Annotating the Real World with Knowledge-Based Graphics on a See-Through Head-Mounted Display

Steven Feiner
Blair MacIntyre
Dorée Seligmann

Department of Computer Science
Columbia University
New York, New York 10027
{feiner, bm, doree}@cs.columbia.edu

Abstract

We describe an experimental, knowledge-based, virtual-world system that uses a monocular "see-through" head-mounted display to overlay graphics on the user's view of the real world. In a simple equipment maintenance domain that we have developed, the overlaid graphics include 3D representations of actual physical objects, textual annotations and callouts, and virtual metaobjects, such as arrows. A set of 3D position and orientation sensors monitor the user's head and several of the objects in the world. Our system includes a knowledge-based graphics component that determines what information to present, a display-list–based display server that represents the models to be displayed, and a set of sensor servers that track the user and selected objects. The sensor servers directly modify the object transformations and viewing specification in the display list. The knowledge-based graphics component also receives the sensor data and uses it to redesign the information being presented.

Résumé

Nous décrivons un système graphique expérimental à base de connaissances, qui enrichit la vision du monde perçue par l'utilisateur en super-imposant des graphiques générés par le système. L'utilisateur porte une lentille monoculaire semi-reflective sur laquelle s'affichent les images que notre système décide de montrer. Nous avons développé le système dans le domaine de la maintenance d'équipement. Dans ce domaine assez simple, les graphiques superimposés comportent des représentations 3D des objets réels, des annotations textuelles, et des "méta-objets" virtuels (par example, des flêches). Un ensemble de détecteurs 3D déterminent l'orientation et la position de la tête de l'utilisateur et de certains objets dans l'environnement. Notre système comprend trois composants principaux: un système de génération de graphiques à base de connaissances détermine non seulement quelles informations doivent être présentées mais aussi comment les présenter; un gestionnaire d'écran à base de liste d'objets représente les modèles des objets à afficher, et un ensemble de serveurs de détecteurs qui suivent la tête de l'utilisateur et certains objets. Les serveurs de détecteurs modifient la position des objets et les spécifications de la scène directement dans la liste des objets. Le composant graphique à base de connaissances reçoit aussi les valeurs retournées par les détecteurs et utilise cette information pour réviser l'information à présenter.

Keywords: knowledge-based graphics, virtual worlds, head-mounted displays, heads-up displays, augmented reality

1 Introduction

Virtual worlds that use 3D displays and interaction devices have the potential to make possible greatly improved explanations of complex physical tasks. These technologies promise to allow a user to view and manipulate information in ways that better exploit our ability to interact with 3D spatial information than does the use of flat 2D displays and input devices. As the richness and variety of the information that a system can present to a user increases, however, so does the difficulty of designing the presentation. Desktop publishing systems require more skill and time to master than do simple word processors; multimedia presentation editors demand yet more design expertise to use effectively. Perhaps the ultimate design demands are posed by virtual worlds, which can require the coordinated design of material that affects all sensory modalities, and that must respond continuously to the user's interactions.

Over the past years, we have been developing knowledge-based systems that address the automated design of presentations that explain how to perform simple 3D tasks. These systems generate static and animated graphics [6, 22, 14], and multimedia presentations [8] that satisfy a high-level expression of the information to be communicated. Here, we describe some first steps that we have taken toward building a testbed system for exploring the automated design of virtual worlds to explain maintenance and repair tasks. In this work, we are concentrating on the knowledge-based generation of graphics that overlay the user's view of the physical world, and which dynamically take into account information about the user, task, and changes in the surrounding world.

Ivan Sutherland, in his pioneering research on head-mounted displays, developed a binocular "see-through" system [23]. Each eye viewed a miniature vector CRT, whose synthesized

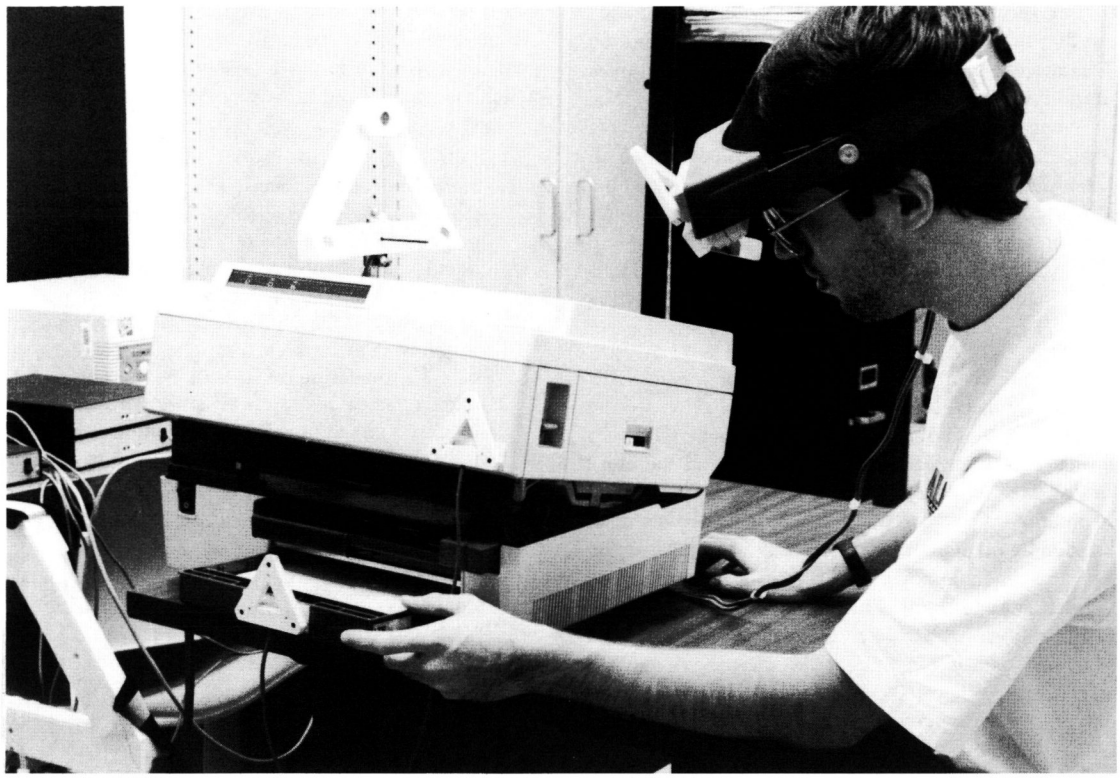

Figure 1: Prototype see-through head-mounted display in use in testbed laser printer maintenance application.

graphics were merged with the user's view of the real world by means of a beam splitter. Since then, other graphics researchers have explored the use of see-through displays, both as part of head-mounted displays (e.g., [11, 4]), and desk-mounted displays (e.g., [15, 21]). We have assembled a relatively inexpensive see-through head-mounted display, using a Reflection Technology Private Eye [20], a Logitech 3D position and orientation sensor [18], and a mirror beam splitter. This system, shown in Figure 1 in use in an experimental maintenance application described below, is a research prototype: it produces a dim, bilevel, monochrome image, and overlays graphics on a relatively narrow portion of the user's visual field, providing approximately a 22° horizontal field of view. Nevertheless, it has proven to be a useful research tool.

In the remainder of this paper, we describe the knowledge-based graphics component that designs what the user sees, a simple maintenance application with which we are experimenting, and the design and implementation of our overall system architecture.

2 Knowledge-Based Graphics Component

2.1 IBIS Overview

The knowledge-based graphics component that we use is based on IBIS (Intent-Based Illustration System) [22]. IBIS is a rule-based system that designs *illustrations*, a term that we use to refer to pictures that are designed to satisfy an input communicative intent. The communicative intent is specified by a prioritized list of *communicative goals*. Each communicative goal specifies something that the picture is to accomplish; for example, to show an object's location or shape. IBIS distinguishes between design and stylistic choice. The *design* of an illustration corresponds to its high-level structure, specifying those visual effects that must be accomplished together to satisfy the illustration's communicative goals; the *styles* used in an illustration represent the different ways each visual effect may be accomplished.

IBIS uses two kinds of rules to design an illustration: methods and evaluators. A *method* specifies how to accomplish a particular style or design; an *evaluator* determines whether a particular style or design has been accomplished. IBIS's rules allow it to examine partial solutions and to backtrack when conflicts occur.

The illustrations that IBIS generates are dynamic, rather than static: IBIS can continuously receive and handle changing constraints and goals, while at the same time realizing a particular intent. For example, IBIS normally determines a viewing specification of its own when designing an illustration. If the viewing specification is provided externally, however, then IBIS attempts to use it in generating a picture that satisfies the goals, and will incrementally redesign the picture if the viewing specification changes. This allows IBIS's user to navigate within the illustrated world by changing the viewing specification. IBIS's navigation facility is

different from that used in traditional "synthetic camera" graphics in that the binding between the input intent and the output illustration is preserved. IBIS continuously examines the illustration as it is altered to attempt to satisfy the illustration's communicative goals. As the user changes the viewing specification, conflicts may occur, causing IBIS to adopt a new design or set of stylistic choices. Thus, the illustrated world itself may change as the user explores it. (A rudimentary precursor of this capability was introduced in early vector systems that made traversal of an object contingent upon the screen size of its precalculated extent, allowing an object's level-of-detail to change with its size [24].)

As a simple example of how the original version of IBIS works, consider the need to maintain object visibility. In the course of satisfying the input communicative goals, IBIS will determine that certain objects must be visible, and therefore *unoccludable*. This typically occurs if the objects participate directly in a communicative goal. Unoccludable objects must not be obscured by others in the world. IBIS can maintain the visibility of unoccludable objects by selecting an appropriate viewing specification for the illustration, by generating an inset sub-illustration, or by altering the appearance of the obscuring objects. For example, IBIS can decide not to render the obscuring objects at all, to render them as partially transparent, or to use a "cutaway" view [9].

2.2 Extending IBIS for a Head-Mounted See-Through Display

In extending IBIS to support a head-mounted see-through display, we have had to take into account a number of important differences:

- The original IBIS assumes that it generates all of what the user sees. In overlaid graphics, however, IBIS must instead enrich the user's view of the world with additional information.
- The user could modify the viewing specification after the original IBIS chose an initial viewing specification. In contrast, when designing overlaid graphics, our extended IBIS must from the beginning relinquish to the user all control of the viewing specification, since only the user can determine where they look.
- The world was assumed to change only after an illustration was completed in the original IBIS. This was easy to enforce since IBIS maintained control over what was visible, and based its illustrations on a world model that was frozen throughout the illustration's life. In contrast, our extended IBIS must take into account ongoing changes in the world as it generates illustrations.
- The original IBIS was responsible for achieving all communicative goals itself. Because the extended IBIS controls only what it generates, the user becomes an active participant in achieving the communicative goals. For example, if a goal is to specify an object's position, and the object's projection does not lie within the window, IBIS must indicate to the user where they should look.

Based on these differences, we have developed a preliminary set of new IBIS rules to handle the input communicative goals. At its core is a set of low-level style rules, each of which may be invoked for any object. These include:

- Make an object *visible*. If the object's projection falls within the overlay window and is not occluded by other objects, nothing need be done: the object is not rendered. (In contrast, in the original IBIS, the object would have to be rendered.) If the object's projection is within the window, but is occluded wholly or in part by other objects, the object is marked for rendering to allow it to be seen through its occluders; the object's rendering style will be determined by other rules. (Occlusion tests are performed with a fast image-precision algorithm described in [9].) If the object does not project to the window, the goal fails.[1]
- *Highlight* an object. If the object is within the window, the object is marked for rendering and tagged with a highlighted style (e.g., solid lines); otherwise, the goal fails.

Representative higher-level design rules that invoke these are:

- *Show* an object. If the object is *visible*, nothing need be done; otherwise, the *find* goal is activated.
- Show the *location* of an object. If the object is *visible*, it is *highlighted*; otherwise, the *find* goal is activated.
- *Find* an object. A callout (currently, a "canned" text string label) is created for the object, fixed in 2D relative to the overlay window. A dotted rubberband leader line connects the callout to the object. (If the object does not project to the overlay window, then the dotted line extends to the edge of the window in the direction of the object, and can be followed by the user to find the object; the callout and its end of the leader line is always in view.)
- Show a *change* to be accomplished in an object's state. This is achieved by generating a "ghost image" of the object in the desired state.
- Show an *action* performed on an object. This is achieved by adding a *metaobject*, such as an arrow, to depict the action (e.g., pushing or opening).
- *Identify* an object. If the object is *visible*, a 3D callout (textual label) is generated near the object; otherwise the *find* goal is activated.

IBIS is initially provided with representations of the physical objects in the environment. It uses these representations both to generate the overlaid graphics and to evaluate it, employing

[1] All goals succeed except where failure is indicated explicitly.

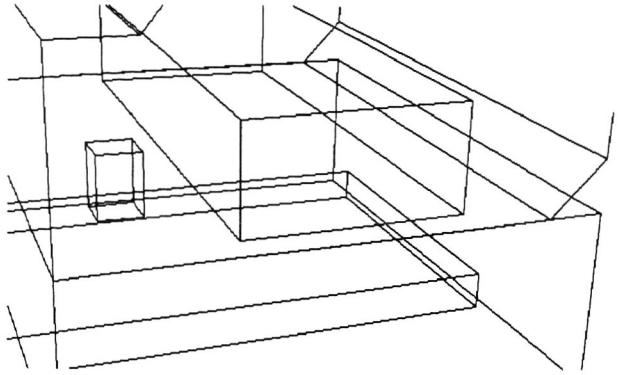

Figure 2: Overlay graphics of simple world model.

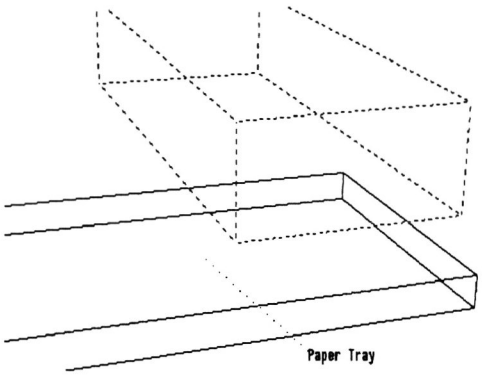

Figure 3: Overlay graphics designed by IBIS to *find* and show *location* of paper tray and *show* toner cartridge.

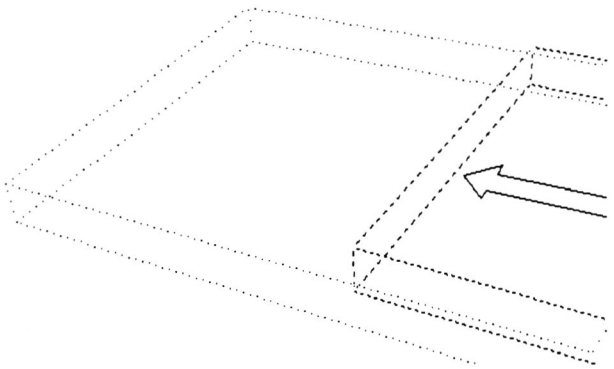

Figure 4: Overlay graphics designed by IBIS to show *action* of pulling out paper tray and resulting *change* in tray's state.

the approaches described in [22, 9].

3 Testbed Application: End-User Laser Printer Maintenance

To test our ideas, we have been experimenting with a simple end-user maintenance application for a laser printer. This entails relatively straightforward operations, such as refilling the paper tray and and replacing the toner cartridge.

Figure 1 shows the system in use. The large white triangles toward the top and bottom of the figure are Logitech sensor transmitters, which have three small ultrasonic sources near their vertices. Small triangles mounted on the display and on the laser printer each contain three microphones. The top transmitter is for the head; the bottom transmitter is for the sensors that track the printer's paper tray and lid.

Figure 2 shows part of our simple laser printer world as displayed in our head-mounted display's overlay bitmap without any design done by IBIS. In contrast, Figure 3 shows an example of an overlay illustration designed by IBIS, using the same viewing specification, to fulfill the *location* and *find* goals for the paper tray and the *show* goal for the toner cartridge. To achieve the *location* goal, the paper tray is highlighted (drawn solid), since the *visible* goal succeeded. To achieve the *find* goal, a 2D label is generated with a dotted leader line that terminates on the tray. This illustration also *shows* the toner cartridge—since it is occluded by the obscuring printer cover, it is drawn with dashed lines to make it visible. Figure 4 *shows* the paper tray, shows the desired *action* of pulling out the paper tray by means of a *metaobject* arrow, and shows the *change* in the paper tray's location that would result as a dotted "ghost" image.

4 System Architecture

The original version of IBIS synchronously designs and then renders each illustration, and then incrementally redesigns the picture in response to user interaction. While the resulting delay is barely tolerable in fully-synthesized animation displayed on a conventional CRT, it is unusable if synthesized graphics must not only change in response to head motion, but must be registered with objects in the real world. In addition, we also need to process data from a number of motion trackers. Based on our own experience [7] and that of other researchers [1, 12, 17], it was clear to us that it would be necessary to distribute processing over multiple processes and processors.

The Private Eye, in its 720×280 resolution mode, presents an interface that appears as a bare single-bit–deep framebuffer. We created a small 3D structured display-list–based *display server*. The server supports line and text primitives, linestyle and font attributes, and a set of structure management operators that allow hierarchical objects to be created, edited, and deleted. Positions in the display server's move and draw commands may be either 2D device coordinates or 3D world coordinates, and may be intermixed in a single primitive, allowing the creation of lines that are anchored to the screen on one end, and to a point in the 3D world on the other. As shown in Figure 5, when the system is initialized, IBIS creates the initial display list by sending a set of object models to the display server.

Each tracker is handled by a low-level *tracker process*. These processes in turn interact with a set of object servers and a head server. Each *object server* is associated with an object in the real world that is monitored with a tracker. At initialization, IBIS provides each object server with the identifier of the display server structure containing the object's vector representation. It also tells the server the position and orientation of the object's tracker relative to the object's coordinate

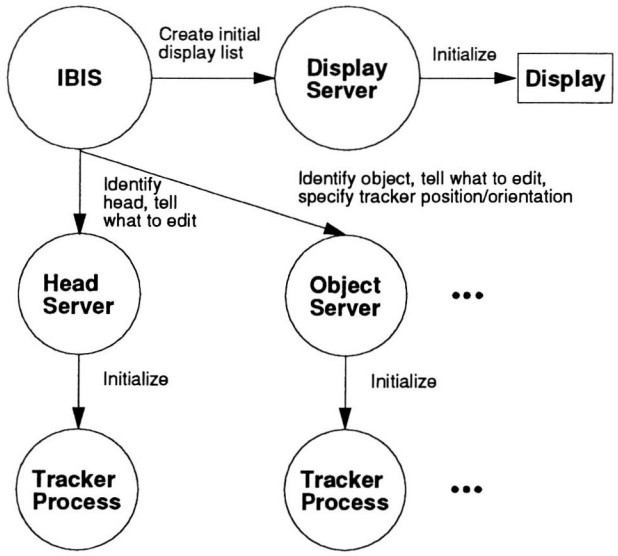

Figure 5: System architecture: Initialization

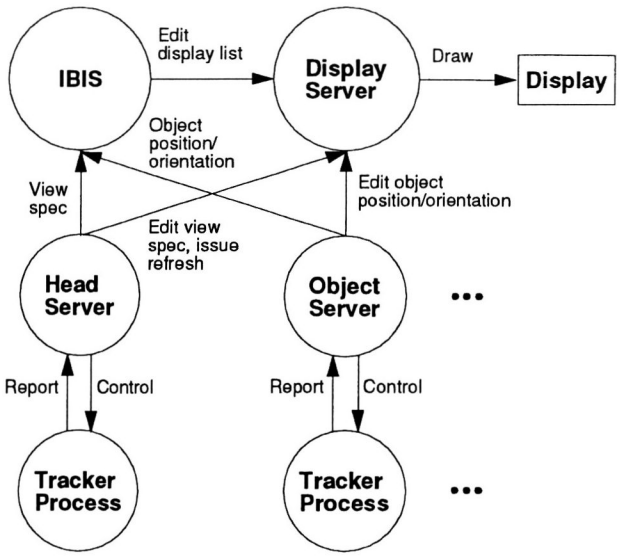

Figure 6: System architecture: Steady state

system. Similarly, IBIS provides the *head server* with the identifier associated with the scene in the display server. (The head and object servers actually use the same executable, so each must also be told whether it is tracking the head or an object.)

Figure 6 shows the system's operation after initialization. The head and object servers are responsible for maintaining the integrity of the object and head motion information that they represent. They edit the display list stored in the display server directly. Each object server regularly edits the display list position and orientation information associated with its object. The head server updates the display list viewing specification for the scene. Both head and object servers also report their information to IBIS. This avoids the delay that would result if IBIS were to serve as a go-between, making possible relatively smooth visual response to head and object motion, while assuring that IBIS always has the latest information on which to base its illustration design.

IBIS determines the presence and appearance of the objects in the display list—all information for which it has not explicitly relinquished control to the head and object servers. This includes the specification of *metaobjects*, such as arrows and text. (IBIS can also dynamically reassign tasks to the head and object servers, a facility that we have not yet used.) In work on the automated design of multimedia presentations, IBIS is presented with a set of communicative goals to satisfy, which are output by other components that determine what to say and which media should be used to say it [8]. In the research reported on here, we have instead used a much simpler content planner to generate the set of communicative goals that IBIS receives.

IBIS first designs an illustration that satisfies the initial set of goals set by its content planner, obeying the constraints imposed by the head and object trackers. Then it loops, using its evaluators to determine whether the current illustration design is satisfactory in the face of changes in the goal set, the user, and the world. Whenever the illustration is determined to be unsatisfactory, IBIS's rules result in modifications that are communicated to the display server. For example, if the 3D label of an object that IBIS determines must be identified no longer lies within the overlaid graphics window, then IBIS will generate a new 2D screen label and 3D leader line that points to the object.

Because IBIS has relinquished to the head and object servers the straightforward matters of display-list traversal and updates of the viewing-specification and monitored object transformations, user interaction with the current illustration stored in the server is fully interactive, and occurs while incremental redesign takes place on a separate processor. We currently achieve about 15 frames per second for a model containing 70 displayed vectors, but with some jerkiness due to network contention.

The head server is responsible for instructing the display server to render the image. Since polling the head's motion tracker requires much less time than rendering the image, we need to avoid building up a backlog for the display server. We accomplish this by having the head server send its commands for the current frame to the display server and then wait until it receives the display server's acknowledgment that the *previous* frame was rendered. This allows rendering and polling to proceed in parallel. All of the actions requested of the display server (e.g., modifying the contents of an object) are processed atomically to ensure that the scene always appears in a valid, drawable state.

5 Implementation

The components of the system run on several different machines under different flavors of UNIX, and communicate through sockets. IBIS is implemented in C++ and the CLIPS production system language [5], and runs under HPUX on an HP 9000 380 TurboSRX graphics workstation, which

provides a fast hardware *z*-buffer–based graphics accelerator that IBIS uses in its illustration design process. The display server is written in C and runs under Mach on an Intel 486-based PC "clone," which supports the Private Eye display entirely in software. (A significant portion of the display server's time is spent implementing double buffering, copying the graphics frame buffer to the Private Eye's frame buffer and clearing it for the next frame.) The head/object server and lower level tracker processes are written in C and C++. Since the tracker hardware requires only an RS232 interface, and the servers can impose a large load on the machine on which they execute, we run them on other workstations.

Our current 6-DOF tracker devices include three Logitech ultrasonic sensors [18] and one Polhemus magnetic sensor [16]. Our software allows both kinds of 6-DOF sensors to be used interchangeably, for example to trade off an ultrasonic sensor's freedom from magnetic interference against a magnetic sensor's ability to work without a direct line of sight to its source.

One important point, that others have noted before [2], is that experimental interfaces that are coupled tightly to the user often require a fair amount of calibration. Our head-mounted display is no exception. Since the generated image must be registered with that of the real world, each user must perform three kinds of calibration:

- *focus.* The Private Eye is focusable from less than a foot to infinity, with each user requiring separate focus adjustments based on their eyesight. After putting on the display, the user must position a slider until a calibration image is comfortably in focus (but see below).
- *visible area.* Due to the physical relationship between the Private Eye, the beam splitter, and the current focus setting, a small portion of the display may not to be visible to a viewer. Therefore, we request that the user determine the viewable area by adjusting the size and position of a visible rectangle until it is as large as possible. This establishes a "safe-title" area that is communicated to IBIS, in which IBIS can assume that anything that is drawn will be visible.
- *viewing specification.* To register the image with the world, the user must first physically adjust the display on their head. Next, they must register a virtual object with its corresponding physical object, which in turn must be viewed in a known position relative to the user. The software takes into account the difference in position and orientation of the user's eye and the measured position and orientation of the sensor.

Registration is a serious issue: Our current motion trackers have neither the spatial nor angular resolution needed to register graphics precisely with the surrounding world. However, we have yet to try mapping sensor inaccuracies to correct for nonlinearities [21], and believe that we can also improve accuracy with a better calibration strategy. (Registration is currently off by about an inch in world coordinates in Figures 2–4.)

We have also found focus to be a particular problem in our current application. We originally built our head-mounted display for a *hybrid user interface*, which embeds the user's view of a "notebook" flat-panel display inside a partial spherical information surround presented on the head-mounted display [10]. The flat-panel display is relatively small and tangent to the sphere, which is centered about the user's head. Assuming that the user's head is stationary, it is easy to adjust the Private Eye so that the material that it displays is focused at the same distance from the user as the flat-panel display. In comparison, "heads up" flight displays are focused at infinity [19], since this is effectively where out-of-the-cockpit targets are located.

In contrast to both, our application requires that graphics be overlaid on nearby objects that necessitate constant changes in visual accommodation to focus in sequence. Since the Private Eye must be focused manually, the need for readjustment as the viewing distance to the object of interest changes is irritating.[2] There is an interesting benefit, however, to the precise focus control provided by the Private Eye. Even though we are currently using a monocular display, when the display is adjusted so that a small synthesized object is in focus at a particular distance, the illusion of it being at the selected position is quite compelling. (Note that one of the difficulties that many users experience in viewing fixed focus stereo displays is developing independence between their ocular convergence and focus accommodation.)

6 Conclusions and Future Work

The work that we have reported on here represents our first steps in designing a testbed for the knowledge-based generation of maintenance and repair instructions using a head-mounted, see-through display. We have developed a preliminary set of rules that allow us to augment the user's view of the world with additional information that supports the performance of simple tasks such as finding designated objects and carrying out simple actions on them. Our software architecture makes a clean distinction between design and rendering to help prevent design decisions from interfering with interactive rendering.

Our experience with the system has suggested many research directions that need to be explored. For example, one important problem is the development of a formal model of how a user's performance will be affected by different decisions made in designing 3D illustrations, taking into account the purpose for which the illustration is generated (specified by our communicative goals), as in the 2D design work of Casner [3].

[2]We have considered automating the process by using a servomotor to adjust the focus to that of a selected object. Note, however, that focus in the entire overlay would be affected uniformly.

Support for visible-line and visible-surface determination is another issue. IBIS currently bases its illustration design in part on whether selected objects are occluded in the current viewing specification, and computes these relationships itself. Our display server, however, does not support visible-line determination. We are particularly interested in incorporating into the display server what Kamada and Kawai [13] refer to as "picturing functions," which determine how a projected line fragment should be rendered, based on the set of surfaces that obscure it. For example, while IBIS currently sets the graphical attributes of an entire object based in part on its visibility, we would like more precise control to be accomplished by the display server, based on the visibility relationships of each line fragment. IBIS would then be responsible for determining the high-level policies used by the display server (e.g., making obscured parts dashed). One challenge is to do this while still maintaining real-time performance. We believe this could be possible on our current hardware if IBIS were allowed to select a subset of objects against which the display server would perform visibility tests. Another intriguing research direction is to explore how to design support facilities that would allow IBIS to specify a rich set of additional high-level policies to be enforced by the display server.

Acknowledgments

Research on this project is supported in part by the Office of Naval Research under Contract N00014-91-J-1872, the Center for Telecommunications Research under NSF Grant ECD-88-11111, NSF Grant CDA-9022123, and an equipment grant from the Hewlett-Packard Company. Work on the original version of IBIS was supported in part by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0165. Our work owes much to the portable computing infrastructure software developed for the Columbia Student Electronic Notebook project directed by Daniel Duchamp, Steven Feiner, and Gerald Maguire. We thank Ari Shamash and Sushil Da Silva for their work on the Logitech 6-DOF tracker process and bitmap scan-conversion package, Brad Paley for sharing his tracker support code, Cliff Beshers for his help in porting his distributed DataGlove server, and Jim Barnes of Logitech for his assistance. Michael Elhadad et Colette Johnen nous ont aidés à traduire le résumé.

References

[1] Blanchard, C., Burgess, S., Harvill, Y., Lanier, J. Lasko, A., Oberman, M., and Teitel, M.
Reality Built for Two: A Virtual Reality Tool.
In *Proc. 1990 Symp. on Interactive 3D Graphics (Computer Graphics, 24:2, March 1990)*, pages 35–36. Snowbird, UT, March 25–28, 1990.

[2] Brooks Jr., F.
Grasping Reality Through Illusion—Interactive Graphics Serving Science.
In *Proc. CHI '88*, pages 1–10. Washington, DC, May 15–19, 1988.

[3] Casner, S.
A Task-Analytic Approach to the Automated Design of Graphic Presentations.
ACM Transactions on Graphics 10(2):111–151, April, 1991.

[4] Chung, J., Harris, M., Brooks, F., Fuchs, H., Kelley, M., Hughes, J., Ouh-young, M., Cheung, C., Holloway, R., and Pique, M.
Exploring Virtual Worlds with Head-Mounted Displays.
In *Proc. SPIE Non-Holographic True 3-Dimensional Display Technologies, Vol. 1083*. Los Angeles, January 15–20, 1989.

[5] Culbert, C.
CLIPS Reference Manual
NASA/Johnson Space Center, TX, 1988.

[6] Feiner, S.
APEX: An Experiment in the Automated Creation of Pictorial Explanations.
IEEE Computer Graphics and Applications 5(11):29–38, November, 1985.

[7] Feiner, S. and Beshers, C.
Worlds within Worlds: Metaphors for Exploring *n*-Dimensional Virtual Worlds.
In *Proc. UIST '90 (ACM Symp. on User Interface Software)*, pages 76–83. Snowbird, UT, October 3–5, 1990.

[8] Feiner, S. and McKeown, K.
Automating the Generation of Coordinated Multimedia Explanations.
IEEE Computer 24(10):33–41, October, 1991.

[9] Feiner, S. and Seligmann, D.
Dynamic 3D Illustrations with Visibility Constraints.
In Patrikalakis, N. (editor), *Scientific Visualization of Physical Phenomena (Proc. Computer Graphics International '91, Cambridge, MA, June 26–28, 1991)*, pages 525–543. Springer-Verlag, Tokyo, 1991.

[10] 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.

[11] 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.

[12] Green, M. and Shaw, C.
The DataPaper: Living in the Virtual World.
In *Proc. Graphics Interface '90*, pages 123–130. Halifax, Nova Scotia, May 14–18, 1990.

[13] Kamada, T. and Kawai, S.
An Enhanced Treatment of Hidden Lines.
ACM Transactions on Graphics 6(4):308–323, October, 1987.

[14] Karp, P. and Feiner, S.
Issues in the Automated Generation of Animated Presentations.
In *Proc. Graphics Interface '90*, pages 39–48. Halifax, Canada, May 14–18, 1990.

[15] Knowlton, K.
Computer Displays Optically Superimposed on Input Devices.
The Bell System Technical Journal 56(3), March, 1977.

[16] Kuipers, J.B.
SPASYN—A New Transducing Technique for Visually Coupled Control Systems.
In *Proc. Symp. on Visually Coupled Systems: Development and Application.* Brooks AFB, TX, September, 1973.
AMRL/WPAFB Report No. AMD TR-73-1.

[17] Lewis, B., Koved, L., and Ling, D.
Dialogue Structures for Virtual Worlds.
In *Proc. CHI '91*, pages 131–136. ACM Press, New Orleans, LA, April 27–May 2, 1991.

[18] Logitech, Inc.
Logitech 2D/6D Mouse Technical Reference Manual (Preliminary).
Fremont, CA, 1991.

[19] Norman, J. and Ehrlich, S.
Visual Accommodation and Virtual Image Displays: Target Detection and Recognition.
Human Factors 28(2):135–151, 1986.

[20] Reflection Technology.
Private Eye Product Literature.
Waltham, MA, 1990.

[21] Schmandt, C.
Spatial Input/Display Correspondence in a Stereoscopic Computer Graphic Work Station.
Computer Graphics (Proc. SIGGRAPH '83) 17(3):253–261, July, 1983.

[22] Seligmann, D. and Feiner, S.
Automated Generation of Intent-Based 3D Illustrations.
In *Proc. ACM SIGGRAPH '91 (Computer Graphics, 25:4, July 1991)*, pages 123–132. Las Vegas, NV, July 28–August 2, 1991.

[23] Sutherland, I.
A Head-Mounted Three Dimensional Display.
In *Proc. FJCC 1968*, pages 757–764. Thompson Books, Washington, DC, 1968.

[24] van Dam, A., Stabler, G., and Harrington, R.
Intelligent Satellites for Interactive Graphics.
Proc. IEEE 64(4):483–492, April, 1974.