Free-Space Transparency: Exposing Hidden Content Through Unimportant Screen Space

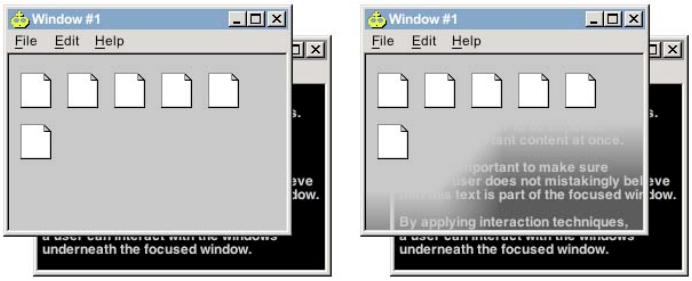
Edward W. Ishak *Steven K. Feiner*
Columbia University, Department of Computer Science
New York, NY 10027
Email: {ishak, feiner}@cs.columbia.edu

ABSTRACT
We present *free-space transparency*, a method that renders unimportant regions of a window as transparent. This allows users to view screen content that lies beneath these regions. By maximizing the amount of simultaneously visible content, while keeping window sizes and locations constant, users can be made more productive with minimal window layout manipulation.

KEYWORDS: Transparency, screen space.

INTRODUCTION
Computer users often move, resize, or "alt-tab" between windows to expose desired content that is currently obstructed by other windows. To maximize visibility, users sometimes resort to using layout managers that cascade or tile windows automatically. This potentially moves or resizes each window without regard to its content, which is not desirable if certain windows need to display more content than others or require a specific aspect ratio. Some have addressed these problems by rendering the entire window and its content as transparent [1] or with manually controlled "magic lenses" to view obscured content [2]. In Macintosh OS X, a text-only terminal window can have an opaque foreground color and a transparent background color [3]. We present *free-space transparency,* which generalizes this last approach to support windows with arbitrary contents that render important regions opaquely and unimportant regions with a smooth transition from opaque to transparent, as shown in Figure 1. This guarantees that important content will be readable at all times, while simultaneously exposing hidden content underneath the unimportant regions.

IMPORTANT VS. UNIMPORTANT REGIONS
In implementing free-space transparency, it is necessary to determine which regions of a window are important and which are unimportant. We have considered several approaches. The rendering engine can classify window regions containing only a particular background color or texture as unimportant. Alternatively, the window's application can inform the rendering engine of the unimportant regions, as in our implementation. The user might also identify unimportant regions manually with a mouse, or automatically by using an eye tracker to detect window regions that do not attract the user's attention.



(a) (b)

Figure 1: window rendered (a) **without** free-space transparency, and (b) **with** free-space transparency exposing content underneath.

TILE COLORING
Our implementation of free-space transparency uses a tile-coloring process to determine the opacity value of each pixel. Every window is divided into a uniform grid of tiles. Then, each tile is classified into one of three categories (as described below). It is drawn either opaque, transparent, or with a linear gradient between opacity and transparency. Rather than classifying each pixel individually, we classify a tile at a time to optimize the classification process.

Since users frequently interact with window decoration (title bar, menus, border), we consider this important content and do not include it as part of any tile. For this reason, pixels that make up these regions are rendered opaque at all times. Therefore, tiles make up only the window body. Rendering the window decoration opaque allows the user to disambiguate window boundaries more easily, as in the Macintosh terminal window.

Classifying Tiles
Determining the category into which each tile falls is a three-stage process. First, tiles containing "important" content are painted opaque. Then, all remaining tiles adjacent to opaque tiles are painted with a linear opaque-

transparent gradient. This provides a smooth transition between entirely opaque and entirely transparent tiles. Finally, all remaining tiles are rendered transparent. Note that no tile is painted 100% transparent since it would completely expose the pixels underneath. This is undesirable since this content could be mistakenly perceived as part of the overlaid window. We have found that an 85% transparency value works quite well.

Rendering Gradient Tiles

It is important that the opaque-to-transparent transition be smooth and appealing to the user's eyes. An abrupt opaque-transparent boundary line could imply a separation of the opaque and transparent sections, leading the user to believe that one window is actually divided into multiple objects. Currently, the tile-coloring process gracefully transitions the opaque to the transparent tiles while also allowing for holes and concavities in the opaque tile regions. A gradient tile is rendered by setting a transparency value for each of its four vertices and using interpolation shading (a *GradientPaint* in our Java2D implementation). Each vertex adjacent to an important tile is given a transparency of zero (fully opaque), and all remaining vertices are given the maximum transparency value.

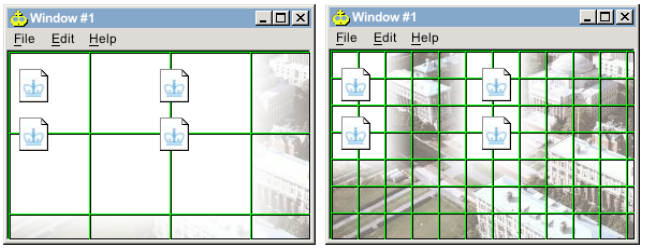
We tested the tile-coloring algorithm with different grid size tiles. Since exactly one tile is used for rendering the gradient between opaque and transparent regions, smaller tiles produce a smaller gradient that renders a sharper opaque-transparent boundary, whereas larger tiles produce a more gradual and unobtrusive transition. However, since larger tiles produce larger gradients, they consequently produce less transparent regions, therefore exposing less content underneath (see Figure 2).

CURRENT AND FUTURE WORK

We have a developed an application that allows users to instantiate within it windows containing icons, text, and images. Users can drag icons from one window to another, thus dynamically creating and destroying unimportant screen space. Currently, every tile is the exact same size, independent of window content. We are considering combining tiles of varying sizes and dimensions to more accurately represent important content regions. Instead of linear gradients spanning one tile length, we are considering nonlinear gradients spanning multiple tiles to produce a more gradual transition between opaque and transparent regions. Additionally, we will investigate using Bezier curves to help define the opaque-transparent boundaries.

Content Disambiguation

A potential problem is the user's inability to disambiguate the content of an overlaid window from content underneath exposed by free-space transparency. Several techniques could help make this clearer. One possibility is to render the overlaid content in full color, and content underneath in grayscale or in a specific tint. Content underneath could also be rendered slightly blurred to give the perception that it is being viewed through a semi-transparent medium [4]. Although these approaches may not allow the user to view details of the content underneath, they allow the user to have an adequate visual understanding of this material while easily disambiguating it from overlaid material.



(a) (b)

Figure 2: Free-space transparency windows. Grid lines have been added to show tiling. (a) Large gradient tiles expose minimal content underneath, but with smoother transitions. (b) Small gradient tiles expose more content, but with sharper transitions.

Interaction Techniques

Various interaction techniques can be applied in conjunction with free-space transparency to make maximum use of its capabilities. We are currently implementing techniques that allow the user to interact with obstructed content through the transparent regions of the overlaid content without relocating or resizing any window. This would make free-space transparency much more useful and desirable, allowing the user to interact with any window from fully viewable to fully obstructed.

ACKNOWLEDGMENTS

This research was funded in part by Office of Naval Research Contracts N00014-99-1-0249 and N00014-99-1-0394, NSF Grant IIS-00-82961 and IIS-01-21239, and gifts from Intel and Microsoft Research.

REFERENCES

1. NVIDIA nView Technology, Advanced Window and Menu Effects, http://www.nvidia.com/docs/lo/1457/SUPP/Q4_nView.jb2_final.pdf.
2. Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and Magic Lenses: The See Through Interface. Proceedings of Siggraph 93 (Anaheim, August), *Computer Graphics Annual Conference Series*, ACM, 1993, pages 73-80.
3. Apple Computer, Mac OS X Terminal Window, http://www.apple.com/macosx/jaguar/unix.html.
4. Robert Kosara, Silvia Miksch, and Helwig Hauser. *Semantic Depth of Field*. In IEEE Symposium on Information Visualization (2001), San Diego, CA, USA, October 22-23 2001. pages 97-104.