Technical Report TR05-010

Department of Computer Science
Univ. of North Carolina at Chapel Hill

Facetop: Integrated semi-transparent video for enhanced natural pointing in shared screen collaboration

Karl Gyllstrom and David Stotts

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175

stotts@cs.unc.edu

May 15, 2005

Facetop: Integrated semi-transparent video for enhanced natural pointing in shared screen collaborations

Karl Gyllstrom
Department of Computer Science
The University of North Carolina at Chapel Hill
karl@cs.unc.edu

David Stotts
Department of Computer Science
The University of North Carolina at Chapel Hill
stotts@cs.unc.edu

General Terms
No terms

Keywords
Video tele-conferencing, telepresence, interpersonal communication, shared workspaces, awareness

ABSTRACT
We present an innovative tool for distributed, synchronous, pair work that seamlessly combines video-teleconferencing and generic application sharing. We integrate video into the workspace as a persistent, semi-transparent, full screen overlay, instead of an independent window that must contend for screen space. Because this complicates traditional, generic framebuffer sharing tools, we present an "overlay-aware" screen sharing facility. We incorporate mechanisms to maintain consistent views between users, enabling physical gestures to correspond to the same shared, visual objects. We tailor this facility to support diverse sharing requirements by simultaneously providing multiple modes of sharing, from generic application windows to arbitrary screen portions.

1. INTRODUCTION
Pair programming research has revealed numerous benefits when collaborators use the same computer, as it assists good programming practices by involving constant, personal interaction and continuous oversight in code writing[5]. However, many subtle aspects of pair programming are rendered difficult or impossible when participants are not located at the same device[1]. While communication facilities such as chat and video-teleconferencing assist face-to-face interaction, and mechanisms such as tele-pointers assist awareness with respect to work on the shared workspace, there are a rich set of tasks that fall between these two interaction modes. Such tasks include subtle but important forms of communication such as gesturing to a visual object (e.g. pointing to a line on a document) and gaze awareness to initiate conversation or communicate attention.

The importance of gestures in communication has been the focus of much research that has revealed they play a critical role in complementing speech[14][27][20][19]. Used collaboratively on shared work, gestures become powerful tools, enabling workers to quickly and intuitively signify points of focus[3][10]. This dynamic of gestures is difficult to recapture when simply using disparate tools such as video-teleconferencing and shared-editing facilities; an effective solution requires that these tools be carefully integrated. To accomplish this, we have developed a system that weaves video-teleconferencing and a screen sharing facility together, to not only minimize the effort required in task switching between communicating and editing, but more importantly, to enable the set of subtle interactions that fall between these two tasks. Our application achieves this by blending the video image of another participant onto one's desktop through a full-screen, semi-transparent overlay that allows both the content of a shared artifact and the image of the other participant to be simultaneously visible (see Figure 2). However, the incorporation of screen sharing in a full-screen overlay introduces complications that prohibit the use of existing sharing solutions such as VNC. We have circumvented this problem through an "overlay-aware" screen sharing facility, and furthermore, we have extended the traditional functionality of screen sharing to provide more versatility to the user.

Past research has introduced many levels of screen sharing granularities along a spectrum ranging from the model to the framebuffer. We find that, to share applications with both (a) no requirement that they expose access to an underlying, abstract representation, and (b) tight consistency at the visual level (no view divergence), we must share at the framebuffer, or pixel level. While numerous framebuffer sharing facilities exist today, many of them are purposefully general in order to support tasks such as remote administration. We instead propose a sharing facility tailored to support collaboration-only tasks. This augments framebuffer-based application sharing with considerably stronger flexibility and agility, creating a system that accomplishes the following tasks:

- It allows the replication of visual features of one user's screen portion to other participants' screens, ensuring that all participants maintain a consistent view.

- It avoids the problems inherent in the combination of framebuffer sharing with full screen video overlays.
- It provides for different, novel forms of sharing, including the sharing of hand-drawn screen selections, single windows, and sub-window portions that are anchored to their window's coordinates and scroll positions.

Finally, we extend this screen sharing system to allow for an integrated "glass board" that applies the idea behind shared whiteboards to the existing visual components, allowing users to apply annotations to generic visual components such as windows and portions of windows.

2. RELATED WORK

Significant research has contributed to facilitating group awareness in collaboration, from developing facilities for easy construction of groupware that supports awareness[11], to retrofitting existing applications[4][16]. A weakness of the former is that it forces collaborators to use tools built with collaboration as an element of design, preventing the use of the vast majority of more mature, single-user applications in collaborative work. Retrofitting existing applications offers the potential to free many single-user applications to be used collaboratively, but still requires per-application development of bridges capable of intercepting and broadcasting application events. Framebuffer sharing systems such as VNC[1] provide the most general support for group awareness and application sharing, but suffer limitations in flexibility. WinCuts[24] presents a more flexible screen sharing system that allows users to share portions of windows. Beyond flexibility issues, VNC, as we will describe later, makes integration of overlaid video particularly problematic.

Recapturing the intuitiveness behind face to face communication has been a heavily researched area, with many interesting approaches tackling various dynamics such as gaze awareness and gestural communication. Hydra Unit[21] presents early work in multiple party video-teleconferencing that enabled gaze awareness through pairing each camera to the display of the user viewing that video feed. MAJIC[18] presents a system that allows multiple parties to view life-size images of each other, removing boundaries between disparate images to recreate the feeling of being physically co-located. Hypermirror[17] integrates the mirrored image of the user into the projected environment of other participants. This allows users to indicate direction through their own orientation in the shared space as an alternative to gaze direction. While all of these solutions remove much of the difficulty associated with using video for remote communications, none of them integrate a shared workspace that allows all users to directly manipulate common artifacts.

Other work extends communication by integrating a shared workspace into the video. VideoDraw[29] shares a shared video representation of each users' hands and a common whiteboard. VideoWhiteboard[28] modifies this work by enabling users to see silhouettes of each other's full body to enable gestural communication.

VideoArms[25][26] presents an overlay of user's arms onto a shared workspace, with various compositing techniques including silhouettes, opaque images of the arms, or semi-transparent images. Like in Facetop, VideoArms allows the arms of remote collaborators to show up in video form in the viewing space of collaborative tools, giving an enhanced sense of presence, more natural pointing, and the ability for users to more easily distinguish one collaborator from another (compared to using shared pointer technology, for example). A number of qualities distinguish our work from VideoArms. First, the camera placement in Facetop allows inclusion of the full image of the collaborating users, increasing the ability of collaborators to recognize each other easily over simply seeing arms; if desired, Facetop can show only arms, giving the same view as VideoArms, but Facetop provides a more powerful range of possibilities. Second, instead of working only on groupware, our work allows for the sharing of arbitrary application windows or screen portions, and automatically handles issues related to ensuring that each user shares a consistent view of the shared document (making gestures to shared objects feasible). We find that the latter distinction is a complicated problem, and have devoted much rigor to enabling this functionality.

ClearBoard[13][12], another closely related work, presents a solution that emulates two users facing each other with a glass pane between them that they can simultaneously draw on, mirroring each's display to allow both members a consistent view of the contents of the shared "clear board". The implementation maintains both the image of the other person and the shared virtual canvas in the same line of sight, facilitating gaze awareness and rapid switching between face to face communication and editing.

Our work departs from ClearBoard in a fundamental way: where ClearBoard involves the drawing space as the shared artifact, we integrate the entire desktop to the shared workspace. Beyond simply sketching pieces to brainstorm and communicate ideas, Facetop users can share and simultaneously edit general application windows. This distinction introduces numerous problems that prohibit simply combining Facetop with a generic screen sharing system; this work presents a careful integration of full screen video and a custom screen sharing facility. In addition, our application of general drawing functionality to on-screen objects provides the ability to anchor annotations to movable objects, allowing annotations to follow the objects they overlay if those objects are moved by the user.

A further distinction from ClearBoard, and the other aforementioned research in video presence, is that Facetop has a greater potential for ubiquity through both physical portability and cost (it can be deployed on a laptop with an inexpensive, off-the-shelf web camera). In addition, where Clearboard requires users to be directly in front of their drawing utility, Facetop allows the incorporation of projected displays. This yields users the flexibility to sit far from the screen and still fully operate the system.

Past work on Facetop[23][22] presented an early implementation of video and desktop integration. However, it lacked an automated screen sharing facility; it could only be used with group aware applications, and required manual synchronization of window coordinates to ensure consistent views be-

[1]http://www.realvnc.com/

tween users. In this paper, we present a system that provides a range of screen sharing methods that work in the presence of a video overlay rapidly updating the framebuffer. Furthermore, we have extended this work by adding facilities for automated transparency adjustment and background removal to avoid cluttered video.

3. OVERVIEW

Facetop includes two main components. The first is a facility for capturing video, transferring video over a network, and displaying transferred video to the screen as a transparent overlay. The other component is an adaptable screen sharing facility which both: (a) circumvents the problems a traditional screen sharing application such as VNC presents, and (b) affords an increased amount of flexibility in types of sharing. For the remainder of this paper, we will make comparisons to VNC as another framebuffer sharing tool, although many of the comparisons are equally applicable to other tools in the framebuffer sharing family, including PCAnywhere[2], Timbuktu[3], and several others.

3.1 Video Integration

Facetop appears to the user as his usual desktop composited with two video feeds, the video of his partner and the video of himself. The video overlays cover the entire screen; in cases where screen dimensions between users are not equal, both instances use an overlay size representing the minimum height and width of each's screen. Figure 1 depicts the physical setup for a computer with Facetop running. Note the video camera sitting on top the LCD panel pointing back at the user; in our current work we use a $100 Sony iBot, giving us an image that is 640 x 480 pixels of 24-bit color, captured 30 frames per second.

Facetop simultaneously emulates two forms of interaction. The first is face to face; users all see the image of the other collaborator before them, displaced by a common view of the shared workspace. As we wish each's view to be consistent with respect to horizontal orientation (e.g. text should read left to right on all displays), we flip each image about the y-axis to create a mirrored image of each participant. This allows the physically impossible illusion of two workers facing each other and maintaining a consistent view of an object between them. Figure 2 depicts a screenshot of two users collaborating on a shared document.

The image depicts the second simultaneous form of interaction: two users working side-by-side and viewing a common workspace. The reasons we display the video of the user to himself are subtle, but important. First, because the video of the user's actions must be captured from a camera that has a separate physical location from the display (i.e. not directly behind it), the image of a user pointing directly at an object may not correspond to the exact location the overlaid video indicates. The user's image of himself provides a feedback mechanism, allowing the user to see exactly what location his gestures refer to, and hence, what screen location his partner will see him referring to. Furthermore, the illusion of two collaborators working side-by-side, potentially mitigates some of the dissonance associated with using computer mediated communication to emulate face-to-face interaction[17].

The extent of blending for either the video of either user is configurable at run time. We provide a toolbox with sliders that allow each user to quickly modify blending. While appropriate blending levels may vary according to the content of users' workspaces, background objects in the video, and user preferences, we believe that the blending should often vary according to the different types of task with which the users are engaged. A user concentrating on editing a document may want the document's window to be more visible, while a user explaining a point may want his partner to be more visible so he can more easily notice facial responses to his discussion. We have explored a few options for automated adjustment of blending. To facilitate visibility of the workspace when the user is editing, we implemented a technique that reduces the opacity of both video overlays when the user is frequently typing or using the mouse, and restores it when a lapse in keyboard events has occurred. We are also working on integrating voice recognition, increasing the opacity of both participants when either is speaking to make conversation more visible.

The physical objects behind a user can potentially add clutter to the video image that obfuscates the view of objects on the user's screen. We find that user control of transparency levels can offer significant reduction of clutter. Furthermore, a solid wall behind the user does much to mitigate this problem. However, in the interest of portability, it is infeasible to require this. Thus, we have implemented a background removal feature using background removal techniques to provide users a cleaner representation of each other when collaborating. By maintaining an image of the background without the user in front of it, we can establish which pixels on an image correspond to the user. We build an image mask from these, then apply erosion and dilation on it mask to remove noise. Applying this mask to the image, such that all pixels not within the mask's boundaries are rendered fully transparent, we are able to recover an image where only the user is visible. Figure 3 depicts the image of a single user on Facetop with and without background removal. Note that, for demonstration purposes, both images feature a higher opacity of the video than would normally be necessary.

3.2 Screen and Window Sharing

Application sharing facilities range in sharing granularity from the model to the pixel representation. While higher-level sharing mechanisms provide more flexibility in view divergence, our goal was to maintain strict, pixel level correspondence in views to ensure precise symmetry in coordinates of visual objects. Furthermore, we required a mechanism that allowed sharing of arbitrary applications, thus prohibiting sharing mechanisms that require applications to comply with any specific multi-user functionality. For this reason, we implemented a screen sharing mechanism that accommodates sharing of the representations of arbitrary windows.

While screen sharing systems such as VNC are readily available, we implemented our own system for several important reasons. First, VNC provides updates at the framebuffer

[2]http://www.symantec.com/pcanywhere/
[3]http://www.netopia.com/software/products/tb2/mac/

Figure 1: A Single User on Facetop.

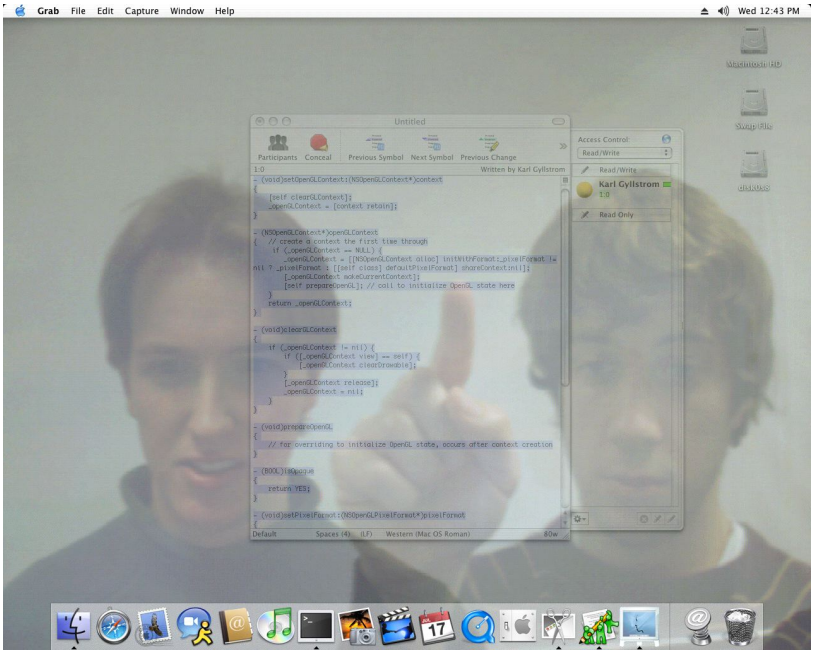

Figure 2: Two users on Facetop.

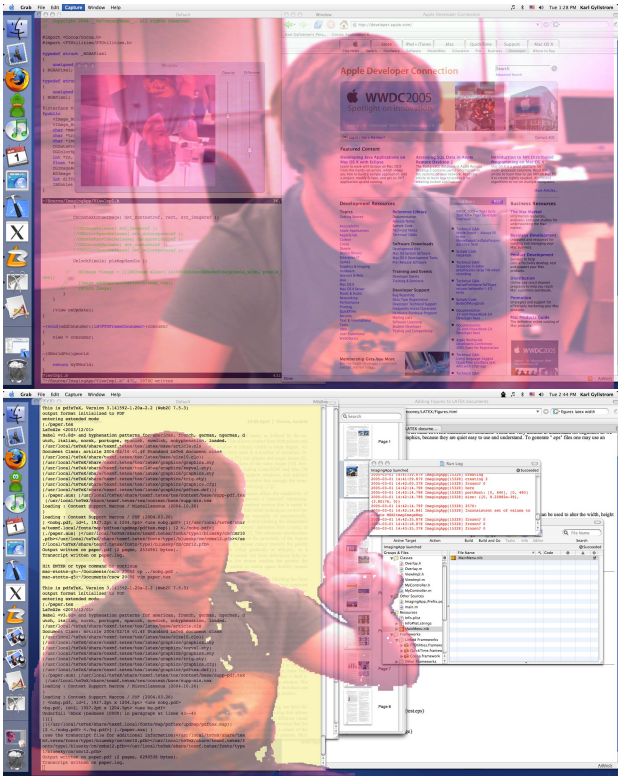

Figure 3: Background removal, before and after.

level only. This creates a fatal conflict with our transparent overlay; as each frame causes a refresh of every on-screen pixel, an immense amount of bandwidth would be necessary to allow for 20-30 frames per second, and since our system sends video as a compressed stream, much of VNC's updates would be spurious. As VNC employs a "pull"-based mechanism, clients could be configured to receive data at a slower rate to mitigate bandwidth issues. This is still problematic for both performance and functional reasons. First, reduced transfer rates compromise the interactive nature of screen sharing. Furthermore, delayed transfers would display overlaid frames from the other's desktop that were in the past, causing potentially confusing images when the real-time video is overlaid.

Another reason we developed our own sharing mechanism is that VNC's full screen sharing does not provide the granularity of sharing that we find useful. In collaborative sessions, for reasons of shared focus or privacy, users may want the ability to share only a single window, or even an arbitrary portion of a window or the screen. Since window sizes, locations, and order may change, a fine grained screen sharing system must maintain "awareness" of the underlying visual objects. Additionally, we desired to support bidirectional sharing, which creates significant difficulty when using VNC's full screen implementation.

We implemented a mechanism for screen sharing that supports a variety of granularities: arbitrary screen rectangles, single windows (although multiple single windows can be shared simultaneously), and window subsections. Arbitrary screen rectangle sharing allows users to use a selection tool to delineate a static portion of the screen to share. Input events from users who do not own the window are forwarded and translated to corresponding input events on the owner's window.

Window sharing allows an entire window to be shared, and furthermore affords a user the flexibility to move, resize, and occlude that window with other windows. To maintain consistency in view, the dimensions of the window are transferred and updated upon changes so that instances running on other collaborators' screens can maintain corresponding dimensions. If a host shares a window that is later partially or entirely occluded by another window, the visual representation of the window will remain complete to other users. However, in this case, we suppress remote input events to avoid the possibility of these events unintentionally affecting occluding windows.

There are many situations in which users may want to share small portions of windows in order to quickly mutually interact with the most relevant part of a document. We support window subsection sharing that, like single window sharing, allows the dimensions of windows to be adjusted. Furthermore, we provide the ability for window subsections that lie entirely within scrollable sections' boundaries to adjust with the scroll. In other words, if a scroll event moves a section of shared text to another screen location, the content of the intended shared region is reflected to the other user, even though the coordinates of that region have changed. Users are not limited to rectangular regions when selecting shared screen portions; they are free to create arbitrary shapes for flexible selection.

We desired the ability for users to easily select the particular sharing technique that was most appropriate for their need. The interface currently allows users to select the technique through a set of selection tools available via a toolbox that can be opened when needed.

As mentioned previously, WinCuts provides similar functionality in that portions of windows can be shared. Our work distinguishes from this in that (a) users can share arbitrary portions – instead of rectangles – by hand drawing a shape, and (b) we provide support for sections that are scrolled to update to the new coordinates of the object. Furthermore, the integration of this sharing into the video allows for more power in collaborative settings.

3.3 Glass board Interaction

Shared whiteboards are popular tools for collaborative developers seeking to visually communicate ideas. However, their usefulness is usually limited to a set of drawing operations on an opaque canvas. We contend that the shared whiteboard paradigm is more powerful when it allows collaborators the ability to make drawable annotations to existing screen objects. We thus extended the versatility of whiteboards by integrating them into Facetop as a "glass pane", or fully transparent window that allows opaque drawing operations to be displayed that appear as drawings on the underlying windows. This integration allows users to circle, underline, or apply other common drawing operations to visual components on their screens such as images and text.

These representations remain visible until users manually remove them.

While this "glass pane" implementation affords the traditional functionality of white boards, we feel its application to the desktop allows for more interesting possibilities. First, the transparent property means the drawing area need not be a separate window that users must manage for screen space, and coordinate with shared use. Second, a shared drawing tool on the desktop creates a more permanent form of telepresence. While our work encourages the use of physical gestures to reference screen objects, one might want a semi-permanent reference, such as an underline to an important portion of text on screen. We find that a shared glasspane over the desktop provides users the useful ability to make generic annotations to any visual object on screen, allowing operations such as circling emphasized objects and drawing connecting lines between related portions of windows. Figure 4 depicts the glass pane use in a Facetop session.

To assist this general screen annotation, we offer some additional useful features to the "glass pane". When the dimensions of a drawn annotation rests entirely within a window, that annotation is anchored to that window's position. If the underlying window is moved after the annotation, the coordinates of the annotation will be adjusted accordingly. In other words, an underline drawn beneath a word will follow that word if its bounding window is moved on screen. Furthermore, the coordinates of an annotation resting entirely within a scrollable region will be adjusted with scroll events.

4. IMPLEMENTATION

We developed Facetop on Mac OS X with a combination of Objective-C and C, using the functionality Apple exposes through the Cocoa, OpenGL, and Quicktime frameworks. Cocoa is the main API exposed for Mac programming, exposing a rich framework for developing and managing graphical widgets. Most of the core applications within Mac OS X are built upon Cocoa, including their web browser (Safari), email client (Mail), text editing applications (TextEdit), as well as their developer's IDE (xcode). Numerous 3rd party applications, such as the group editor SubEthaEdit and chat program Adium are also based on Cocoa.

4.1 Transparent Overlay

We constructed the transparent overlay by connecting the feed from the video to an OpenGL texture, displayed on a Cocoa window sized to the dimensions of the screen. The window is set to a "floating" level, which means it appears above all normal (i.e. not alert panels) windows on the screen. The window is set to ignore mouse events to prevent interfering with the user's ability to interact with applications.

4.2 Window Sharing

Because the overlay taints the pixels of windows with a partial representation of its image, we sought to use the internal representation of windows as the shared data source. While a detailed, platform specific description of our implementation follows, note that the method of obtaining a window's internal representation is not specific to the Macintosh platform, and a similar method could be used on a platform such as Windows.

Mac OS X does not provide a direct mechanism for accessing the off-screen buffers of windows. We thus developed a workaround that allowed us to acquire the underlying representation of windows. InputManagers provide application developers the ability to inject code between input device events and their graphical destinations, and are loaded automatically when any Cocoa-based application is launched. We developed a custom InputManager that exploits functionality that allows developers to replace Cocoa components with custom components. Objective-C, as a dynamic object-oriented language, provides the ability for programmers to universally modify classes at runtime. In other words, programs can extend the functionality of GUI widget classes used by other programs when those processes are launched. We thus replaced various widget object class with custom classes that allowed for replicating their internal representations and saving them to a shared buffer that could be used to distribute to remote applications.

In addition, our custom components could trap and broadcast external windowing events, such as resizing and moving, to ensure common coordinates on remote applications. Additionally, we customize existing scroll frame widgets to notify the sharing facility that a shared portion existing entirely within a scroll pane should be adjusted to coincide with the new location of visual objects within that scrollable region.

It is important to note that this custom code injection does not require a recompile of existing third party code. In theory, any application developed using Cocoa will work with our code at launch time. While roundabout, none of the techniques required accessing private API's or otherwise deprecated practices.

4.3 Generic Sharing

While most OS X applications are built upon Cocoa, there are some that instead use Carbon, an API specifically designed for C programming for ease in portability. We have not implemented a Carbon method similar to that for Cocoa windows, although we are exploring ways to emulate that functionality. Instead, we implemented an alternate method for screen sharing that operates more similarly to VNC in that it accesses pixels at the framebuffer level. This technique can be applied on any window, regardless of what framework it was developed on, and presents a technique that may be useful on platforms that expose no way to capture the underlying representation of windows.

This method introduces two main challenges. First, pixels obtained from the framebuffer memory reflect the effect of the overlaid video. To deal with this problem, we implemented a method for overlay removal. Traditional alpha blending functions use the following function:

$$overlay_{xy} \times \alpha + source_{xy} \times (1 - \alpha) = visible_{xy} \quad (1)$$

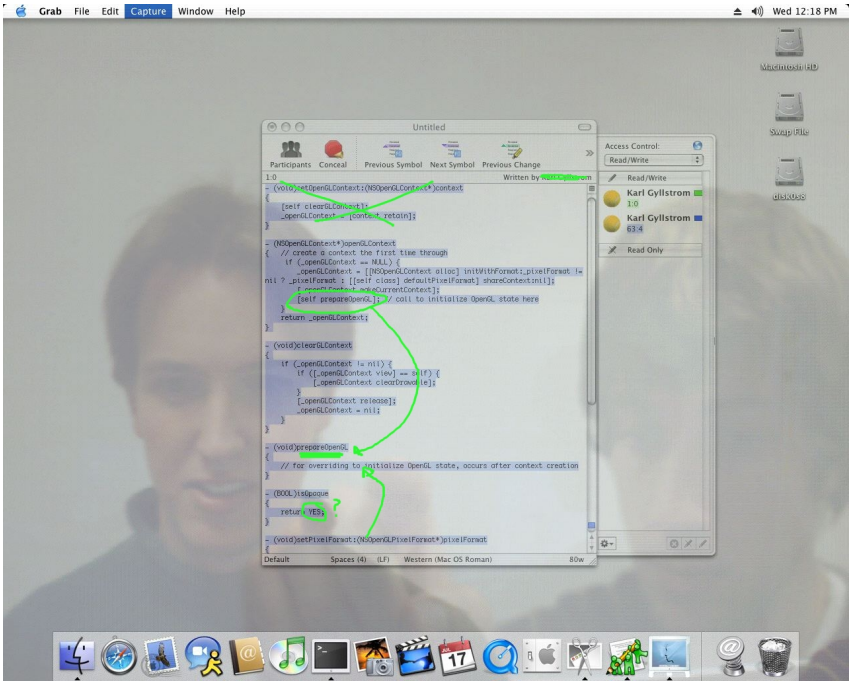

Figure 4: Glass pane.

Where *overlay* represents the pixel value of the overlaid image, *source* represents the pixel value of the underlying image, and *visible* represents the pixel that is visible on the framebuffer. *visible* is accessible through the framebuffer memory, and α is available programmatically, as it is a variable controlled by Facetop. Furthermore, since we have access to the memory the camera writes to, we can also recover the value of $overlay_{xy}$. Therefore, it is possible, with small error, to solve for $source_{xy}$, and recover a visual representation of a visual object beneath the Facetop overlay with minimal effects of the overlay. Since the image of the shared window is overlaid on the remote site, artifacts pertaining to error are even less visible.

The second problem is that the refresh rate of the video overlay will generate a large volume of updates to the screen that may not indicate that the pixels of the shared window have changed. An existing implementation of VNC updates its representation of the screen pixels by the Mac OS X API function *CGRegisterScreenRefreshCallback*, which allows a programmer-defined function to be registered to be invoked with updated screen rectangles whenever portions of the screen are refreshed. Since a full screen update occurs with every new video frame, this method will not suffice to track local window pixel updates. We've implemented a periodic update mechanism to subvert this, allowing users to specify the desired refresh rate for shared windows. While this does not support the dynamic update rate a window might generate when used heavily, we find that a constant update periodicity of once per half second to once per two seconds to honorably emulate an interactive update rate.

We have explored an alternate method for tackling the refresh problem, that employs a difference-based update mechanism to remote hosts to minimize bandwidth usage. By maintaining a snapshot image of a shared screen portion, we can compare the current image to find which pixels are different. However, while the overlay-removal error is insignificant to the user's eye, per pixel image comparisons will still yield many differences that are due to the error. We deal with this by maintaining a pixel difference threshold, such that any pixel difference between a new image and the previous snapshot that is smaller than the threshold will not count as an update to be transmitted. As a consequence, minor pixel differences in a window that are not due to the overlay will not be reported. To deal with this, we periodically update, instead of update on each refresh of the window, the base snapshot to account for any drift associated with minor pixel changes from the window itself.

The ability for shared window portions or screen annotations to be adjusted when non-Cocoa windows are scrolled is not supported at this point.

4.4 Glass Pane

We implemented the shared glass pane by creating a screen-sized window with a transparent background, and setting its depth permanently above other windows in the desktop environment. When a user selects a screen drawing pen from a toolbox, we toggle a parameter on the window that makes it respond to mouse events, and renders drawn objects as opaque. We consider each drawing event – that is, the path created between a user's mouse down and mouse up events – to be a separate object, allowing us to connect drawn objects that fall completely within window or scroll area boundaries to those areas' movement or scroll events. This allows drawn objects to be anchored to movable objects, following those objects when they are moved on the screen.

5. CONCLUSIONS AND FUTURE WORK

We have presented a significant enhancement to Facetop, overcoming the challenges inherent in the integration of framebuffer sharing and semi-transparent video overlays, through both platform-dependent and platform-independent methods. The platform-independent solution uses a video analysis technique to remove the video overlay from shared portions of the framebuffer. The platform-dependent method exploits facilities specific to the operating system, gaining some functionality and efficiency, but sacrificing generality in implementation.

Furthermore, we extended existing methods of screen sharing to provide different, useful granularities of sharing, from arbitrary screen sections, to whole windows, to anchored portions of windows. Finally, we have tightly woven the concept of shared whiteboards into the desktop, assisting collaboration by allowing users the ability to easily and universally annotate their shared workspace, and we have integrated the ability to anchor these annotations to movable screen objects.

Despite the fact that a majority of user applications on the Macintosh are built upon the Cocoa framework, we're pursuing a technique similar to that used in our Cocoa window sharing facility to apply to Carbon windows. Rentzsch[4] presents methods for overriding general C functions on the Macintosh OS X platform, which could potentially allow for extending Carbon window management functionality to save window representations to shared buffers.

While we find Facetop to effectively integrate with many forms of shared work, any use of semi-transparency in window management carries the risk of obfuscating content boundaries when an overlay contains similar visual content to its underlying windows[2]. We're currently pursuing methods to make Facetop more amenable for use on human-based visual data such as x-rays or photographs.

We are currently pursuing methods for integrating more than two users into Facetop. VideoArms effectively handles multiple users by showing only hands and arms, limiting the amount of video introduced by each additional user. While Facetop can also show only hands and arms, we are interested in using the full images of multiple users, in the context of wall-sized screen spaces. One approach to this problem is enabling multi-modal operation, where users can intuitively switch to full-screen collaborative sessions with specific users when needed, but otherwise keep all participants in smaller windows along the side of the screen.

6. ACKNOWLEDGMENTS

This work was partially supported by a grant from the U.S. Environmental Protection Agency, # R82-795901-3.

7. REFERENCES

[1] P. Baheti, E. Gehringer, and D. Stotts. Exploring the efficacy of distributed pair programming. In *XP Universe 2003*, volume 2418, pages 208–220.

[2] P. Baudisch and C. Gutwin. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proceedings of the 2004 conference on Human factors in computing systems*, pages 367–374. ACM Press, 2004.

[3] M. M. Bekker, J. S. Olson, and G. M. Olson. Analysis of gestures in face-to-face design teams provides guidance for how to use groupware in design. In *Symposium on Designing Interactive Systems*, pages 157–166, 1995.

[4] L.-T. Cheng, S. L. Rohall, J. Patterson, S. Ross, and S. Hupfer. Retrofitting collaboration into uis with aspects. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 25–28, New York, NY, USA, 2004. ACM Press.

[5] L. W. e. a. D. Stotts. Virtual teaming: Experiments and experiences with distributed pair programming. pages 129–141, 2003.

[6] C. Gutwin, J. Dyck, and C. Fedak. The effects of dynamic transparency on targeting performance. In *Graphics Interface*, pages 105–112. CIPS, Canadian Human-Computer Commnication Society, A K Peters, June 2003. ISBN 1-56881-207-8, ISSN 0713-5424.

[7] C. Gutwin and R. Penner. Improving interpretation of remote gestures with telepointer traces. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 49–57, New York, NY, USA, 2002. ACM Press.

[8] B. L. Harrison, H. Ishii, K. J. Vicente, and W. A. S. Buxton. Transparent layered user interfaces: an evaluation of a display design to enhance focused and divided attention. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 317–324, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[9] B. L. Harrison, G. Kurtenbach, and K. J. Vicente. An experimental evaluation of transparent user interface tools and information content. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 81–90, New York, NY, USA, 1995. ACM Press.

[10] S. Harrison and S. Minneman. A bike in hand: a study of 3d objects in design. In H. Cross N., Christiaans and K. Dorst, editors, *Analysing Design Activity*. Wiley, 1996.

[11] J. Hill and C. Gutwin. Awareness support in a groupware widget toolkit. In *GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 258–267, New York, NY, USA, 2003. ACM Press.

[12] H. Ishii and M. Kobayashi. Clearboard: a seamless medium for shared drawing and conversation with eye contact. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 525–532. ACM Press, 1992.

[4] J. Rentzsch. Overriding MacOSX, 2003. http://rentzsch.com/papers/overridingMacOSX

[13] H. Ishii, M. Kobayashi, and J. Grudin. Integration of inter-personal space and shared workspace: Clearboard design and experiments. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 33–42. ACM Press, 1992.

[14] R. Krauss and R. Dushay. The communicative value of conversational hand gestures. *Journal of Experimental Social Psychology*, 31:533–552, 1995.

[15] R. E. Kraut, D. Gergle, and S. R. Fussell. The use of visual information in shared visual spaces: informing the development of virtual co-presence. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 31–40, New York, NY, USA, 2002. ACM Press.

[16] D. Li and R. Li. Transparent sharing and interoperation of heterogeneous single-user applications. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 246–255, New York, NY, USA, 2002. ACM Press.

[17] O. Morikawa and T. Maesako. Hypermirror: toward pleasant-to-use video mediated communication system. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 149–158, New York, NY, USA, 1998. ACM Press.

[18] K.-I. Okada, F. Maeda, Y. Ichikawaa, and Y. Matsushita. Multiparty videoconferencing at virtual social distance: Majic design. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 385–393, New York, NY, USA, 1994. ACM Press.

[19] F. Rauscher, R. Krauss, and Y. Chen. Gesture, speech and lexical access: The role of lexical movements in speech production. *Psychological Science*, 7:226–231, 1996.

[20] M. G. Riseborough. Physiographic gestures as decoding facilitators: Three experiments exploring a neglected facet of communication. *Journal of Nonverbal Behavior*, 5:172–183, 1981.

[21] A. J. Sellen. Speech patterns in video-mediated conversations. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 49–59, New York, NY, USA, 1992. ACM Press.

[22] D. Stotts, J. Smith, and K. Gyllstrom. Support for distributed pair programming in the transparent video facetop. In *XP/Agile Universe 2004*, pages 92–104, 2004.

[23] D. Stotts, J. M. Smith, and K. Gyllstrom. Facespace: endo- and exo-spatial hypermedia in the transparent video facetop. In *Proceedings of the fifteenth ACM conference on Hypertext & hypermedia*, pages 48–57. ACM Press, 2004.

[24] D. S. Tan, B. Meyers, and M. Czerwinski. Wincuts: manipulating arbitrary window regions for more effective use of screen space. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1525–1528, New York, NY, USA, 2004. ACM Press.

[25] A. Tang, C. Neustaedter, and S. Greenberg. Embodiments and videoarms in mixed presence groupware. Technical report, Univ. of Calgary, 2004.

[26] A. Tang, C. Neustaedter, and S. Greenberg. Videoarms: Supporting remote embodiment in groupware. video. In *Video Proceedings of the ACM CSCW Conference on Computer Supported Cooperative Work*, New York, NY, USA, 2004. ACM Press.

[27] J. C. Tang. Findings from observational studies of collaborative work. *Int. J. Man-Mach. Stud.*, 34(2):143–160, 1991.

[28] J. C. Tang and S. Minneman. Videowhiteboard: video shadows to support remote collaboration. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 315–322, New York, NY, USA, 1991. ACM Press.

[29] J. C. Tang and S. L. Minneman. Videodraw: a video interface for collaborative drawing. *ACM Trans. Inf. Syst.*, 9(2):170–184, 1991.