Proc. UIST '93 (ACM Symp. on User Interface Software and Technology), Atlanta GA, November 3-5, 1993, 145-155

Windows on the World: 2D Windows for 3D Augmented Reality

Steven Feiner
Blair MacIntyre
Marcus Haupt
Eliot Solomon

Department of Computer Science
Columbia University
New York, NY 10027

212-939-7000
{feiner, bm, haupt, esolomon}@cs.columbia.edu

ABSTRACT

We describe the design and implementation of a prototype heads-up window system intended for use in a 3D environment. Our system includes a see-through head-mounted display that runs a full X server whose image is overlaid on the user's view of the physical world. The user's head is tracked so that the display indexes into a large X bitmap, effectively placing the user inside a display space that is mapped onto part of a surrounding virtual sphere. By tracking the user's body, and interpreting head motion relative to it, we create a portable information surround that envelopes the user as they move about.

We support three kinds of windows implemented on top of the X server: windows fixed to the head-mounted display, windows fixed to the information surround, and windows fixed to locations and objects in the 3D world. Objects can also be tracked, allowing windows to move with them. To demonstrate the utility of this model, we describe a small hypermedia system that allows links to be made between windows and windows to be attached to objects. Thus, our hypermedia system can forge links between any combination of physical objects and virtual windows.

KEYWORDS: augmented reality, virtual reality, virtual worlds, head-mounted displays, portable computers, mobile computing, window systems, X11, hypertext/hypermedia.

INTRODUCTION

When we think of the use of head-mounted displays and 3D interaction devices to present virtual worlds, it is often in terms of environments populated solely by 3D objects. There are many situations, however, in which 2D text and graphics of the sort supported by current window systems can be useful components of these environments. This is especially true in the case of the many applications that run under an industry standard window system such as X [13]. While we might imagine porting or enhancing a significant X application to take advantage of the 3D capabilities of a virtual world, the effort and cost may not be worth the return, especially if the application is inherently 2D. Therefore, we have been exploring how we can incorporate an existing 2D window system within a 3D virtual world.

We are building an experimental system that supports a full X11 server on a see-through head-mounted display. Our display overlays a selected portion of the X bitmap on the user's view of the world, creating an X-based augmented reality. Depending on the situation and application, the user may wish to treat a window as a stand-alone entity or to take advantage of the potential relationships that can be made between it and the visible physical world. To make this possible, we have developed facilities that allow X windows to be situated in a variety of ways relative to the user and the 3D world.

In this paper we first present related work and provide an overview of our system. Next, we describe the different kinds of windows that we support, and show how these windows can be used to advantage by a simple hypermedia system. Finally, we explain the underlying system architecture and describe our current implementation.

Figure 1: User of our window system wearing a see-through head-mounted display. The large black cube and white triangle are the transmitters for two 3D tracking systems. Tracker receivers are worn on the head-mounted display, waist, and wrist.

RELATED WORK

Fisher et al. [7] describe a virtual environment in which the user can be presented with a collection of virtual information display windows and input control panels. Other groups have also built virtual world systems that support the creation of general purpose virtual control panels [2, 14]. In general, the extremely low resolution currently provided by the wide field-of-view, opaque, head-mounted displays used in most virtual environments projects has understandably discouraged researchers from porting or developing their own full-fledged window systems. Because the head-mounted display that we are using subtends a relatively small visual angle, what we sacrifice in field of view is compensated for by pixels that are sufficiently small to accommodate the detailed text and graphics displays that are typical of 2D window system applications.

We previously developed an X window manager that allows users to move regular X windows freely between a flat panel display and a much larger virtual surround in which it is embedded, presented on our see-through head-mounted display [5]. We refer to that system as a *hybrid user interface* because it merges two different kinds of display and interaction technologies. We run X on the flat panel and extended a regular window manager so that it maintains the virtual surround using a simple set of graphics routines that draw skeletal outlines of the windows with named title bars. The ideal approach would be to have a single X environment that seamlessly encompasses both the flat panel display and virtual surround. However, the pixels on both displays are of different size and aspect ratio (our head-mounted display has nonsquare pixels), so the same window would need to be of different pixel resolution on each display for it to appear to be the same size. Since X supports only a single address space of uniformly sized pixels, we could not support a single X environment across the flat panel and the virtual surround without either customizing each client or interpreting all X commands at both resolutions.

The system described here differs in several significant ways from this previous work. First, we show how to provide full X functionality on a head-tracked, see-through head-mounted display. Building on top of X, we support three different kinds of windows, including ones fixed to the head-mounted display itself, ones fixed to a virtual surround that the user carries about, and ones fixed to movable objects in the 3D world. While our previous work tracked only head orientation relative to the flat panel display, our current system takes into account the position and orientation of the user's head and body, and of objects in the world. We have implemented our system using an efficient multilayer bitmap software architecture that composites bitmaps at interactive frame rates. Finally, we demonstrate our system with a hypermedia system that we have developed. (Other researchers have also suggested the benefits of incorporating hypermedia capabilities in virtual worlds [12, 8].)

While preparing the final version of this paper, we learned of two other ongoing projects that have goals similar to ours. Dykstra [4] has modified an X server so that its entire display can be texture-mapped on a 3D polygon displayed on a high-performance graphics workstation. Current hardware-supported texture-map size is quite small compared to typical X display resolution, however, and texture-map preprocessing is still too slow to support real-time modifications to the X display. Reichlen [11] uses a high-resolution, head-tracked, head-mounted display to index into a large X bitmap, much as we do, and achieves better real-time performance through the use of custom hardware. However, as in our earlier hybrid user-interface window manager, his system ignores head position. The user is assumed to be stationary, and is seated in a rotatable swivel chair in the center of the surround. Reichlen's head-mounted display is opaque, and his system makes no attempt to correlate windows with objects or positions in the surrounding 3D world.

SYSTEM OVERVIEW

As shown in Figure 1, our user wears a see-through head-mounted display, based on a Reflection Technology Private Eye 720×280 resolution display with a memory-mapped frame buffer. The display's bilevel red image is reflected by a mirror beamsplitter that merges the image with the user's view of the physical world. The head-mounted display is equipped with the receiver for a 3D tracking system that reports the position and orientation of the user's head, making it possible to change the information being presented based on this data. As described below, we also track the user's body and hand, and selected 3D objects. The white triangle and the large black cube in Figure 1 are the transmitters for two different 3D trackers: a Logitech Red Baron ultrasonic system and an Ascension Technology Extended Range Flock of Birds electromagnetic system. Using different tracker technologies allows us to trade off their relative advantages. For example, the ultrasonic system is not sensitive to the presence of metallic objects and magnetic fields, as is the magnetic system, whereas the magnetic system does not require a clear line of sight between transmitter and receiver, as does the ultrasonic system. (In Figure 1, the ultrasonic tracker is being used for the head, and the electromagnetic tracker for the body and hand.)

Because of the relatively small display, and our desire to present a large, encompassing environment, we take advantage of our head-tracking facilities and use the orientation of the user's head to index into a much larger information space than could be presented at one time on the display. This information space, the rectangular bitmap maintained by the X server, is mapped onto a portion of a sphere positioned about the user's head, just like the information surround of our earlier work [5]. To avoid confusing singularities, we use a relatively small portion of the sphere, roughly 170° longitude by 90° latitude, corresponding to a 6K by 2K bitmap (note that the display pixels are unfortunately nonsquare). As before, we make use only of yaw (rotation about an axis up through the neck, as in shaking the head "no," corresponding to the *x* coordinate, which is mapped to longitude) and pitch (rotation about an axis through the ears, as in shaking the head "yes," corresponding to the *y* coordinate, which is mapped to latitude). We ignore roll (rotation about an axis from the front through the back of the head), which would require rotating the 2D bitmap to support. Thus, at any given time, the user sees an upright rectangular portion of the bitmap, providing a piecewise rectangular approximation to a spherical surround.

As described so far, in this model the absolute orientation of the user's head in the environment would determine which part of the surround is visible. Because our user is free to roam within the range of the tracking system, this model is often undesirable: the direction in which the user is facing would impose physical limits on how the head can be comfortably oriented, making it difficult to see parts of the surround without turning around. For example, since the window system's bitmap is mapped to a relatively small portion of the surround's sphere, if the user were facing in the "wrong" direction, information could be displayed only behind the user. To avoid this problem, we have also outfitted the user's body with an additional tracker positioned at their waist. We use the difference between the head-tracker and body-tracker orientation to determine which portion of the surround is mapped to the display. This models a surround that is fixed to the user's body, rather than to the world, a sort of virtual "portable desk" that is always in front of the user.

TYPES OF WINDOWS

We have developed support for three kinds of windows: surround-fixed, display-fixed, and world-fixed.

Surround-fixed windows. We refer to windows that are displayed at a fixed position within the surround as *surround-fixed* windows. These are the most commonly used windows in our system and are not intended to have a specific relationship to the physical world. As the user looks around, the portion of the surround (and its surround-fixed windows) that is visible changes.

Display-fixed windows. Quite a lot of head motion may be needed if we are interested in the relationships between two or more distant surround-fixed windows or if we would like to make frequent use of a particular surround-fixed window as we look around. Therefore, we have developed support for *display-fixed* windows that are positioned at a fixed location relative to the display itself, no matter how the user's head is oriented. (These windows would be the default if the entire bitmap—or the same part of it—were always mapped to the display.) A precedent exists for display-fixed windows within window managers such as *vtwm* [16] that support a virtual desktop that is larger than the physical display. In these systems, display-fixed windows, such as a control panel of window names, are implemented conventionally, whereas the illusion of a larger desktop is provided by actually moving the regular windows across the X bitmap as the user scrolls the display. Since we need to maintain a high frame rate, executing opaque moves for either kind of window is undesirable.

Therefore, we have implemented a compositing approach, described later, that overlays desired windows at specified locations in the coordinate system of the display.

World-fixed windows. Just as we may wish to fix some windows to the display, we may wish to fix others to locations or objects in the 3D world. We call these *world-fixed* windows. Our current implementation supports world-fixed windows by allowing users to specify a known object by name or a location by pointing in 3D. Taking into account the position and orientation of the user's head and the orientation of the user's body, a specified window is moved to an appropriate place in the X bitmap using regular X facilities (but see the conclusions). In general, if the user moves, the position in the bitmap must also change because it is a projection onto the virtual surround of a vector from the user's eye to the 3D window position. Furthermore, since we allow objects to be tracked, if the window is attached to a moving tracked object, the window must be moved as well. Since all windows, including world-fixed windows, are displayed by indexing into the X bitmap, each is always perpendicular to the user's direction of gaze and upright relative to the user's head.

A HYPERMEDIA APPLICATION

To demonstrate the utility of our system, we have developed a simple hypermedia application that supports the ability to make links between arbitrary X windows and to attach windows to objects and locations. To support the concept of linking as a universal system-wide resource [10], we use a display-fixed window for the hypermedia control panel, which allows us to make, break, and follow links. This assures that the control panel is always available wherever the user is looking.

Figures 2 shows a portion of the surround, directly in front of the user, who is looking at the two tracker transmitters. (This and all subsequent photographs were photographed directly through our see-through head-mounted display.) The wide display-fixed window at the bottom of the figure is the hypermedia control panel. The two other windows visible are a pair of *xeyes* and part of a weather map. Figure 3 shows another portion of the surround, seen from the same position, but a different orientation. Here we see the righthand side of the weather map and part of the manual entry for the program used to display it.

Linking two windows causes an arc to be drawn between them, and makes it possible for the link to be followed later to cause a linked window to be displayed and brought to the top of the window stack if it has been iconified or covered. The two windows in Figure 3 have been linked, as indicated by the diagonal arc drawn between them. We use a simple link manager that maintains a database of links between windows. To support linking to or from a physical object or 3D location, a transparent world-fixed window is associated with the object (as described later) and becomes the destination or source of the link, as appropriate. If the object is tracked, the window will move in the surround with the object, and the link will appear to follow the object. We have outfitted the user's hand with a 3D tracker. To specify a 3D location, the user selects a button from a submenu of the control panel using the mouse and then points to a location by moving their tracked hand and clicking the mouse button.

An arbitrary application window can also be positioned at a desired location or relative to a (possibly tracked) object. In Figure 4, we have associated an *xpostit* [3] note with a tracked person, who is wearing a tracker around his neck, and an *xload* load-average meter with the corner of its computer's display. Figure 5 shows the same windows as seen from a different location and after the person has moved. Note that in both cases the windows remain perpendicular to the line of sight of the person wearing the head-mounted display.

IMPLEMENTATION

Our system architecture, shown in Figure 6, has six main components: the X server, the display server, the trackers, the world-fixed window server, the display-fixed window server, and the hypermedia application. Several additional utilities, not shown in the figure for clarity, are discussed later.

X Server

We modified a standard X11 R4 server, running under Mach [1], to use a virtual memory bitmap for the display, instead of the host machine's console. This allows us to create an arbitrarily large display, limited only by available memory. The X display bitmap is made available to our display server as a shared memory bitmap through the file system using the UNIX *mmap* facility. This is the only modification that we made to the server.

Display Server

The display server is written in C and runs under Mach on a 50 MHz Intel 486DX-based PC that supports the Private Eye display entirely in software. The display server was originally written to allow simple wire-frame and polygonal 3D graphics to be displayed on the Private Eye for use in the KARMA augmented reality system [6]. It has been enhanced to allow an arbitrary number of *overlays* to be placed on top of the original graphics display.

An overlay is defined by specifying a rectangular *viewport* on the Private Eye display, a *bitmap* to be overlaid, a 2D *offset* into the bitmap and a *raster operation* (e.g., *copy*, *xor*, *or*, etc.). The viewport-sized portion of the bitmap, whose upper left corner is specified by the offset, is overlaid onto the display by combining it with the image under the viewport using the specified raster operation. The redisplay process is optimized by creating a display list that takes advantage of the fact that certain raster operations are *opaque*, meaning they ignore the image under the overlay. As a result, only those portions of the original graphics screen and the overlay bitmaps that are actually visible in the final image need be examined. An overlay's index number indicates its priority relative to the other overlays.

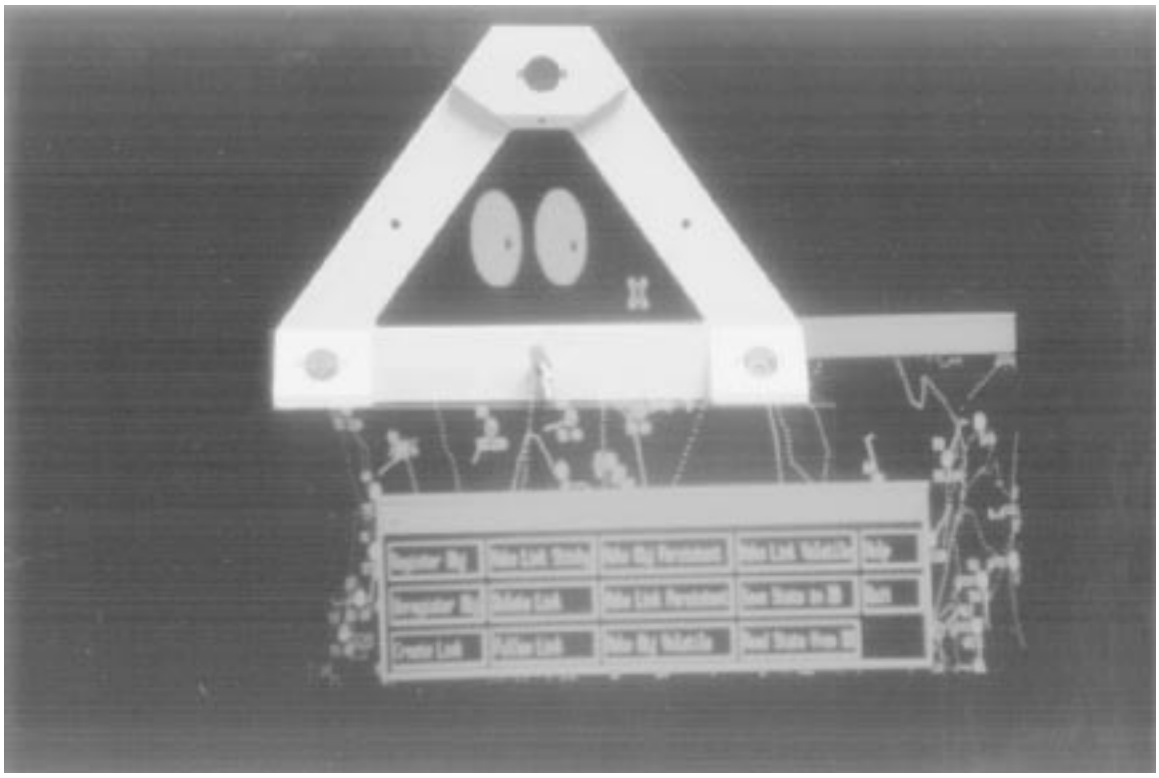

Figure 2: View of part of the surround seen through the head-mounted display.

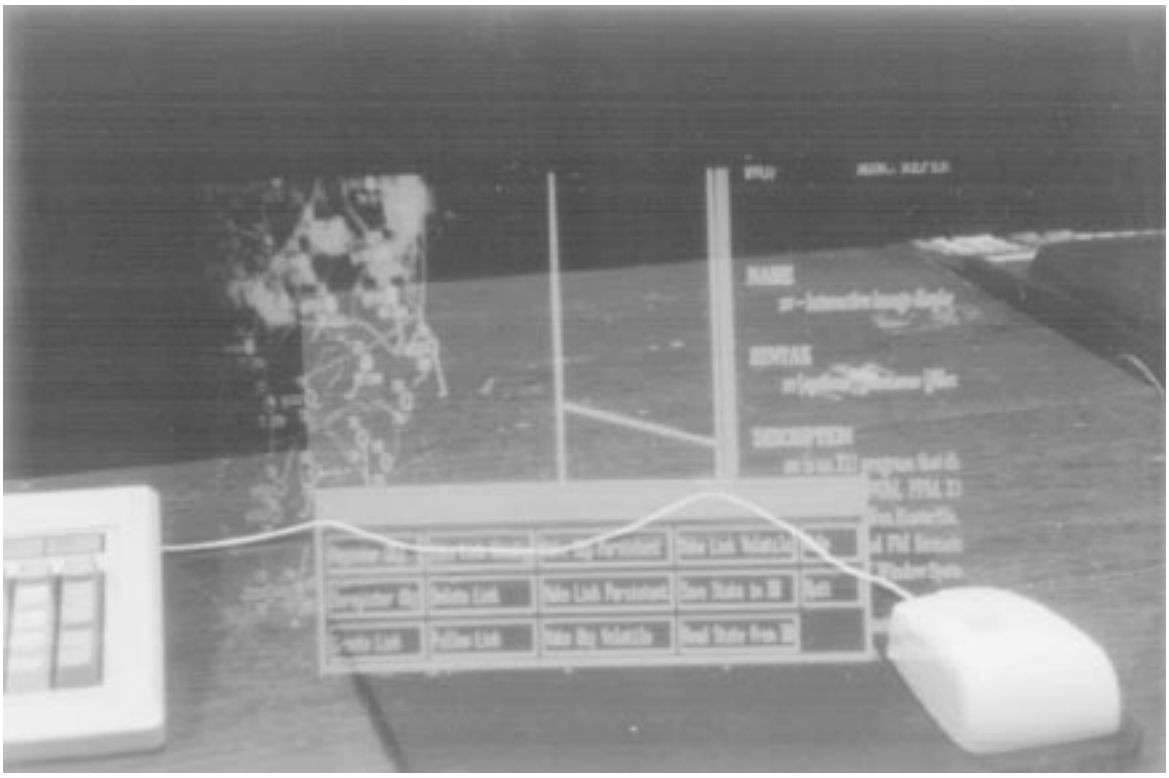

Figure 3: View of part of the surround from the same position as Figure 2, but a different orientation.

Further refresh optimizations are accomplished by noting that each write or read of the Private Eye's memory-mapped frame buffer involves a substantial number of wait states. To avoid the overhead that would be associated with copying an entire new frame to the Private Eye's frame buffer each time, we maintain a buffer in main

Figure 4: A note is attached to a tracked person and a load-average meter to a 3D location.

Figure 5: The note and load-average meter of Figure 4 seen from a different position and after the person has moved.

memory that contains a copy of the Private Eye's frame buffer. Each word in the new frame is compared with the buffer, and those words that have changed from the previous frame are copied to the Private Eye's frame buffer.

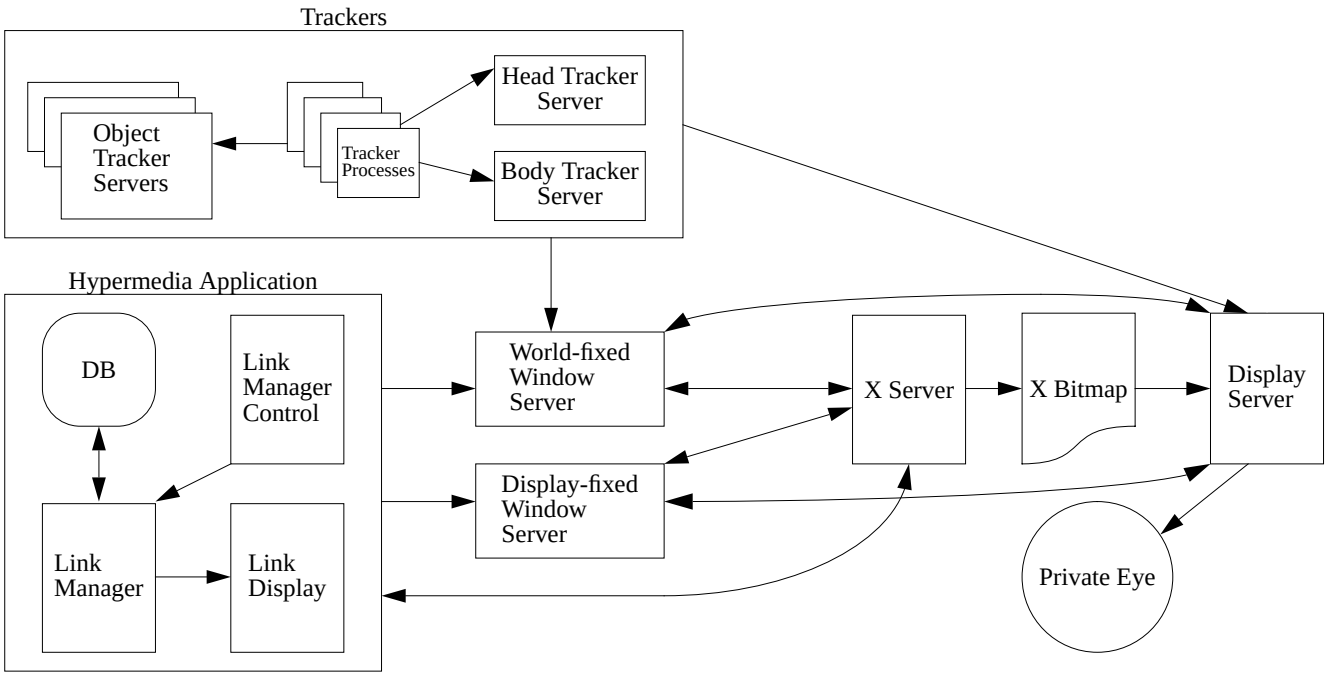

Figure 6: System architecture.

Our display list is organized so that all references to locations in the overlay bitmaps are relative to the 2D bitmap offset. This allows us to pan around an overlay bitmap without incurring the overhead of rebuilding the display list. For example, by having the head and body tracker servers store their position and orientation in the display server, the offset into the X bitmap overlay is recomputed each time the screen is refreshed, based on the user's head and body orientation, as discussed below.

Since the orientation values are changing constantly because of tracker noise and small movements of the user's head and body, the server performs *smoothing* and/or *thresholding* on the changes to the overlay offset resulting from head and body motion. In both cases, the difference between the current offset and the newly computed offset is examined. With *smoothing*, if this difference is less than a specified amount, the offset change is spread over successive refresh frames. *Thresholding*, on the other hand, simply ignores the new offset if the difference is below a specified threshold.

Smoothing, as the name implies, was designed to smooth out small changes and thus eliminate the jitter caused by tracker noise and small head and body movements. However, because it constantly adjusts the offset in the bitmap by a small amount, smoothing adversely affects the performance of the double buffering scheme. Thresholding attempts to eliminate jitter by ignoring small changes. However, if the threshold is sufficiently large to remove the jitter caused by tracker noise, intentional small head movements are ignored. A combination of the two techniques seems to work best. When combined with Kalman filtering of the head tracker motion (see the section on trackers), the image being viewed remains extremely stable when the user is not moving, but is responsive to large movements.

Projecting the Surround

There are two similar projections of interest here. First, we need to project the appropriate part of the X bitmap onto the head-mounted display to create the illusion of a virtual surround. Second, we need to project world-fixed windows onto the X bitmap so that they appear in the correct place on the virtual surround.

As mentioned above, the head-tracker and body-tracker servers store their position and orientation in the display server so that the offset into the X bitmap overlay may be recomputed each time the screen is refreshed. Computing the offset from these values is a two-step process. First, the user's *view direction vector* is determined by applying two transformations to the *z* axis: the *surround viewing transformation* and the *centering transformation*. The surround viewing transformation is the composition of the head-tracker orientation and the inverse body-tracker orientation and gives us the direction of the user's head relative to their body. The centering transformation is initially the identity transformation. Whenever the user requests that the virtual surround be centered about the current viewing direction, the system saves the inverse of the current surround viewing transformation as the new centering transformation. Second, the view direction vector is converted to polar coordinates. The two angular components are multiplied by the number of pixels per degree of horizontal or vertical visual angle (determined for each user during calibration) to compute the offset into the X bitmap (the *surround view offset*).

Projecting world-fixed windows onto the X bitmap is accomplished similarly, with two differences. First, the *window direction vector* (analogous to the view direction vector) is determined by applying two transformations to the vector from the viewers eye to the window's 3D position: the *window transformation* (analogous to the surround viewing transformation) and the *centering transformation*. The window transformation is simply the inverse body-tracker orientation, as the orientation of the user's head does not affect the projection of an object onto the surround. The centering transformation is the same as before. The second difference is that we can no longer ignore the third orientation component of the user's head, the *roll* or twist along the *z* axis. To account for this, we rotate the 2D projection of a window on the surround around the current surround view offset by the inverse of the roll of the user's head.

Trackers

The body and object servers, and the associated lower-level tracker processes, are written in C. We typically run them on other workstations to avoid imposing a large load on the machine that runs the X server and display server. The tracker servers provide a uniform, high-level interface to the different tracking systems. All trackers report their position and orientation to the world-fixed window server. The head and body servers also update their position and orientation in the display server.

We use a Kalman filter [9] in the head tracker to smooth the motion and decrease lag. The head server is also calibrated to report the position of the user's eye, as opposed to the position of the physical tracker.

Display-Fixed Window Server

Display-fixed window support consists of two separate components: the compositing of a specified area of the X bitmap into a fixed area viewport of the display and control of the X cursor to provide the user with the illusion that the display-fixed windows are actually where they appear to be on the X display.

Compositing the windows onto the display is handled by placing the X windows in a portion of the X bitmap not visible anywhere on the virtual surround and adding two display-sized overlays in the display server with priorities higher than that of the X bitmap's overlay. The first of these overlays is used to mask out those portions of the display containing display-fixed windows. Its bits are zero at pixels occupied by a display-fixed window and one at all other pixels. The second overlay is used to *or* display-fixed windows on to the display. It references the portion of the X bitmap in which the display-fixed windows reside. Together, both overlays create the appearance of opaque display-fixed windows. The mask overlay is contained in shared memory and is updated by the display-fixed window server to reflect the current window structure of the display-fixed portion of the X bitmap.

To handle the interaction of the cursor with the display-fixed windows, the display-fixed window server watches all X cursor motion. When the cursor enters a portion of the display in which a display-fixed window is visible, the cursor is warped to the actual window position in the X bitmap. When the cursor leaves the actual window boundaries, the cursor is warped back to the appropriate visible part of the user's display. This is accomplished quickly by testing the appropriate bit in the mask overlay. To the user, it simply appears that the cursor moves in and out of the display-fixed windows.

This solution is sufficient for most applications. However, it falls short of providing the user with a completely transparent implementation of display-fixed windows. For example, display-fixed windows do not interact with the window manager properly since their visibility priority is always higher than that of any surround-fixed or world-fixed windows. While it would be simple to implement a scheme where the stacking order was taken into consideration when creating the mask overlay, allowing display-fixed windows to appear behind world-fixed and surround-fixed windows, we chose not to do so for performance reasons. Another shortcoming is the result of our decision to suspend cursor warping when any mouse button is depressed. This was done to prevent unexpected results from occurring during window manager operations such as window moving and resizing. While the results are reasonable in most situations, there are occasional surprises, such as when the cursor disappears behind a display-fixed window while moving a surround-fixed window.

World-Fixed Window Server

The main responsibility of the world-fixed window server is to maintain a database of known objects in the physical world. The object state information retained by the server includes whether or not the object is tracked, its current 3D location, and the 2D X address of a small, transparent, input-only *proxy window* created by the world-fixed window server. Proxy windows provide client applications with a convenient method to determine quickly the projection of a physical object on the X display. The server continuously updates the position of an object's proxy window based on the user's head and body positions and the position of the object. Additionally, the server allows clients, such as our hypermedia application, to attach X windows to each object. The server ensures the location of each attached X window is consistent with the projection of its associated object on the virtual surround.

Hypermedia Application

The hypermedia application consists of a link manager, link database, link manager control, and link display facility.

The *link manager control* provides an interface to the linking subsystem via the *link manager control panel*, which is presented as a display-fixed window. Using the control panel, arbitrary, hypertextual links may be placed between any two windows or physical objects in the virtual surround. Basic features include following links and deleting

links, and persistent storage of objects and links in the *link database*.

The engine for the linking subsystem is the *link manager*. The link manager processes the actions requested by the user via the link manager control panel. The link manager internally maintains state information of all objects and links in the user's environment. This information may be read from or stored into the link database to provide consistency between sessions.

Links between windows and/or objects (represented by their proxy windows) are indicated by drawing an arc between them. This function is carried out by the link manager. The arc is refreshed whenever either of its endpoints is moved and is removed when the window associated with either of its endpoints exits. Currently, links are drawn directly upon the root window. The link line is not drawn over inferiors of the root window. No damage events occur in any window other than the root window when link lines are added or removed.

Miscellaneous Utilities

Unlike our previous work [5], no changes to the window manager are required.[1] We currently run an unmodified version of the *mwm* window manager. To accomplish additional window-manager-like activities, such as moving the cursor to the center of the area currently being displayed by the display server, we bind simple utility programs to function keys using the window manager. These utilities contact the various servers, depending on their purpose.

For example, to move the cursor to the center of the displayed area, the *movetocenter* program queries the display server to determine the part of the X bitmap that is visible, and moves the cursor to the center of this region. The *setthreshold* and *setsmoothing* utilities adjust the smoothing and threshold values in the display server. The *centerview* program contacts the display server to center the virtual surround about the user's current viewing direction. The *togglefixed* program contacts the display-fixed window server to toggle a selected window in and out of display-fixed mode.

Performance

We currently achieve between 6 and 20 frames per second, double-buffered, for the figures in this paper, which use three overlays (one for the main part of the surround, and two for the display-fixed windows, as described in the section on the display-fixed window server). The exact frame rate depends on whether or not there are display-fixed or world-fixed windows. If there are no display fixed windows, the display-fixed window overlays are not needed, increasing the frame rate about 10 frames per second. If there are world-fixed windows, they must be continuously moved, forcing the X server (which runs on the same processor as the display server) to consume a significant amount of processing time. Unfortunately, with even one world-fixed window, the amount of moving and redrawing performed by the X server is substantial, reducing the frame rate from 18–20 to as low as 6–10 frames per second.[2]

CONCLUSIONS AND FUTURE WORK

We have described an approach to presenting full X window system functionality on a head-tracked, head-mounted display. Minimal server modifications were needed to allow the server to create an arbitrarily-sized X bitmap and make it accessible to others. We developed a fast software display server that supports multiple overlaid bitmaps and the ability to index into and display a selected portion of a larger bitmap. Coupled with tracking of the user's head, body, and hand, and objects in the world, we used this to support windows that were fixed to the display, to an information surround, and to the 3D world.

An important question to ask about any user interface that uses both experimental hardware and software is what stands between the current implementation and a practical system? We see a number of practical limitations, that we expect will be overcome during the next five to ten years. Our head-mounted display, although relatively lightweight (14 oz.) compared to commercial opaque systems is still relatively heavy and socially unacceptable in appearance. Its image is dim and small (22° horizontal field of view). Although its focus is user-settable, it can only be adjusted manually, limiting what can be in focus in the physical and virtual worlds simultaneously. The short range and relative inaccuracy of the 3D trackers restricts the workspace within which a user can roam (currently a 12' square), as does the tether from our display and trackers to what are currently nonportable workstations. We note, however, that our current frame rate is quite comfortable, yet is supported entirely in software on an inexpensive commodity personal computer.

There are a number of directions in which we are interested in taking this work. For example, in theory, display-fixed and world-fixed windows could both be implemented by the same mechanism (both through X or through overlays). However, in our current system, as mentioned above, initial set-up time is required each time an overlay is added, deleted, or changed in location or size. Therefore, changing the location of an overlay dynamically has a significant transient impact on the frame rate. We expect to improve our implementation to reduce this impact, however, and are interested in comparing the two approaches and their impact both on performance on the user interface. Compositing is clearly fastest, but does not behave like X in the sense that each overlay maintains its priority. Using X to do the moving means that the windows act like X windows,

[1] Nevertheless, there are several places in which window-manager modifications would be extremely helpful (e.g., to assure that window-manager dialogue boxes always appear in a visible location) [11].

[2] We expect to improve this significantly by using thresholding when tracking world-fixed windows.

but damage repair can exact a significant performance penalty. One particularly interesting prospect would be to write an X window manager used our display manager to support all window movement operations through real-time compositing. (A commercially available example of this approach was the Lexidata Lex90's hardware window system, developed in the early 1980's.)

Our current support for showing links between windows is extremely unsatisfactory since they are drawn only on the root window. Although we could refresh them whenever they are overwritten by window movement, we instead intend to support link display by means of an additional overlay that is *or*'ed on top of our other overlays. Rather than using the stand-alone 3D graphics primitives supported by the display server, this overlay could be the bitmap of a separate output-only X server.

One of our most important directions will involve incorporating our X support with the knowledge-based 3D graphics generated by our KARMA augmented reality system [6]. This will make it possible to integrate 2D windows with virtual as well as physical objects. The display software facilities to support this are already in place since the display server is an enhanced version of the server originally developed for KARMA. We are also adding 3D spatial sound [15] to KARMA, and expect to explore its implications for our X environment. For example, X activity in windows that are not currently within the user's field of view may be indicated by appropriately positioned sonic cues intended to direct the user's attention in the appropriate direction.

ACKNOWLEDGMENTS

Research on this project is supported in part by the Office of Naval Research under Contract N00014-91-J-1872, a gift and summer internship from NYNEX Science & Technology, the New York State Center for Advanced Technology in Computers and Information Systems under Contract NYSSTF-CAT-92-053, the Columbia Center for Telecommunications Research under NSF Grant ECD-88-11111, NSF Grant CDA-92-23009, and equipment grants from the Hewlett-Packard Company and Ascension Technology. Thomas Magdahl built the camera mount used to take pictures through our head-mounted display.

REFERENCES

[1] M. Accetta, et al.
Mach: A New Kernel Foundation for UNIX Development.
In *Proc. Summer Usenix*, pages 93–112. July, 1986.

[2] Butterworth, J., Davidson, A., Hench, S., and Olano, T.
3DM: A Three Dimensional Modeler Using a Head-Mounted Display.
In *Proc. 1992 Symp. on Interactive 3D Graphics (Special Issue of Computer Graphics)*, pages 135–138. Cambridge, MA, March 29–April 1, 1992.

[3] Curry, D.
Xpostit: X window System Post-It Notes (UNIX Man Page).
West Lafayette, IN, 1991.

[4] Dykstra, P.
X11 in Virtual Environments.
In *Proc. IEEE 1993 Symposium on Research Frontiers in Virtual Reality*. San Jose, CA, October 25–26, 1993.

[5] Feiner, S. and Shamash, A.
Hybrid User Interfaces: Breeding Virtually Bigger Interfaces for Physically Smaller Computers.
In *Proc. UIST '91 (ACM Symp. on User Interface Software and Technology)*, pages 9–17. Hilton Head, SC, November 11–13, 1991.

[6] Feiner, S., MacIntyre, B., and Seligmann, D.
Knowledge-Based Augmented Reality.
Communic. ACM 36(7):52–62, July, 1993.

[7] Fisher, S., McGreevy, M., Humphries, J., and Robinett, W.
Virtual Environment Display System.
In *Proc. 1986 Workshop on Interactive 3D Graphics*, pages 77–87. Chapel Hill, NC, October 23–24, 1986.

[8] Fitzmaurice, G.
Situated Information Spaces: Spatially Aware Palmtop Computers.
Communic. ACM 36(7):38–49, July, 1993.

[9] Liang, J., Shaw, C., and Green, M.
On Temporal-Spatial Realism in the Virtual Reality Environment.
In *Proc. UIST '91 (ACM Symp. on User Interface Software and Technology)*, pages 19–25. Hilton Head, SC, November 11–13, 1991.

[10] Meyrowitz, N.
The Missing Link: Why We're All Doing Hypertext Wrong.
In Barrett, E (editor), *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, pages 107–114. MIT Press, Cambridge, MA, 1989.

[11] Reichlen, B.
Sparcchair: A One Hundred Million Pixel Display.
In *Proc. IEEE VRAIS '93*. Seattle, WA, September 18–22, 1993.

[12] Robinett, W.
Synthetic Experience: A Taxonomy.
Presence 1(2):229–247, Spring, 1992.

[13] Scheifler, R., and Gettys, J.
The X Window System.
ACM Trans. on Graphics 5(2):79–109, April, 1986.

[14] Shaw, C., Green, M., Liang, J., and Sun, Y.
Decoupled Simulation in Virtual Reality with the MR Toolkit.
ACM Trans. on Information Systems 11(2), July, 1993.

[15] Wenzel, E., and Foster, S.
Realtime Digital Synthesis of Virtual Acoustic Environments.
In *Proc. 1990 Symp. on Interactive 3D Graphics (Computer Graphics, 24:2, March 1990)*, pages 139–140. Snowbird, UT, March 25–28, 1990.

[16] Williams, N. and Edmondson, D. (with LaStrange, T.).
vtwm documentation.
1990.