Interacting with Hidden Content Using Content-Aware Free-Space Transparency

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

ABSTRACT
We present *content-aware free-space transparency*, an approach to viewing and manipulating the otherwise hidden content of obscured windows through unimportant regions of overlapping windows. Traditional approaches to interacting with otherwise obscured content in a window system render an entire window uniformly transparent. In contrast, content-aware free-space transparency uses opaque-to-transparent gradients and image-processing filters to minimize the interference from overlapping material, based on properties of that material. By increasing the amount of simultaneously visible content and allowing basic interaction with otherwise obscured content, without modifying window geometry, we believe that free-space transparency has the potential to improve user productivity.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical user interfaces (GUI), Interaction styles, Screen design, Windowing systems*

General Terms: Human Factors, Design

Additional Keywords and Phrases: Transparency, screen space, interaction techniques, content disambiguation, space management, pie menu

1. INTRODUCTION

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, or bring the obscured window to the top. To avoid this problem, users sometimes resort to using window managers that automatically cascade or tile each window without regard to its content [5]. However, this is undesirable when certain windows need to display more content than others or require particular aspect ratios. *Free-space transparency* [11] 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. In this paper, we extend this work to introduce *content-aware free-space transparency* (FST), which takes into consideration various characteristics of the obscured and overlaid content and applies image-processing filters and gradients to further reduce content ambiguity [12]. This guarantees that the important content of overlaid windows will be readable at all times, while simultaneously exposing hidden content beneath the unimportant regions.

UIST '04, October 24-27, 2004, Santa Fe, New Mexico, USA.

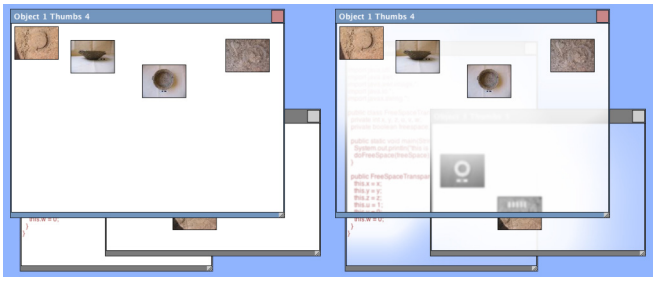

Figure 1: Window rendered (left) *without* free-space transparency, and (right) *with* content-aware free-space transparency, exposing blurred hidden content underneath.

We also present a set of interaction techniques that utilize the benefits of FST. 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* allows a user to temporarily transform a filtered portion of obscured content to its original unfiltered form, which clarifies 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 the *mouse-over pie menu*.

2. RELATED WORK

To increase the amount of simultaneously visible content, some systems have tried rendering the entire obscuring window semi-transparently [1,2,7,13,14]. 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. Multi-blending [4] addresses this issue when interacting with an application's palette windows in visual workspaces. 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. FST takes a different approach by selectively rendering every important region opaque. By preventing obscured content from showing through those important regions, FST increases the legibility of even unfamiliar overlapping information.

In Macintosh OS X [3], 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

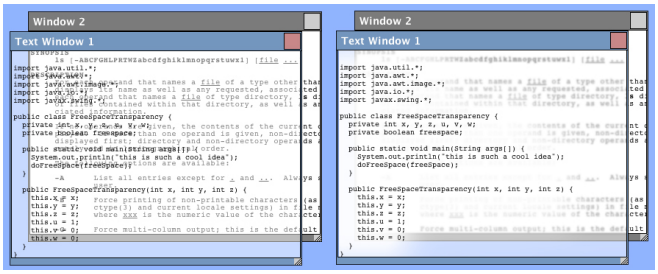

Figure 2: Window rendered (left) with opaque text on transparent background (alpha = 0.5), and (right) using FST, where background pixels of important regions are rendered opaque, producing more legible overlaid content.

background. Although this approach may produce more legible overlaid content than does uniform semi-transparency, content 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 are from content of the same type as the overlaid region (e.g., text, in the case of the Macintosh terminal window). FST generalizes the Macintosh approach by rendering the content pixels, as well as those in the background on which the content lies, opaque, making the content more legible, as shown in Figure 2.

3. FST OVERVIEW

FST allows a window's application to inform the rendering engine of the unimportant regions, as we have done in the implementation described in Section 4. It evaluates and compares content characteristics, such as type (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 [4] 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.

Although many approaches allow visualization of overlaid content, they rarely allow interaction with the obscured content (with the Task Gallery [15] as a notable exception). One reason may be that with uniform semi-transparency, disambiguating obscuring content from hidden content can be hard, making user interaction with 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 FST 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. This facilitates unambiguous interaction with all visible window content, even if it is visible through one or more unimportant window regions.

4. FST IMPLEMENTATION

A key issue in implementing FST, 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 75% transparency value works well. We have also found that rendering important regions with a 10% transparency (90% opacity) value allows for some visualization beneath overlaid content, with little risk of content ambiguity; this supports earlier evaluations of usable and efficient transparent user interfaces, such as in the Stroop Experiment [9,10].

Since users often interact with window decoration (title bar, menus, border, scroll bars), we consider this important content. Therefore, FST does not affect the pixels that make up these regions, and consequently, considers only the window body when classifying important and unimportant regions. In the case of opaque windows, keeping the window decoration opaque allows users to disambiguate window boundaries more easily, as in the Macintosh terminal window mentioned above. Inherently transparent windows maintain their transparent window decoration, while the gradient between important and unimportant regions transitions from the window's inherent alpha value to a more transparent value.

4.1. Important vs. Unimportant Regions

We have considered several approaches to determining important and unimportant regions in an FST window. In one approach, the rendering engine can classify window regions containing only a particular background color or texture as unimportant. Alternatively, the user could 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 white space (i.e., ones devoid of text, icons, and images) can also be automatically classified as unimportant, as in our implementation. However, since some applications utilize white 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 simply specify a set of geometric bounds with characterizations of their contents, as in our testbed application.

4.2. Classifying Content

An FST 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 overlapped data.

4.2.1. 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 our implementation, we evaluate overlapping window contents to apply the most suitable filters. For example, in comparing overlaid versus obscured contents, if the types are both text or both image, we find that

using a Gaussian blur (radius=3) is sufficient for content disambiguation. When both types are icons (or thumbnail images), we find that using desaturation followed by a Gaussian blur (radius=5) is less ambiguous than the Gaussian blur alone. This is because icons (or thumbnails) can have arbitrary shapes and locations within both the overlaid and obscured windows, sometimes making it difficult to associate the icons with the window in which they reside. Text and images, on the other hand, tend to be more structured and less freeform in their locations and dimensions. Figure 3 illustrates the conditions under which we apply different image-processing filters to hidden content.

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

Figure 3: Table showing 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. NC signifies "not considered."

4.2.2. Content-Dependent Gradients. FST considers several gradients varying in slope and distance and uses the appropriate one based on characteristics of the overlapping content. We tried both linear and exponential gradients, and found that an exponential gradient works well. Using a 2D isosurfaces equation, for each pixel (px, py), a scalar value s is computed as a weighted sum (ssum) of N intermediate scalars si, each computed relative to one of N objects with location (Xi, Yi) in the window, added to the weighted scalar maximum, smax, for that pixel. Adding a weighted smax allows a pixel closer to an object of content to have a higher overall scalar value than another pixel with an equal ssum value. We tested many different weights, and values of 0.3 and 0.7 for smax and ssum, respectively, 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:

$$s = (0.3)s_{\max} + (0.7)s_{\text{sum}},$$

where $s_{\text{sum}} = \sum_{i=1}^{N} \frac{s_i}{d}$ and $s_i = e^{-5\left((p_x - X_i)^2 + (p_y - Y_i)^2\right)}$.

We experimented with exponent values, 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.

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. 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. Users can specify whether to render the windows using content-aware FST, 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. Running on an Apple Powermac G5 desktop (dual 2GHz, 1GB RAM), basic moving and resizing of windows operate at about 15–20 fps without the use of filters, but less than 1 fps with filters. We foresee these numbers improving with faster hardware and the use of hardware-accelerated image processing.

6. INTERACTION WITH FST

With content-aware FST, 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.

6.1. 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 allows a user to use pressure to interact with an obscured window [16,17]. In our implementation on a MERL DiamondTouch table [8], the user can apply pressure to the unimportant regions of an obscuring window to allow a hidden window directly underneath the obscuring window to "pop through" and become focused and fully unobstructed. We are currently expanding this approach to detect several pressure thresholds, allowing windows at various layers, proportional to the pressure applied, to "pop through" the topmost window. 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.

6.2. Focus Filter

The use of various image-processing filters for content disambiguation may, at times, make 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 4, acting as a "magic lens" [6]. 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. The focus filter can be extended to use a touchpad and to correlate higher pressure with larger large radii. 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,

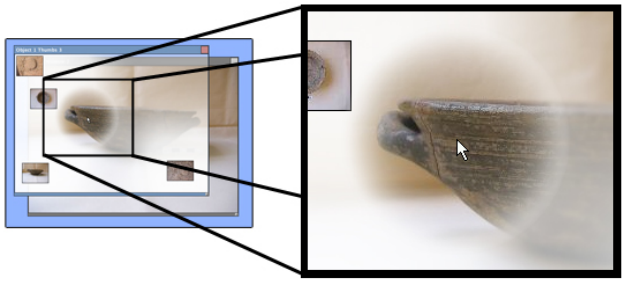

Figure 4: The focus filter allows a user to temporarily restore an image-processed portion of obscured content to its original form for increased legibility. Here, using the mouse, the user temporarily focuses on part of a blurred, obscured image of a pot.

dragging the right button moves the focus filter appropriately. Our implementation of the focus filter operates at about 10 fps.

6.3. 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 windows directly underneath the top-most window; however, they make it cumbersome to interact with any other window when using a standard two-button mouse. We provide a mouse-over pie menu to allow a user to determine with which window 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 the 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 lie beneath that selected pixel. A user can then left mouse click on the thumbnail representation of the window with which she wishes to interact.

7. ADDITIONAL BENEFITS OF FST

FST 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 [3], or to content not contained within the current bounds of the window (e.g., when the user must scroll to visualize content). Additionally, with the use of shorter, less fluid gradients, a spatial grouping of objects can be visually reinforced through the isosurface property of FST. Finally, by knowing which regions of windows are unimportant, one could use space management [5] to place information not only in totally free screen space, but also in unimportant window regions.

8. DISCUSSION AND FUTURE WORK

Using our demo application in informal studies, several users, commenting about FST, stated they thought they would benefit from visualizing and interacting with hidden content if working with limited screen space. Compared to uniform semi-transparency, they found that FST decreased the ambiguity of content within its containing window. Some did not see the benefit in rendering small unimportant regions transparent, saying it was somewhat distracting, and would be advantageous only if large unimportant regions were treated. Additionally, some users did not see the need to visualize or interact with content that was arbitrarily deep, stating that visualization of two to three layers of window content was sufficient. We will be conducting user studies to determine whether FST allows users to disambiguate content faster, and whether it improves their ability to perform certain window manipulation tasks. We also plan on experimenting with different ways to identify important regions, such as detecting high mouse activity, or monitoring a user's gaze through eye tracking. Finally, we will explore additional ways to classify content, by considering additional material properties, as well as by using vision-based techniques.

ACKNOWLEDGMENTS

We wish to thank Lawrence Wang for his work on DiamondTouch code. This research is funded in part by NSF Grant IIS-0121239, Office of Naval Research Contract N00014-04-1-0005, and gifts from Microsoft Research and Mitsubishi Electric Research Laboratory.

VIDEO

The techniques presented in this paper can be previewed in a digital video available for download from www.cs.columbia.edu/graphics/projects/FST.

REFERENCES

[1] ActualTools Corporation. *Actual Transparent Windows*, www.actualtools.com/transparentwindows/

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

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

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

[5] Bell, B., and Feiner, S. "Dynamic space management for user interfaces." *Proc. UIST 2000*, New York, NY, Nov. 2000, 239–248.

[6] Bier, E., Stone, M., Pier, K., Buxton, W., and DeRose, T. "Toolglass and magic lenses: The see through interface." *Proc. SIGGRAPH 1993*, Anaheim, CA, Aug. 1993, 73–80.

[7] Colby, G., and Scholl, L. "Transparency and blur as selective cues for complex visual information," *Proc. SPIE Vol. 1460, Image Handling and Reproduction Systems Integration*, Mar. 1991, 114-125.

[8] Dietz, P. and Lehigh, D. "DiamondTouch: A multi-user touch technology." *Proc. UIST 2001*, Orlando, FL, Nov. 2001, 219–226.

[9] 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*, 317–324.

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

[11] Ishak, E. and Feiner, S. "Free-space transparency: Exposing hidden content through unimportant screen space" [poster], *Conf. Supplement, UIST 2003*, Vancouver, B.C., Canada, Nov. 2–5, 2003, 75–76.

[12] Kosara, R., Miksch, S., and Hauser, H. "Semantic depth of field." *Proc. IEEE InfoVis 2001*, San Diego, CA, USA, Oct. 2001, 97–104.

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

[14] NVIDIA nView Technology, Advanced Window and Menu Effects, www.nvidia.com/object/LO_20020201_6041.htm

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

[16] Zeleznik, R., Miller, T., and Forsberg, A. "Pop through mouse button interactions." *Proc. UIST 2001*, Orlando, FL, Nov., 2001, 195–196.

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