Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending

Patrick Baudisch
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
baudisch@microsoft.com

Carl Gutwin
Computer Science, University of Saskatchewan
57 Campus Drive, Saskatoon, SK S7N 5A9, Canada
carl.gutwin@usask.ca

ABSTRACT

Alpha blending allows the simultaneous display of overlapping windows—such as palette windows in visual workspaces. Although alpha blending has been used in some applications, such as games, it has not been widely adopted. One reason for the limited acceptance is that in many scenarios, alpha blending compromises the readability of content. We introduce a new blending mechanism called *multiblending* that uses a vector of blending weights, one for each class of features, rather than a single transparency value. Multiblending can in most cases be automatically optimized to preserve the most relevant features of both the palette and the background window. We present the results of a user study in which multiblended palettes provided higher recognizability of *both* the background and the palette than the best participating version of alpha blending.

Categories & Subject Descriptors: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General Terms: Human Factors, Design.

Keywords: Alpha blending, semitransparency, windows.

INTRODUCTION

Overlapping windows and 2½-D interfaces were developed to let applications use limited screen space multiple times. However, since overlapping windows occlude each other, users have to switch back and forth between windows in order to access the different tools and information. This switching becomes especially cumbersome when the overlapping windows belong to the same application. Many visual applications make tools and controls available in permanently visible interface components, such as tool palettes. Palettes can be positioned on top of the workspace to allow tools to be closer to the area where the work is being done. However, this causes the palettes to occlude an area of the underlying data in the workspace (e.g. Figure 1a). Palettes are a tradeoff of 2½D interfaces, between the availability of objects in the foreground and background layers. The more available (i.e. larger or closer) the tools in a palette, the more difficult it is to see and use the visual data in the workspace below. In the image processing application of Figure 1a, tasks such as selecting a larger object, checking whether retouched elements still match the overall lighting situation, or getting an overview of the picture will require the user to hide palettes before proceeding.

CHI 2004, April 24–29, 2004, Vienna, Austria.

Figure 1: (a) An image processing application with a selection of frequently used tool palettes. (b) Alpha blended palettes give a better idea of the periphery, but color, brightness, and contrast of the semi-occluded areas are still affected.

In order to deal with the many palette windows that come with many professional applications, such as CAD or software development environments, users often use additional screen space such as a second monitor [7]. The drawback of this solution is that palettes are further away, and acquiring them thus takes more time. Also, adding screen space is not an option for users on mobile computing devices.

Semitransparent palettes have been proposed and implemented as a solution to the problem of occlusion (e.g. [9]). Semitransparency, using a technique called alpha blending [15], allows two windows to be displayed on the same piece of screen space. Semitransparent palettes show the contents of both windows, reducing the need for switching between overlapping windows.

However, semitransparency still seems far from reaching its potential. While it has been adopted into some gaming applications such as EverQuest [8], and is available on a window level in some operating systems (e.g., Linux and MacOS X), it has not been implemented as part of any multi-window/multi-palette application even though it seems to be an obvious answer to an ongoing problem. One reason for the limited acceptance seems to be that for many scenarios, alpha blending affects the readability of window contents too much (Figure 1b).

In this paper, we argue that we can make window blending applicable to a wider range of applications by extending alpha blending to selectively preserve image features. The new technique, called *multiblending*, blends the individual color and texture features of palette and window separately, using a range of image processing techniques. This allows

us to assign higher visibility to the features most relevant to the user's task at hand. Using multiblending, we can create blended palettes that better preserve the visibility of both the background and foreground windows (Figure 2).

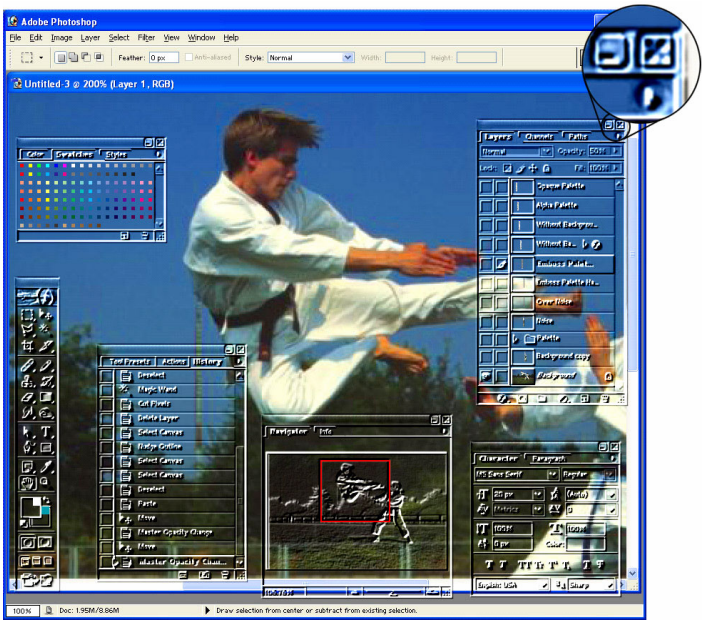

Figure 2: This multiblending style ("glass palettes") allows higher fidelity in the background image, and still shows relevant features of the palettes. (More close-ups: Figure 3, Figure 7, and Figure 8).

In this paper, we introduce multiblending and report on user studies showing that multiblending improves readability compared to alpha blending. First, we review related work on the problem of window space in 2½D interfaces.

RELATED WORK: OVERLAP IN 2½D INTERFACES

In applications that use palettes and other floating windows, several techniques have been proposed to help manage the tradeoff between the visibility of palette and background. Although some research has been done in techniques for placing palettes into areas that result in the least amount of conflict [4], the majority of work looks at various methods for *diminishing* palettes in situ.

These techniques reduce either the palette's size or opacity. The techniques that reduce palette size either shrink the entire palette (e.g., *expanding targets* [11]), eliminate everything but the palette's title bar (e.g., *window shades* [13], a particular instance of semantic zooming [14]), or hide the palette altogether (e.g., Adobe® Photoshop® [2]).

The techniques that reduce opacity either remove some pixels entirely or all pixels by a certain amount. *Overlays* make part of the palette's surface transparent. They have been used for adding titles to video (i.e. chroma-keying), in head-up displays to show flight data (e.g. [16]), in games to show maps and messages (e.g. *Diablo II*), for workspace tools such as magic lenses [5], and in workspaces to show global overviews [6]. Alpha blending [15] also known as semitransparency) blends palette windows with the background by computing a weighted sum of pixel colors. This can be done on a per-pixel basis, as defined by the so-called alpha map. This allows content from both palette and background window to remain visible to some degree. Alpha blending has been used in a number of scenarios, including popup menus [9], tool glasses and magic lenses [5], and multiple representations of the same image [10]. Several operating systems including Linux/KDE and Mac OS 10 also allow for semitransparent windows, and add-ons for window managers are available that manage the transparency of application windows (e.g., [1]).

Since all diminishing techniques affect the readability of the palette, they typically provide a 'restored' representation in addition to the diminished version that is easier to read and manipulate. The individual techniques switch between representations either manually or automatically. Manual techniques include double-clicking the diminished palette [13], selecting a 'restore' operation from a menu, or hitting a keyboard shortcut [2]. The automatic mechanisms proposed so far are based on the proximity of the mouse cursor [8]. Moving the mouse cursor towards the palette restores the palette either gradually or abruptly, as a threshold distance is passed. All these techniques fully restore palettes when the mouse reaches the palette; the boundaries of the palette in motor space thereby remain unchanged, aiding acquisition.

Of all the diminishing techniques, alpha blending has been paid the most attention, and several results have appeared that consider the usability of semi-transparent tool windows. Harrison and colleagues [9] found that 50% transparent palettes can greatly improve workspace visibility without degrading icon selection performance. Other studies have shown that users can perform targeting tasks well with 25% or even 10% opacity, depending on the complexity of the workspace data [8].

Unlike techniques that reduce the size of the palette, alpha blending has the benefit that all palette content remains on the screen—although in diminished form. Alpha-blending thus potentially allows users to monitor changing palette content, such as a display of the cursor coordinates or warning signs. The drawback of alpha blending is that it causes the contents of palette and background to interfere with each other. This interference will often make both palette and background hard to read and makes it hard to say whether two features belong to the same window. We will discuss these issues in more detail as we introduce our proposed blending technique in the following section.

MULTIBLENDING

To address the drawbacks of alpha blending, we have developed a new technique called *multiblending* that allows the use of different blending functions for different visual features. Multiblending is based on the observation that for a particular task, users typically need only a subset of the visual features in the involved windows or palettes. Multiblending modifies the blending process to enable it to preserve more of these relevant features—if necessary at the expense of reducing other, less relevant features. To explain this, we first describe what we mean by "features."

Feature selection is based on human vision

Our current approach to multiblending is based on four classes of features: three color classes and one class of textural features. We focus on 'apparent' features—that is, those that are particularly easily perceived by humans.

The three classes of color feature used by multiblending are the ones encoded by the CIE Lab color model [12]. CIE Lab is a perception-oriented color model that represents color as luminance, red-green and blue-yellow difference. We picked this model because it largely matches the color model the human visual apparatus uses when sending color information to the brain.

Multiblending uses only a single textural feature class—the presence of high-frequency data, such as edges or other areas of high contrast in the image. The human visual apparatus has specialized cells that perceive contrast (e.g. *receptive fields* [16]).

When analyzing Figure 1 with respect to colors and high frequencies, we see that the screen is not as crowded as it seems. Most parts of the palettes are grayscale, i.e., contain no red/green and blue/yellow information; the photograph, on the other hand, contains large areas that contain no visible edges. It is these "unpopulated" areas that we exploit in our approach.

The limits of alpha blending

Having defined this set of features, we can now better express the limitations of alpha blending. In terms of color, alpha blending computes the output image as a weighted sum of the two source windows. This makes the red, green, and blue channel, as well as the channels of the CIE Lab representation, into weighted sums—each weighted by the same ratio (the blending parameter *alpha* of that pixel). In other words, all color channels get diluted by the respective contribution of the other window. In terms of texture, alpha blending is subject to interference effects, since image frequencies in the two images will reinforce each other in some cases, while they will cancel out in others, depending on how the two images are aligned. If a feature is sensitive to that dilution or interference, it disappears. If that feature would have been relevant to the user's task, alpha blending becomes unsuitable.

A further drawback that affects all features is that alpha blending introduces visual ambiguity. Looking at a blended image, it is hard to say which layer a specific feature belongs to (see, for example, Figure 6a). Also, the relationships between features become obscured: it can be difficult to determine whether two observed features belong to the same window and thus are semantically related.

Overcoming those limits with multiblending

In order to prevent relevant features from disappearing, multiblending assigns each class of features an individual weight, instead of using the single global weight used by alpha blending.

Even though theoretically, multiblending allows weights to range from zero to one, it will often be beneficial to limit the technique to weights that are either zero or one. In other words, one of the two windows will give up a feature class entirely in order to allow the respective feature in the other window to stand out. A palette window may, for example, be desaturated in order to not affect color in the background image. Limiting multiblending to Boolean weights eliminates visual ambiguity; now each type of feature can stem from only one of the two windows, and all features of the same type belong to the same window, which clarifies their grouping. Interference that is caused by the same features stemming from different windows is eliminated.

Converting a regular opaque palette into a multiblending palette thus mainly requires deciding for each of the four feature classes which windows to take these features from. The actual computation is then straightforward. Given that color is represented using the CIE Lab model, computing output color only requires picking each channel from either palette or window and reassembling them. The removal of texture information is accomplished with filters that blur or sharpen images. These filters generally do not interact with the *overall* color of a window; whenever making one pixel darker, these filters make some nearby pixel lighter.

MAKING MULTIBLENDED PALETTES

In this section, we detail the steps used to create a type of multiblended palette introduced earlier as the glass palette, using Boolean weights for each feature class. These Boolean values are provided by the designer, an effort roughly equivalent to the input required for alpha blending, where only a single, but therefore real-valued opacity value needs to be provided. We will show methods for manual improvement in a later section. Note that many of the following examples require seeing a color version of this paper.

The scenario for our walkthrough is an image editing/retouching session in Adobe Photoshop [2]. The goal of palette creation is to support this task by showing as much of the surfaces and color of the photograph. The final result of this walkthrough is shown in Figure 2.

A walkthrough in five steps

Figure 3 shows a part of the screen that contains several palettes floating on top of a photograph the user intends to retouch. By default, the tool palette is opaque and occludes the photograph (Figure 3a). When the palette is alpha-blended with the background (Figure 3b), the photograph shows through, but its contrast and colors are still affected.

Step 1: Desaturating: The Photoshop tool palette contains no useful color information, so we eliminate it by desaturating (Figure 3c). To fully preserve the color of the photograph (red/green and blue/yellow, but not luminance), we use a blending function that *combines* the palette's luminance with the red/green and blue/yellow information from the underlying photograph. Each of the color channels is taken from only one of the two windows, which prevents them from getting diluted.

Step 2: Making surfaces transparent: The tool palette mainly consists of icons, and it is the icon's contours that are most important for recognition. The icon's surfaces, on the other hand, seem to play a lesser role. We therefore

make the palettes surfaces transparent as shown in Figure 3d. In detail, this is done as follows. First, we apply a high-pass filter (here an emboss filter) to bring out edges. This produces a grayscale image with light and dark edges. The edges can be interpreted as a 3D effect; but most importantly, the effect makes the edges stand out against light and dark backgrounds. We then remap that grayscale image to translucency by using an appropriate blending function (a so-called "linear light" blending function, for details on filters and blending functions, see [2]).

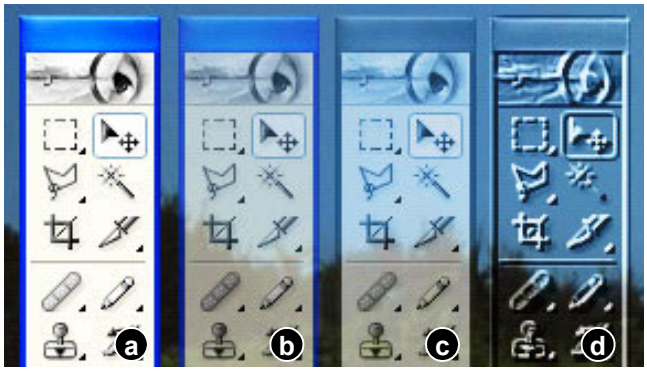

Figure 3: A tool palette blended using (a) opacity, (b) alpha blending, (c) luminosity, and (d) emboss.

Step 3: Blurring noisy backgrounds: Noisy backgrounds interfere with multiblended palettes as much as they do with alpha-blended palettes (Figure 4a and b), making both palettes virtually disappear. Multiblending therefore eliminates high frequencies from the background image by applying a blur filter to the background behind the palette (Figure 4c). Multiblending uses a smart filter that moves with the palette, and that blurs only those areas that exceeds a certain contrast threshold, a concept similar to the "unsharp mask" filter in [2]. The resulting palette is easy to read; all high contrast content is clearly *on* the palette, while all low contrast content is in the photograph behind it. The resulting palette seems to be made of a piece of frosted glass[1], a palette style we will refer to as the *glass palette*.

Step 4: Area-based opacity based on usage data. In Figure 5a and b, a significant part of the palette consists of window decoration, unused icons, or labels that never change, such as "R", "G", and "B". Once users have learned such static palette elements, they offer little information to the user. While varying opacity across alpha palettes leads to a noisy appearance, diminishing parts of glass palettes works well and can be used to make additional background space visible (Figure 5c and d). Frequently used areas are determined automatically based on click data, frequently changing areas by monitoring the palettes. See [3] for a survey of related techniques.

Step 5: Remapping channels: In most cases, steps 1-4 will suffice to produce a satisfactory image; however, some situations of high interference require an additional step where information from one source is remapped to an alternate channel. Figure 6 shows a worst case scenario—two pieces of text in bitmap format, both using the same font and font size. Both windows need to preserve the same features in order to be readable – and given that text contains less redundancy than photographs, both windows are more sensitive to mutilation than the image content we have looked at so far. When alpha blended, *both* text segments become unreadable (Figure 6a).

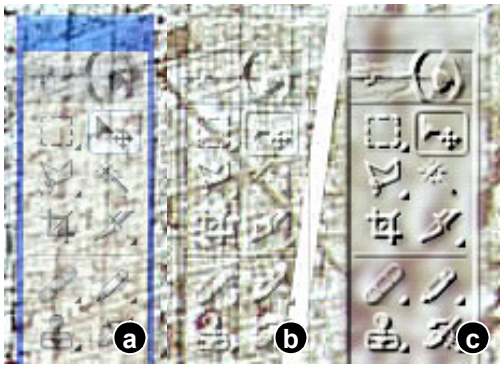

Figure 4: (a, b) Over a high-frequency background, alpha-blended and embossed palettes become unreadable. (c) A background blur solves this issue.

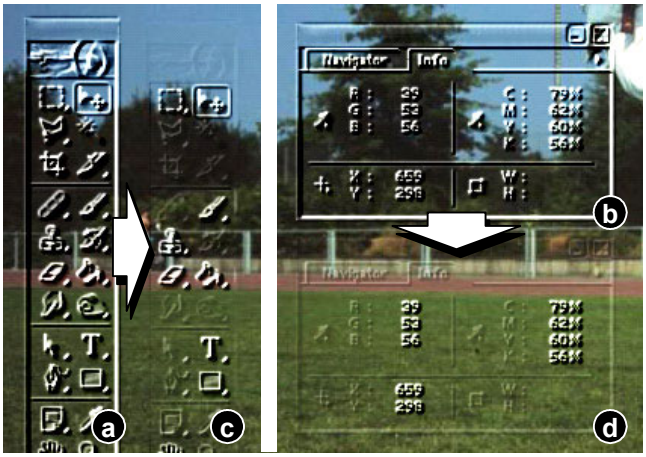

Figure 5: (a and b) Borders and unused icons use up screen space (c and d): Frequently clicked icons and dynamic numeric information are preserved, but decoration and constant text labels fade.

When applying steps 1–4, only blurring actually affects the palette. The problem is that both windows use the same color channel (luminance) to convey their information. We address this by remapping the luminance channel of the foreground text to a different color channel, here the red-green difference (Figure 6). Then we use a blending function that assembles the final image from the hue of the foreground and luminance and saturation from the back-

[1] The palette can also be thought of as a relief palette with the photograph pressed *onto* it. This underlines that when blending windows, the notion of Z-order as a means for defining an occlusion order goes away. Z-order is only needed to decide which window receives mouse input, and if only one window can receive mouse input, Z-order becomes unimportant.

ground. The resulting image allows the blurry text to be read based on its luminance (hold the figure an arm's length away), while the crisp text can be read based on its color (hold the paper close). Note that this is a worst case scenario. We created multiblending with graphical material in mind; even with the enhancements of multiblending, blending text will generally remain undesirable. Nonetheless, pre-filtering text segments creates a limited amount of readability where alpha blending does not.

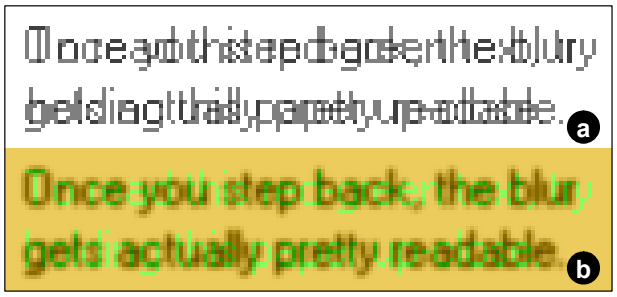

Figure 6: (a) Alpha-blending text on text. (b) Blurring one text and encoding the other text in hue (hue encoding is *invisible* in b/w hardcopy. Please see the ACM digital library for a color version of this figure).

Summary of the walkthrough

In this walkthrough, steps 1-3 removed features in order to preserve the respective class of features in the other window from interference. We applied a blur filter to remove textural features and we used customized blending function to selectively process color channels. In step 4, we extended the approach by allowing different blending parameters for individual areas. In step 5, finally, we solved collisions in requirements by remapping a channel. As a result, each feature class is now used by either palette *or* background. This eliminates visual ambiguity, as each feature is clearly associated with only one window.

Figure 7 gives an idea of the applicability of the glass palette. In this example, we merge two windows that have identical features, as the shown overview palette shows the same photograph as the background. We decide that the overview palette contains less task-relevant information than the background photo and thus turn the overview into a glass palette. While the outline information in the overview is still sufficient for showing which part of the photograph is currently visible, this palette avoids the visual ambiguity that the alpha palette introduces.

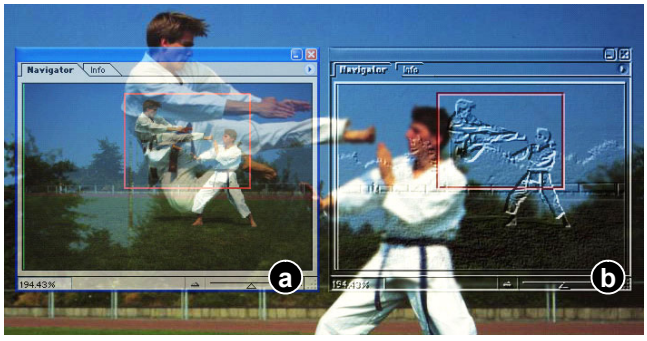

Figure 7: (a) Alpha overview palette. (b) The glass overview interferes less with the background.

Manual palette optimization

When converting opaque palettes to multiblended palettes, the individual weights are best chosen such that the window with the more prevalent features of that class 'wins.' These initializations can generally be done automatically; and loading a different picture or moving palettes to a different background can even be used to trigger a change in the palette's representation. These initializations may, however, need manual correction. For example, removing a red-eye effect requires preservation of color, even if the rest of the picture has little saturation. Allowing users to manually switch between palette representations at runtime allows obtaining the best results for the task at hand.

Also during palette creation, the quality of multiblended palettes can be improved by manual input. Figure 8 shows an example of manual background removal. Alpha blending color swatches results in diluted, thus inaccurate colors (Figure 8a). The swatches thus need to be rendered as fully opaque. Manual cropping of swatches (Figure 8b) allows preservation of the colors with minimal occlusion, while the decoration of the palette uses the known glass effect.

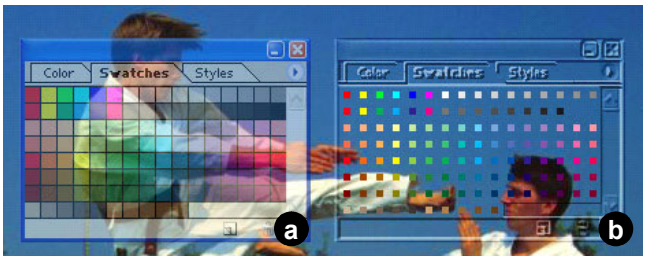

Figure 8: (a) Alpha blending dilutes colors in color palette. (b) Glass palette with manually cropped swatches (see ACM digital library for color version).

IMPLEMENTATION

We implemented an initial Java version of the glass palettes described above. The program works by rendering an opaque version of each palette into an off-screen buffer, applying all required filters to that off-screen image, merging it with a copy of the respective fragment of the screen buffers, and then copying the resulting bitmap back onto the screen. Since our prototype does not yet use graphical acceleration its rendering performance is fairly limited; rendering is therefore not done while palettes are moving, and there is a noticeable pause after moving a palette before the multiblending effect appears. However, for stationary palettes, our experience with this prototype suggests that the technique is viable from an implementation standpoint; future versions in native code will easily overcome the current performance limitations.

USER STUDIES

In order to validate the multiblending approach, we conducted two user experiments comparing the glass palette to alpha-blended palettes at different levels of opacity. Our main hypothesis was that the glass palette would simultaneously deliver better recognizability of foreground and background than any alpha palette. Each of the two studies measured one of these aspects using a distinct task. The

palette recognizability task required participants to find and click on a series of palette icons. The *background recognizability* task required participants to match the background picture to one of several candidates.

Background recognizability study

The first study considered the recognizability of backgrounds that were covered with palettes.

Methods

Twenty-four participants were recruited from a local university. All had normal or corrected-to-normal visual acuity and normal color vision. All had extensive experience (more than 10 hours/week) with applications that used palettes and visual workspaces; 8 participants had experience (> 1 hour/month) with an image-processing application.

The study was conducted on a Pentium4 Windows PC running a custom-built Java application. The study system was displayed on a 21-inch monitor at 1280x1024 resolution.

The study compared three alpha palette types and a glass palette type (types shown in Figure 14). Palette visuals were taken from Adobe® Photoshop® [2] and converted automatically. Alpha-blended palettes computed pixel colors as a weighted sum of palette and background using opacities 10%, 25%, and 50%, as suggested by [8]. Glass palettes computed pixel colors using the following four steps: emboss (2-pixel height) and desaturate to the palette, Gaussian blur (1-pixel radius) to the background underneath the palette, and blending (using "linear light").

The task asked participants to look at a source image that was covered by palettes (Figure 9, top left quadrant), and click on the exact match of that image from among a set of three candidate images (other three quadrants in Figure 9). This simulated the real-world task of image retouching, where the user must assess the correctness of the overall image after every stroke. In two of the three candidates, one image feature (the flowers in the top left of the picture) had been altered by changing either its brightness or its contrast by 1, 2, or 3 steps in either direction. The modified feature was either light (Figure 10) or dark (Figure 11). Participants were given four practice trials with each of the palettes, and then completed 10 test trials in each condition.

Most of the palette surface (the background) was light. Alpha Palettes thus formed a stronger contrast with the dark background features than with the light ones. Since alpha blending reduces contrast, we hypothesized that it would affect the recognizability of the dark features more.

Measures in this task included completion time and error magnitude—that is, the number of steps difference between the two images when an error was made.

The study used a 4x2 mixed factorial design. The factors were *palette type* (glass palette, alpha-50, alpha-25, alpha-10) and *feature type* (light or dark). Palette type was a within-participants factor, and feature type was a between-participants factor. Order and spatial position in the quadrants were counter-balanced so that each condition was seen equally in each quadrant. The study system collected completion time and error data.

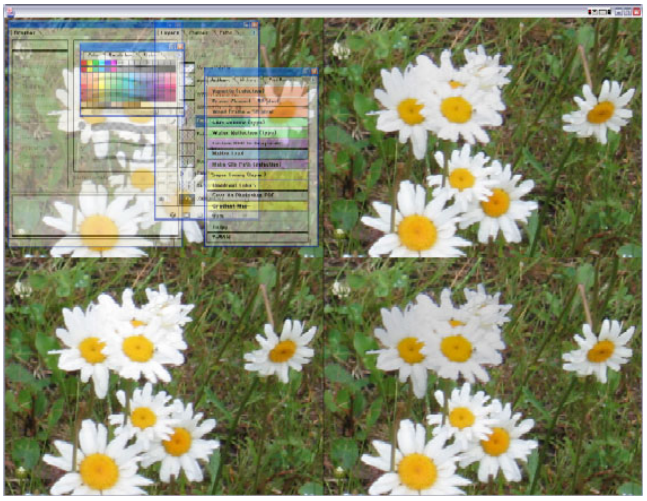

Figure 9: Background recognizability task (alpha-50 condition). A source image with overlaid palettes is shown at the top left, and three candidate images—near copies of the source image—are shown in the other three quadrants.

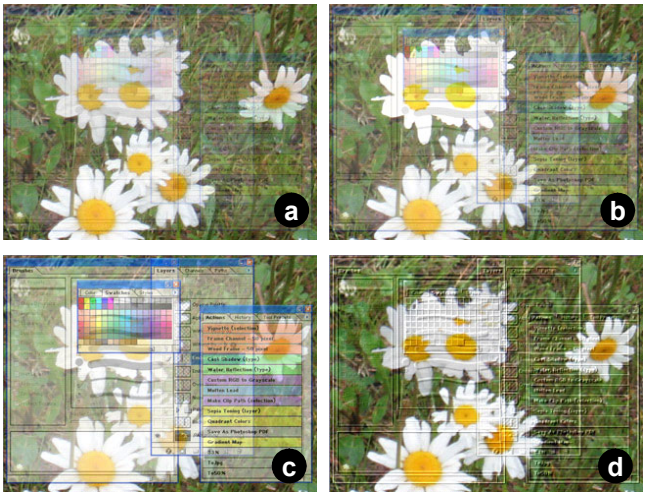

Figure 10: The 4 conditions in the background recognizability task: (a), alpha-blended at 10% opacity, (b) alpha 25%, (c) alpha 50%, and (d) glass palettes.

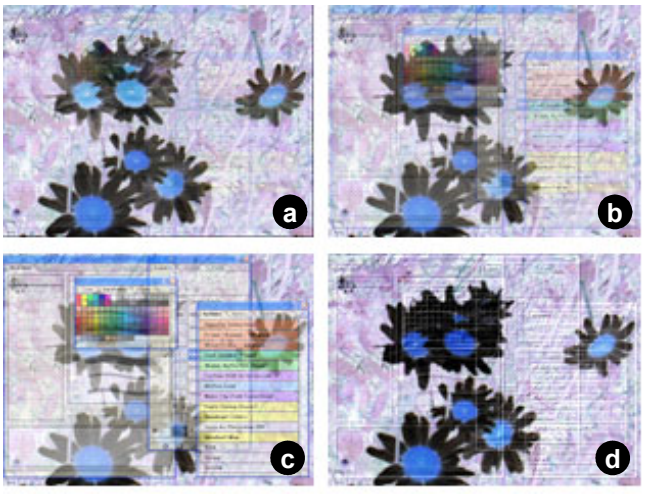

Figure 11: …same palettes over dark image features

Results – background recognizability

Participants' accuracy in matching the source image using different palette types is shown in Figure 12. Analysis of variance (4 (palette) x 2 (feature type) ANOVA) with feature type being a between subjects factor was used to test the effects of the two factors. For errors, there was a significant main effect of palette type ($F_{3,66}$=11.14, p<0.001) but not of feature type ($F_{1,22}$=0.59, p=0.45). Interaction between palette style and feature was borderline significant ($F_{3,66}$=2.56, p=0.06).

We carried out follow-up analyses to compare individual conditions. Errors were significantly lower for the glass palette than any alpha palette; (glass palette vs. alpha-10 palette, $F_{1,22}$=5.26, p<0.05). The only other significant difference was between alpha-10 and alpha-50 ($F_{1,22}$= 11.15, p<0.005).

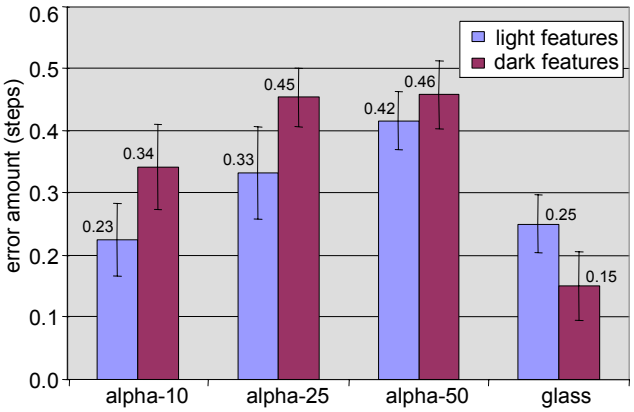

Figure 12: Mean error amounts for background recognizability task. Error bars show std error.

Completion time data was also analyzed using ANOVA, but no main effect ($F_{3,66}$=1.73,p=0.17) or interactions were found. On average, each trial took between 40 and 70 seconds. Experience with image-processing applications did not have any effect on performance.

We will now first present the second study, which will allow us to discuss the results of both studies in conjunction.

Palette recognizability study

The second study investigated the recognizability of the images on the palettes themselves.

Methods

Twelve participants were recruited in the same way as for the first study. The study was conducted on the same apparatus, with a similar Java application. The same four palettes types were used.

Participants were presented the apparatus shown in Figure 13. In each trial, an icon was shown in the middle of the screen. The participants' task was to click on the matching icon located in one of the four six-icon palettes on the screen as quickly as possible. The task consisted of 24 trials per palette type. Each icon was presented once per condition; the same icons and palettes were used in all conditions. We used the same background image as in the previous study. As with the previous study, target palettes icons were either located over a light or dark background feature, for 50% of the trials each.

Participants were given 12 practice trials when starting a different palette type. Since participants thus had a good general idea of where each icon was (as would be the case for an experienced user of a program like Photoshop), the task did not test visual search over a large area, but rather assessed localized search, icon recognizability, and target acquisition. Measures for this task were task completion time and number of incorrect clicks.

This study used a 4x2 within-participants factorial design with the same factors (palette type and background feature type) used previously; however, in this study both factors were within-subject factors.

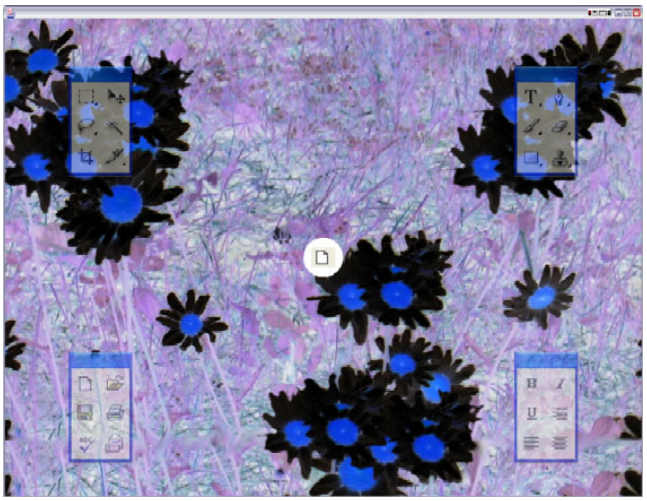

Figure 13: Palette recognizability task (alpha 50% condition). The next icon to be selected is shown in the centre circle.

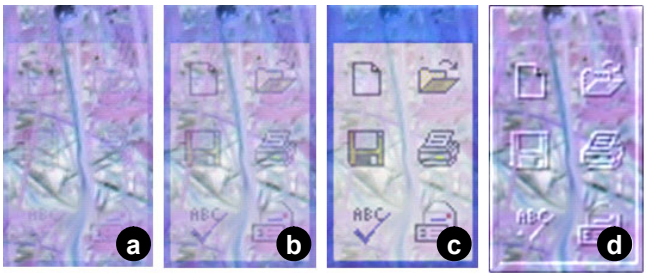

Figure 14: Alpha palettes at (a) 10% (b) 25%, and (c) 50%, and (d) glass palette used in user study.

Results – palette recognizability

Using a 4x2 ANOVA, the main result was the clear difference between alpha-10 and the other three palette types. There were main effects of both errors ($F_{3,33}$=9.15, p<0.001) and completion time ($F_{3,33}$=7.56, p<0.005). Where the error rate with the other three types was about one in 25 trials, the rate for alpha-10 averaged more than one in three for light backgrounds, and more than 1.5 per trial for dark (see Figure 15). Completion time ranged from more than five seconds on average for the alpha-10 condition, to less than two seconds for all the other palette types. Post-hoc analyses confirmed that these differences were

significant ($p<0.05$). The large difference in errors between light and dark backgrounds for the alpha-10 palette also resulted in significant main effects of background type on errors ($F_{1,11}=6.32$, $p<0.05$), and on completion time ($F_{1,11}=8.77$, $p<0.05$). There were also interactions between background and palette type (for errors, $F_{3,33}=5.25, p<0.01$; for completion time, $F_{3,33}=4.57, p<0.05$).

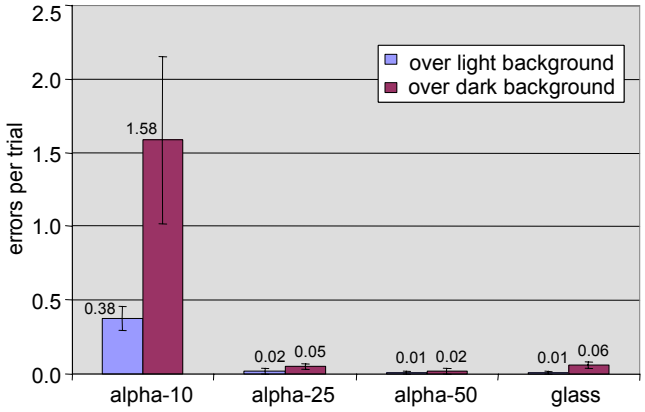

Figure 15: Mean targeting errors for palette recognizability task. Error bars show std error.

Preferences

Participants in the first study were also shown the targeting task at the end of their session, so they could compare the conditions both for background and foreground visibility. We then asked them which technique they felt best supported *both* tasks, considered together. Of the 24 participants, 20 chose the glass palette and 4 the alpha-25 palette.

DISCUSSION

The tradeoff of alpha blended palettes is that increasing opacity to perform better on foreground tasks necessarily implies worse performance on background tasks. The studies showed that a multiblended palette is able to offer a better tradeoff, and perform well on both tasks. Glass palettes were at least as good as the best alpha palettes for both tasks, and were also significantly better than the best overall alpha palette (25% opacity) for certain image types. Moreover, the majority of participants preferred the glass palettes.

Although our study tested only a comparably small sample of alpha values, it seems unlikely that a different choice of alpha values would have lead to a different outcome of the experiment: opacities above 50% should perform even worse in the background recognizability task that alpha-50; opacities below 10% should be even less recognizable in the foreground task than alpha-10.

The better performance of the glass palettes seemed to be caused by two main properties of this palette style. First, by making most of the palette surface completely transparent they provide an *unaltered* view on larger parts of the background. This allows users to see and check important image background features, such as color and brightness. Second, the emboss effect applied to palettes produces outlines with both light and dark components, making edges stand out on a variety of background color and brightness.

CONCLUSIONS

By eliminating the drawbacks of alpha blending, such as visual ambiguity, loss of contrast, and unfaithful reproduction of colors, multiblending helps optimize the readability of palettes and background. For the tasks examined in our user study, multiblending maintained recognizability of palette *and* background significantly better than any of the tested alpha-blended palettes. On the other hand, multiblending is computationally more expensive and optimization of palettes requires a certain understanding of the application scenario.

For future work, we plan to test multiblended palettes in a variety of applications scenarios, ranging from games, image editors and CAD systems to instant messengers, audio players, and task bars.

Acknowledgements

Thanks to Christopher James Fedak for his work on the implementation and to Mary Czerwinski, Andy Wilson, Steven Drucker, and Ed Cutrell for their comments.

REFERENCES

1. ActualTools Corporation, *Actual Transparent Windows Software,* www.actualtools.com/transparentwindows
2. Adobe Photoshop www.adobe.com/products/photoshop
3. Baudisch, P., DeCarlo, D., Duchowski, A., and Geisler, B. Focusing on the Essential: Considering Attention in Display Design. *CACM* 46(3), pp. 60–66.
4. Bell, B. Feiner, S., Höllerer, T. View management for virtual and augmented reality. In *Proc. UIST'01*, pp. 101–110.
5. Bier, E., Stone, M., Pier, K., Buxton, W., and De Rose, T. Toolglasses and Magic Lenses: the See-Through Interface. In *Proc. SIGGRAPH '93*, 73–80.
6. Cox, D., Chugh, J.S., Gutwin, C. and Greenberg, S. The Usability of Transparent Overview Layers. In *CHI'98 Companion*, 301–302.
7. Grudin, J. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In *Proc. CHI'01*, pp. 458–465.
8. Gutwin, C., Dyck, J., and Fedak, C. The Effects of Dynamic Transparency on Targeting Performance, In *Proc. Graphics Interface 2003*, 101–110.
9. Harrison, B.L., Kurtenbach, G., and Vicente, K.J. An Experimental Evaluation of Transparent User Interface Tools and Information Content. In *Proc. UIST '95*, 81–90.
10. Lieberman, H. Powers of ten thousand: Navigating in large information spaces. In *Proc. UIST'94*, 15–16.
11. McGuffin, M., Balakrishnan, R. Acquisition of expanding targets. In *Proc. CHI 2002*, 57–64.
12. McLaren, K. The development of the CIE 1976 (L*a*b*) uniform colour-space and colour-difference formula, *Journal of the Society of Dyers and Colourists,* 92, 1976, pp. 38–341.
13. Palma, W. *WinRoll*, www.palma.com.au/winroll
14. Perlin, K. and Fox, D. Pad: An alternative approach to the computer interface. *Proc. SIGGRAPH'93*, 57–64.
15. Porter, T. and Duff, T. Compositing Digital Images, *Computer Graphics* 18, 3, July 1984, pp. 253–259.
16. Wandell, B. *Foundations of Vision*. Sinauer Assoc, 1995.
17. Wickens, C. *Engineering Psychology and Human Performance*, Harper Collins, 1992.