Multimodal Interaction with a Wearable Augmented Reality System

Mathias Kölsch
Naval Postgraduate School

Ryan Bane
Microsoft Corporation

Tobias Höllerer and Matthew Turk
University of California, Santa Barbara

An augmented reality system enhances a mobile user's situational awareness and provides new visualization functionality. The custom-built multimodal interface provides access to information encountered in urban environments.

Wearable computers have evolved into tremendously powerful and versatile devices: PDAs, cell phones with integrated video recorders, wristwatches, game consoles, and even garments with built-in computational power. Unfortunately, their human-interface capabilities have not evolved as rapidly. Rather, the devices' continuously shrinking form factors severely limit their interfaces. Traditional interfaces, such as keyboards and LCD screens, can only be as big as a device's surface.

Fortunately, the conflicting goals of device size and interface area can both be met by expanding the interaction area beyond the device's dimensions: augmenting the reality through head-worn displays allows for information visualization in the entire field of view, extending far beyond the display's physical size. Equally, hand gestures performed in free space, recognized with a head-worn camera, are not constrained to the hardware unit's dimensions. Combining these advantageous I/O modalities harbors the promise of more complex interaction than what is possible with a keypad.

In this article, we detail our experiences with various input devices and modalities and discuss their advantages and drawbacks in the context of interaction tasks in mobile computing. We show how we integrated the input channels to use the modalities beneficially and how this enhances the interface's overall usability.

Motivation

We use a head-worn display and a small camera mounted on the glasses (see Figure 1) to provide video-based, see-through augmented reality (AR). Our system first processes the camera image with vision-based gesture-recognition algorithms, and then renders atop 3D graphics, registered with the camera view.

The motivation for this work is to support important tasks in urban environments, such as building maintenance, emergency rescue, and reconnaissance missions (see the "Augmented Reality Uses" sidebar). Our goal is to provide roaming workers with advanced visualization equipment to improve situational awareness, ease, and effectiveness in these jobs. The testbed system implements various interactive 3D visualization features in support of these roles, including tools for interactive inspection of volumetric data—such as room temperature within a building—and tools for insertion and 3D manipulation of virtual objects. These tools are suited for authoring georegistered annotations of pipes, conduits, and so on into building blueprints and for labeling urban landmarks for AR tour guides.

Our work shows how multimodal interface techniques can stretch the boundaries of the interactional complexity of tasks that can be performed on a wearable platform. Our contributions fall mainly into two categories: interactive information visualization techniques and the multimodal interfaces that control them.

Providing users with more powerful applications and the means to control them often fails because of the interaction devices' limitations. On the other hand, many nontraditional interaction devices and paradigms are unnecessary and too cumbersome for most conventional applications. Thus, it's important to develop novel interaction metaphors concurrently with the user interfaces that control them. We developed the tunnel tool and its interaction metaphors with custom means of interaction in mind that use multimodal user input.

New interaction metaphors

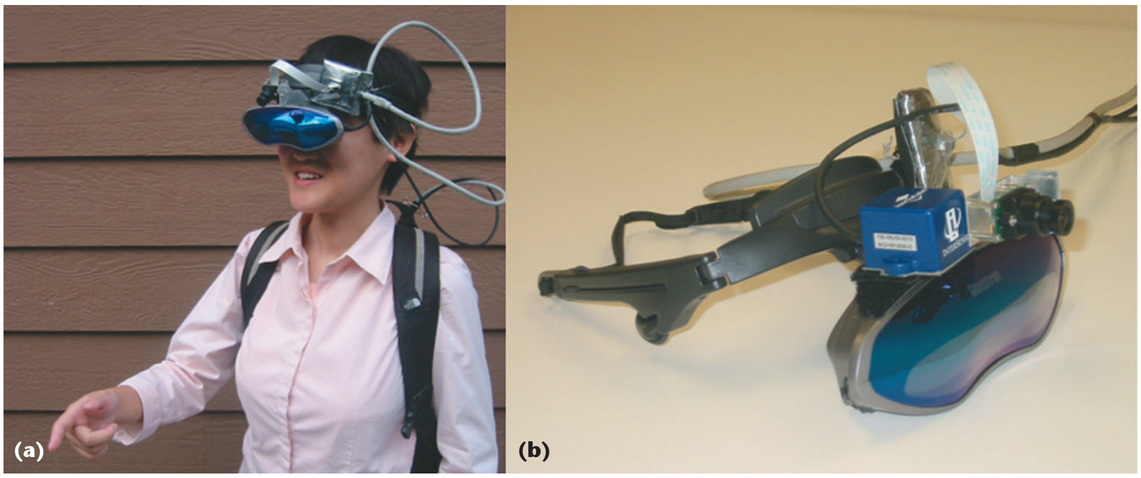
Our wearable system offers users visualization aids. Display techniques visualize multiple layers of "invisible" information on top of the physical world. Interactive mechanisms allow for data filtering and ultimately enable the user to understand and manage the wealth of available information. Because filter control parameters are so complex, traditional interaction methods are limited and unsuitable for mobile use.

Visualization environments

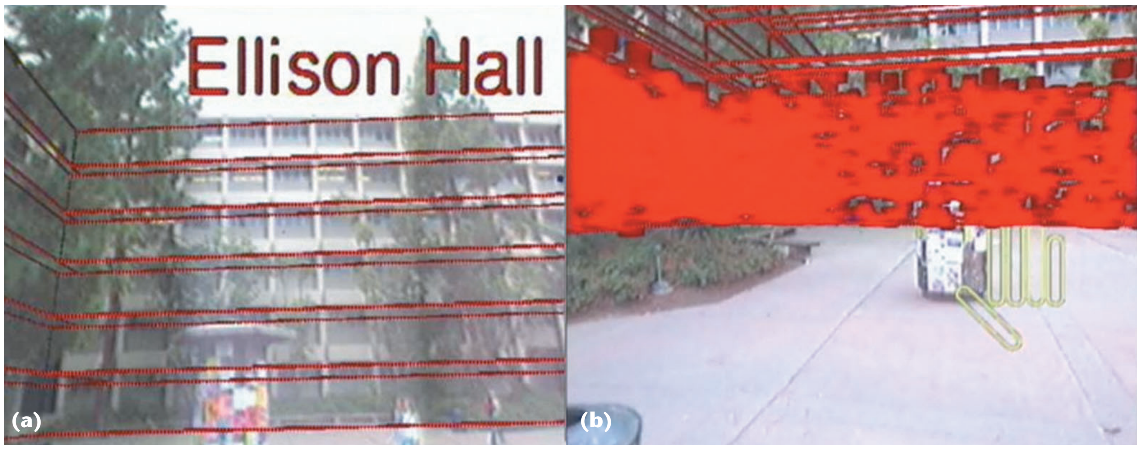
Our system divides geometric primitives into different semantic groups: structural walls, furniture, temperature clouds, building wireframes, and so on. The AR module renders these virtual augmentations differently depending on their momentary screen position. For example, the system could show a building's opaque structural walls when they are close to the edge of the screen, but not at the center of the screen. Instead, the central screen area could show building wireframes and furniture. The user can pick and choose which groups are displayed where. For this, we distinguish three rendering environments: the finger environment represents the area in close proximity to the user's hand. It can be used, for example, for selective probing of the virtual world. The tunnel environment is specific to the screen area covered by the tunnel tool. The third environment, surroundings, comprises the remaining screen area. Figure 2 has only the surroundings environment turned on; activated are build-

Augmented Reality Uses

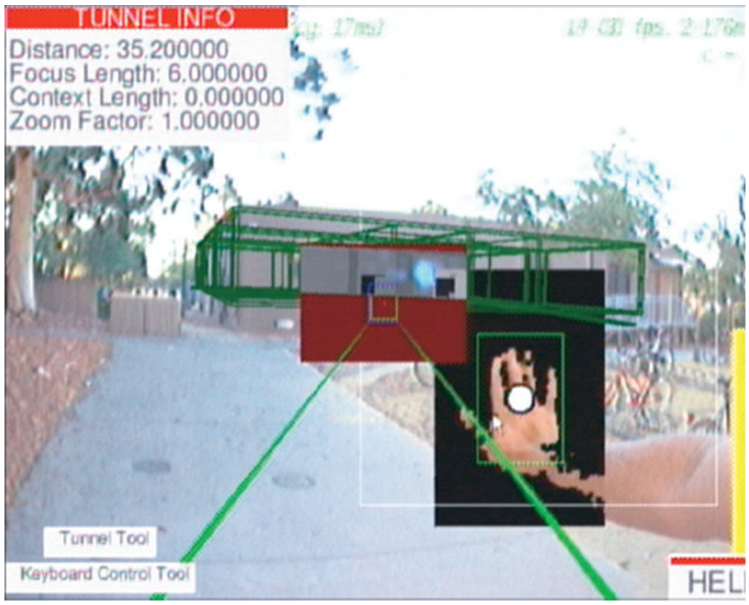
Ubiquitous information access is desirable for many tasks in life. For example, as a tourist you would like to know your location in a city, nearby sights, and types and locations of services (restaurants, ATMs, and so on). The wealth of information that could be made available needs filtering: based on proximity and your general preferences, but also in response to your current needs. Human–computer communication is thus an integral component of any information system. Information access is particularly important to support situational awareness in unknown and potentially life-threatening environments. Emergency response teams, firefighters, and soldiers in urban environments all depend on the availability of information in the most convenient, efficient, and effective ways possible. Augmented reality and nontraditional interaction methods can open new avenues to making information accessible and manageable.



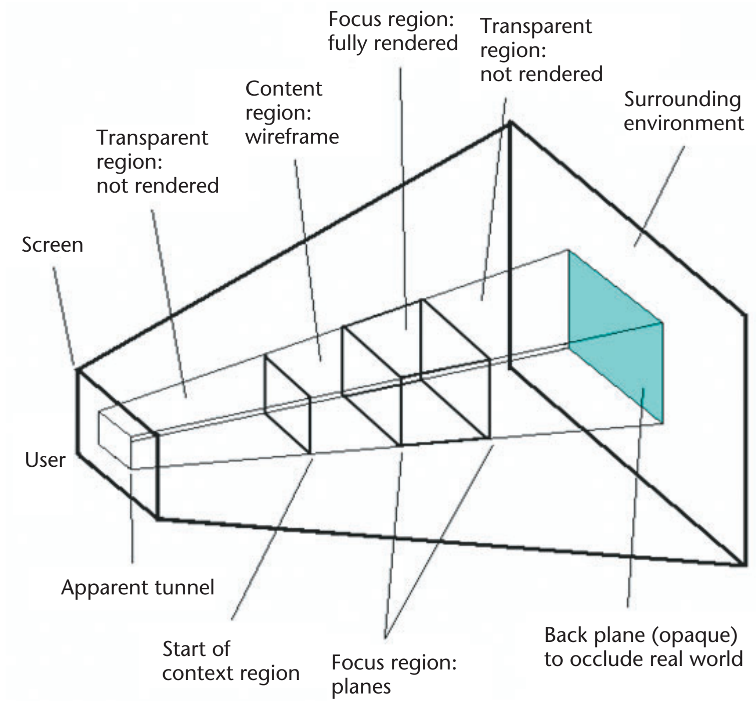
1 (a) A user wearing our system. (b) The head-worn display with the tracker attached.



2 (a) Augmented reality view of one of the two buildings for which we built models. (b) Visualization of spots with "insufficient wireless" signal strength. The density of the data prohibits discerning differences in the data intensity.



3 The view through the head-worn display as a maintenance worker would use our system when in search of an overly active air-conditioning duct.


Focus region: fully rendered
Transparent region: not rendered
Content region: wireframe
Surrounding environment
Transparent region: not rendered
Screen
User
Apparent tunnel
Start of context region
Focus region: planes
Back plane (opaque) to occlude real world

4 Schematic view of the tunnel tool. The system only renders in full those objects inside the focus region. It renders as wireframes those objects that fall within the context region. Objects to either side of the apparent tunnel are rendered as specified in the surroundings environment.

ing wireframes and "insufficient wireless" signal strength. Figure 3 additionally shows the tunnel environment. The surroundings environment contains a building wireframe in green. The back plane is brown, preventing the real world to shine through; and a cold spot of temperature data is visible in the focus region. For demonstration purposes, we have shown skin color segmentation by the hand tracking library (all non-skin-colored pixels are blackened out in the rectangular area around the hand). A faint dark-green box around the hand indicates the area in which the hand was detected (not to be confused with the trailer's wireframe). The white circle on top of the hand and the black rectangle around it indicate system stede information about hand tracking (tracked position and learned hand color). During normal use, only a small red circle is shown as feedback.

Tunnel tool

The tunnel tool is our main visualization aid. It enables the user to mask out items that obstruct the view onto hidden infrastructure, while providing orientation and navigation cues as context in the surroundings environment around the tunnel tool.

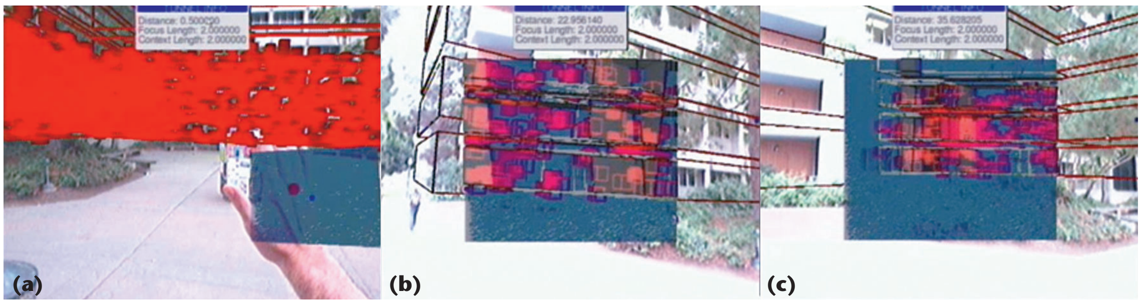
The tool is apparent to the viewer as an opaque plane that occludes the real world and occupies part of the screen. For the area in front of the plane, the system only renders those items that the user has added to the tunnel environment using voice commands. Around the apparent tunnel, the system renders items that are active in the surroundings environment. Figure 4 shows the tunnel tool's 3D layout; more detail is available elsewhere.[1]

The tunnel defines a frustum shape into the virtual representation of the world. Within this shape, the user controls a series of three vertical planes, defining four regions within the tunnel. The first of these regions starts at the user and extends to the first plane. Objects within this first transparent region are not rendered. The second or context region provides context on the objects the user is viewing, so objects within this region are always rendered in wireframe (sample uses of this feature are available elsewhere[1]). The focus region lies between the second and third planes. This region represents the user's current area of interest. Objects falling within the focus region are fully rendered. To avoid signal mixing, the transparent region behind the focus region is not rendered.

The user can slide the focus region forward and backward in the tunnel, making it possible to see objects at any distance from the current position even when other objects would normally occlude them. This functionality allows users to interactively explore complex 3D information, such as the structure of a building or volumetric visualization of wireless network strength. For more control over the view, the user can also adjust the length of the context and focus regions.

Slicing. The tunnel tool presents volumetric data in short segments, or slices. To investigate the inside of a dense data distribution, the user adds the item in question to the tunnel environment then interactively moves the focus region through the data cloud. This masks out data in front of and behind the focus region, simplifying the user's view and allowing for selective data exploration. Figure 5 illustrates how this technique helps localization of a hotspot—a higher density area deep inside a data cloud.

In initial trials, the speech command "tunnel follow finger" attached the tool's main axis to the location of



5 **Tunnel tool (the blue rectangular area): (a) a first version equipped with functionality to attach to the hand location and move across the display. (b) A slice of "insufficient wireless" data is cut out of the entire data cloud seen on the left, allowing the density (that is, the number of reports of low signal strength) to be judged. (c) A hotspot area with many blobs clustered close together.**

the tracked hand, allowing 2D relocation with the hand. We wanted to enable the user to interactively sample the space with the tunnel tool. However, we found it more convenient to leave the tunnel tool fixed at the center of the view. In hindsight, this makes more sense because the tool is likely to the user's center of attention while being displayed.

We recently extended the tunnel tool's functionality so that it can be spatially confined by snapping to semantic objects such as building floors, walls, or individual rooms.[1] This avoids artifacts caused by partially displayed walls. The volume slicer can also operate on selective rooms only, permitting an exploration style that more closely resembles humans exploring building spaces.

X-ray vision. The tunnel tool allows for virtual superman-style X-ray vision, letting the user see through physically present walls to the virtual representations of the objects beyond. This differs conceptually from the tunnel tool's slicing functionality (that helps explore volumetric data) since the view of the objects of interest is obstructed by real, physical objects instead of purely virtual data.

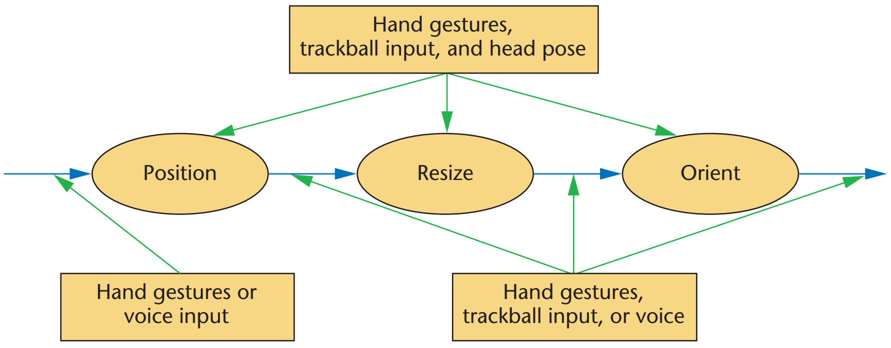
Manipulation of virtual objects

Users can select and manipulate virtual representations of semantic entities such as a desk, as well as insert simple geometric objects (boxes, spheres, and lines) into the virtual scene and subsequently manipulate them. This is useful for annotating a physical scene or for creating virtual models of physical geometry. The object manipulation occurs in a three-step process: the object is first positioned, then resized, and finally rotated. We use a mix of hand gestures, trackball input, and voice commands to perform these tasks (see Figure 6).

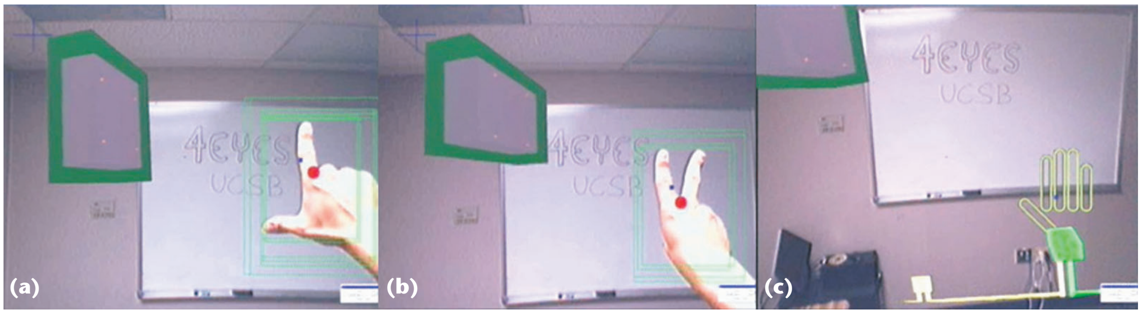
After issuing the voice command "select picking tool for finger," the user can select an object by pointing a finger at it and saying "select object." The position mode is then activated by saying "move object." Alternatively, the user can insert a new virtual object with "insert insertable," where insertable is a box, sphere, or line. After insertion, the user interface automatically enters position mode.

Our user interface mixes three input signals to allow for concurrent object positioning in three dimensions: 2D head pose and hand gestures for (x, y) and 1D trackball input for (z). The gesture commands work as follows: the user first makes the lock hand gesture (see Figure 7a, next page), which sets the system to track hand motions and apply them to the object's position. Subsequent head and hand motions will move the object in a plane parallel to the screen.

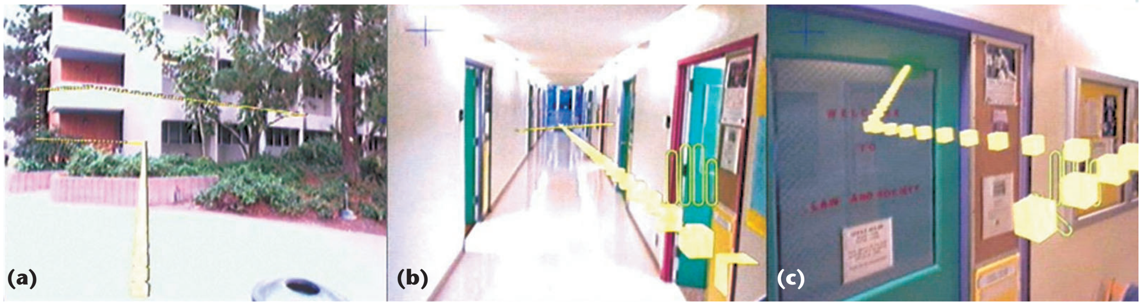
When satisfied with the object's position, the user makes the release hand gesture (see Figure 7b), which stops the system from applying head and hand motions to the object (see Figure 7c). Note that the input range is not limited to the camera's or HMD's field of view since the head orientation is tracked. The voice command "finished," clicking with the trackball, or the release gesture prompt the transition into the resize mode. There, the same input modalities allow 3D object resizing by dragging its proximal, right, bottom corner. Again, clicking, gesturing, or a "finished" voice command exits this mode and causes a prompt transition to the next mode in which the user can rotate the object around each axis—with the same input modalities except head pose. (We initially used the head pose as input for the object's orientation as well, but found it inconvenient that the object rotated while attempting to look at it from multiple angles.)


Hand gestures, trackball input, and head pose
Position
Resize
Orient
Hand gestures or voice input
Hand gestures, trackball input, or voice

6 **State machine showing the three-step process for manipulating virtual objects.**



7 Resizing an object with gestures. (a) The user's hand in the lock posture from which hand location changes rescale the virtual object (the gray box with a green selection frame around it). (b) The user performs the release posture, which decouples hand movements and object scale again. (c) The placement of the virtual object in the upper left corner and representations for power and networking equipment near the floor.



8 The path (a) enters the building, (b) then it follows the stairs (in a straight vertical) to (c) the second floor.

The voice commands "move object," "resize object," and "orient object" cause the system to immediately enter the respective manipulation modes.

In the future, when recognition of more precise hand gestures becomes available, we might replace this modal interaction with two-handed manipulation, which can support concurrent grabbing, positioning, resizing, and rotating without any mode changes. In the meantime, the described modal interface proved to be easy to understand and perform, relying only on two generic hand gestures and a few voice commands. The simultaneous input of 3D commands was preferable over even more staggered, sequential input of fewer dimensions at a time, say, with a trackball or a mouse.

Selection and manipulation with hand gestures has advantages over the frequently used selection by gaze (that is, head) direction.[2] With those systems, users had to hold their heads quite still to make a selection. Our approach integrates multiple resolutions: the head determines the main direction (in the entire 4π steradians) and the hand facilitates fine selection (within the current field of view). This is also important for explorative probing of the environment, for example, with dynamic labels on objects inside the finger environment.

Paths

A path finding and navigational guidance capability is critical to many emergency response situations, for example, building evacuation. We implemented a modification of Dijkstra's shortest path algorithm and a visualization that displays the result (similar to those found in Kalkusch et al.[3]). The user can generate a path to either a previously picked object, or to the center of the focus region in the tunnel tool. To initiate path computation, the user uses the voice command "set path from here to tunnel." Figure 8 shows the visual result.

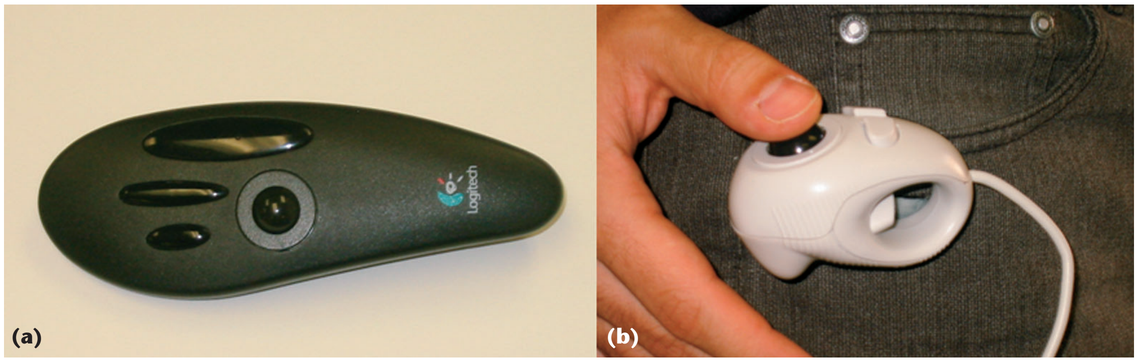
Interacting with the new metaphors

We experimented with different input modalities to control the visualization tools. Here, we detail our experiences, the lessons we learned for future implementations, and the choices we made.

Design choices

We started operating the visualizations with a keyboard- and mouse-controlled, screen-stabilized GUI—our development interface. We spread out all hardware components on a park bench and operated the HMD/ tracker unit independently of the other input devices. It quickly became apparent that the inconvenience of controlling complex interaction metaphors with conventional devices was unbearable while immersed in AR. Thus, we experimented with additional interface devices: a wireless handheld trackball, ring-style trackball, hand-gesture recognition, speech recognition, and head-orientation tracking. With informal experimentation, we gathered first-hand results on the suitability of the devices for diverse tasks.

We feel that discrete, binary toggle parameters—previously mapped to one key each—are best accessed by equally binary speech commands. Furthermore, speech allows for the natural extension of enabling commands



9 Our two favorite devices to provide 1D input to our system: (a) a wireless handheld trackball and (b) a trackball that can be worn in a similar manner as a ring. Here it is attached to the user's pants with a Velcro strip.

with parameters: for example, the "add" speech command is parameterized by item names, such that "add item temperature to tunnel" prompts the display of temperature information along with previously shown information inside the tunnel area. Other typical commands that we mapped to voice input are "take snapshot," "save," "discard," and "open/close tunnel."

Our visualization interface requires multidimensional control for positioning, sizing, and orienting objects. This could be done with multiple sequential 1D and/or 2D input steps, but this is awkward and differs starkly from the direct manipulation of real, physical objects. To achieve concurrent input of 3D data into our system, hand tracking provides two dimensions and an auxiliary modality supplies the third (quasicontinuous 1D input). Device choices for the 1D input include mice, trackballs, trackpoints, touchpads, and key-press duration. As expected, regular mice and keyboards turned out unsuitable for the task environment. The mouse wheel by itself could supply 1D input. However, the most common hardware implementation of mouse wheels does not permit input of parameters from a continuous domain. Instead, they generate discrete stepping commands with optical switches and are thus unsuited for continuous input.

Our favorite candidates were a wireless handheld trackball and a ring trackball (see Figure 9). We eventually chose the latter device because the user can leave it dangling from the index finger to enable use of the same hand for gesturing, thus permitting single-handed hand gesture input. Our system only needs one dimension of the 2D ball motions, but the device has the full functionality of a three-button mouse and thus allows for user-interface extensibility. As mentioned previously, for certain command-issuing tasks such as mode switching, button clicks serve as one of a set of redundant alternative input modalities (see also Table 1).

Mapping the trackball's two continuous dimensions to the planar interaction parameters in our interface would preclude the advantages of registered interaction of the hand with virtual objects.

Multimodal integration

The system integrates four input modalities: hand gestures, voice commands, unidirectional trackball motions, and head orientation. Feature extraction and interpretation happens independently on each channel. That is, the modalities are combined with late integration after grammatically correct sentences have been extracted and the location and posture of the hand is determined. The style of interpretation differs according to input commands and system state. Our system blends three styles of late integration to maximize the overall usability while choosing input from the best-suited modality for a given task.

Independent, concurrent interpretation. Our system immediately interprets input of this style as

Table 1. Application parameters and which modalities control them.*

Control Parameter	Speech	Gesture	Trackball	Head Pose
0D				
Take/save/discard snapshot	Yes			
Tunnel mode	Yes			
Add/remove/visualization from environment	Yes			
Enter relocate mode	Yes			
End relocate/resize/orient mode	Yes	Yes	Yes	
1D				
Adjust focus region distance			Yes	
Adjust focus region depth			Yes	
2D				
Pencil tool for finger		Yes	Yes	
Select virtual objects		All		All
3D				
View direction				Yes
Position virtual objects		All	All	All
Resize virtual objects		All	All	All
Orient virtual objects		All	All	

*Multiple notations of "yes" in a row indicate that any one of many modalities can supply the input to control the respective application parameter. "All" indicates that every modality's input contributes to the application parameter.

Related Work

Feiner et al.'s Touring Machine was the first outdoor, mobile, augmented reality (AR) system that overlaid 3D computer graphics on top of the physical world.[1] Follow-up research explored a series of mobile AR user interfaces, including world- and screen-stabilized navigational aids. These mobile user interfaces did not make use of vision-based gestural input or focus particularly on multimodal interaction techniques. An advantage of our gesture-based selection (over the head-orientation-based selection mechanism employed in Feiner et al.'s work) is that the user's hand can operate in the world reference frame, so that rotating the head does not interfere with fine-grain selection tasks.

Researchers at the University of South Australia have implemented a series of mobile AR systems for terrestrial navigation; their more recent prototypes employ a pinch-glove-based interface for creating and manipulating augmented material in the outdoors.[2] Visual markers allow 3D location and 3D orientation estimation of the hand with a camera. The researchers show many compelling interface applications with these gloves. The input device is bulky, however, and prevents users from using their hands for other tasks. Our gesture interaction does not require a glove, and our speech recognition component takes over tasks that are not naturally spatially arranged, such as command selections.

Sawhney et al. designed and evaluated speech-controlled audio interfaces for wearable computers.[3] We use speech to let the user issue commands that would be more difficult to express in other media, but we provide backup interaction techniques that can be employed when speech input fails or is impossible due to situational constraints (noisy environments or imposed silence).

Some functionality of our 3D tunnel tool is reminiscent of concepts pioneered by Bier et al. in their work on magic lenses.[4] We extend the idea of different layers that can be visualized by 2D lenses to a physical 3D space and allow for the layers to be selected and controlled by voice commands.

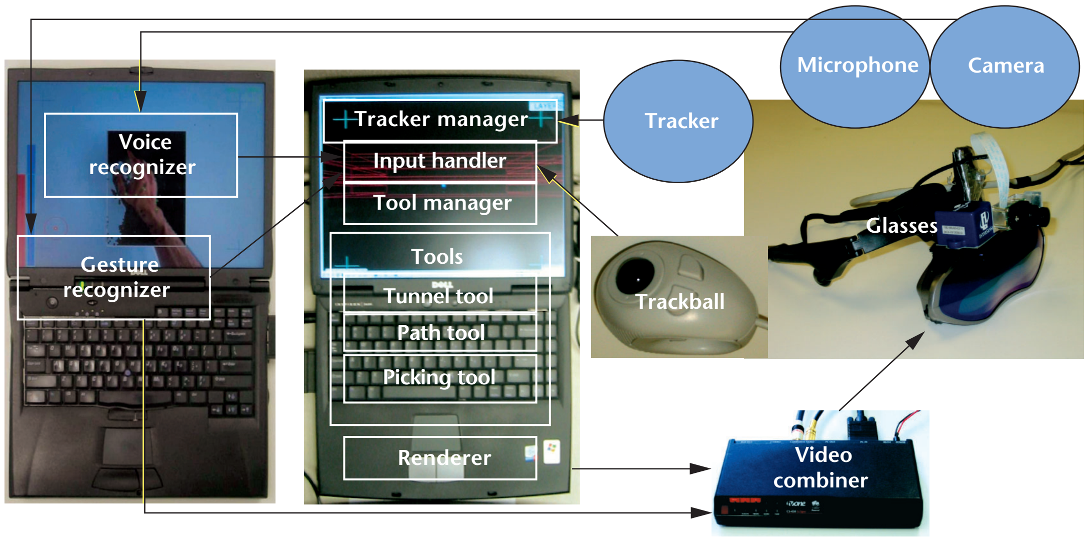
A number of researchers have tackled the problem of multimodal integration for speech, gestures, gaze, and 3D tracking information,[5,6] but this has not been implemented on a mobile platform. Oviatt and colleagues on the other hand demonstrated the positive influences of multimodal interaction on error rates and robustness in a mobile setting involving speech and pen input on tablets.[7] Rather than focusing solely on increased reliability, we explore multimodality for expanding the user's interaction capabilities and for increasing the dimensionality of user input.

References

1. S. Feiner et al., "A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment," *Proc. Int'l Symp. Wearable Computers* (ISWC), IEEE CS Press, 1997, pp. 74-81.
2. B.H. Thomas and W. Piekarski, "Glove Based User Interaction Techniques for Augmented Reality in an Outdoor Environment," *Virtual Reality: Research, Development, and Applications*, vol. 6, no. 3, 2002, pp. 167-180.
3. N. Sawhney and C. Schmandt, "Speaking and Listening on the Run: Design for Wearable Audio Computing," *Proc. 2nd Int'l Symp. Wearable Computers* (ISWC), IEEE CS Press, 1998, pp. 108-115.
4. E.A. Bier et al., "Toolglass and Magic Lenses: The See-Through Interface," *Proc. Siggraph*, vol. 27, ACM Press, 1993, pp. 73-80.
5. D.M. Krum et al., "Speech and Gesture Multimodal Control of a Whole Earth 3D Visualization Environment," *Proc. Symp. Data Visualization* (VISSYM), Eurographics Assoc., 2002, pp. 195-200.
6. E. Kaiser et al., "Mutual Disambiguation of 3D Multimodal Interaction in Augmented and Virtual Reality," *Proc. Int'l Conf. Multimodal Interfaces* (ICMI), ACM Press, 2003, pp. 12-19.
7. S.L. Oviatt, "Multimodal System Processing in Mobile Environments," *Proc. ACM Symp. User Interface Software and Technology* (UIST), ACM Press, 2000, pp. 21-30.

atomic commands. Users can give most speech commands at any time, having the same effect every time. For example, the speech directive "add networking to surroundings," which causes visualization of wireless networking signal strength, can occur simultaneously with gesture or trackball commands, with the system interpreting both commands independently of its state. Another example is the 2D hand tracking and 1D trackball input that combine into 3D input.

Singular interpretation of redundant commands. Redundant commands, that is, commands from one channel that can substitute for commands from another, give the user a choice of picking the most convenient way to issue an instruction. The system treats the case of multiple, mutually redundant commands as a single instruction. We currently have two cases of this style: speaking "select picking tool for finger" achieves the same result as performing a dedicated hand posture, and the release gesture during object manipulation is equivalent to the "finished" speech command. We chose 2 seconds between commands as an appropriate interval in which the system will treat them as one. Wherever possible, we avoid arbitrary thresholding by changing into a state in which the commands are not associated with a meaning and can thus do no harm if issued twice. A more sophisticated signal integration, as in Oviatt,[4] could replace the simple "or" operation and improve robustness through mutual disambiguation.



10 An overview of the hardware components that comprise the wearable augmented reality system and a functional diagram of the main software modules.

Sequential, moded interpretation. This style does the opposite of redundant commands. It requires users to provide input first in one modality, then in another. This is a common style within the desktop metaphor—first a mouse click to give focus to a window, then keyboard interaction with that window—which has the drawback of an associated switching time between mouse and keyboard. In our system, however, there is no such switching penalty since the two involved modalities don't both use the same resource (for example, the same hand). The drawing and virtual object manipulation modes use gestures for spatial input and voice commands for mode selection. In fact, we chose this style because it makes the best use of each modality without creating a conflict.

Overall, the modalities work together seamlessly, allowing for interaction that has almost conversational character. Voice commands allow the user to easily switch features or tools on and off, and to enter nonspatial input such as adding items to environments. Hand gestures provide a natural input interface for spatial data. Also, select hand postures allow for input of simple action sequences entirely with gestures. Finally, the trackball provides for exact, continuous 1D input in situations where hand gestures are less convenient or 3D input is desired. Table 1 summarizes how the four modalities are combined to control various application parameters.

Multiple modalities offer various benefits. First, more diverse input modalities increase the chance that one of them provides a natural way to execute a certain action, such as performing registered interaction with hand tracking. Second, redundancy lets the user select the momentarily most convenient interface. Redundancy can also be exploited to increase robustness and to reduce error rates.[4] Third, users can simultaneously control more input dimensions. The first two points are of particular importance in wearable computing, where the computer is usually not in the foreground or the sole object the focus of attention. The third point is increasingly important as application complexities rise beyond what conventional input modalities can control.

System description

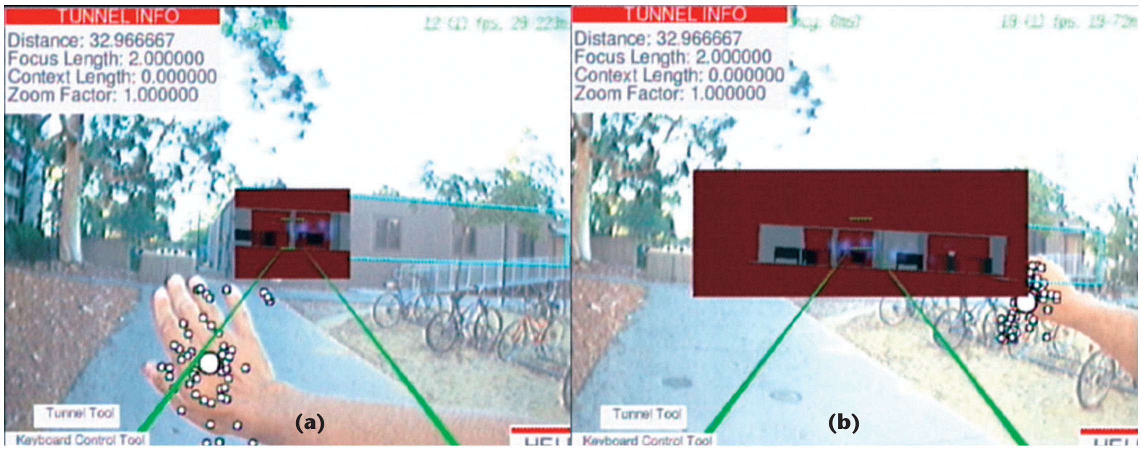
Figure 10 shows a diagram of our system implementation structure, overlaid on pictures of the actual devices. We use two laptops (1.1-GHz Pentium IIIs running Windows XP) because the performance of the gesture recognizer drops significantly if it has to compete with other compute- or bus-communication-intensive applications. The second laptop hosts a custom-built OpenGL-based visualization engine.

Sony Glasstron LDI-A55 glasses serve as the base for our head-worn unit, delivering mono National TV Standards Committee (NTSC) resolution in color. Mounted atop are a Point Grey FireFly color camera and an InterSense InertiaCube^2 orientation tracker. A commercial off-the-shelf microphone is clipped to the side of the glasses. A TV One CS-450 Eclipse scan converter overlays the luminance-keyed output from the rendering computer over the mostly unmodified video feed through from the first computer, providing input to the head-worn display and to a digital–video camera that recorded the images shown in this article.

Visualization engine

The visualization engine stores data about each used rendering primitive in a custom scene graph, including a tag specifying the item that it belongs to. Each environment stores a record of active items that determine whether an object should be rendered within this environment.

When the tunnel tool is not active, the system renders the items currently selected as part of the surroundings environment in a single pass over the scene graph. When the tunnel tool is active, rendering becomes a series of passes over the scene graph, each using different clipping planes. Objects within the finger environment are dynamically added to the scene graph.



11 **(a) Each feature of a flock of features is shown as a little dot; their average is the big dot. (b) The user is resizing the tunnel tool's area with the tracked hand. The white circles on top of the hand represent the tracked features and the feature median. During normal use, only a small red circle (as in Figures 7a and 7b) is shown as feedback.**

Hand-gesture recognition

We used our HandVu computer-vision module to implement hand-gesture recognition.[5] This library is similar to Kurata et al.'s Hand Mouse,[6] but HandVu doesn't require a high-speed LAN connection to a computing cluster to facilitate real-time hand tracking. We also employ multiple image cues for more robust hand detection and tracking.

HandVu's hand detection algorithm detects the hand in a standard pose based on texture and color with a recognition rate of more than 90 percent, with a few false positives per hour of live video. Upon detection, the system learns the observed hand color and uses it for tracking. Hand tracking mainly uses gray-level texture, but resorts to local color information as backup. This multicue integration of gray-level texture with textureless color information (or flock-of-features tracking[7]) increases the algorithm's robustness, letting us track the hand despite vast and rapid appearance changes. The algorithm's last stage attempts posture recognition at the tracked location—a classification into a set of predefined, recognizable hand configurations. Figure 11 shows two images with active hand tracking and verbose output.

HandVu's vision processing methods typically require less than 100 milliseconds combined processing time per frame. They are mostly robust to the different environmental conditions: lighting changes, color temperature, lens distortion, and so on. Overall, HandVu delivers the interactive speed, accuracy, and robustness qualities that are imperative to a user interface's quality and usability. The final output of the vision system is the hand's location in 2D-image coordinates and possibly its posture. More detailed descriptions of HandVu's architecture,[5] robust hand detection,[8] and hand tracking[7] are available elsewhere.

To obtain 3D input, we interpret trackball input, which is unbounded, as the third dimension in addition to HandVu's two dimensions. Three-dimensional hand tracking would not achieve the same goal: the limited range (distance from the camera) of hand motion does not allow for interaction with distant objects. Introducing a scaling factor would trade off range for precision. For all other mappings, a clutching mechanism would have to be provided, making interaction less natural and less smooth.[9]

Speech recognition

We chose a prototype automatic speech recognition library, Panasonic Speech Technology Laboratory's ASRlib, to provide computationally efficient, speaker-independent recognition of a simple grammar. We use the English data set, about 70 keywords, and a grammar that allows for around 300 distinct phrases with these keywords. Command phrases must be preceded and followed by a brief pause in speech, but the words can be concatenated naturally. The library performed well in our tests, producing no false positives despite listening in on all of our conversation. It did occasionally require the repetition of a command. The speech recognition module consumes few enough resources to run alongside the power-hungry computer-vision application. The interface to the rendering application consists of unidirectional speech events.

Conclusions

Wearable computers offer new applications to new fields of deployment. We have shown how a novel, versatile visualization aid for AR environments can be controlled with a multimodal interface in a mobile scenario. We emphasize the importance of concurrent development of nontraditional applications and novel user interfaces capable of operating them, even in adverse environments. We conclude that multimodal user interfaces broaden the realm of applications for wearable computers and that they can satisfy the diverse input needs of demanding application interfaces. ■

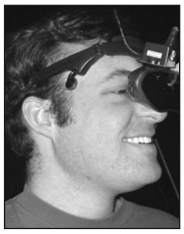
References

1. R. Bane and T. Höllerer, "Interactive Tools for Virtual X-

Ray Vision in Mobile Augmented Reality," *Proc. IEEE and ACM Intl. Symp. Mixed and Augmented Reality* (ISMAR), IEEE CS Press, 2004, pp. 231-239.

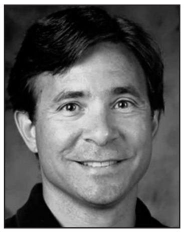
2. S. Feiner et al., "A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment," *Proc. 2nd Int'l Symp. Wearable Computers* (ISWC), IEEE CS Press, 1997, pp. 74-81.
3. M. Kalkusch et al., "Structured Visual Markers for Indoor Pathfinding," *Proc. IEEE Int'l Workshop ARToolKit* (ART), IEEE CS Press, 2002.
4. S.L. Oviatt, "Multimodal System Processing in Mobile Environments," *Proc. ACM Symp. User Interface Software and Technology* (UIST), ACM Press, 2000, pp. 21-30.
5. M. Kölsch and M. Turk, "Fast 2D Hand Tracking with Flocks of Features and Multi-Cue Integration," *Proc. IEEE Workshop Real-Time Vision for Human–Computer Interaction*, IEEE CS Press, 2004, p. 158.
6. T. Kato, T. Kurata, and K. Sakaue, "VizWear-Active: Towards a Functionally-Distributed Architecture for Real-Time Visual Tracking and Context-Aware UI," *Proc. 2nd Int'l Symp. Wearable Computers* (ISWC), IEEE CS Press, 2002, pp. 162-163.
7. M. Kölsch, M. Turk, and T. Höllerer, "Vision-Based Interfaces for Mobility," *Proc. Int'l Conf. Mobile and Ubiquitous Systems* (Mobiquitous), IEEE CS Press, 2004, pp. 86-94.
8. M. Kölsch and M. Turk, "Robust Hand Detection," *Proc. IEEE Int'l Conf. Automatic Face and Gesture Recognition*, IEEE CS Press, 2004, pp. 614-619.
9. I.S. MacKenzie, "Input Devices and Interaction Techniques for Advanced Computing," *Virtual Environments and Advanced Interface Design*, W. Barfield and T.A. Furness III, eds., Oxford Univ. Press, 1995, pp. 437-470.

***Mathias Kölsch** is an assistant professor of computer science at the Naval Postgraduate School in Monterey, California. His research interests include computer vision, human–computer interaction, computer graphics, and AR. Kölsch has a PhD in computer science from the University of California, Santa Barbara. Contact him at kolsch@nps.edu.*



***Ryan Bane** is a developer with the Windows Live Mobile team at Microsoft, in Redmond, Washington. His research interests include AR, mobile devices, and distributed systems. Bane has an MS in computer science from the University of California, Santa Barbara. Contact him at rbane@microsoft.com.*

***Tobias Höllerer** is an assistant professor of computer science at the University of California, Santa Barbara, where he codirects the Four Eyes Laboratory. His research interests include AR, 3D interaction, visualization, mobile and wearable computing, and adaptive user interfaces. Höllerer has an MS and PhD, both in computer science, from Columbia University, and a graduate degree in informatics from the Technical University of Berlin, Germany. Contact him at holl@cs.ucsb.edu.*



***Matthew Turk** is a professor of computer science at the University of California, Santa Barbara, where he is also the chair of the Media Arts and Technology Graduate Program and codirects the Four Eyes Laboratory. His research interests include imaging, interaction, and innovative interfaces. Turk has a PhD from the Massachusetts Institute of Technology. Contact him at mturk@cs.ucsb.edu.*