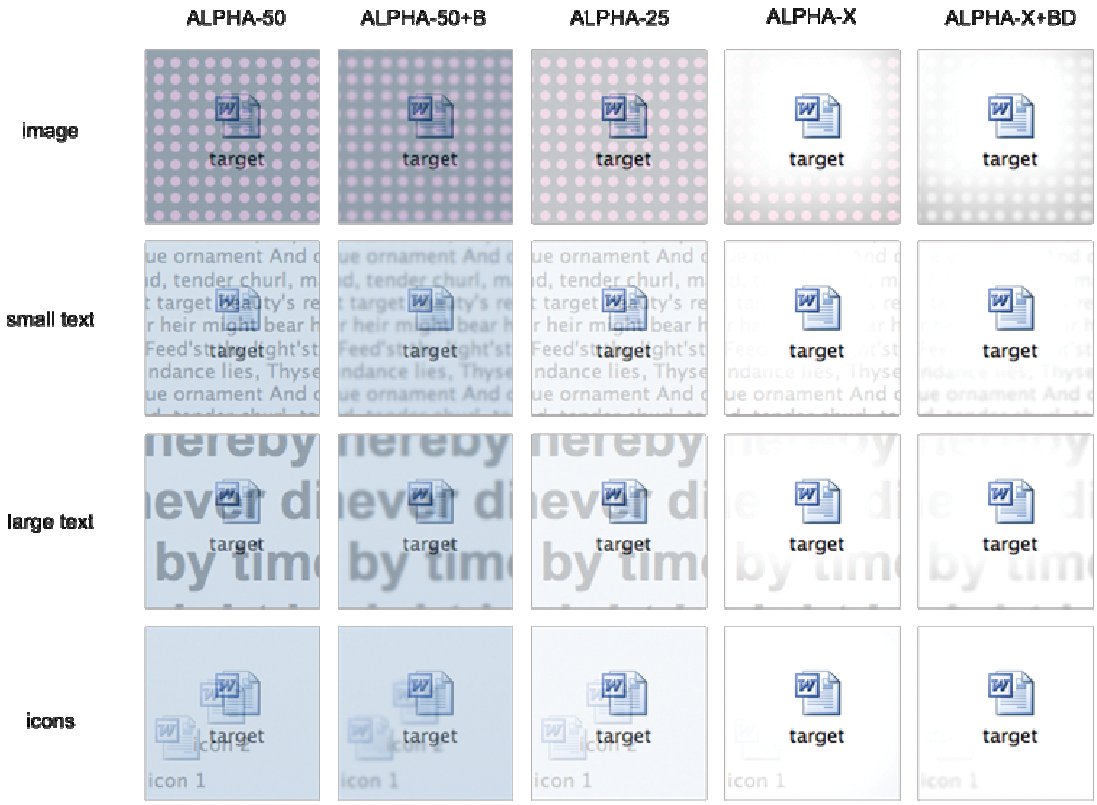

Figure 3.17: Screenshot snippets of the search task, where an overlaid "target" icon is visualized using 20 different combinations of transparency technique and underlying content type. The techniques used were (from left to right column) ALPHA-50, ALPHA-50+B, ALPHA-25, ALPHA-X, and ALPHA-X+BD. Four different underlying content types were used: high spatial frequency imagery, small text, large text, and similar-looking icons.

3.6.8.1. Procedure

A within-subjects, repeated measures design was performed using the five techniques described earlier and the four types of underlying content shown in Figure 3.17: (1) high-spatial frequency images, (2) continuously flowing text written in black 13pt Lucida Grande font, (3) continuously flowing text written in black 26pt Lucida Grande font, and (4) similar-looking icons to those in the top window. Each participant completed 400 trials (5 techniques × 4 content types × 20 repetitions). The order in which the techniques were used was counterbalanced across all participants. The type of underlying content

used in each trial was chosen randomly, but the same order was used across all participants.

3.6.8.2. Hypotheses

We formulated the following hypotheses before running the search experiment:

(H6) Within each technique, due to the dissimilarity in appearance of the target icon and high spatial frequency imagery, searching for the target icon over imagery will be *faster* than over any other type of content.

(H7) When searching for the target icon over any type of content, the ALPHA-50 technique will be *slower* than any other technique, due to the interference of the overlapping contents.

(H8) When searching for the target icon over text (both large and small), the use of opaque foreground and background of the overlaid content will cause the ALPHA-X+BD technique to be *faster* than any other technique.

3.6.9. Results

We analyzed the results, measuring completion-time, success rate, and subjective preference. Only successful trials were included in the completion-time analysis. Again, to compensate for right-skewing frequent to human response times, as well as to reduce the significance of outliers, we used median values for our completion-time analysis. Again, we used $\alpha = 0.05$ to denote statistical significance.

Completion-time analysis: We performed a 4 (Technique) x 4 (Content Type) repeated measures ANOVA on median completion times for successful trials using the 10 participants as the random variable. (Again, since three of the participants did not attempt the ALPHA-X technique, we excluded it from this particular analysis.) All factors had a significant main effect on completion time, including Technique ($F_{(3,27)} = 7.816$, $p < 0.001$), Content Type ($F_{(3,27)} = 3.966$, $p < 0.05$), and the interaction between them ($F_{(9,81)} = 3.930$, $p < 0.001$).

As we see from Figure 3.18, the data fails to fully confirm H6, since there are several techniques that are slower when searching over imagery vs. other types of content (e.g., the use of ALPHA-X+BD over similar icons outperforms ALPHA-X+BD over imagery). However, within the ALPHA-50 technique, we see that the use of ALPHA-50 over imagery significantly outperforms ALPHA-50 over small text ($t_9 = 2.843$, $p < 0.01$), partially confirming H6.

We also see that ALPHA-50 seems to perform worse than any other technique. Within each type of underlying content, we performed t-tests between ALPHA-50 and the next worse performer, and found borderline to no significant differences, thus failing to confirm H7.

In order to confirm H8, we compared ALPHA-X to the next fastest technique of the ALPHA-50, ALPHA-50+B, and ALPHA-25 techniques, within each of small text and large text content types. Although H8 originally hypothesized that ALPHA-X+BD would outperform the rest of the techniques, the original intent for this hypothesis before the

experiment was to see if a technique that rendered both the foreground and background of overlaid content opaque would cause a significant performance improvement over a technique that did not. Since ALPHA-X and ALPHA-X+BD both incorporate this functionality, and since ALPHA-X outperformed ALPHA-X+BD, we chose to compare ALPHA-X to the other three techniques rather than ALPHA-X+BD. In both cases, the next fastest was the ALPHA-25 technique. A sample-paired t-test shows a significant difference between ALPHA-X and ALPHA-25 when searching over either small text ($t_6 = 2.128$, $p < 0.05$) or large text ($t_6 = 3.108$, $p = 0.010$), thus confirming H8.

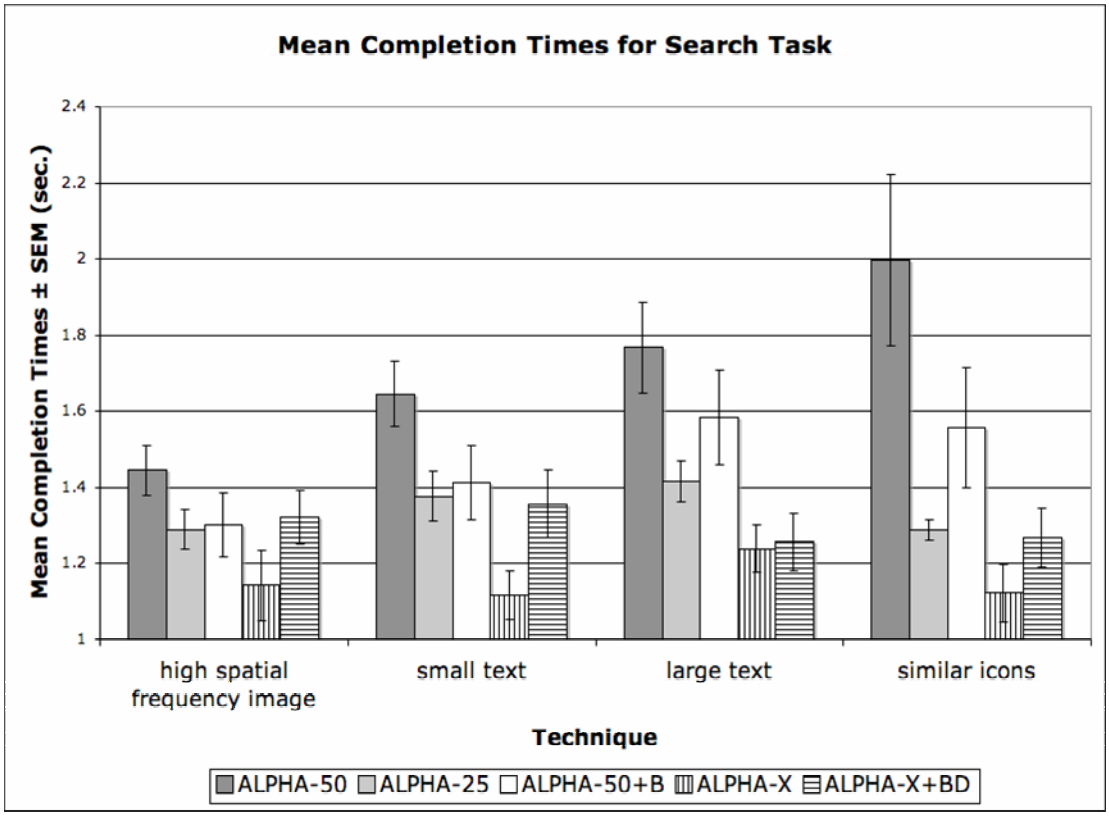

Figure 3.18: Average completion times for the search task using the five transparency techniques and four different types of underlying content.

Error-rate analysis: We performed a 4 (Technique) x 4 (Content Type) repeated measures ANOVA on error rates using the 10 participants as the random variable.

(Again, since three of the participants did not attempt the ALPHA-X technique, we excluded it from this particular analysis.) There were no significant effects on error rate. We suspect that, although completion times significantly differed, error rates were similar across techniques because of the nature of the search task. Searching involves looking for something until one has found it. If the participant has not found it, they usually search again to confirm that the item does not exist. In other words, although participants were told to perform trials as fast as they could, the nature of a search task caused them to value correctness as much as, or more than, speed.

Qualitative Evaluation Analysis: On the same questionnaire, participants rated each technique for the search task. Figure 3.19 shows the average ratings for each technique, evaluating its ease of use, satisfaction, and intuitiveness. All ratings scored relatively, although slightly, lower than the ratings for the container-identification task. We feel this is because the search task was inherently a harder task, and this caused some frustration for participants in general. However, relative to each other, the overall ratings for the techniques were similar in both the search task and the container-identification task. Again, the ALPHA-50 technique seemed to be the hardest, followed by the ALPHA-50+B technique. Several comments about the ALPHA-50 technique included “terrible” and “an extremely frustrating experience.”

Some participants made comments about the use of the same technique in a different task seeming to not have the same facilitative effect. For example, one participant said that the blurring effect in the ALPHA-50+B technique “did not help much in this task” as compared to the ALPHA-50 technique, whereas in the container-identification task, the

same participant rated the ALPHA-50+B technique to be significantly easier, more satisfying and more intuitive than the ALPHA-50 technique. Another participant said that the use of the ALPHA-25 technique in the search task was "unexpectedly more frustrating than the same technique in the container identification task."

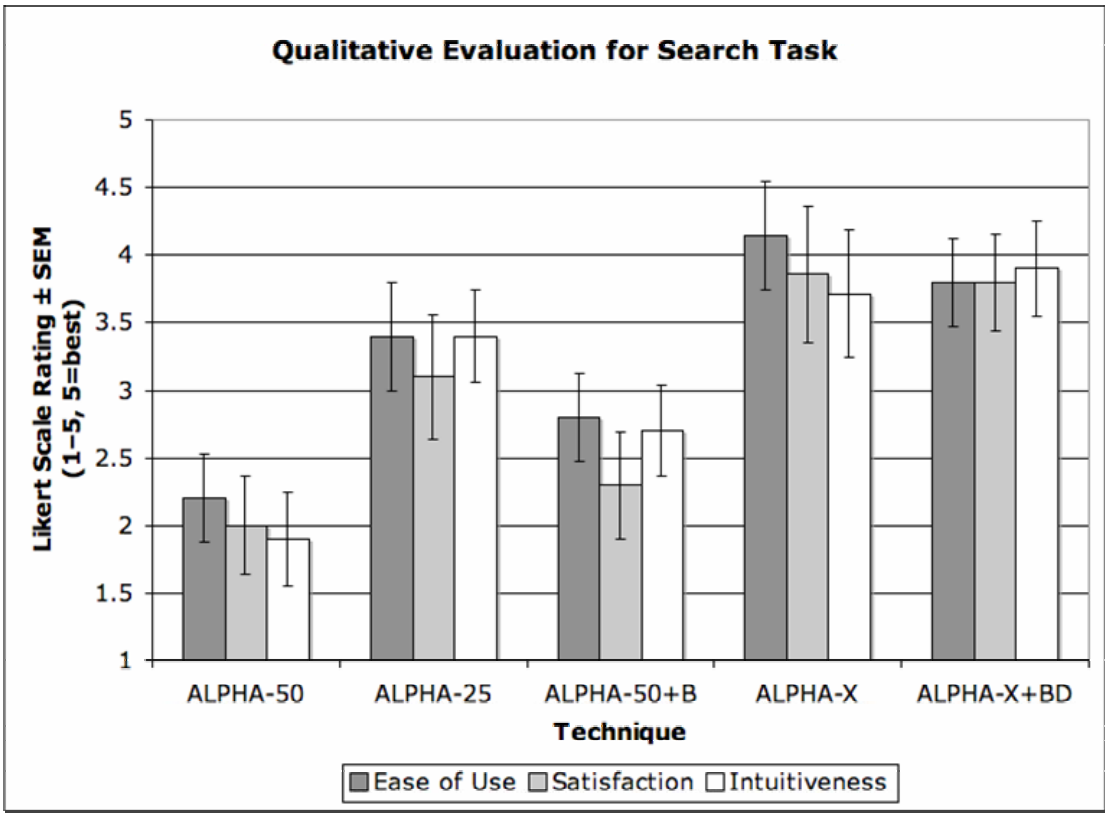

Figure 3.19: Results from the participant-filled questionnaire regarding the search task show that the ALPHA-X and ALPHA-X+BD techniques were rated to be the easiest, most satisfying and most intuitive, while the ALPHA-50 technique were rated to be the most difficult, most frustrating, and most confusing. Overall, each technique received relatively lower scores in the search task than in the container-identification task indicating that it was a relatively more difficult task.

For the most part, the opaque background and foreground in the ALPHA-X and ALPHA-X+BD techniques facilitated an easier and more satisfying experience for the participants. One participant said the following about the ALPHA-X+BD technique: "because it

blocked out the background, it was the easiest and quickest technique for this task." Another said "great technique for finding target."

3.6.10. Discussion

In summary, we conducted a user study comparing the use of various transparency techniques for a search task and a container-identification task. In both tasks, alpha-blending at 50% (ALPHA-50) was both the worst performer and the least preferred by participants. Adding a blur effect to the underlying contents seen through a transparent object (ALPHA-50+B) improved both the performance and preference slightly. However, decreasing the transparency to 25% without blurring (ALPHA-25) seemed to provide a greater performance benefit. This technique was also preferred by participants over the ALPHA-50 and ALPHA-50+B techniques. In the end, apparently due to the higher opacity of important content, our implementation of CAT, both with (ALPHA-X+BD) and without (ALPHA-X) image-process filtering of the background content, proved to be the best performers and the most preferred techniques.

Interestingly, our CAT implementations did not perform significantly better than other techniques when searching for text over high spatial frequency imagery, as opposed to over textual content. Although not confirmed, this leads us to believe that the level of transparency applied to overlaid content can safely vary depending on content type. This can allow certain user interface scenarios to take advantage of higher transparency values in some situations, allowing more content to be seen through overlaid objects without compromising performance. However, other factors to consider are the effect of varying

the transparency technique used from one scenario to another and how that might affect the user experience in the long run.

3.7. Interaction with CAT

With CAT, a user can view more content simultaneously and unambiguously. To further increase the usefulness of our approach, we have developed techniques that allow a user to interact with and manipulate any visible content, using either a touchpad or a standard two-button mouse.

Although many approaches allow visualization of overlaid content, they rarely allow interaction with the obscured content. (The Task Gallery [Robertson2000], Adobe Photoshop, and Autodesk 3ds Max are notable exceptions, as they each allow one to cycle through overlapping windows or layers.) One reason may be that with uniform semi-transparency, disambiguating overlapping contents can be hard, making user interaction with otherwise hidden content difficult. With full opacity, each pixel represents part of at most one window, and therefore, interaction with that pixel is unambiguous as to the selected window. In contrast, with uniform semi-transparency, each pixel is a blended representation of any number of windows and background; therefore, what is being manipulated at a particular pixel can be ambiguous when a user wishes to interact with a window underneath the top-most window. Since CAT does not allow important window regions to be rendered semi-transparently, each pixel on the screen represents an important region from at most one window. In the case that unimportant content is identified as white or empty space, this facilitates unambiguous

interaction with all visible window content, even if it is visible through one or more unimportant window regions.

3.7.1. Pop-Through

We allow a user to manipulate content beneath transparent regions of obscuring windows through the use of the *pop-through* interaction technique, which makes it possible to use pressure to interact with an obscured window [Zeleznik2002, Zeleznik2001]. In our implementation on a MERL DiamondTouch table [Dietz2001], when the user applies more than a threshold amount of pressure (i.e., increases their area of hand contact with the table beyond a threshold) to an obscuring window, the topmost hidden window at the point of contact will "pop through" and become focused and fully unobstructed, as shown in Figure 3.20. Currently, when using a non-pressure-sensitive input device, such as a standard two-button mouse, a user invokes a pop-through with a left button mouse-down and half-second delay.

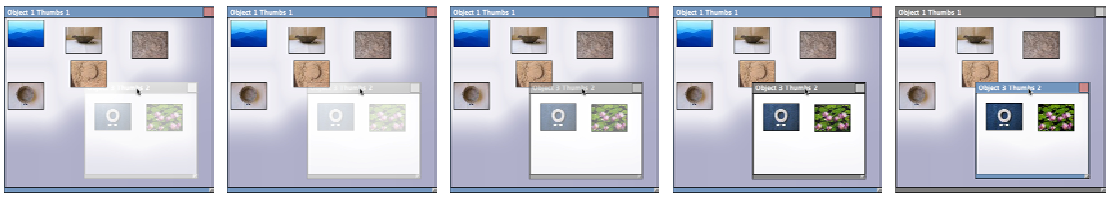

Figure 3.20: A sequence of frames showing the pop-through technique (from left to right): upon a mouse-down and 500ms delay over a pixel within the bounds of the obstructed window in the lower right, the obstructed window pops through the overlaid window and becomes focused for immediate interaction.

3.7.2. Focus Filter

The use of various image-processing filters for content disambiguation may, at times, make overlapped content illegible through the unimportant regions of an overlaid

window. We provide a technique that permits a user to temporarily view filtered content in its unfiltered form. Applying the *focus filter* causes image-processed content underneath the overlaid window to be restored to its original unfiltered form, as shown in Figure 3.21, acting as a "magic lens" [Bier1993]. In our implementation, content within a fixed radius around the point-of-interest is restored; we have found that a 100-pixel radius provides adequate coverage in our informal experience. Currently, when using a standard two-button mouse, holding down the right button, followed by a left button click, invokes the focus filter. At this point, dragging the right button moves the focus filter appropriately.

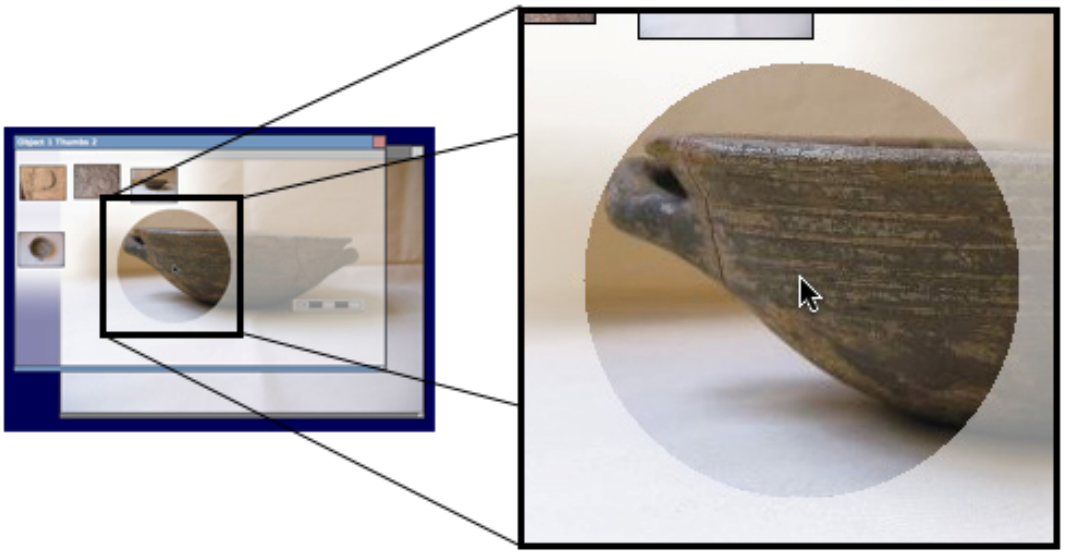

Figure 3.21: The focus filter allows a user to temporarily restore an image-processed portion of obscured content to its original appearance for increased legibility. Here, using the mouse, the user temporarily focuses on part of a blurred, obscured image of a pot.

3.7.3. Mouse-Over Pie Menu

When using many windows, a user may wish to interact with a window at an arbitrary depth. Techniques such as pop-through and focus filter facilitate interaction with a single

layer underneath the top-most window; however, they make it cumbersome to interact with multiple underlying windows when using a standard two-button mouse. We provide a *mouse-over pie menu* [Hopkins1987], shown in Figure 3.21, to allow a user to determine the window with which to interact at any level. Using a two-button mouse, a user can invoke the mouse-over pie menu by holding down the left mouse button, and then clicking the right mouse button on a pixel representing blended (and possibly image-processed) content from more than one window. A pie menu appears with choices containing thumbnail representations of all the windows that contain the selected pixel. As the user hovers over a particular slice of the pie, that window highlights, moves to the front, and becomes focused.

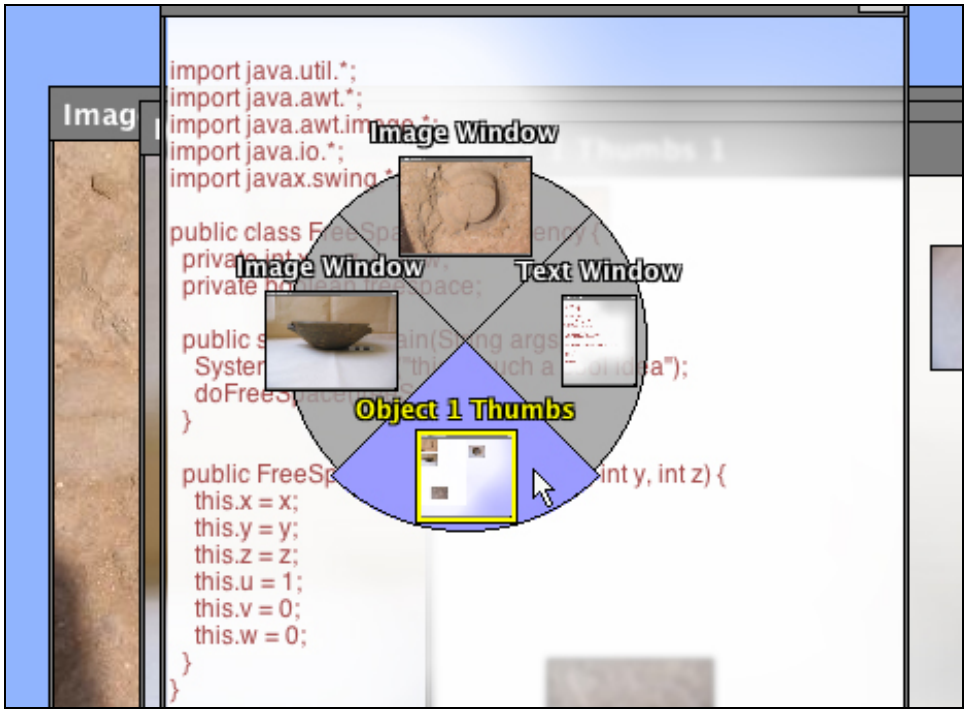

Figure 3.22: The mouse-over pie menu allows a user to determine with which window to interact at an arbitrary depth. The menu contains only those windows beneath the mouse position upon invocation.

The design decision to implement a pie menu over a marking menu [Kurtenbach1991] was based on the fact that the menu choices can differ dramatically from one invocation to the next. Since one of the advanced features of marking menus relies on the user memorizing the menu items and their placements, a pie menu seemed to suffice for this type of interaction.

3.8. CAT Performance

Running on an Apple Power Mac G5 desktop (dual 2GHz, 1GB RAM), basic moving and resizing of windows operate at about 15–20 fps without the use of filters. We foresee these numbers improving with faster hardware and the use of hardware-accelerated image processing. Our implementation of the focus filter operates at about 10 fps, which is adequate for normal interaction.

3.9. Additional Benefits of CAT

CAT provides additional benefits in certain window layout scenarios. The use of a gradient between important and unimportant window regions allows one to infer the approximate distance from almost any pixel within a window to important content in that same window, and possibly even to off-screen content (as shown in Figure 3.23), similar to the Halo technique [Baudisch2003], which surrounds off-screen locations with rings just large enough to reach into the border region of the display window.

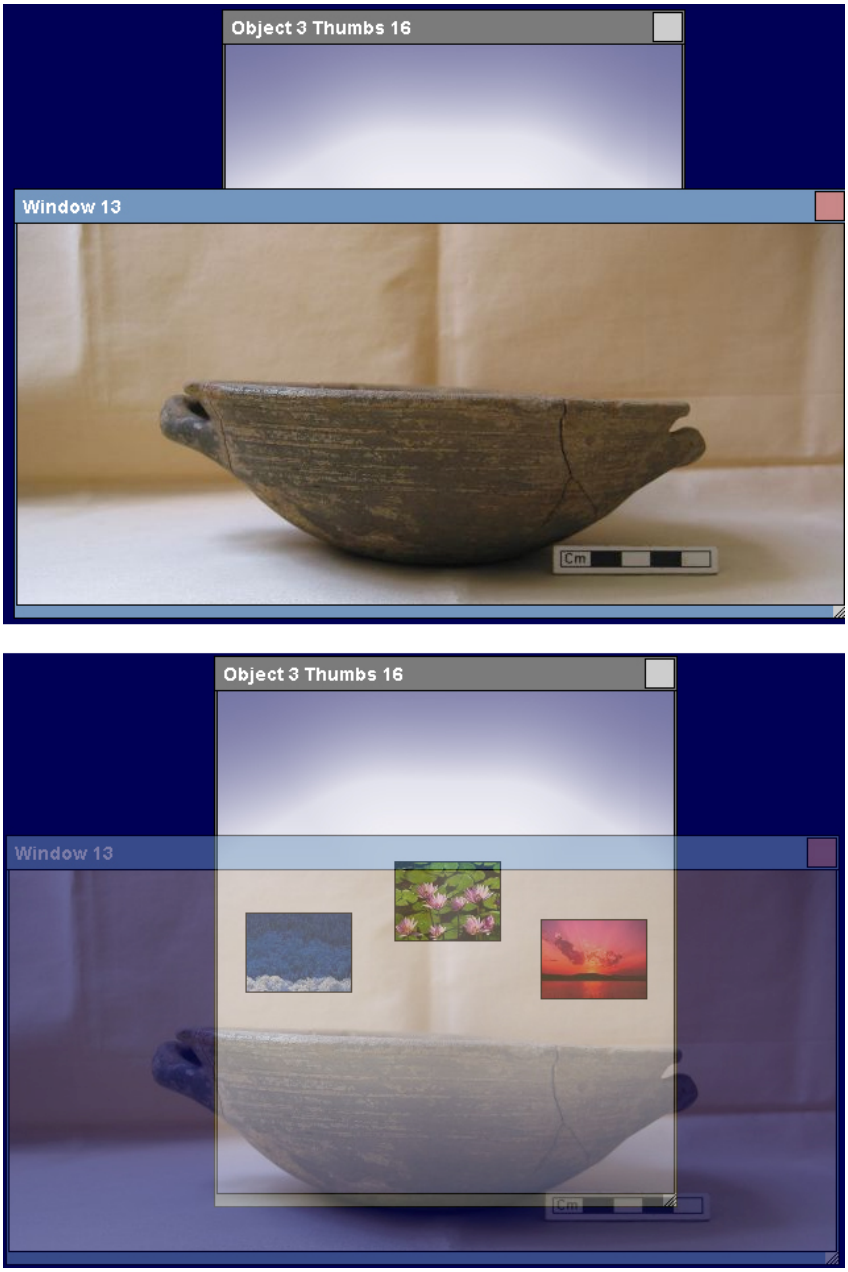

Figure 3.23: The use of a gradient between important and unimportant regions allows one to infer the approximate distance from almost any pixel within a window to important content in that same window, and possibly even to off-screen or obscured content. The obscured window's gradient implies that important content lies beneath the obscuring window (top). For demonstration only, we render the obscuring window semi-transparent to reveal what lies beneath it (bottom).

Additionally, spatial groupings of objects can be visually reinforced with the use of steep gradients (i.e., short gradients containing a large alpha discrepancy) by encapsulating important regions with opaque pixels thus forming an opaque "blob" around those

regions. Finally, by knowing which regions of windows are unimportant, one could use space management [Bell2000] to place information not only in totally free screen space, but also in unimportant window regions, an idea we present in Chapter 5.

Chapter 4

Content-Aware Scrolling (CAS)

Scrollbars are conventionally used to navigate large documents on small screens. When using scrollbars, scrolling along either axis can be done independently, but usually not simultaneously. Panning, vector scrolling, and various hardware devices allow users to scroll in an arbitrary 2D direction, but all require precise steering for complex 2D reading paths (e.g., multi-column, multi-page documents).

In this chapter, we present *content-aware scrolling* (CAS) [Ishak2006], an approach that varies the scroll direction and speed based on the properties of the content within the document. We have developed an implementation of CAS, which automatically extracts the reading path of text PDF documents, and varies the direction, speed, and zoom while

scrolling along this path, based on content properties, as shown in Figure 4.1. We have also designed and implemented a document viewer that uses the *CAS widget*, a user interface widget that allows the user to scroll along any linear or nonlinear path using traditional scrolling gestures, which will be described in Section 4.4. We have evaluated our CAS implementation and have found that participants prefer CAS when reading documents. We have also found that CAS outperforms both traditional and vector scrolling when used to navigate short distances.

In CAS, when the user scrolls, content is not treated as a single undifferentiated layer of information. Instead, properties of the content and task are taken into account to identify *important regions* to vary the scroll direction, speed, and zoom. This is a special case of what Smith and Taivalsaari call “generalized scrolling” [Smith1999], which they define as a mapping between user input and an object’s intrinsic position in an abstract space of display attributes.

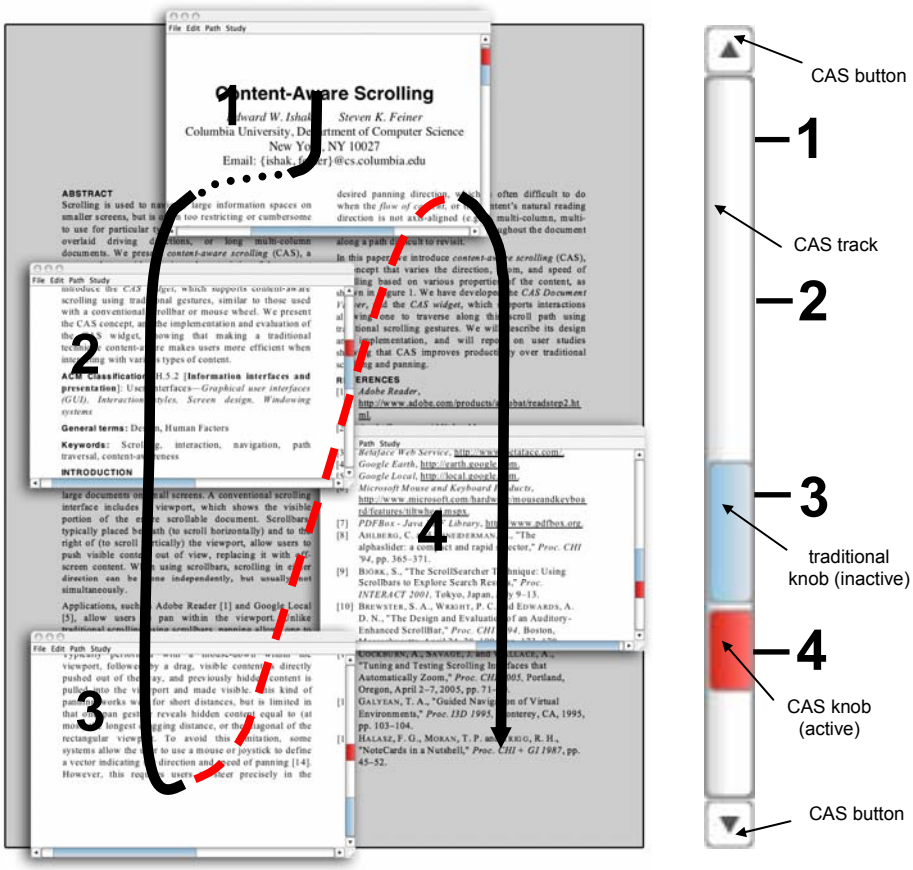

Figure 4.1: Content-aware scrolling (CAS) allows scrolling along the document reading path (the path in which the document is intended to be read) using the CAS widget (enlarged on the right). The reading path is shown *roughly* as the overlaid arrow, where black dots indicate a small unimportant region (traversed with a different scrolling distance mapping) and red dashes indicate a large unimportant region (traversed with an animation). CAS window snapshots 1–4 on the left correspond to track locations on the right, and are traversed in a single scroll. The light blue knob indicates the traditional knob location (inactive during CAS) for the current CAS knob location (currently at position 4).

4.1. Related Work

Almost every application allows one to scroll using scrollbars placed to the right (to scroll vertically) and beneath (to scroll horizontally) the viewport. Some applications, such as Adobe Reader[Adobe] and Google Local [Google], 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 allows one to control distance and direction in two spatial dimensions with a

single gesture, rather than specifying the distance and direction along the *x*- and *y*-axis separately, as would be the case when using conventional scrollbars. Furthermore, precise horizontal or vertical scrolling can be performed using a modifier key to separately control of *x* and *y*. Panning works well for short distances, but is limited in that the longest dragging distance is equal to the diagonal of the rectangular viewport. There are also some systems that allow various versions of *free scrolling*, a first order technique that allows one to scroll continuously by specifying a velocity. A conventional scrollbar affords the use of restricted 1D free scrolling with a mouse down on a scrollbar button, located on each end of the scroll track. However, this restricts the user to a system-defined constant rate control in either the up-and-down (using the vertical scrollbar) or left-and-right direction (using the horizontal scrollbar). Some systems like Google's Picasa allow the use of a joystick knob that, when dragged, defines a variable rate control in only the up-and-down direction. Another type of free scrolling is *vector scrolling*, where a mouse or joystick is used to define a vector, indicating the direction and speed of continuous scrolling [Zhai1997]. This offers rate control without any repeated scroll gestures performed by the user, which is useful when scrolling long distances. However, it requires users to steer precisely along a particular path, which is often difficult to do when the reading path for the content is not axis-aligned, but rather a complex route through the document (e.g., consider a multi-column, multi-page text document). Furthermore, none of these techniques allow one to revisit non-linear paths.

4.1.1. Augmenting and Replacing the Scrollbar

Some systems have augmented the scrollbar to provide additional information about off-screen content. For example, the bookmark scrollbar [Laakso2000] 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 [Björk2001] indicates search results for the document within the scrollbar. The Thumbar [Graham1999] provides a draggable lens over a thumbnail version of the entire document to quickly scan and navigate to any position within the document. The auditory-enhanced scrollbar [Brewster1994] uses non-speech sounds to help identify off-screen locations.

Other researchers have developed alternative techniques to try to outperform scrollbars, such as the Alphaslider [Ahlberg1994], LensBar [Masui1998], and FineSlider [Masui1995], which allow for quick visualization of and traversal through a large list of data items. Some researchers have developed scrolling gestures to allow users to scroll in only one dimension [Moscovich2004, Smith2004], as well as in two dimensions simultaneously [Igarashi2000]. There are also those who have developed [Igarashi2000] and analyzed [Cockburn2005] systems that automatically zoom while scrolling in two dimensions. However, most of these techniques are typically applied towards one-dimensional data, such as lists, and long linear documents (e.g., a single-column document), and none allow one to revisit a previously traversed path without requiring the user to emulate the same (and often complex) set of scrolling gestures.

4.1.2. Hardware Interaction Devices

There have also been hardware devices created to scroll across documents in one (e.g., the WheelMouse) and two dimensions (e.g., the JoyStickMouse, the MightyMouse, and TrackPoint) [Zhai1997]. The Logitech MX Revolution contains the MicroGear™ Precision Scroll Wheel, which allows users to spin the scroll wheel with minimal friction

for continuous scrolling through long documents. Although direction control is limited to one dimension as with a standard WheelMouse, scrolling long distances requires only an initial effort, since the near frictionless weighted wheel continues to scroll for several seconds after the initial spin. Unlike CAS, however, none of these devices allows one to revisit a complex 2D path. However these input devices could be used in conjunction with CAS. For example, using the Precision Scroll Wheel, one can continuously scroll through long paths, flicking the wheel to indicate "forwards" or "backwards" along the path, rather than "up" or "down" along the *y*-direction.

4.1.3. Navigating Hypertext and 3D Environments

Some hypertext systems, such as Scripted Documents [Zellweger1989], allow authors to create conditional and programmable paths within and between online documents. Similarly, CAS allows manual path creation, but also automatically creates paths based on attributes of the content. LineDrive [Agrawala2001], a real-time system for automatically generating and rendering route maps, automatically creates driving direction paths that accentuate the important features, such as turning points, while diminishing unimportant ones, such as exact lengths and angles of each road along the route, and may use distortion to render such regions. Similarly, CAS automatically creates paths and identifies important features within them, but rather than use distortion when rendering, it varies scrolling speed to allow users to traverse through unimportant regions faster than important ones. This is an example of how a content-aware technique varies the display of content across the *temporal domain*, such that, given a constant scroll gesture, the user is spending more *time* visiting important regions than unimportant ones.

There have also been many systems developed to traverse a 3D path [Tan2001, Zeleznik1999]. For example, Khan et al.'s HoverCam [Khan2005] automatically determines a camera's motion when closely inspecting a 3D object. CAS also automatically constructs the viewport's motion, but uses animations and varying scrolling speed to reduce the amount of time spent in unimportant regions of the path, which depends on the user task. Galyean's "river analogy" [Galyean1995] 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 contrast, CAS supports strict following of a predetermined path, but at a user-controlled speed. 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.

4.2. The Approach: Maintain Visual Momentum

Hochberg and Brooks [Hochberg1978] explain that a successful model of scene transitions within cinematography should employ what they call "visual momentum." *Visual momentum*, within this context, refers to how a switch from one scene viewpoint to another affects an observer. In other words, high visual momentum allows the viewer to increase their understanding of a scene upon each successive viewpoint switch, whereas low visual momentum may disorient or confuse the viewer.

Woods [Woods1984] extended this model and applied it to user interfaces, stating that, by taking into account cognitive psychological models about perception, information extraction across multiple displays (or multiple glances at the same display) can be improved. He stated that high visual momentum results from continuity across successive

views, which aids in information comprehension following each view transition. Our CAS implementation uses animations through unimportant regions to maintain continuity between important regions when scrolling through the reading path of a document, for example. This allows CAS to maintain high visual momentum while a reader scrolls through this path, rather than break the continuity, which traditional scrolling often requires when transitioning from the end of one column to the beginning of the next, for example. Woods also discusses several *spatial representation techniques*, or *spatial access techniques*, that provide information about the location of one view with respect to the successive one. Some of these techniques focus on *route knowledge*, or inter-display movements perceived as paths through a virtual space. In short, route knowledge allows a user to correctly anticipate where she is going and from where she came when switching views. This model helped inspire our implementation of CAS.

Most traditional scrolling interfaces provide what Woods calls "display overlap," or overlapping successive views, where the next view intersects a part of the previous view to maintain continuity. Display overlap increases viewer comprehension between views, a property that our CAS implementation maintains, as described later in Section 4.3. CAS further incorporates route knowledge, as described above, but also combines this with creating an effective route that satisfies the needs of the task. For example, in a reading task within a text document, the user may want to travel along the reading path, whereas in a search task in that same document, the user may want to travel through each search result in the order in which it appears in the document. More simply, the combination of route knowledge and route preference makes it possible for the user to know where she is going, know from where she came, and to find the next view beneficial in completing the

task. In Section 4.3, we discuss how our implementation of CAS within a text document employs route preference when it scrolls along a path in which the document was meant to be read when performing a reading task, but scrolls through search results when performing a search task.

4.3. CAS Implementation

We present our implementation of CAS and how system-defined important regions help vary the direction, speed and zoom of scrolling.

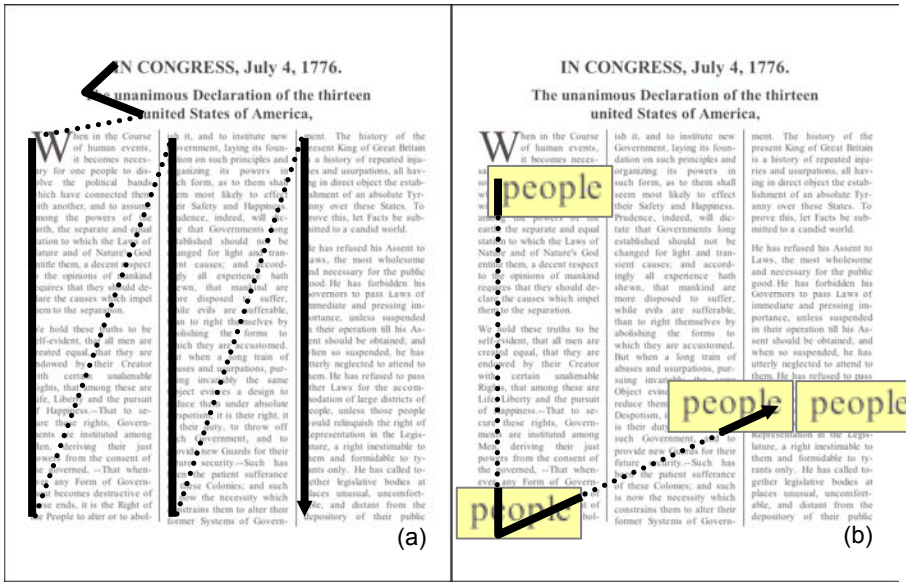

Figure 4.2: The path, defined by the flow of content, can vary depending on the task, as shown in (a) a reading task, and (b) a text search task for "people" in same page. Solid and dotted black parts of path indicate important and unimportant regions, respectively. (Arrows and popouts are included for purposes of illustration only.)

4.3.1. Varying Direction

Our CAS implementation is useful for 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 path of a text document, or a path through the results of a text search, as shown in Figure 4.2. Adobe Reader automatically infers the reading path of documents for both its "Read Out Loud" feature [Adobe] and third-party screen readers to speak the text, as an accessibility feature. We use this same reading path.

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 our CAS implementation an advantage over reflow in spatial memory tasks (e.g., remembering where a passage occurred in the fully laid out document).

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 [Burtnyk2002] supports continuous 3D interaction, allowing a user to control the speed and direction with which they traverse a manually authored 3D camera path. Likewise, our CAS implementation allows manual control of direction and position along a 2D path within a document. However, in contrast, our CAS implementation

affords manual control of *automatically-generated* task-specific 2D document paths. Furthermore, unlike article threads, our CAS implementation automatically extracts the reading path, making it effective on documents that do not contain author-constructed hyperlinks.

4.3.1.1. Identifying Important Regions

Our CAS implementation identifies important regions of a document to create a scrolling distance mapping between the scrollbar and document pixels. For a reading task (e.g., as shown in Figure 4.1), 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. Our paths are continuous; therefore, the unimportant regions along a 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, as shown in Figure 4.2, the distance along the path that lies within unimportant regions can be as large as thousands of pixels. Paths are traversed using a piecewise linear interpolation of the lower left points of the important regions.

4.3.2. Varying Speed

We vary the scrolling speed based on the locations of important regions, in the spirit of the adaptive control/display ratio of Semantic Pointing [Blanch2004], which improves target acquisition by changing the control-to-display ratio according to cursor distance to nearby targets. Control/display ratio is the ratio of movement of the input device, such as

a mouse cursor, to the movement of the object that it is controlling, such as a document within a viewport [MacKenzie1994]. We define the *scrolling distance mapping, m,* as $\Delta d_{knob}/\Delta d_{doc}$, or the ratio of the pixel distance that the CAS widget's knob is dragged (Δd_{knob}) to the pixel distance the document is scrolled (Δd_{doc}). This ratio varies as a function of the distance *D* to the next important region relative to the size of the viewport, as shown in Figure 4.3. For example, consider traversing a search results path within a textual document, as shown in Figure 4.2. 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 scrolling distance mapping through these regions (shown as dotted parts of the path), despite a constant physical scrolling gesture, decreasing the control/display ratio. We have found that a 3× speedup works well. For *very* large distances (those larger than the viewport's diagonal, *d*), the viewport automatically flies to the next important region using a "slow-in, slow-out" animated change in speed. Consider the scrollbar length *s*, scrollable path length *p* (parts of the path not containing large unimportant regions through which animations are used), the sum of the parts of the path containing important regions, p_i, and the speedup factor through unimportant regions not traversed by an animation, *x* (e.g., to triple the scrolling speed, $x = 3$). We define the scrolling distance mapping, *m*, as:

$$m = \begin{cases} \dfrac{s_i}{p_i} & \text{if } D < \dfrac{d}{2} \\ \dfrac{s_i}{p_i x} & \text{if } \dfrac{d}{2} \le D < d \\ 0 & \text{if } D \ge d \end{cases},$$

$$\text{where } s_i = s\left(1 - \frac{\beta}{x}\right), \text{ and } \beta = 1 - \frac{p_i}{p}.$$

The variable s_i is defined as the sum of the parts of the scrollbar through which important regions are scrolled. The variable β is the fraction of the path that contains unimportant regions.

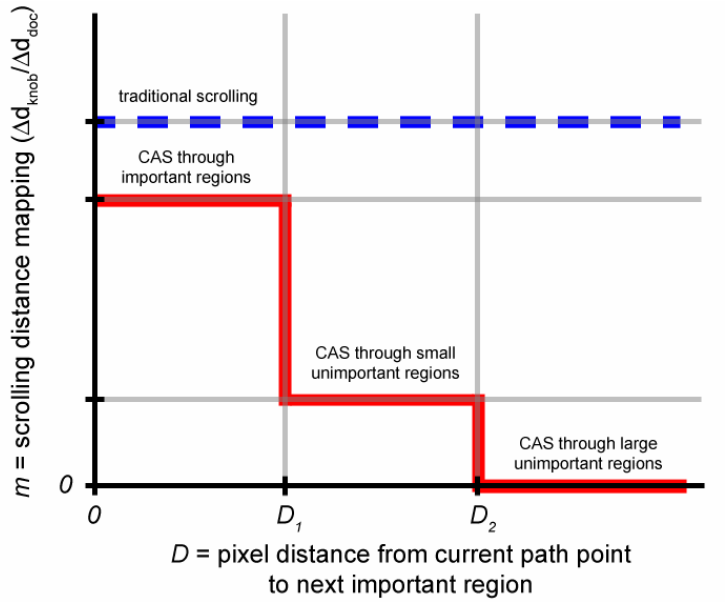

Figure 4.3: While scrolling along a particular path using CAS, the scrolling distance mapping ($\Delta d_{knob}/\Delta d_{doc}$) depends on the pixel distance between the viewer's current location and the next important region along the path, *D*. The value *m* is the ratio of scrollable pixels in the scrollbar to those along the entire path, which is typically constant in conventional scrolling (indicated by horizontal dotted blue line). In our CAS implementation (indicated by solid red lines), if *D* is somewhat far (i.e., greater than 0.5 × diagonal of the viewport; call this D_1), the mapping is reduced by 1/*x* (e.g., to triple the scrolling speed *x* = 3). If *D* is substantially large (i.e., greater than the diagonal of the viewport; call this D_2) the viewer performs a "slow-in, slow-out" animation to the next region of interest and no scrolling is required.

4.3.3. 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" Sony Vaio XU 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++ [Bederson1996]. Igarashi and Hinckley [Igarashi2000] 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.

4.4. CAS Widget

The CAS widget looks and feels like a traditional scrollbar in many ways. Consider, for example, how the CAS widget behaves when its scroll path is mapped to the reading path of a text document, as shown in Figure 4.1. 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 the "display overlap" discussed earlier in Section 4.2. Mouse wheel scrolling is also

supported, which helps users maintain precise control when traversing long CAS paths, especially on small displays.

Although the CAS widget looks and feels like a traditional scrollbar, its resulting actions may significantly differ. As mentioned earlier, various reading and search paths may contain a non-uniform scrolling distance mapping. When the knob of a traditional scrollbar is dragged halfway down the track, it navigates halfway through the document. In contrast, when the CAS widget's knob is dragged halfway down its track, this only ensures navigation halfway down its associated 2D path, which may or may not be through half of the document, all depending on the content properties. For example, navigating halfway through a search path of a 100-page document may travel to page 5 if there are only three result hits, where the second one is on page 5.

4.5. Application

To explore how CAS can be implemented and to determine its effectiveness, we have built an application that uses the CAS widget to support content-aware scrolling for different types of content. Our *CAS Document Viewer* is written using the Java 5.0 SDK [SunMicrosystems2004] for cross-platform operation, and supports both text-based PDF documents and JPG-encoded EXIF 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.

4.5.1. PDF Viewer

4.5.1.1. Reading

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

To scroll along the reading path, which is stored in memory, the user must first change the *scrolling mode* from "normal" to "CAS," through a menu option or temporary depression of the "alt" modifier key. Upon enabling CAS mode, if the current viewport position does not contain a part of the document along the reading path, the viewer automatically pans the document to the nearest path point. The vertical CAS widget in this mode now traverses that 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 [Laakso2000], this creates an implicit bookmark, allowing the user to pan and scroll freely until shift is released. The viewer then automatically springs back directly (i.e., linearly) to the anchored location using a 500 ms slow-out animation

When designing the scrolling interaction along a reading path, we adhered to the principle that the viewer should always vertically center the user's current path point within the viewport's bounds. However, during initial pilot studies, we found that certain boundary conditions violated this principle, causing the current path point not to always be vertically centered. For example, we found that users tend to vertically center content that is surrounded by other content (e.g., text not near the beginning or end of a column) within the viewport. However, users interact with content near column and page boundaries a bit differently. Strong visual cues, such as page edges, cause a user to not center content within the viewport. To account for this, we modified the scrolling interaction to anticipate column and screen edges, causing an animation between the bottom of a column to the top of an adjacent column to occur as soon as the page edge is visible within the viewport. Likewise, animations that lead to the beginning of a new column (or page) position the document within the viewer to show the leading page break, as shown in Figure 4.4.

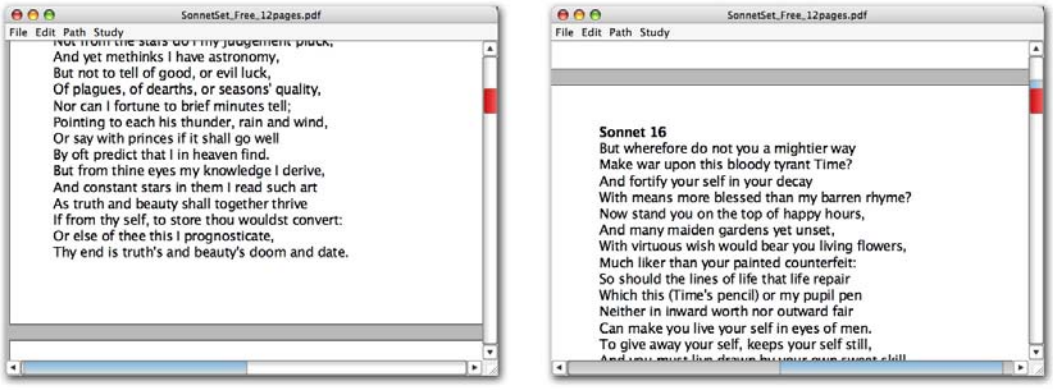

Figure 4.4: Our CAS implementation detects visible page breaks (left) before animating from one important region to another. Likewise, after the animation, the document is positioned such that a preceding page break is visible. If no page break is detected either before or after the animation, the current path point (i.e., line of text) is centered vertically within the viewport.

4.5.1.2. Searching

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 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.

Figure 4.5: "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).

4.5.2. 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 [Betaface]. 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, as shown in Figure 4.5.

4.5.3. 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 a path is recreated every time the document is opened and used as the default reading path. However, we also allow paths to be created manually by the user using a simple path-creator program (similar to CyberCoaster [Satou1999]) invoked from a menu. For user-created paths to be persistent, we require the newly created path to be explicitly saved using a menu option and the document to be in JPEG EXIF-compliant format (if not, we convert it), to store the path as metadata using Adobe XMP [Adobe]. If a file's content has changed since the last path was created, the user must explicitly update the path or create a new one using our path editor and again explicitly save it. If a document contains multiple created and saved paths, they can be stored within the file's metadata and made available the next time the document is opened.

4.6. CAS User Studies

To measure the effectiveness of our CAS implementation, we performed three experiments to see if users were more effective when scrolling using our CAS implementation versus more traditional scrolling methods, all using a standard three-button mouse. Using each of traditional, vector, and content-aware scrolling, each participant completed a reading task, a long-column navigation task, and a short-column

navigation task, all using multi-column, multi-page text documents. Each participant completed the reading task first, followed by the navigation tasks in a counterbalanced order, using each technique within a particular task in a counterbalanced order, all within a single 45-minute session. The purpose of the reading task, as described in Section 4.6.3, was to allow each participant to become familiar and comfortable with each of the three scrolling techniques while perusing a document at a self-controlled pace. The two navigation tasks, as described in Section 4.6.4, required participants to quickly navigate from specified start points to specified end points as fast as possible. The purpose of the two navigation tasks was to analyze the performance and user experience using each of the three techniques, as the height of columns varied within a document. Our goal was to identify the scenarios where CAS outperformed or was preferred over other methods. Completion times, occurrence of mouse clicks, and other quantitative data were gathered using our custom timer software embedded into the scrolling interfaces. Qualitative results were gathered from hand-written questionnaires completed by the participants after performing each of the three tasks.

Ten paid participants (6 male, 4 female), ages 21–31, were recruited by email to undergraduate and graduate student mailing lists at Columbia University. Each participant was a frequent computer user and was familiar with traditional scrolling methods using a standard three-button mouse, such as mouse-down-and-drag panning, use of the scrollbars, as well as vertical scrolling using the mouse wheel. All used the mouse with their right hand and those with corrected vision wore their glasses or contacts during the experiment as they would when normally using a computer.

4.6.1. Scrolling Techniques

We asked the participants to perform the three tasks using each of three scrolling techniques: traditional scrolling, vector scrolling, and content-aware scrolling.

4.6.1.1. Traditional (T) Scrolling

Traditional scrolling (referred to as "T-scrolling" in the experiment) refers to the use of a combination of several standard scrolling and panning methods, including dragging either the vertical or horizontal scrollbar's knob to scroll up and down or left and right (respectively), mouse-down-and-drag panning, and the use of the scroll wheel for strict up-and-down scrolling. Almost every commercial application employs the use of vertical or horizontal scrollbars, as well as the mouse scroll wheel. Many commercial applications, such as Google Local and Adobe Reader, also support mouse-down-and-drag panning. Because of the popularity of these approaches, we felt the T-scrolling was a good baseline method.

4.6.1.2. Vector (V) Scrolling

Vector scrolling (referred to as "V-scrolling" in the experiment) refers to the use of the mouse cursor location to define a vector indicating the direction and speed of scrolling, where scrolling occurs continuously until explicitly stopped by the participant. A middle–mouse-button click is used to define the anchor location, and the direction of scrolling is determined by the vector from the anchor to the current mouse location, and the speed is determined by the length of the vector. Once V-scrolling is invoked, an icon appears at the anchor denoting its position, as shown in Figure 4.6. Another middle–mouse-button click stops scrolling and repositions the anchor, while a left–mouse-button click simply

stops scrolling. Some commercial systems, such as Microsoft Word and the Mozilla Firefox web browser, employ this interaction technique to allow users to quickly scroll through long documents, which is why we chose this technique as another baseline. All T-scrolling features, including mouse-down-and-drag panning, mouse-wheel scrolling, and scrollbar knob-dragging, were disabled for this technique.

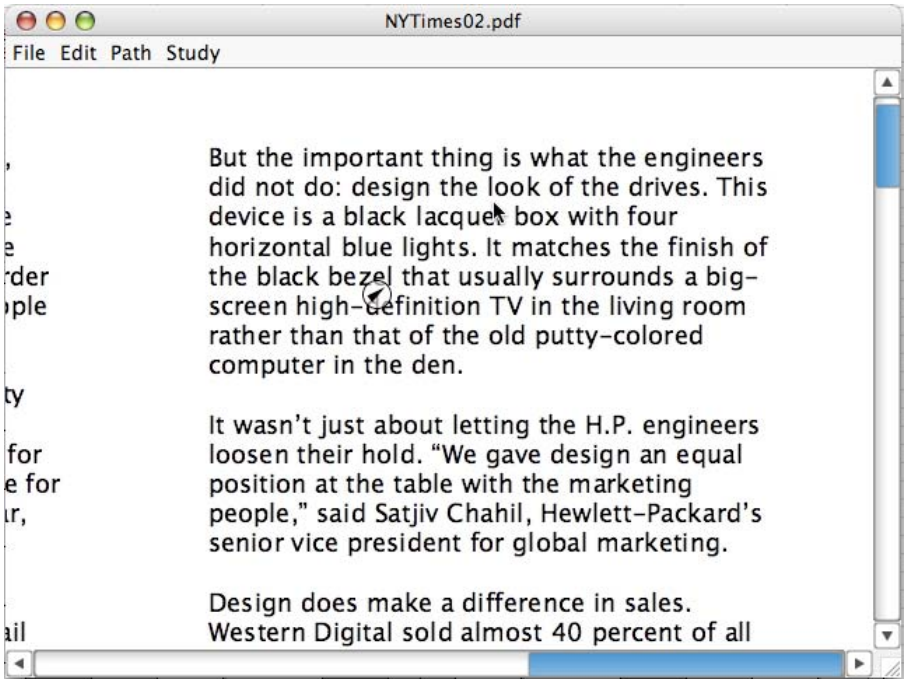

Figure 4.6: A screenshot of the reading task using V-scrolling. The anchor position is denoted by an icon (appearing over the word "high-definition" above). The angle formed and distance between the mouse cursor and the anchor define the direction and speed of continuous scrolling.

4.6.1.3. Content-Aware (C) Scrolling

Content-aware scrolling (referred to as "C-scrolling" in the experiment) refers to our CAS implementation, where the participant is required to only use the mouse scroll wheel to scroll *forwards* and *backwards* along the reading path of the document. A rotation of one scroll wheel notch advances one line of text in a text document. The participant was verbally informed to use only the scroll wheel when performing C-scrolling. This was

because we felt that dragging the scrollbar knob is usually reserved for scrolling long distances to coarsely navigate to a known part of the document. In this case, participants were not familiar with the document, and therefore dragging the knob was not the best way to navigate to a relatively unknown part of the path. Since we felt that the mouse wheel had the best potential for user performance, we restricted content-aware interaction to use of the mouse wheel alone.

4.6.2. Experimental Setup

The experiments were performed on an Apple MacBook Pro running Mac OS X using a single Dell UltraSharp 2407WFP 24” external LCD display running at 1920×1200 pixel resolution at 60Hz. A standard three-button Microsoft optical mouse was used for input.

4.6.3. Reading Task

We initially asked each participant to peruse current New York Times articles, one article for each of the three scrolling techniques. Each article contained multiple 11"×8.5" pages, with each page containing multiple columns of text, as shown in Figure 4.8. The viewport’s dimensions were fixed at 640×480 pixels, which was wide enough to fit at most one column of ten-point Lucida Grande text, rendered at a legible 200% zoom value. Our goal was to understand the user experience while scrolling within smaller viewports when *reading*, which often occurs in full-screen mode on smaller displays, as well as in smaller windows on larger displays when the user is viewing several documents simultaneously. To read the article successfully at the specified resolution, up-and-down scrolling was required to read a column of text, and scrolling in an off-axis direction was required to advance from the end of one column to the beginning of the

next one, as well as from the end of one page to the beginning of the next one. Participants were not allowed to change the size of the viewport while reading and were asked to peruse the document at a pace that was comfortable to them. We gave a demonstration to each participant showing how to use a particular technique immediately before they had to use it, but they were not given any practice time. Participants were not timed during this task. Each participant spent between 3–7 minutes reading each article, which served as valuable practice time for the subsequent tasks, described in Section 4.6.4. The order in which the techniques were used was counterbalanced across participants; however, each participant completed the reading task before any other task.

4.6.3.1. Hypothesis

Our initial motivation for developing CAS stemmed from the interruptions and discontinuities in navigation that occurred when scrolling between columns and between pages while reading text documents within smaller viewports. For this reason, we hypothesized that participants would prefer using C-scrolling when reading documents over T- and V-scrolling, even though they may be much more familiar with T- or V-scrolling.

4.6.3.2. Usability Feedback

Immediately following the reading task, each participant filled out a hand-written questionnaire rating each technique, evaluating its ease of use, satisfaction, and intuitiveness on a Likert scale of 1–5 (5, being the best). Figure 4.7 shows the average ratings. They were also asked to rank the techniques in order of preference, as well as comment on their scrolling experience while reading the articles.

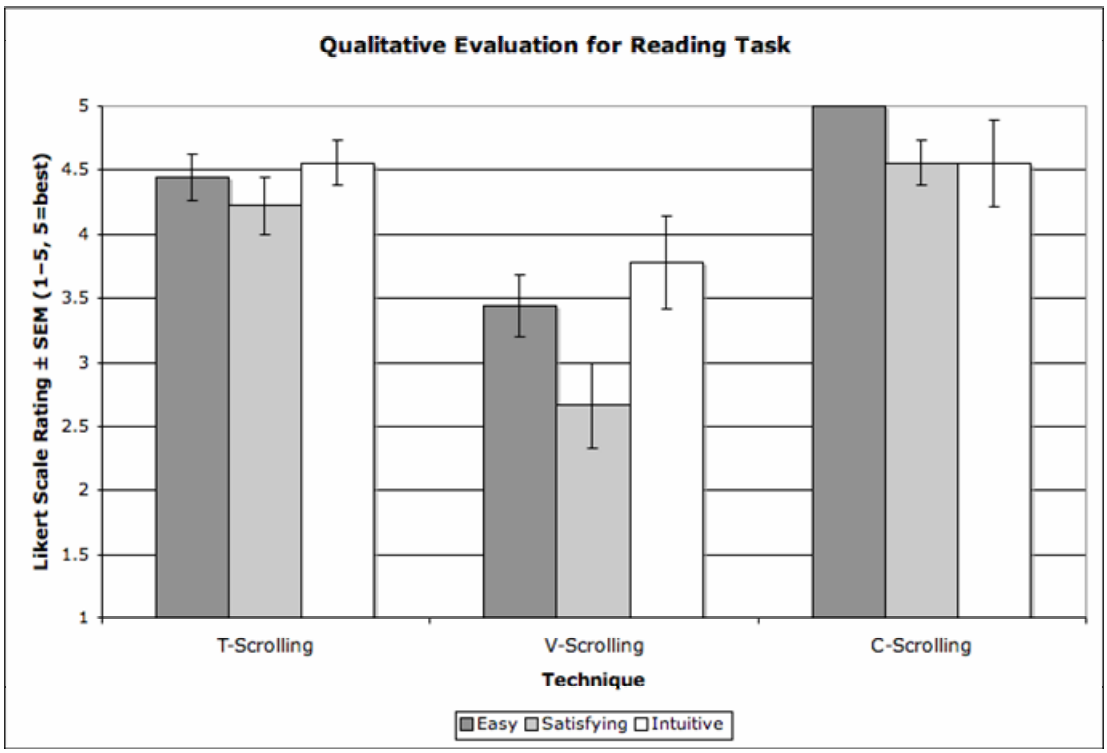

Figure 4.7: Results from the participant-filled questionnaire for the reading task show that the V-scrolling technique was not rated as easy to use, satisfying, or intuitive as the T- or C-scrolling techniques.

When asked to rank the techniques by preference during the reading task, seven out of the ten participants preferred C-scrolling, while the remaining three chose T-scrolling. This was a significant result ($\chi^2_{(2,\ N=10)} = 7.400$, $p < 0.05$), confirming our hypothesis (our null hypothesis expected each technique to be equally preferred). Comments about C-scrolling included "this method made it easier to view a document" and "for non-random-access, this would be my preferred scrolling method." Participants who ranked T-scrolling the highest made comments about their familiarity with the technique, such as "I am used to this technique," and "easiest, because it's what I'm used to."

Nearly all of the comments indicating frustration were directed at V-scrolling, including "little pain in the eye," and "the center icon is distracting while reading," while one

participant stated that “I’d like V-scrolling if it were easier to stop.” This most likely resulted from the use of the middle mouse button, to which most users are not accustomed. V-scrolling didn’t receive all the criticism, however. One participant did mention that, with T-scrolling, “changing columns is frustrating,” while another mentioned that, while using C-scrolling, she was confused because she sometimes did not know spatially where she was located within the document. This may have been due to the brief animations that occurred automatically, as opposed to the manual repositioning of the document within the viewport when using T- or V-scrolling.

Overall, participants preferred C-scrolling for reading a document, even after using it for only a short time (approximately five minutes), and they were more familiar with traditional methods, such as T- and V-scrolling.

4.6.4. Long-Column and Short-Column Navigation Tasks

Immediately following completion of the reading task questionnaire, each participant was asked to perform two navigation tasks using the same experimental setup. At this point, each participant had been introduced to and was familiar with each technique. The 640×480 pixel-sized viewport contained a two-column, multiple-page PDF document. For one of the tasks, each page was 8.5"×11", with two long columns per page (we call this the long-column navigation task). For the other task, each page was 8.5" ×5.5", with two short columns per page (we call this the short-column navigation task). The order in which the two tasks were performed was counterbalanced across participants (i.e., participants #1, 3, 5, 7, and 9 completed the long-column navigation task using all three techniques before starting the short-column task, while participants #2, 4, 6, 8, and 10

completed the short-column navigation task using all three techniques before starting the long-column task). Each document contained numbered Shakespearean sonnets, sorted in order of increasing number, where each sonnet was preceded by a bold heading indicating its number (e.g., "**Sonnet 2**"). Pages were arranged from top-to-bottom, similar to the layout by most commercial PDF viewers, as shown in Figure 4.8. Both the sonnets and their headings were rendered in a ten-point Lucida Grande at a 200% zoom level. For each trial, the participant was presented with a popup dialog box indicating the sonnet to which they would need to navigate and click on its heading. The trial did not finish until the correct sonnet heading was clicked. Thus, all trials were considered successful. Before each technique was used, participants were verbally told that they were being timed and to complete each trial as fast as possible using the specified technique.

4.6.4.1. Procedure

For each task (long-column and short-column navigation), we used a within-subjects, repeated measures design with the three scrolling techniques described earlier and two measures of distance (one-column and two-column transitions). A *one-column transition* that started at a particular *y*-location (relative to the top of the column) ended at the same *y*-location (relative to the top of that column) in an adjacent column (which could possibly appear on the next page). Similarly, a *two-column transition* ended at the same *y*-location two columns away, as shown in Figure 4.8.

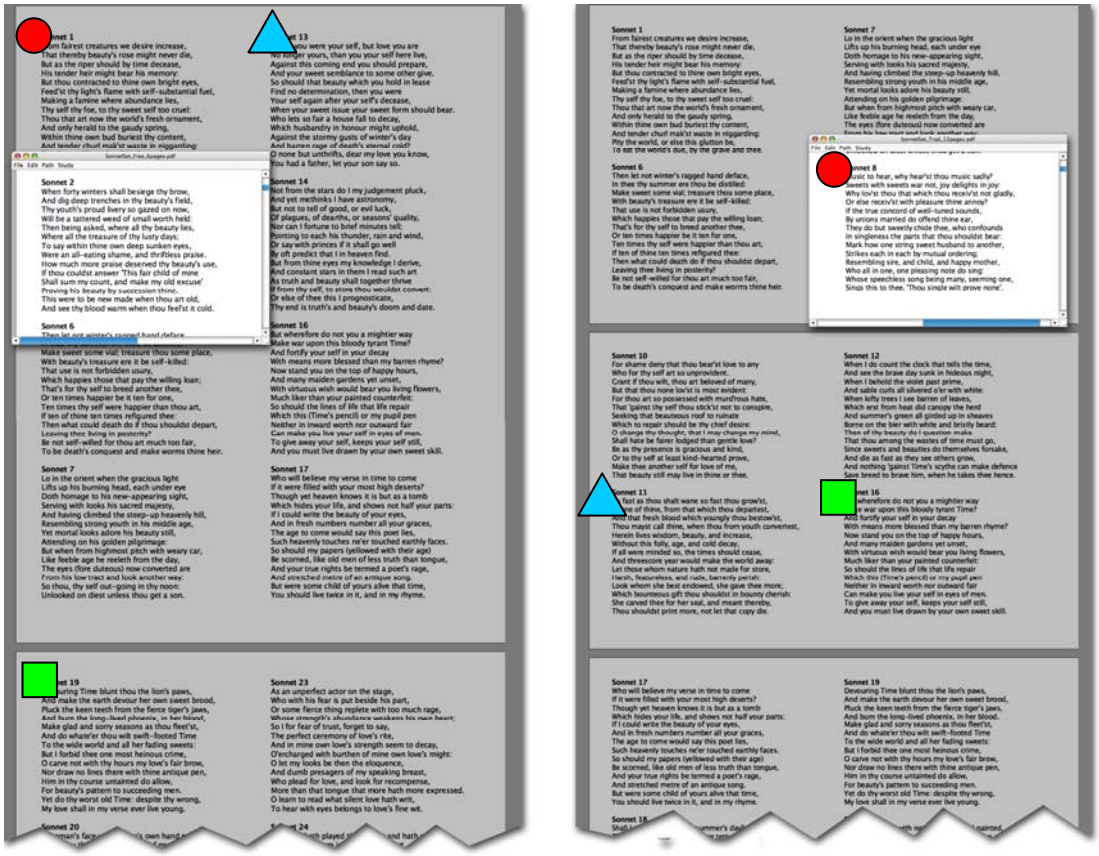

Figure 4.8: Screenshots of the viewer overlaid onto a PDF document in the long-column (left) and short-column (right) navigation tasks. Experiment trials timed participants while scrolling from a starting point (denoted by a red circle) to a destination point located either one column (denoted by a blue triangle) or two columns (denoted by a green square) away. (Overlaid colored shapes are used in the above image for illustration only).

Long-Column Navigation Task: Each participant completed 60 measured trials (3 techniques × 2 distances × 10 repetitions), as well as 30 interspersed unmeasured trials (3 techniques × 10 repetitions).

Short-Column Navigation Task: Each participant completed 90 measured trials (3 techniques × 2 distances × 15 repetitions), as well as 45 interspersed unmeasured trials (3 techniques × 15 repetitions).

Unmeasured trials contained non-integer column transitions (e.g., from the top of one column to the center of the next column) and were included so that participants could not as easily anticipate the approximate location of the target sonnet (i.e., one or two columns away). Since all the sonnets in the document were ordered, the participant knew whether they would have to scroll forwards or backwards in the document, but did not know exactly how far. Distances were randomly ordered, but the order was consistent across all participants. The order in which the techniques were used was counterbalanced across all participants and all trials for a particular technique were completed before a participant used the next technique.

Immediately following completion of each technique within a particular task, each participant filled out a hand-written questionnaire rating the technique in context of that task in terms of ease of use, satisfaction, and intuitiveness on a Likert scale of 1–5 (5, being the best).

4.6.4.2. Hypotheses

We formulated the following six hypotheses before running the navigation experiments:

(H1) For both the long-column and short-column navigation tasks, due to the automatic transitions from the end of one column to the beginning of an adjacent column, C-scrolling will be the *fastest* when transitioning one column.

(H2) For both the long-column and short-column navigation tasks, due to the required use of two mouse buttons to use V-scrolling and click on targets, it will be the *slowest* when transitioning one column.

(H3) For the long-column task, due to the participants' comfort with T-scrolling, it will be the *fastest* when transitioning two columns.

(H4) For the long-column task, due to the requirement of having to scroll through the entire height of each column, C-scrolling will be the *slowest* when transitioning two-columns.

(H5) For the short-column task, due to the automatic transitions from the end of one column to the beginning of an adjacent column, C-scrolling will be the *fastest* when transitioning two columns.

(H6) For the short-column task, due to required use of two mouse buttons to use V-scrolling, it will be the *slowest* when transitioning two columns.

In summary, we felt that each technique had its own strengths depending on how far one had to scroll and how much scrolling effort and time was saved due to the animations. Since the participants were unfamiliar with the document, we made an assumption that, for the documents with short columns, participants would navigate them in the order in which they were meant to be read (i.e., navigate through each sonnet on each page before proceeding to the next page), with little anticipation of how far they needed to scroll.

Therefore, we felt C-scrolling would outperform the others in scenarios involving short-column transitions. However, for long column transitions, we felt that participants would soon find navigating in the reading order a bit tedious. Therefore we thought they would soon begin to "hedge their bets" by, for example, skipping one column (and consequently several sonnets) with the hope that they would only need to advance forward or backward a few lines, and we believed they would be successful in doing so most of the time. Consequently, we felt T-scrolling would outperform the others when navigating two long columns.

4.6.4.3. Results for Long-Column Navigation Task

We analyzed the results from the long-column navigation task, measuring completion-time and subjective preference. To help reduce the significance of outliers, we used median values for our completion-time analysis. We used an α of 0.05 to denote statistical significance.

Completion-time analysis: We performed a 3 (Technique) × 2 (Distance) repeated measure ANOVA on median completion times for 9 of the 10 participants as the random variable. (Since one participant did not successfully attempt the C-scrolling technique, we excluded his results from this particular analysis). While Distance had a significant main effect on completion time ($F_{(1,8)} = 67.759$, $p << 0.001$), Technique did not ($F_{(2,16)} = 1.091$, $p = 0.360$). However, the interaction between Technique and Distance did have a significant main effect ($F_{(2,16)} = 15.431$, $p << 0.001$).

Figure 4.9 shows mean completion times for the long-column navigation task for the two Distance values, broken down by Technique. For one-column transitions, C-scrolling performed significantly better than T-scrolling (t_8 = 2.538, p = 0.017), partially confirming H1. T-scrolling performed significantly better than V-scrolling for one-column transitions (t_8 = 3.181, p < 0.01), partially confirming H2. For two-column transitions, there was no significant difference between the completion times for T-scrolling and V-scrolling (t_8 = 0.472, p = 0.325), thus failing to confirm H3. However, V-scrolling performed significantly better than C-scrolling for two-column transitions (t_8 = 2.836, p = 0.011), confirming H4.

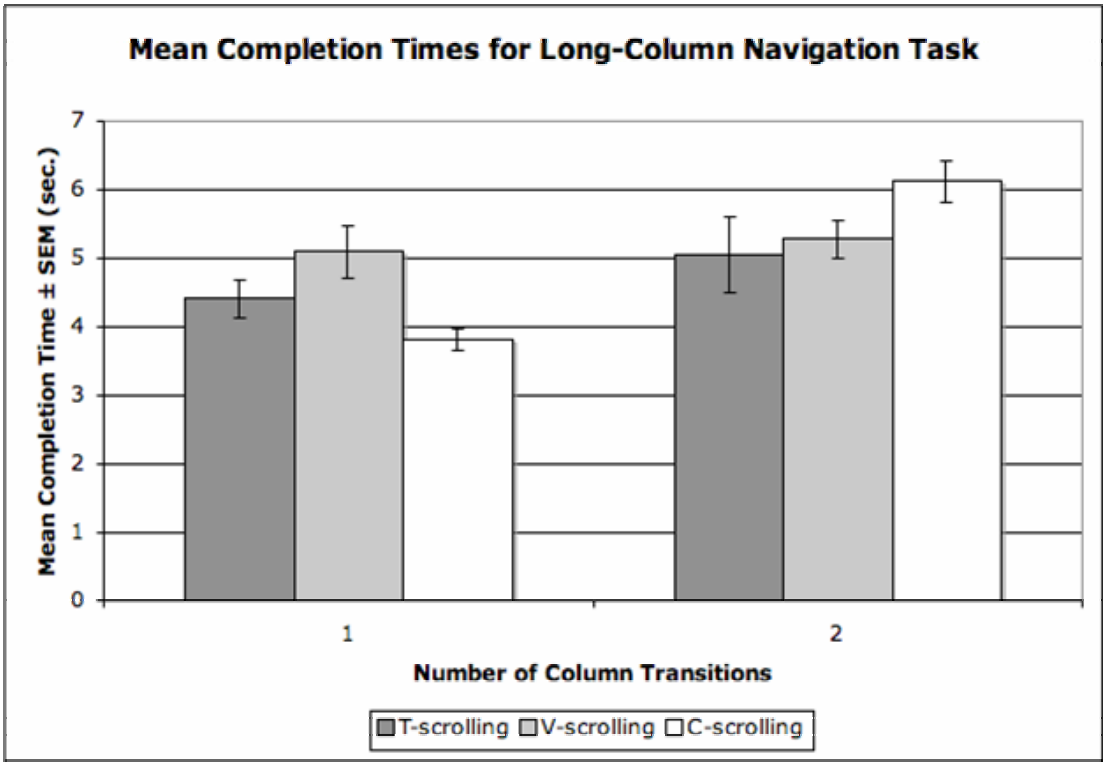

Figure 4.9: Mean completion times for the long-column navigation task, showing that the C-scrolling technique performed the best when transitioning one column, but performed the worst when transitioning two columns.

T-scrolling and V-scrolling performed better than C-scrolling when transitioning two columns, most likely due to the combination of the spatial layout of the two-column document and the hedging of bets hypothesized earlier, where participants were observed to have successfully guessed the target sonnet's approximate location, and then performed "fine-tuning" scrolling to its exact location, as demonstrated in Figure 4.10. For example, consider a trial starting at the bottom of a left-hand column at sonnet 7, requiring that the participant scroll forward to sonnet 17 (with the participant not knowing spatially where it appears). Rather than initially performing relatively time-consuming pan gestures up and then to the right to see the following sonnet at the top of the right-hand column, the participant could make a quick single pan gesture to the right (to see the next column) or downwards (to see the next page) to verify whether the pan to the top right is necessary. Many participants found this initial time investment worthwhile and we feel it helped them perform tasks quicker with T-scrolling and V-scrolling.

Qualitative Evaluation Analysis: Following the use of each technique in the long-column navigation task, each participant filled out part of a hand-written questionnaire rating the technique in context of the task in terms of ease of use, satisfaction, and intuitiveness on a Likert scale of 1–5 (5, being the best). All techniques seemed to be ranked comparably, as shown in Figure 4.11. We also asked which technique they would prefer to use, given this task. Preferences were again mixed, where three participants preferred T-scrolling, three preferred V-scrolling, and three preferred C-scrolling.

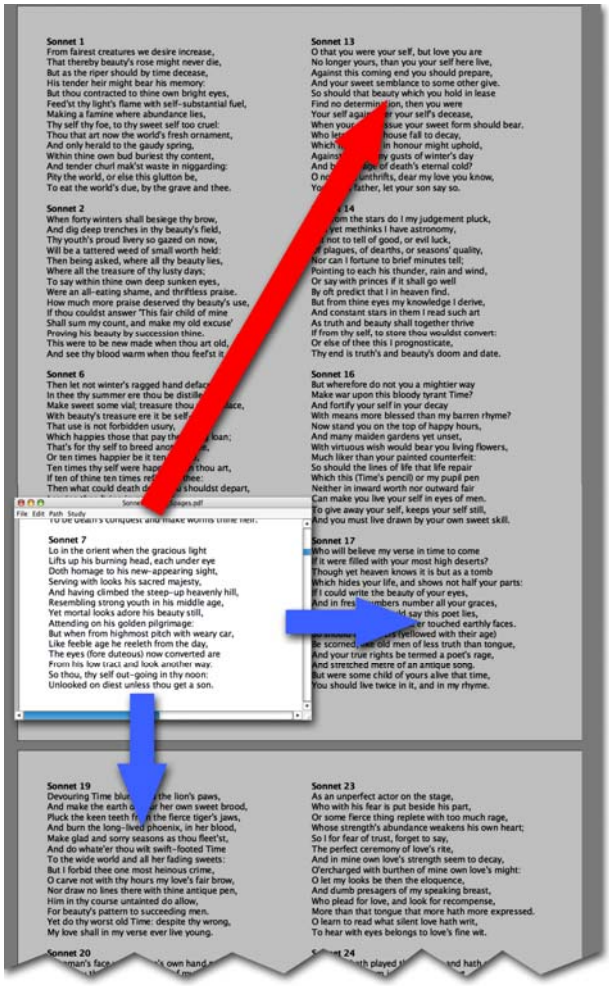

Figure 4.10: When using T-scrolling and V-scrolling, some participants "peeked" at parts of the long-column document out of order (indicated by the short blue arrows), taking a chance of not having to scroll through every sonnet in order to find the target. This often prevented long unnecessary scrolling motions (such as that indicated by the long red arrow).

Participant comments about the use of the different scrolling techniques for this task included frustrations about the long scrolling distances. Although some felt V-scrolling was very suitable for long-distance scrolling, two felt it made them dizzy or caused some strain on their eyes. Some participants noted that C-scrolling was great for short distances, but tedious for long distances, a comment that is consistent with C-scrolling performance measurements. Most complaints about C-scrolling targeted the repeated scroll wheel rotation gestures needed to complete longer scrolls. Several participants

mentioned they would have liked a fourth technique that combined the use of C- and V-scrolling into a continuous scrolling interaction along the reading path without having to repeatedly rotate the scroll wheel. Comments about the T-scrolling technique included how repetitive mouse-down-and-drag panning gestures caused fatigue in the hand.

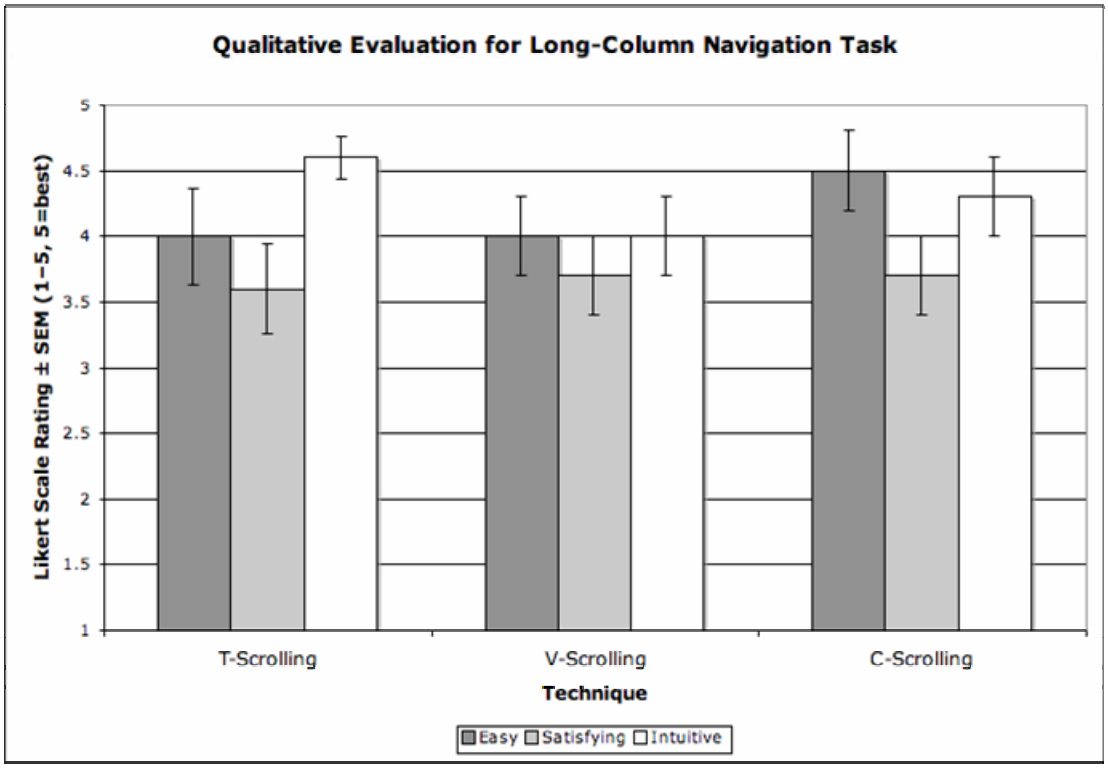

Figure 4.11: A qualitative evaluation of the long-column navigation task shows that the three techniques were rated comparably in terms of ease of use, satisfaction, and intuitiveness.

4.6.4.4. Results for Short-Column Navigation Task

We analyzed the results from the short-column navigation task, measuring completion-time and subjective preference. Although we present the results from the short-column navigation task after those from the long-column navigation task, we again note that the order in which the participants completed the tasks was counterbalanced. To help reduce the significance of outliers, we used median values for our completion-time analysis.

Completion-time analysis: We performed a 3 (Technique) × 2 (Distance) repeated measured ANOVA on median completion times for the 10 participants as the random variable. Distance ($F_{(1,9)} = 132.57$, $p << 0.001$), Technique $F_{(2,18)} = 27.943$, $p << 0.001$), and the interaction between them ($F_{(2,16)} = 16.714$, $p << 0.001$) all had a significant main effect.

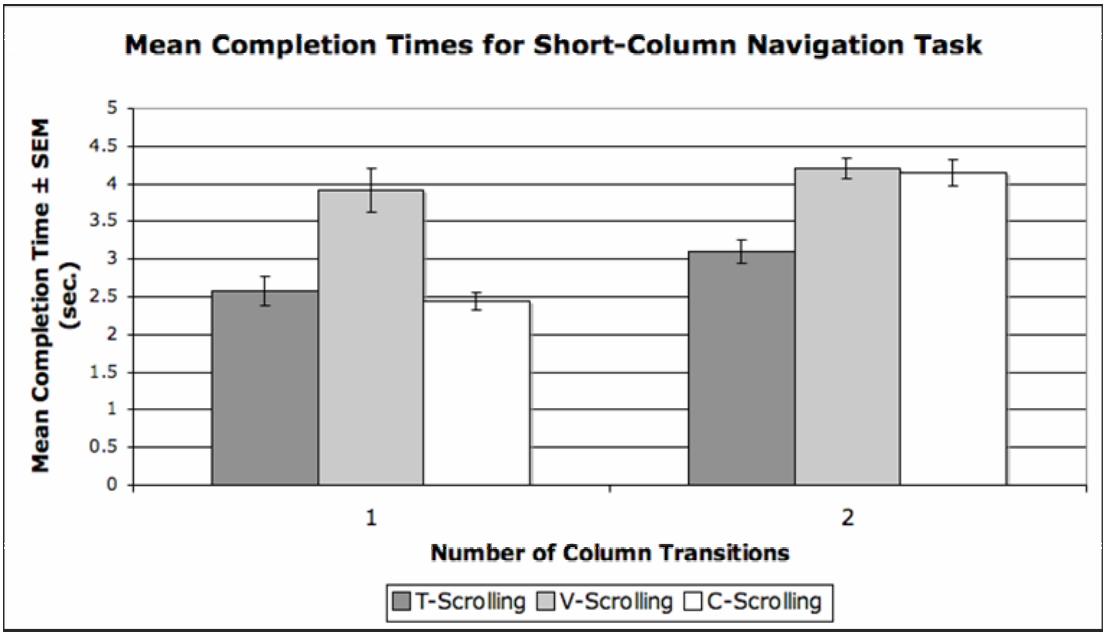

Figure 4.12: Mean completion times for the short-column navigation task, showing that the C- and T-scrolling techniques performed better than V-scrolling when transitioning one column, and the T-scrolling technique performed the best when transitioning two columns.

Figure 4.12 shows mean completion times for the short-column navigation task for the two Distance values, broken down by Technique. For one-column transitions, there was no significant difference between C-scrolling and T-scrolling ($t_9 = 0.723$, $p = 0.244$), thus failing to fully confirm H1. Both T-scrolling ($t_9 = 4.769$, $p < 0.001$) and C-scrolling ($t_9 = 5.535$, $p < 0.001$), however, performed significantly better than V-scrolling, thus fully confirming H2. For two-column transitions, we were surprised to find that T-scrolling

performed significantly better ($t_9 = 5.252$, $p < 0.001$) than C-scrolling, thus failing to confirm H5. Finally, for two-column transitions, there was no significant difference between C-scrolling and V-scrolling ($t_9 = 0.393$, $p = 0.352$), thus failing to confirm H6.

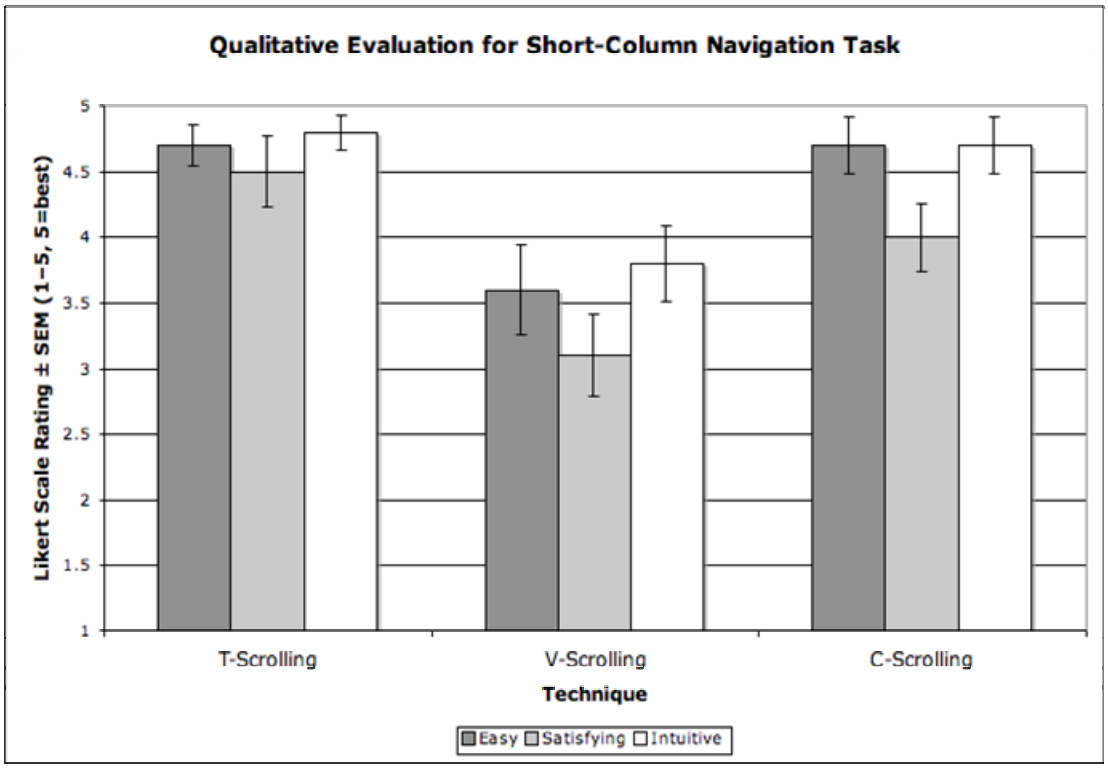

Figure 4.13: A qualitative evaluation of the navigation task (short columns) shows that the T- and C-scrolling techniques were rated better than the V-scrolling technique in terms of ease of use, satisfaction, and intuitiveness.

Qualitative Evaluation Analysis: Immediately following completion of the short-column navigation task, each participant filled out a hand-written questionnaire identical to that used in the long-column task, rating each technique in context of the short-column navigation task in terms of ease of use, satisfaction, and intuitiveness. Both the T- and C-scrolling techniques had higher mean scores than the V-scrolling technique, as shown in Figure 4.13. Preferences were also mixed for this task, where four participants preferred T-scrolling, four preferred C-scrolling, and two preferred V-scrolling.

Participant comments about the use of the V-scrolling for this task mostly included suggestions on how to improve it, such as alternative ways to stop the scrolling rather than with a left mouse-button click. The need to use two different mouse buttons (middle button to initiate V-scrolling, left button to click on the target) was also very frustrating for some participants, especially for short scrolling distances, even though this two-button interaction was copied from commercial interfaces, such as Internet Explorer and Mozilla Firefox, which incorporate this technique. Additional comments about V-scrolling included that it made them feel dizzy, or that it caused some strain on their eyes. These comments were from the same two participants who stated this regarding the long-column navigation task.

Consistent with the long-column navigation task, favorable participant comments about T-scrolling mostly mentioned their familiarity with the technique, although some mentioned that with T-scrolling, "changing columns is not as intuitive," and that it is "not good for reading as we need too many mouse movements."

Those who preferred C-scrolling commented that they chose it over the other techniques because "it doesn't require much thought, I don't have to worry about positioning the text like with the other two," and "you don't have to bother where on the page the sonnet is." Many had suggestions for improvement to the C-scrolling technique, as they did with the long-column navigation task, such as "the scroll speed should be adjustable." Another commented that she would have liked to use C-scrolling with a different mouse, indicating that she could not achieve as many mouse wheel rotations with one gesture with the mouse used in the experiment as she could with her own mouse.

4.7. Discussion

We have shown how our CAS implementation has advantages over traditional scrolling methods in some user interface scenarios involving a need to transition from column to column and page to page. Whether slowly perusing or quickly navigating short document distances, using our CAS implementation, one can take advantage of the automatic scrolling transitions that occur from one part of a document to another, while incurring an effortless task interruption, as compared to other scrolling techniques, which require more effort when transitioning from column to column and page to page. Our user study participants favored our CAS implementation for reading, and although their preferences were mixed with regard to navigation tasks, they performed better using our CAS implementation in relatively short navigation tasks. Based on their reactions, we feel that both their performance and overall experiences could be further improved with some slight modifications to our CAS implementation, described below.

4.7.1. Untimely Animations

Our initial pilot studies (performed before the formal experiments) indicated that a visible document page break or substantial white space acted as a sufficient visual cue to indicate the end of a column or page and a need to scroll to continue reading. Consequently, our CAS implementation functioned accordingly; that is, animated transitions were invoked upon a scroll gesture that occurred immediately following a gesture that made a page break visible, as shown in Figure 4.14. However, two participants in the reading task experiment commented that the C-scrolling technique animated too early. In other words, upon scrolling towards the end of a column or page, they weren't finished reading before the viewer automatically transitioned to the beginning of the next column or page, even

though the page break was visible before the animation occurred. Therefore, we feel that it may be worth investigating possible modifications to alleviate this drawback.

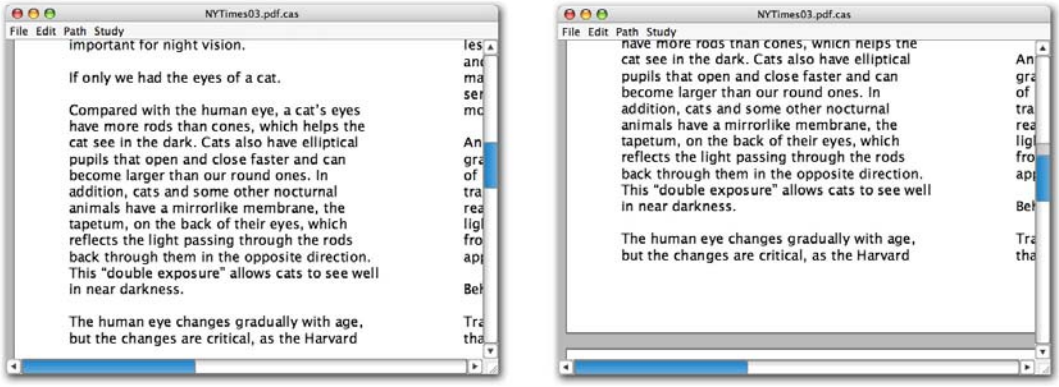

Figure 4.14: A page break is not visible in the viewer on the left to indicate the end of a column (or page), whereas the visible page break in the viewer on the right indicates the end of the column. Thus, with C-scrolling, the viewer on the right would animate to the next column upon the next scroll gesture.

One potential modification is to implement a variant of our initial CAS prototype implementation, which required that the current line be vertically centered within the viewport at all times, regardless of visible page breaks. As our early pilot studies indicated, this did not sit well with most users, but one possibility is to use a different scrolling gesture to allow the user to manually preempt the system indicating that they are ready to move on to the next column. A more aggressive mouse-wheel scroll gesture (such that multiple wheel notch rotations occur, rather than just one) is a possible candidate for this kind of interaction. This kind of functionality, however, might need to be accompanied by some sort of indication that such an interaction is possible at that moment. Otherwise, an aggressive scroll gesture will simply scroll normally advancing the document a distance proportional to the magnitude of the wheel rotation. Such an

indication could be in the form of a visual cue, such as an icon near the last line of the column.

Another possibility is for the system to provide some sort of visual indication to the user that an animation is about to occur, such that the user can stop scrolling if she is not finished reading to prevent the animation from occurring, or continue scrolling to proceed with the animation. Chang [Chang1993] discusses how using cartoon-like anticipation techniques can aid the user in preparing for such an animation by providing a quick motion contrary to the animation direction in the spirit of Wile E. Coyote in the Road Runner cartoons, who springs onto his back leg before dashing off after the Road Runner.

4.7.2. "C-V-Scrolling"

Several participants suggested a potential combination of C- and V-scrolling into what they called "C-V-scrolling." The motivation behind this suggestion stemmed from the appreciation of the different affordances of the two techniques. With C-scrolling, they appreciated the use of a 1D gesture to move forward and backward within a document that would normally require precise navigation in two dimensions. However, for long distances, repeated mouse-wheel rotations caused some frustration and fatigue. On the other hand, V-scrolling allowed rapid and continuous scrolling with minimal effort beyond the initial mouse click required to invoke the use of the technique. However, the precise steering required to navigate the document caused some frustrations among the participants. A technique that allowed users to expend little effort to scroll along the reading path of a document was, in their minds, achievable through "C-V-scrolling," a hybrid technique allowing users to scroll along the reading path of a document (a la C-

scrolling) in a continuous fashion until explicitly stopped (a la V-scrolling). Essentially, this has the advantage of the precise position control obtained with C-scrolling, but now with the added rate control advantage of V-scrolling.

Soon after the completion of the experiments, we implemented a version of CAS that incorporates vector scrolling. This required remapping the unrestrictive "joystick" interactions of V-scrolling (that is, an arbitrary user-defined 2D vector from the anchor position to the mouse cursor, which indicates direction and speed of scrolling) into a 1D vector that determines rate control limited to traveling either forwards or backwards along the reading path of a document.

Chapter 5

Content-Aware Layout (CAL)

Computer users often need to interact with more than one document at a time. Consequently, they often rearrange their windows by resizing and/or repositioning them. Most window managers provide ways to bring obstructed content to the front through the use of “alt-tab” (or, in the case of Mac OS X, “command-tab”), which invokes a menu containing a list of open windows (in Mac OS X, a list of open applications) from which the user can choose.

Some commercial window managers automatically rearrange windows, such that each one is unobstructed. For example, Microsoft Windows can tile or cascade each window ordered by recency of use. In contrast, Apple’s Exposé scales, tiles, and neatly rearranges

all open windows (or all open windows of a particular application) at the press of a key. However, none of these approaches take into account window content (as opposed to window type).

In this chapter, we present *content-aware layout* (CAL) [Ishak2007], an approach that considers the content of a window to help determine if and where it should be placed on the screen. For example, in Figure 5.1, after a user has selected a text string in a window, CAL presents other windows that contain the selected text, arranging them so that the search results are aligned horizontally, irrespective of the windows' bounds. We will show that taking into account the characteristics of the content, along with the user's task, potentially makes for more effective layouts.

We implemented a testbed application, called CASTLE (Content-Aware Scrolling, Transparency, and Layout Environment) that demonstrates the use of CAL coordinated with our other content-aware techniques. Rearranged windows can overlap by taking advantage of CAT to allow otherwise obstructed unimportant content (e.g., during a search, window regions not containing search results) to be seen through transparent regions of neighboring windows. CASTLE also allows a user to scroll through every search result within a window (using CAS) and between windows (using CAL) using only the mouse scroll wheel.

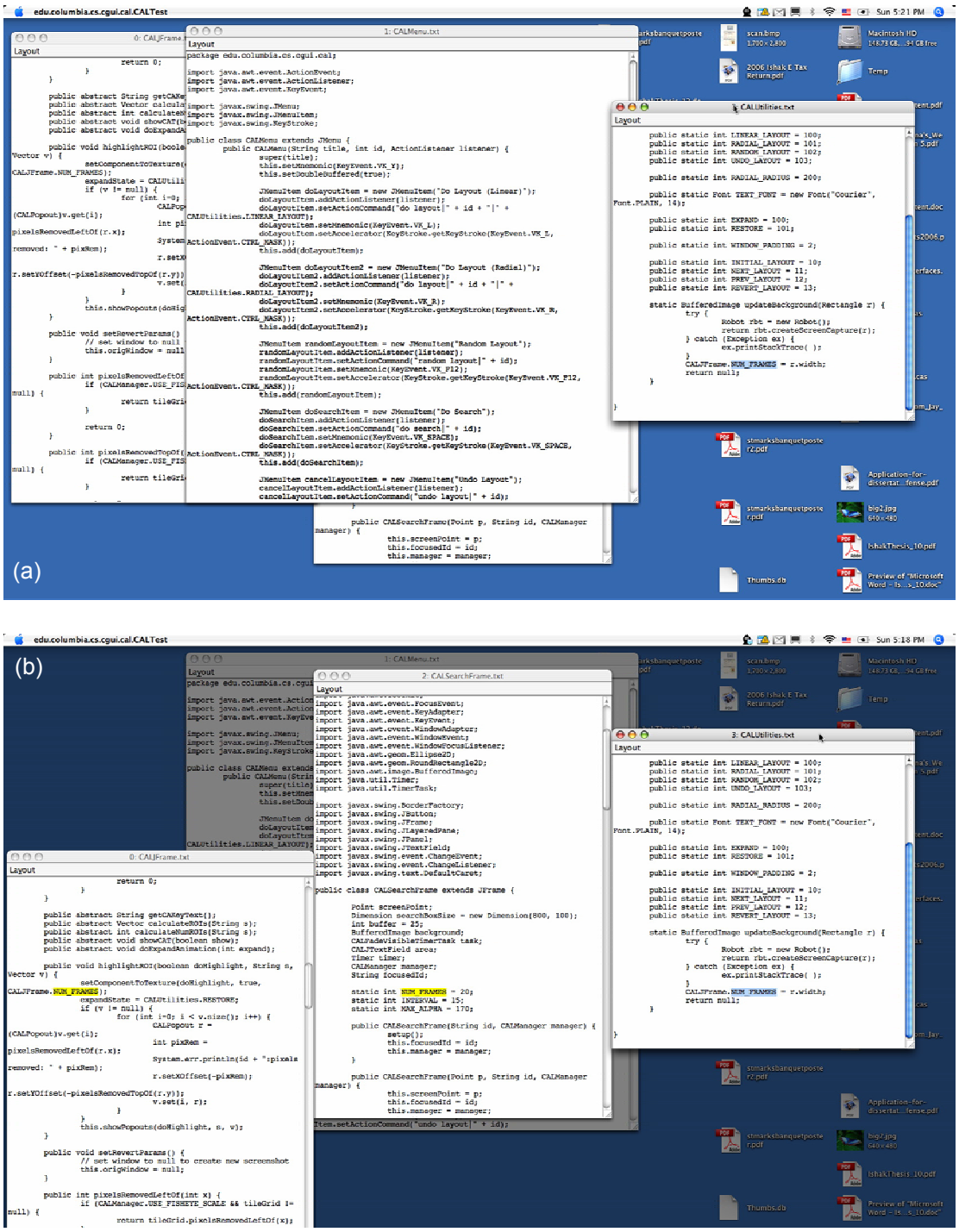

Figure 5.1: (a) A user wants to view other code windows containing the java variable "NUM_FRAMES," which is selected in the focused window. (b) The user invokes a layout-peripheral command, causing the system to search and present other windows that contain that text, with the first occurrence in each window highlighted and aligned with the selected text.

5.1. Related Work

We summarize relevant work in bringing obscured content into focus and automatically arranging content in user interfaces.

5.1.1. Bringing Obscured Content to Focus

Both alt-tab and Exposé are familiar techniques used to bring an obstructed object into focus. 3ds Max [Autodesk], Adobe Photoshop [Adobe], and our CAT mouse-over pie-menu (see Section 3.7.3) allow users to cycle through a list of overlapping objects beneath the current mouse cursor position. However, these techniques require a user to know the approximate bounds of an obscured window to invoke the list containing it.

5.1.2. Automatic Layout Using Constraint Solvers

There has been extensive research in automatic layout using design grids [Feiner1988, Jacobs2003], machine learning [Borning1986, Myers1993, Stille1996, Zhou1999], and evaluation techniques [Comber1995, Comber1988, Jeffries1991, Sears1993, Sears1997, Tullis1984]. There has also been research in layout using constraint solvers [Badros2001, Borning1986, Cohen1988, Cohen1986, Graf.1992, Hudson1990, Hudson1996, Kochhar1991, Myers1994, Vander Zanden1990, Weitzman1994], some of which attempt to adhere to aesthetic design criteria [Lok2004, Purvis2002]. (See Lok and Feiner [Lok2001] for a survey of this work.) However, these approaches (with the exception of [Lok2004]) do not consider the properties of the contents within the objects to which the constraints are applied, and most require manually-specified constraints. Some constraint solvers use information about the content to automatically decide where an object should be placed. For example, Lok et al.'s WeightMap [Lok2004] assigns weights to each

object based on its visual attributes to determine where it should be placed on the screen. However, constraints are then applied to the entire object, rather than to its content, whereas CAL applies constraints to the actual content that lies within a window.

5.2. The Approach

Before developing our approach to content-aware layout, we hypothesized that by making an existing layout approach content-aware, user interaction could be improved by facilitating the automatic arrangement of multiple windows that are *related* to each other based on their *contents*. This involves applying layout criteria to the window content. In contrast, conventional window managers, and even constraint-based research window managers [Badros2001, Cohen1988], perform layout based on geometric window bounds. CAL applies constraints automatically by having each application report to the window manager the locations and dimensions of *important regions* within each of its windows to which the constraints should be applied (e.g., window regions containing search results), thus rearranging the windows accordingly. CAL is an example of how a content-aware technique varies the display of content across a *spatial domain*, such that the *location* of content on the screen changes to allow the user to switch context more easily between important regions of multiple windows.

5.2.1. Visual Scope

We designed CAL with the intent to increase what Norman and colleagues refer to as *visual scope*: "the degree to which the user is able to integrate information across a display of multiple windows or screens and to grasp the whole of whatever is being displayed" [Norman1986]. Maintaining visual scope is important because many tasks

require visual access to multiple screen objects. Since it is not necessarily true that presenting more information to the user will make her more productive, CAL attempts to display no more than what is needed at a given time. For example, consider copying text from one window to another. If the system presents the user with more visual information than just the source and destination windows, the task may not be completed as quickly, or even correctly at all. In fact, according to Norman and colleagues, if there is no relationship spanning the various screen objects, this may decrease the visual scope. Our goal is to match the user's visual expectations of what is to be displayed on the screen with what is actually displayed.

5.3. CAL Design

5.3.1. Anchoring the Working Set

With large displays, if content is not placed strategically, users may spend several seconds trying to locate it. During normal interaction, users often view or interact with only a part of the display (or what has been referred to as the *working set* [Card1991]). This can vary both physically and semantically from a paragraph of text to some meaningful subset of windows. CAL regards the area containing the working set as important for two reasons: (1) it is the region where the user is currently working, and therefore, contains information that is important to the user at that time, and (2) it is the physical part of the screen containing content around which the user may wish to see additional contextual information. For these two reasons, if this physical region is known by the system during a layout operation, CAL uses it as an *anchor area*, a portion of the screen around which a layout occurs.

CAL allows the user to directly specify an anchor area. Although one could use an eye tracker to determine where the user is gazing, or record the user's mouse activity to determine what parts of the screen she is interacting with more than others, these methods may not accurately reflect the parts of the screen containing content in which the user is *currently* interested. For example, a user's eyes may be fixated on one part of the screen, while her mouse is on another, making it ambiguous to the system as to which area is more important. However, if the user directly indicates her area of interest, this gives CAL a well-defined screen region containing content in which the user is interested. We allow a user to directly indicate her area of interest using text selection, as described in Section 5.4.2.

5.3.2. Layout Scheme

A variety of layout schemes could be used for rearranging various types of content. When dealing with text, for example, layout schemes such as an array layout (horizontal or vertical), a grid layout (i.e., 2D array), or a radial layout (e.g., pie menu layout) could be used. Furthermore, alternative layout schemes may be more appropriate for other types of content. We have chosen a *linear horizontal array* layout when dealing with text because it resembles side-by-side book pages, facilitating an easier visual context switch between the working set area and peripheral textual content. However, we made two distinct content-aware modifications. First, only those windows containing relevant content are laid out. Second, for each of these windows, the layout is performed with regard to the content within the window, not just the window bounds, as with most other layout managers. Based on an analysis of window content, we could have, in theory, modified an existing constraint-based layout system to enforce the geometric constraints. For

example, one could extend RTL's Neighborhood or Regional strategies [Cohen1988] to apply the constraints to objects within a window, rather than to the window bounds. Similarly, one could use a variant of SCWM's Vertical/Horizontal Alignment constraint [Badros2001] to align *offsets* of the top and left edges of windows, rather than enforcing strict edge alignments.

5.4. Application

To demonstrate the CAL concept, we have developed a prototype text processor application in which users can view and create text documents, called the *CALViewer*. It is a cross-platform application and allows any user to create and interact with custom text editor windows that behave like any other window on their desktop for the purpose of creating or viewing plain text content (e.g., source code).

5.4.1. Implementation

We implemented the CALViewer using the Java 1.4 SDK on top of a custom Swing library containing abstract Java classes that inherit from standard Swing base classes. For example, our `CALJFrame` class inherits from Swing's `JFrame` class. This abstract class requires any inherited class to implement several key methods to report pertinent information about the application during layout. For example, our `CALViewer` class, which inherits from our `CALJFrame` class, contains a `JTextArea` to display its textual content. The `CALViewer` class has to implement `CALJFrame`'s abstract `getImportantRegions(Object obj)` method, which returns an array of `Rectangle` objects reporting the locations and dimensions of all the important regions

within the viewport of its `JTextArea` object that matches the search criteria specified by the variable `obj` (in this case, it would be textual string match). Similarly, other types of applications can be built on top of the `CALJFrame` class and their functionalities would behave depending on the implementations of the `getImportantRegions` method. We use Bell's dynamic space manager [Bell2000] to control free space.

5.4.2. Rearranging Similar Documents

When a user interacts with text using the CALViewer, she can focus on other text windows containing similar content. In our implementation, the CALViewer uses textual string matches to identify these windows. This works well for various tasks in which one must locate other windows containing the same term or phrase. In the case of writing source code, for example, this may be useful when trying to identify the occurrences of a particular variable in other code windows. In Section 5.5, we discuss how this was helpful to physicians who often simultaneously peruse multiple notes containing identical words and phrases, such as standard section headings that usually appear in every note.

By first selecting text in a window using any conventional selection mechanism (e.g., double clicking on a word), the user can then invoke a *layout-peripheral* operation across all other open windows (through either a menu option or a ctrl-L shortcut key). Once invoked, the CALViewer takes the selected text as input, sets its current screen location as the anchor area, and for each window containing that search string, highlights the first occurrence, and then rearranges it using a linear horizontal array layout scheme, such that the first occurrence within each document is lined up with respect to the anchor area, as shown in Figure 5.1. This allows the user to visually scan the search results in a single

spatial dimension. The rearrangement is performed using a slow-in-slow-out animation over 500 ms, similar to that of Apple's Exposé and earlier work on automated layout [Bell2000]. The location of each window is restored using the same shortcut key, or by pressing the "escape" key.

Since the CALViewer application manages multiple text editor windows, the layout-peripheral command requires the focused window to communicate the selected text, as well as its absolute screen position, to all other CALViewer windows to search and possibly rearrange them. For each of those other windows, if the search term is found, it identifies the location of the search term's first occurrence relative to the window's viewport. The window is then immediately placed (via an animation) as close to the focused window as possible, such that the search term in the animated window is horizontally aligned with the selected text of the focused window (i.e., the anchor area). It attempts to do this without resizing the window and without scrolling within the viewport. However, in the case that the first search term of the animated window is not visible within the viewport (i.e., scrolling is needed to view it), it automatically scrolls as little as possible, such that it is made visible and aligned with the anchor area.

5.5. CASTLE: Combining CAT, CAS, and CAL

To demonstrate the coordinated implementations of our other content-aware techniques, we designed and implemented CASTLE (Content-Aware Scrolling, Transparency, and Layout Environment), which incorporates all three of our content-aware techniques. At the heart of CASTLE is CAL. CASTLE also incorporates content-aware transparency (CAT) to allow regions of windows to be seen through transparent portions of

neighboring windows, and content-aware scrolling (CAS) to allow users to navigate within and between adjacent windows to visualize and interact with content across multiple windows using a 1D input device. For example, in Figure 5.2, a user can search across all windows, invoking CASTLE to present only those that contain the search term, arranging them so that their search results are aligned horizontally in the center of the screen. Using the mouse scroll wheel, the user can then scroll through each occurrence of that string within each of the windows.

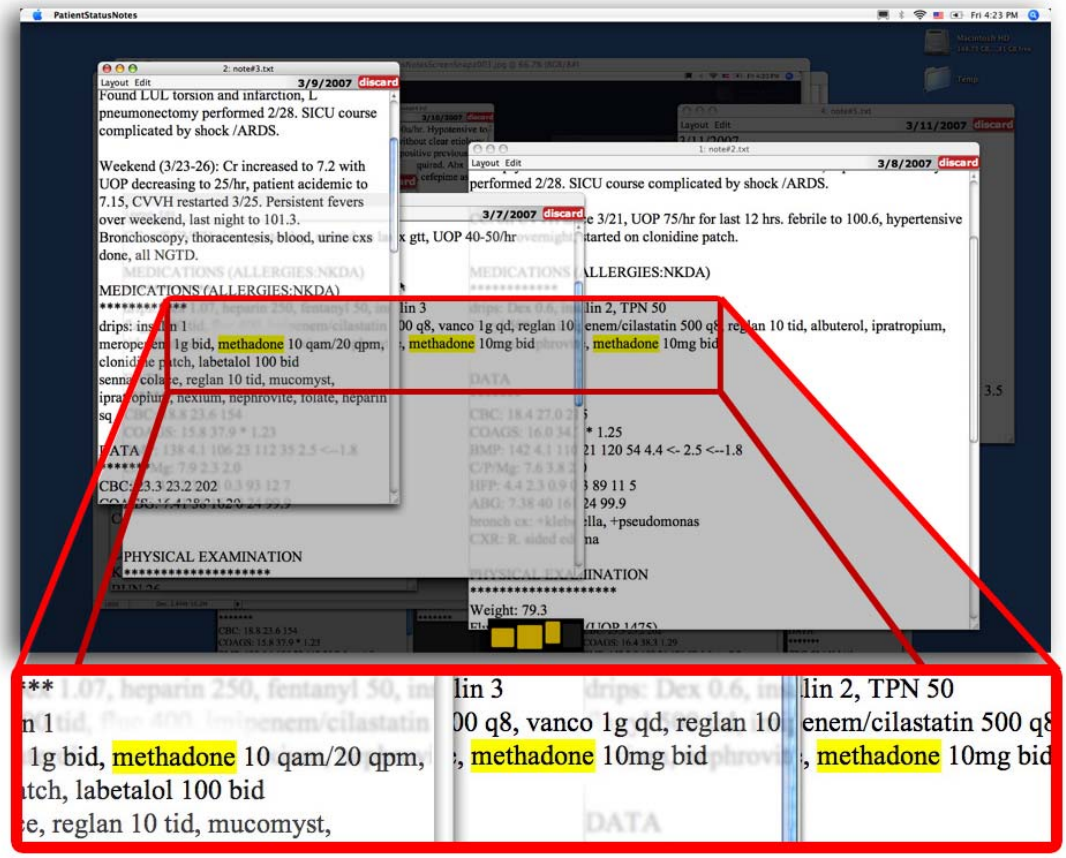

Figure 5.2: CASTLE can rearrange only those windows containing a search term, such that the first result in each window is horizontally aligned in the center of the screen. If necessary, it allows windows to overlap, such that content-aware transparency can allow otherwise obstructed unimportant regions to be seen through neighboring windows.

We have asked two physicians with whom we are collaborating in the Cardiothoracic Intensive Care Unit at New York Presbyterian Hospital to use CASTLE to peruse patient daily status notes to obtain feedback about its use in a hospital environment. CASTLE has shown that taking into account the characteristics of the content, along with the user's task, can make possible a more effective user interface.

5.5.1. Patient Notes Application

We have developed a prototype application in which physicians can peruse patient daily status notes using CASTLE. It is cross-platform and allows physicians to interact with custom text editor windows that behave like any other windows on their desktop. Our implementation uses the Java 1.4 SDK and a custom Swing library.

5.5.1.1. Search All Patient Status Notes

A physician is initially presented with a cascaded stack of all the daily status notes for a particular patient. Each note is registered with CASTLE to allow CASTLE to manipulate its location during layout. When a physician peruses a particular note (either written by her or another physician on a previous day), she can choose to make visible additional notes containing similar content by invoking CAL's layout-peripheral operation across all other open notes through either a menu option or a ctrl-L shortcut key (details discussed in Section 5.4.2).

A physician can alternatively perform a *search-and-layout* operation across all open windows, without first selecting any text. Pressing ctrl-space causes an input dialog to appear, which allows the user to enter a text string. Upon pressing return, windows

containing that string are located and the first occurrences of each search result are rearranged in the center of the screen, using the linear array layout scheme with its search results highlighted, as shown in Figure 5.3.

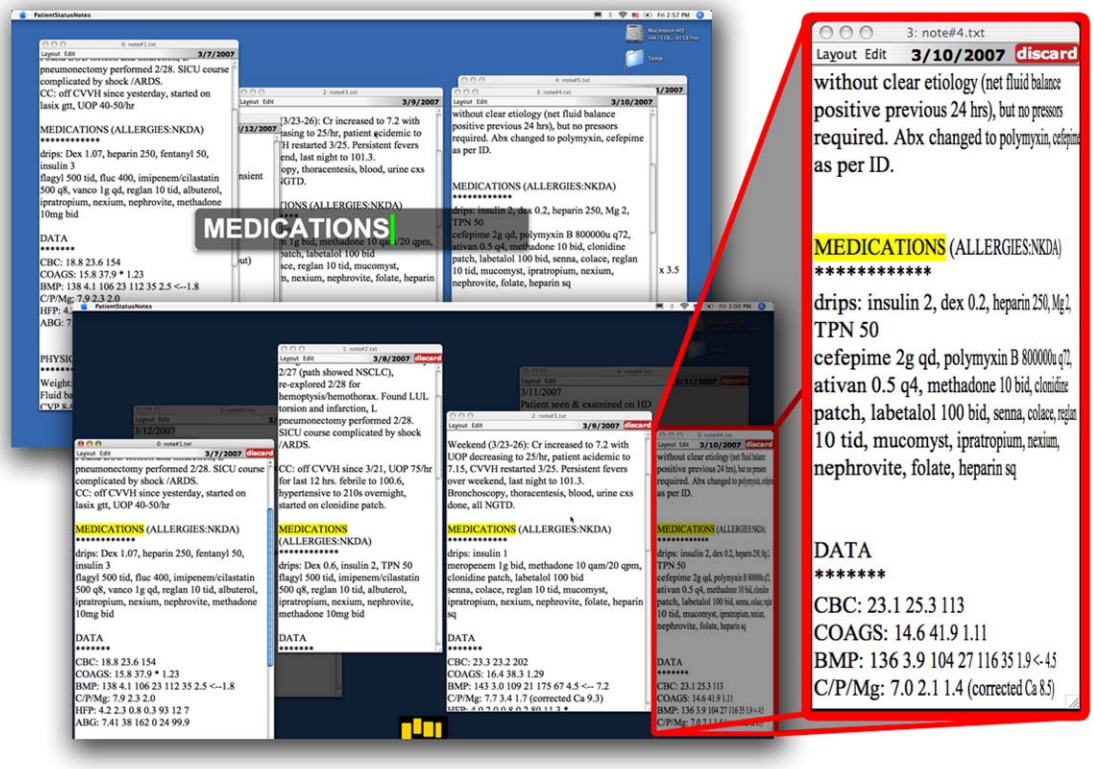

Figure 5.3: When a physician performs a search for "MEDICATIONS," CASTLE horizontally aligns every daily status note containing that term in chronological order, providing an effective timeline of drugs taken by the patient. The callout on the right shows the non-linearly scaled view of a window in its entirety that would otherwise overlap the screen edge.

5.5.1.2. Layout Order

Using a linear horizontal array layout scheme, as described in Section 5.3.2, allows the physician to visually scan the search results in a single spatial dimension, ordered left-to-right. In the case of a search-and-layout operation, the default sort is in decreasing order of result hit count. However, for physicians, the chronological order in which the notes were created is important to understand the patient's status across multiple days. For our

patient notes domain, we use this order by extracting each note's status date, usually written by the physician within the first few lines of each note. In case this information is missing from the note, we use the file's creation date instead.

5.5.2. Seam-Awareness

For cases in which the screen cannot accommodate all relevant windows in one view, we have incorporated seam-awareness [Mackinlay2004] by implementing an orthogonal non-linearly scaled view of window content that would otherwise overlap a monitor seam, as shown at the right of Figure 5.3. This is a distortion effect similar to that used in the Fishnet web browser [Baudisch2004b], except that, in CASTLE, distortion is performed non-uniformly, where content nearer to the screen edge is scaled by a larger factor. We only scale windows that would otherwise overlap the screen edge, and we do not allow any part of the window to be scaled at less than 50% of its original size to maintain its legibility. Before the layout is reverted, a user can click on the scaled window to view it temporarily in its original size. In all cases, the scaled window's original size is restored when the layout is reverted.

5.5.3. Incorporating CAT

If possible, we prevent window bounds from overlapping. However, CASTLE allows windows to overlap such that important regions remain unobstructed to accommodate the simultaneous view of additional windows. To provide a visualization of obstructed content, CASTLE incorporates an implementation of CAT to take advantage of screen space occupied by *unimportant regions* of neighboring windows, as shown in Figure 5.2. CAT varies the transparency level of different window regions based on their importance,

and therefore renders important content and its background opaque. If unimportant regions are rendered transparent, they can overlap otherwise hidden regions of a neighboring window and thus expose them to the user. As discussed in Section 3.4.2.2, CAT also applies a Gaussian blur to visually similar underlying content (in this case, text) to reduce visual interference. We use Bell's dynamic space management system [Bell2000] to query and control unallocated space on the display.

If the screen is still insufficiently wide, we display an initial set of windows, and then allow the user to iterate through the next set by pressing ctrl-right or ctrl-left to advance forwards or backwards, respectively. The current visible set of windows is highlighted within a miniature view of all relevant windows at the bottom of the screen, indicating any additional windows that are currently invisible, as shown in Figure 5.2 and Figure 5.3. Unwanted windows can be removed from the current set by clicking the "discard" button in the upper right corner of the window, allowing other windows to immediately occupy the newly available screen space.

5.5.4. Incorporating CAS

CASTLE incorporates CAS to support the use of a zero-order control device (i.e., the mouse scroll wheel) to navigate through each search result in every window. After the initial layout, if the user positions the mouse cursor over a window and rotates the mouse scroll wheel, CASTLE navigates between search results by automatically panning to the next result within the current window using CAS. Once the user reaches the last result in that window, a subsequent scroll wheel rotation uses CAL to rearrange the windows' locations to the left when scrolling down (or to the right when scrolling up), such that the

first result of the next window is repositioned in place of the last result of the current window. Upon reaching the last result in the last window, the next scroll wheel rotation cycles around to the first result in the first window.

5.5.5. Physician Feedback

We have asked two resident physicians with whom we are working in the Cardiothoracic Intensive Care Unit at New York Presbyterian Hospital to informally evaluate the use of CASTLE for perusing daily patient status notes. After a series of initial formative evaluation sessions, they interacted with the current version of the system for two hours. Based on their experience trying the system, they stated that a system such as CASTLE could greatly improve the way in which they perform everyday tasks involving monitoring patient status notes. They only have several minutes to spend on each patient and therefore are time-constrained when making such references. Using our system, a physician can easily make inferences about a patient's daily progress. For example, a physician typically needs to extract temporal information about a patient, such as the drugs taken over several days, by visually inspecting each note's section entitled "MEDICATIONS." Where this section appears in the note depends on its author. In CASTLE, the physician can simply search for that section heading using the search-and-layout command. They appreciated that CAL automatically rearranges the notes, lining up their MEDICATIONS sections (each with a list of drugs underneath the section heading) sorted in chronological order, as shown in Figure 5.3. This effectively produces a display of the drugs a patient has taken over the course of several days in a matter of seconds. Although they preferred neighboring windows not to overlap if possible, they said that they liked how CAT reduced the obstruction of window regions, since these

otherwise hidden regions were now visible through unimportant regions of neighboring windows. They also appreciated how CAS made it possible to use the mouse scroll wheel (or key commands), to scroll through the search results across all of their open windows. In addition the physicians told us that they appreciated the speed and ease with which they could search for and compare this information, which would otherwise take up to several tedious minutes to perform by hand with the software that they currently use.

Chapter 6

Conclusions and Future Work

In this dissertation, we have described a unique approach to modifying traditional user interface techniques to improve both user productivity and user experience when viewing or interacting with 2D content.

Through this chapter, we will summarize our contributions and present future work. We will present alternative content-aware modifications to the various traditional techniques we have successfully modified throughout this dissertation. We will also discuss how our contributions have opened the door to alternative interaction design and investigations of other traditional techniques that can take advantage of our approach.

Finally, we will discuss how different domains outside 2D user interface design can take advantage of content-aware modifications. Although we have applied this approach to 2D user interfaces, we believe that being content-aware is applicable to a variety of domains, including 3D user interfaces.

6.1. Summary of Contributions

We have developed a set of user interface techniques that take into account an understanding of the content with which one interacts. We have taken the approach of modifying traditional user interface techniques and making them content-aware, exploring their ability to improve user performance and experience. Our contributions are:

- **Content-aware transparency:** The affordance of non-interfering and unambiguous visualization of overlapping window content by selectively applying different levels of transparency, as well as various image-processing filters, to overlapping content, based on various properties of the content.

- **Content-aware scrolling**: The automatic construction of 2D paths through documents based on various properties of the content within them, with the ability to scroll through those paths using a 1D position control input device, varying scroll speed and direction based on various properties of the content, as well as the user task.

- **Content-aware layout:** The automatic determination of which windows should be rearranged, and where those windows should be placed on the screen by applying layout constraints to window contents, instead of or in addition to window bounds, such that the new layout affords easier visual access to window contents.

- **Content-aware user interface:** A user interface that coordinates the implementations of content-aware transparency, content-aware scrolling, and content-aware layout to allow a user to view and interact with related information across multiple windows using a 1D position control input device.

6.2. Future Work

In this dissertation, we have presented specific implementations of content-aware modifications to traditional techniques. We now explore alternative implementations and future direction for each of the techniques.

6.2.1. CAT

We explore some scenarios in which it might be advantageous to consider different content properties, and present prototypes for two alternative CAT implementations, and speculate on their effectiveness. We also explore future enhancements to our mouse-over pie menu technique.

6.2.1.1. Considering Additional Textual Properties

In our CAT implementation, we allow applications to report their windows' high-level content types. For example, a text viewer application reports that its windows contain text. It is a coarse categorization that specifies whether or not its windows contain text, but without information concerning low-level textual properties. Consider how users interact with text. For example, computer system administrators can have many terminal windows that monitor the status of a computer network. Computer programmers frequently interact with many source code and documentation windows. Financial analysts typically visualize large tables of textual and numerical information. The content in each of the aforementioned situations is mostly textual, and therefore it may be worth exploring if user interaction could benefit from a more in-depth analysis of these textual properties. Properties, such as font color, face, style and size produce a plethora of visualization scenarios. Although text can be generically classified as having predominantly high spatial frequencies, various properties can be considered to determine different kinds of interferences that could potentially occur when texts of different styles overlap [Paley2003]. Consider also line spacing and character separation as additional determining factors. A variety of user tasks and scenarios could greatly benefit from an analysis that regarded such properties to prevent overly aggressive modifications to content, thus providing increased legibility of overlapping materials.

6.2.1.2. Vision Approach

Our approach to applying image-processing filters to an obscured window's content considers the properties of overlapping windows' contents. This information is reported to the system by the applications' windows, requiring communication between the

application and the system. However, a more general approach could allow the graphics card to use vision algorithms to automatically determine what to report to the system. For example, consider a text viewer application reporting that an underlying window contains high spatial frequencies (e.g., black non-anti-aliased text on a white background), similar to [Leykin2004], who presents a method to automatically determine if a given text will be readable over textured background for augmented reality systems. One could use various high-pass filters (e.g., edge detectors) to understand the spatial frequency domain and amplitudes of underlying content to determine if it might interfere with overlaid content. This analysis could be performed on a sub-window level, as opposed to on a window level, as in our implementation. This could result in an automatic determination of transparency levels applied to an overlaid window and various filters being applied to different regions within the same underlying window. Also, consider that, in our implementation, the application reports the *type* of content its windows contain, such as icon, image, or text. Various object segmentation algorithms can help determine a more granular level of type and if these similar objects overlap. In other words, there is potential for automatic determination of content properties' values, which is useful for when applications do not know how to communicate with the system the necessary information the system needs to make beneficial decisions when using CAT.

6.2.1.3. Level of Awareness

As we have mentioned before, content-aware interaction is built upon having an understanding of the properties of the content with which the user is interacting. In Chapter 2, we introduced several guidelines for designing content-aware interaction. However, the resulting modification can vary based on the *level of content awareness*. In

other words, how content-aware does a technique have to be for its modifications to be effective? Granted, the level of being content-aware can vary, but is there a sufficient level of being content-aware, such that further analysis does not produce additional benefits? We have shown how our level of awareness within the context of transparent user interfaces can improve user interaction, but there may be additional content characteristics that would further improve user performance and the user experience. Furthermore, in addition to content properties, contextual information, such as eye-tracking information, can help determine levels of importance.

6.2.1.4. Alternative 1: Assigning Multiple Levels of Importance

Consider assigning multiple levels of importance [Furnas1986] to overlapping content. In other words, rather than assigning regions of windows a binary value of importance (i.e., important or not important) as we did in our implementation, one could build an importance model that would assign relative importance between overlapping contents that could vary based on the task. For example, consider a region of an overlaid window containing content behind which obscured content is potentially *more* important at that time (e.g., the underlying content contains a search result or some information that has recently changed of which the user has not seen yet). In this situation, one may want to momentarily allow the legibility and accessibility of the more important obscured content. This differs from our implementation described earlier, where we assigned a binary level of importance to content and took the approach that overlaid important content always took precedence over underlying important content. However, it has potential, and should be explored and evaluated as a viable alternative technique.

6.2.1.5. Alternative 2: Using Orientation for Disambiguation

In Section 3.2.3, we discussed how research in perception has shown that orientation differences alone do not help observers perceptually scission overlapping layers. However, as discussed earlier, Akerstrom and Todd's experiments [Akerstrom1988] used textured layers containing random-line stereograms, rather than familiar figural objects. Therefore, consider more familiar objects that may be expected to be seen at a particular orientation (e.g., text is often expected to be oriented parallel to top and bottom edges of a window) where an orientation change may allow for correct perceptual scission. It also may be worth exploring how altering the orientation using an animation may help the scission process. Consider applying an *animation filter* to underlying content, such that it slightly jitters or is perceived to be seen through a water medium with slow moving ripples. Consider also a slight uniform pulsating-like variation of the transparency over time Some may be skeptical about these approaches, since animations are normally used to draw attention to a part of the screen, whereas in this case, it is used to reduce its interference with overlaid content.

6.2.1.6. Enhancements to the Mouse-Over Pie Menu

Since an arbitrary number of windows can exist at a particular pixel location, the mouse-over pie menu can encounter scaling problems when displaying a large number of menu items. We have found that when the pie menu must display more than eight items, it becomes difficult to display thumbnails of each window at a recognizable size. Possible future modifications to accommodate this limitation could include displaying only the window title, or increasing the radius of the pie menu, thus allowing more room to display more items. Alternatively, a hierarchical pie menu can include, in the primary

menu, an icon representing applications whose windows are beneath the specific pixel location, and then upon selection of an application, all of its windows in a secondary menu. Alternatively, the most important (e.g., most recently used) windows could be displayed in the primary window, with less important windows in the secondary menu.

6.2.2. CAS

We now discuss how our CAS implementation can be extended to navigate alternative paths for different kinds of content. We also comment on future work for our implementation to allow for more robust path creation within text documents.

6.2.2.1. Scrolling Through Large Documents and Across Multiple Documents

CAS allows scrolling through the reading path of text documents, which we have found to be beneficial in reading tasks and short distance navigation tasks. Long distance navigation tasks, however, require a user to scroll through potentially longer paths than through a strict up-to-down path traversed with conventional scrolling. A potential enhancement to CAS is to incorporate Speed-Dependent Automatic Zooming [Igarashi2000], such that depending on the user scrolling speed, complex 2D paths at the current zoom level (e.g., close enough to the document to comfortably read the text) could be approximated by simpler and more coarse paths at a smaller zoom level (i.e., further from the document). This could provide better scrolling performance and a better user experience when navigating large distances when using CAS.

In CASTLE, physicians can scroll through every search result across multiple documents using the mouse scroll wheel. After performing a search-and-layout using CAL, only those windows that contain that search result are presented to the user through which scrolling can take place. An alternative approach would be to lay out *all* the windows on the desktop and allow a user to employ CAS to traverse through each window, such that windows that better match the search query are scrolled through differently than those that do not.

6.2.2.2. 3D Scrolling

In our CAS implementation, one can travel along a path within a document (e.g., the reading path of a text document) using a mouse scroll wheel. Although we were able to automatically extract paths from text documents and photographs, other types of documents could potentially take advantage of being content-aware. Consider, for example, driving directions overlaid onto a 2D or 3D map (e.g., using Google Earth). The user may want to scroll, such that they virtually travel the route from its start to its destination. This feature is available in Google Earth, but not at a user-controlled rate. Since there may exist parts of the route that the user is not as interested in as others, we feel this interaction can be improved. Consider the LineDrive system [Agrawala2001], which automatically generates routes by distorting unimportant features (e.g., road lengths, angles, etc.) and enhancing important ones (e.g., turning points). Rather than visually distort unimportant features, CAS can distort the scrolling speed across those regions. For example, consider the following set of driving directions from Holmdel, NJ to Garfield, NJ taken online from local.google.com:

1. Head NW on RT-34 toward Ashley Dr.	2.4 mi	3 min
2. Turn right at CR-3/Lloyd Rd.	2.1 mi	5 min
3. Turn right at Clark St.	0.3 mi	1 min
4. Take Garden State Pkwy N ramp	285 ft	0 min
5. Take right fork to Garden State Pkwy N	0.1 mi	0 min
6. Take right fork to Garden State Pkwy N, merge onto Garden State Pkwy N	39.6 mi	43 min
7. Merge onto US-46 E via RT-20/US-46 exit 156 to River Dr.	0.5 mi	1 min
8. Exit US-56 E toward Elmwood Park/New Jersey Tpke/Garden State Pkwy	0.2 mi	0 min
9. Merge onto River Dr. via River Dr/CR-507 S exit to Garfield	1.5 mi	3 min
Total	46.7 mi	56 min

Step 6 in the route process asks the user to "merge onto Garden State Pkwy N" and to proceed 39.6 miles before continuing on to the next step in the route process. Although this single step is only 11% of the total number of steps, it represents over 85% of the total distance traveled. If one were to use CAS to traverse this route, one could scroll through this step at a faster rate than the rest of the route despite a consistent scrolling gesture, since there may not be a need to virtually experience the entire 39.6 miles at the same rate as other steps in the route process. Just as the system determined that lines of text were considered important regions within a text PDF document, turns/merges can be considered important regions along a map containing overlaid driving directions, and therefore, are scrolled through at a different rate than those parts in between them (i.e., the unimportant regions).

6.2.2.3. Conditional Scrolling

Another example of a potential use for CAS is within the context of conditional navigation. Documents are not always read from beginning to end where one visualizes the content in the order in which it appears. Zellweger introduced Scripted Documents

[Zellweger1989] as a way for authors to create conditional paths within and between documents. Similarly, consider a product user's manual containing a broad range of information, including safety information, step-by-step instructions on how to complete a task, and explanations of different warning labels. One manual is often written for an entire class of products (e.g., standard, professional, and enterprise versions of a software application), rather than one manual per specific model. Conditional statements within the document usually direct the reader to relevant parts of the manual containing information related to their specific model (e.g., after step 2 of an installation task, it might say: "for the professional model, proceed to step 5."). Some manuals are also generated from master documents that allow variants to be automatically extracted. A user could take advantage of CAS to scroll through the master document, such that they navigate along the appropriate path specific to their product model, skipping irrelevant parts of the document. This has a similar flavor to a famous series of children's books published by Bantam Books throughout the 1980's and 1990's, called "Choose Your Own Adventure," where each story was written such that the reader took the role of the main character [CYOA]. The reader was then occasionally presented with choices on how the story should progress and would then navigate to a different part of the book depending on which choice was made. Thus, the plot could unfold in different ways and eventually lead to a variety of different endings.

6.2.2.4. Intelligent Path Creation

Our CAS implementation automatically extracts reading paths from a text PDF document using the same reading path that Adobe Reader's "Read Out Loud" feature would use to help the visually impaired read a document. This works well for most documents, but

sometimes fails when the text resides outside the continuous main body. Some text (e.g., inline quotes, captions, titles) may spatially appear inline with the main body's text, but logically appear out of order in a separate text box after the main body. This results in paths that seemingly skip over these sections but return to them after the last word on the page—most likely an undesirable route.

To account for this, the path creation process could take a double pass approach. That is, after extracting all the text and its locations and dimensions, it could take a second pass to detect sections that seem out of place. A simple approach is to detect when a path skips over (and possibly intersects) a block of text with the appropriate spatial and visual properties to be visually integrated with the surrounding content. With the combination of a natural language processing approach, the path could more intelligently detect where such sections can logically be placed.

6.2.2.5. Path Parameter Enhancements

CAS paths are currently created using piecewise linear interpolations of a sequence of points defined by important regions throughout the document. Although this works well for textual content, various path parameter adjustments can help improve the interaction for other types of content, such as images, as in our CAS Image Viewer. For example, if we make use of polynomial interpolations, or splines, a smoother path can be constructed through the faces of people in photographs. This may also work well for paths determined by driving directions on 2D or 3D maps.

6.2.3. CAL

We now discuss potential future work for CAL, including modifications for the Patient Notes application and alternative layout schemes.

6.2.3.1. Patient Notes Application Enhancements

With the help of our colleagues, we hope to incorporate a high-level medical classification algorithm to help retrieve documents that are "relevant" to completing a daily patient status note without having to match an exact textual search. We would also like to take into account a model of the document structure to refine the scope of the important regions (e.g., to include the full list of drugs under "MEDICATIONS" rather than just the heading).

With the help of our Computer Science colleagues specializing in natural language technology, we plan on displaying a *summary note*, which is automatically created from the information contained within multiple manually created daily status notes. This gives the physician a single note containing all the pertinent information about a patient. However, since the note contains system created information, the physician may wish to see the parts of the original daily status notes from which this information was created. Using CAL, these parts of other daily status notes that were used as inputs to create the summary note can be rearranged contextually sensitive to the physical part of the summary note it supports. Overall, we believe that CAL has the potential to improve productivity in this environment.

6.2.3.2. Multi-Display Environments

We believe CAL has great potential in multi-display environments (MDEs). Using multiple displays affords multiple layouts to occur simultaneously. Since users often prefer to use separate displays for separate tasks, CAL could potentially behave differently depending on the display, allowing users to conduct different types of layouts depending on the content, as well as the display.

6.2.3.3. Alternative Layout Schemes

In our CAL implementation, we used a single layout scheme: the horizontal array layout scheme. This affords a quick visual context switch between rearranged windows due to a side-by-side placement. We also considered alternative layout schemes, such as a radial layout scheme. Using a radial layout, important content would be placed equidistant from the anchor area (or the center of the screen) in different directions, similar to the way a pie-menu or marking menus might work. However, the specific task for which our CAL implementation was designed was to visually compare similar textual contents across multiple windows, which by using a horizontal linear array scheme could provide an easier visual context switch between the windows being compared.

Google Search presents search results using a vertical linear array to present snippets of web pages matching a particular search query. The vertical layout facilitates a quick up-and-down scan of the results to find a particular item. A possible enhancement to the interaction model of the Patients Notes application is to first present snippets of all documents matching a particular search query using a vertical linear array, similar to Google Search results, allowing the doctor to initially select the documents that she

would like to compare. This could immediately be followed by our horizontal linear array layout of the selected documents.

6.3. Additional Content-Aware Techniques and Beyond

Throughout this dissertation, we have shown how three specific classical interaction techniques can be improved by allowing them to modify some display property of the content to which the technique is applied. This modification occurs across one or more particular domains. For example a modification in the *spatial* domain may rearrange the location of content, whereas a modification in the *temporal* domain may vary how long the content remains visible. The variance across these domains depends on the properties of that content. In other words, given a particular task, we have shown how successful content-aware techniques are able to present to the user *what is important to complete a particular task* in the appropriate physical place, at the appropriate moment in time, and with the appropriate visual appearance to complete a task sufficiently fast, accurately, and enjoyably.

What other content-aware techniques to be created? Can they arise from modifications of traditional techniques, or can they be created from scratch? Within what other domains can they be developed other than the spatial, temporal, or appearance domains? We believe the door has just opened for the development and evaluation of such techniques. Content-aware techniques can potentially aid users in 3D and mobile environments, public and private scenarios, audio interfaces, cursorless (or “eyes-free”) interfaces, collaborative environments, and beyond. Such environments require content modifications across additional display domains, such as audio and haptic.

We have enjoyed performing the research that inspired this dissertation topic, as well as building the tools that allowed us to complete this dissertation. We are excited to continue to innovate and look forward to using future content-aware interaction techniques, both modified from existing ones, and newly constructed by other user interface researchers.

Bibliography

ActualTools, *Actual Tools Corporation, Actual Transparent Windows*, www.actualtools.com/transparentwindow/.

Adobe, *Adobe Extensible Metadata Platform (XMP)*, http://www.adobe.com/products/xmp/.

Adobe, *Adobe Photoshop*, http://www.adobe.com/products/photoshop/.

Adobe, *Adobe Reader*, http://www.adobe.com/products/acrobat/readstep2.html.

Agrawala, M. and Stolte, C., "Rendering effective map routes: Improving usability through generalization," *Proc. SIGGRAPH 2001*.

Ahlberg, C. and Shneiderman, B., "The alphaslider: a compact and rapid selector," *Proc. CHI 1994*, pp. 365–371.

Akerstrom, R. B. and Todd, J. T., "The perception of stereoscopic transparency," *Perception and Psychophysics,* Vol. 44, pp. 421–432.

Apple, *Apple Computer / Mac OS X Terminal Window*, www.apple.com/macosx/features/unix.

Autodesk, *Autodesk 3ds Max 8.0*, http://usa.autodesk.com/adsk/servlet/index?id=5659302&siteID=123112.

Badros, G. J., Nichols, J. and Borning, A., "SCWM: An extensible constraint-enabled window manager," *Proc. FreeNIX 2001,* Boston, MA, June 2001.

Baudisch, P. and Gutwin, C., "Multiblending: Displaying overlapping windows simultaneously without the drawbacks of alpha blending," *Proc. CHI 2004,* Vienna, Austria, April 24–29, pp. 367–374.

Baudisch, P., Lee, B. and Hanna, L., "Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view," *Proc. AVI 2004,* Gallipoli, Italy, May 25–28, 2004, pp. 133–140.

Baudisch, P. and Rosenholtz, R., "Halo: A technique for visualizing off-screen locations," *Proc. CHI 2003,* Ft. Lauderdale, FL, April 5–10, pp. 481–488.

Beck, J. and Ivry, R., "On the role of figural organization in perceptual transparency," *Perception and Psychophysics,* Vol. 44 (6), 1988, pp. 585–594.

Beck, J., Prazdny, K. and Ivry, R., "The perception of transparency with achromatic colors," *Perception and Psychophysics,* Vol. 35, 1984, pp. 407–422.

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* (7), pp. 3–31.

Bell, B. and Feiner, S., "Dynamic space management for user interfaces," *Proc. UIST 2000,* San Diego, CA, November 5–8, pp. 239–248.

Beryl, *Beryl*, http://www.beryl-project.org/features.php.

Betaface, *Betaface Web Service*, http://www.betaface.com/.

Bier, E. A., Stone, M. C., Pier, K., Buxton, W. and DeRose, T., "Toolglass and magic lenses: The see-through interface," *Proc. SIGGRAPH '93,* Vol. 27, August, pp. 73–80.

Björk, S., "The scrollSearcher technique: Using scrollbars to explore search results," *Proc. INTERACT 2001,* Tokyo, Japan, July 9–13.

Blanch, R., Guiard, Y. and Beaudouin-Lafon, M., "Semantic pointing: Improving target acquisition with control-display ratio adaptation," *Proc. CHI 2004*, pp. 519–526.

Borning, A. and Duisberg, R., "Constraint-based tools for building user interfaces," *ACM Transactions on Graphics,* Vol. 5 (4), October 1986, pp. 345–374.

Brewster, S. A., Wright, P. C. and Edwards, A. D. N., "The design and dvaluation of an auditory-enhanced scrollbar," *Proc. CHI 1994,* Boston, Massachusetts, April 24–28, 1994, pp. 173–179.

Burtnyk, N., Khan, A., Fitzmaurice, G., Balakrishnan, R. and Kurtenbach, G., "StyleCam: Interactive stylized 3D navigation usingintegrated spatial and temporal controls," *Proc. UIST 2002*, pp. 101–110.

Card, S. K., Robertson, G. G. and Mackinlay, J. D., "The information visualizer, an information workspace," *Proc. CHI 1991,* New Orleans, LA, pp. 181–186.

Chang, B.-W. and Ungar, D., "Animation: From Cartoons to the User Interface," *Proc. UIST 1993*, November 3–5, 1993, pp. 45–55.

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.

Cohen, E. S., Berman, A. M., Biggers, M. R., Camaratta, J. C. and Kelly, K. M., "Automatic strategies in the Siemens RTL tiled window manager," *Proc. Computer Workstations 1988,* Santa Clara, CA, March 7–10, 1988, pp. 111–119.

Cohen, E. S., Smith, E. T. and Iverson, L. A., "Constraint-based tiled windows," *IEEE Computer Graphics and Applications,* Vol. 6 (5), May 1986, pp. 35–45.

Colby, G. and Scholl, L., "Transparency and blur as selective cues for complex visual information," *SPIE,* Vol. 1460 (Image Handling and Reproduction Systems Integration), March 1991, pp. 114–125.

Comber, T. and Maltby, J., "Evaluating usability of screen design with layout complexity," *Proc. OZCHI 1995*, 1995, pp. 175–178.

Comber, T. and Maltby, J., "Investigating layout complexity," *Proc. Graphics Interface 1988*, June 1988, pp. 192–197.

CYOA, *Choose Your Own Adventure Book Series*, www.cyoa.com.

Dietz, P. and Lehigh, D., "DiamondTouch: a multi-user touch technology," *Proc. UIST 2001,* Orlando, FL, November 2001, pp. 219–226.

Feiner, S. K., "A grid-based approach to automating display layout," *Proc. Graphics Interface 1988*, June 1988, pp. 192–197.

Fleming, R. W. and Bülthoff, H. H., "Low-level image cues in the perception of translucent materials," *ACM Transactions on Applied Perception,* Vol. 2 (3), July 2005, pp. 346–382.

Furnas, G. W., "Generalized fisheye views," *Proc. CHI 1986,* Boston, Massachusetts, U.S.A., April 13–17, pp. 16–23.

Galyean, T. A., "Guided navigation of virtual environments," *Proc. I3D 1995,* Monterey, CA, 1995, pp. 103–104.

Google, *Google Earth*, http://earth.google.com.

Google, *Google Local*, http://local.google.com.

Graf., W. H., "Constraint-based graphical layout of multimodal presentations," *World Scientific Series in Computer Science,* Vol. 36, 1992, pp. 365–385.

Graham, J., "The reader's helper: A personalized document reading environment," *Proc. CHI 1999,* Pittsburgh, PA, May 15–20, 1999, pp. 481–488.

Harrison, B. L., Buxton, W. and Zhai, S., *Graphical user interface with anti-interference outlines for enhanced variably-transparent applications*, US Patent 6317128, November 13, 2001.

Harrison, B. L., Ishii, H., Vicente, K. and Buxton, W., "Transparent layered user interfaces: An evaluation of display design space to enhance focused and divided attention," *Proc. CHI 1995*, pp. 317–324.

Harrison, B. L., Kurtenbach, G. and Vicente, K., "An experimental evaluation of transparent user interface tools and information content," *Proc. UIST 1995,* Pittsburgh, PA., pp. 81–90.

Hochberg, J. and Brooks, V., *Film cutting and visual momentum*, in J. W. Senders, D. F. Fisher and R. A. Monty, eds., *Eye Movements and the Higher Psychological Functions*, Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1978.

Hopkins, D., "Directional selection is easy as pie menus!," *login: The Usenix Association Newsletter,* Vol. 12 (5), September 1987.

Hudson, S. E. and Mohamed, S. P., "Interactive specification of flexible user interface displays," *ACM Transactions on Information Systems,* Vol. 8 (3), July 1990, pp. 269–288.

Hudson, S. E. and Smith, I., "Ultra-lightweight constraints," *Proc. UIST 1996*, 1996, pp. 147–155.

Igarashi, T. and Hinckley, K., "Speed-dependent automatic zooming for browsing large documents," *Proc. UIST 2000,* San Diego, CA, 2000, pp. 139–148.

Ishak, E. W. and Feiner, S. K., "Content-aware layout," *CHI 2007 Work-In-Progress (poster),* San Jose, CA.

Ishak, E. W. and Feiner, S. K., "Content-aware scrolling," *Proc. UIST 2006,* Montreux, Switzerland, October 15–18, 2006, pp. 155–158.

Ishak, E. W. and Feiner, S. K., "Free-space transparency: Exposing hidden content through unimportant screen space," *Conference Supplement, UIST 2003,* Vancouver, B.C., Canada, November 2–5, 2003, pp. 75–76.

Ishak, E. W. and Feiner, S. K., "Interacting with hidden content using content-aware free-space transparency," *Proc. UIST 2004,* Sante Fe, NM, October 24–27, 2004, pp. 189–192.

Jacobs, C., Li, W., Schrier, E., Bargeron, D. and Salesin, D., "Adaptive grid-based document layout," *ACM Transactions on Graphics*, July 2003, pp. 838–847.

Jeffries, R., Miller, J. R., Wharton, C. and Uyeda, K. M., "User interface evaluation in the real world: A comparison of four techniques," *Proc. CHI 1991*, 1991, pp. 119–124.

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.

Kochhar, S., Marks, J. and Friedell, M., "Interaction paradigms for human-computer cooperation in graphical-object modeling," *Proc. Graphics Interface 1991*, June 1991, pp. 180–191.

Kurtenbach, G. and Buxton, W., "Issues in combining marking and direct manipulation techniques," *Proc. UIST 1991,* South Carolina, pp. 137–144.

Laakso, S. A., Laakso, K.-P. and Saura, A. J., "Improved scroll bars," *Proc. CHI 2000*, April 1–6, 2000, pp. 97–98.

Leykin, A. and Tuceryan, M., "Automatic determination of text readability over textured backgrounds for augmented reality systems," *Proc. ISMAR 2004*, pp. 224–230.

Lieberman, H., "Powers of ten thousand: Navigating in large information spaces," *Proc. UIST 1994,* Marina del Rey, CA., November 2–4, 1994, pp. 15–16.

Lok, S. and Feiner, S., "A survey of automated layout techniques for information presentations," *Proc. SmartGraphics Symposium 2001*, March 2001, pp. 61–68.

Lok, S., Feiner, S. and Ngai, G., "Visual balance for automated layout," *Proc. IUI 2004*, January 2004, pp. 101–108.

MacKenzie, I. S. and Riddersma, S., "Effects of output display and control-display gain on human performance in interactive systems," *Behaviour and Information Technology,* Vol. 13, 1994, pp. 328–337.

Mackinlay, J. D. and Heer, J., "Wideband displays: mitigating multiple monitor seams," *Proc. CHI 2004,* Vienna, Austria, pp. 1521–1524.

Masui, T., "LensBar - visualization for browsing and filtering large lists of data," *Proc. InfoVis 1998*, 1998, pp. 113–120.

Masui, T., Kashiwagi, K. and Borden, G. R., "Elastic graphical interfaces for precise data manipulation," *Proc. CHI '95 Conference Companion*, 1995, pp. 143–144.

Metelli, F., "The perception of transparency," *Scientific American,* Vol. 230 (4), 1974, pp. 90–98.

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.

Myers, B. A., "The garnet user interface development environment," *Proc. CHI 1994 Conference Companion,* Boston, MA, 1994, pp. 25–26.

Myers, B. A., McDaniel, R. G. and Kosbie, D. S., "Marquise: Creating complete user interfaces by demonstration," *Proc. INTERCHI 1993,* Amsterdam, The Netherlands, pp. 293–300.

Norman, K. L., Weldon, L. J. and Shneiderman, B., "Cognitive layouts of windows and multiple screens for user interfaces," *International journal of man-machine studies,* Vol. 25, pp. 229–248.

NVIDIA, *NVIDIA nView Technology, Advanced Window and Menu Effects*, www.nvidia.com/object/LO_20020201_6041.html.

Paley, W. B., "Designing better transparent overlays by applying illustration techniques and vision findings," *Proc. Conference Supplement, UIST 2003,* Vancouver, B.C., Canada, November 2–5, 2003, pp. 85–86.

PDFBox, *PDFBox - Java PDF Library*, http://www.pdfbox.org.

Porter, T. and Duff, T., "Compositing digital images," *Computer Graphics,* Vol. 18 (3), July 1984, pp. 253–259.

Purvis, L., "A genetic algorithm approach to automated custom document assembly," *Proc. Workshop on Intelligent Sytems Design and Application,* Atlanta Georgia, 2002, pp. 131–136.

Rao, R. and Card, S., "The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information," *Proc. CHI 1994,* Boston, MA, pp. 318–322.

Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinkley, K., Risden, K., Thiel, D. and Gorokhovsky, V., "The task gallery: A 3D window manager," *Proc. CHI 2000,* New York, NY, November 2000, pp. 494–501.

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, pp. 202.

Sears, A., "Layout appropriateness: A metric for evaluating user interface widget layout," *IEEE Transactions on Software Engineering,* Vol. 19 (7), July 1993, pp. 707–719.

Sears, A. and Lund, A. M., "Creating effective user interfaces," *IEEE Software,* Vol. 14 (4), July/August 1997, pp. 21–24.

Singh, M. and Anderson, B. L., "Toward a perceptual theory of transparency," *Psychological Review,* Vol. 109 (3), 2002, pp. 492–519.

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.

Smith, R. and Taivalsaari, A., "Generalized and stationary scrolling," *Proc. UIST 1999,* Asheville, NC, pp. 1–9.

Stille, S., Minocha, S. and Ernst, R., "A^2DL-an adaptive automatic display layout system," *Proc. HICS 1996*, pp. 243–250.

Stroop, J. R., "Studies of interference in serial verbal reactions," *Journal of Experimental Psychology,* Vol. 18, pp. 643–662.

SunMicrosystems, *Java 2 Platform Standard Edition 5.0*, http://java.sun.com/j2se/1.5.0/.

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.

Tullis, T. S., "A computer-based tool for evaluating alphanumeric displays," *Proc. IFIP INTERACT 1984*, 1984, pp. 719–723.

Vander Zanden, B. and Myers, B. A., "Automatic, look-and-feel independent dialog creation for graphical user interfaces," *Proc. CHI 1990*, pp. 27–34.

Weitzman, L. and Wittenburg, K., "Automatic presentation of multimedia documents using relational grammars," *Proc. Multimedia 1994,* New York, October 1994, pp. 443–452.

Woods, D. D., "Visual momentum: a concept to improve the cognitive coupling of person and computer," *International Journal of Man-Machine Studies,* Vol. 21, pp. 229–244.

Zeleznik, R. and Forsberg, A., "UniCam - 2D gestural camera controls for 3D environments," *Proc. I3D 1999,* Atlanta, GA, pp. 169–173.

Zeleznik, R., LaViola, J., Acevedo, F. and Keefe, D., "Pop through button devices for VE navigation and interaction," *Proc. IEEE Virtual Reality 2002*, pp. 127–134.

Zeleznik, R., Miller, T. and Forsberg, A., "Pop through mouse button interactions," *Proc. UIST 2001,* Orlando, FL, November 2001, pp. 31–40.

Zellweger, P. T., "Scripted documents: A hypermedia path mechanism," *Proc. Hypertext 1989*, November 1989.

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.

Zhou, M. X. and Ma, S., "Toward applying machine learning to design rule acquisition for automated graphics generation," *Proc. 2000 AAAI Smart Graphics,* Stanford, CA, March 20–22, 1999, pp. 16–23.