The Mercator Environment: A Nonvisual Interface to X Windows and Unix Workstations

Elizabeth Mynatt and W. Keith Edwards

Multimedia Computing Group
Georgia Institute of Technology
Atlanta Georgia, 30332-0280
beth@cc.gatech.edu, keith@cc.gatech.edu

ABSTRACT

User interfaces to computer workstations are heavily dependent on visual information. These Graphical User Interfaces, commonly found on powerful desktop computers, are almost completely inaccessible to blind and visually impaired individuals. In order to make these types of computers accessible to non-sighted users, it will be necessary to develop a new interface which replaces the visual communication with audio and tactile communication. This paper describes the Mercator Environment—an auditory and tactile interface to X Windows and Unix workstations designed for the visually impaired.

KEYWORDS: user interfaces, auditory interfaces, visual impairment, auditory icons, rooms, three dimensional audio

INTRODUCTION

The goal of human-computer interfaces is to provide a communication pathway between computer software and human users. The history of human-computer interfaces can be interpreted as the struggle to provide more meaningful and efficient interactions. One of the most important breakthroughs in HCI in recent years was the development of graphical user interfaces. These interfaces provide innovative graphical representations for system objects such as disks and files, and for computing concepts such as multi-tasking by way of windows. Unfortunately, these Graphical User Interfaces (or GUIs) have left a percentage of the computing population behind. Presently Graphical User Interfaces are all but completely inaccessible for computer users who are blind or severally visually-disabled. Historic strategies for providing access for these users are inadequate.

In this paper, we will explain why GUIs are currently inaccessible to computer users who are blind and why providing this access is a non-trivial problem. We will review historic strategies for providing access and discuss why we feel these strategies are inadequate for providing access to contemporary GUIs. Then we will describe our system, the Mercator[1] Environment. We will discuss the principles behind our design and what techniques we are using to satisfy those goals. We will then cover two key implementation strategies for our environment. Finally, we will review the status of our project and identify some key issues for further research.

THE GUI PROBLEM

For much of their history, computer displays have presented only textual and numeric information to their users. This manner in which users interacted with their computers was only adequate at best. One benefit of this character–based interface, however, was that users who were blind could have fairly easy access to such systems. Users with visual disabilities could use computers with character–based interfaces by employing devices and software that translated the

[1] Named for Gerhardus Mercator, a cartographer who devised a way of projecting the spherical Earth's surface onto a flat surface with straight–line bearings. The Mercator Projection is a mapping which aids in navigation, just as the Mercator Project is a mapping from a graphical interface to an auditory/tactile one which aids computer users with visual impairments in navigating their systems.

characters on the screen to auditory information (usually a synthesized human voice).

Next to vision, audition is the human sense which can convey the most information. The sense of hearing has a "large bandwidth" for information transfer. Users who are blind therefore naturally rely on hearing as a means to gather information from their computers. In the case of textual computer interfaces (in which not a great deal of information was presented on the screen at a time), the translation of text to speech was generally sufficient for the needs of most users with visual disabilities. An important characteristic of access to textual interfaces is that the metaphors of information presentation are practically equivalent. Sighted and non-sighted users are presented information in a line-by-line format. Both types of users generally relied on the keyboard to enter text and execute commands.

Since the mid 1980's, the computer industry has seen a remarkable increase in the use of Graphical User Interfaces, or GUIs, as a means to improve the bandwidth of communication between sighted users and computers. These interfaces have several defining characteristics. First, they often use pictorial representations of applications and files. These representations are often called icons. The image associated with an application or file gives an indication of what that particular object is. Secondly, GUIs typically break the screen area into visually segmented areas, called windows. Each window acts as a container for information (much like a sheet of paper) and allows users to manipulate the information they are currently viewing by rearranging their windows. Third, GUIs typically involve the use of some pointing device (usually a mouse) which is used to change the position of an on–screen pointer.

These graphical user interfaces can convey a great deal more information at a time than the older character–based interfaces. While the use and acceptance of GUIs has been a great boon for most computer users, those users with visual impairments have been left behind. [BBV90, Bux86] Unlike the character–based computer displays of a few years ago, there is no simple mapping from graphical window–based systems to the auditory and tactile domains.

Each of the traits of GUIs enumerated above presents a distinct set of challenges for users who have no way to distinguish pictorial representations on the screen. Windows present a unique challenge because even with a direct auditory representation of the various windows on the screen, there is still the problem of occluded windows. Many screen readers are not able to retrieve the contents of windows that are obscured by other objects on the user's display. In both the cases of icons and windows, users who are visually impaired do not have instantaneous access to all the information on the screen, as sighted users do. Instead they must search through the space of auditory information provided to them (which represents the on–screen information) attempting to locate a desired piece of information, application, or file. Finally, the use of a mouse is problematic because moving the mouse moves the screen cursor relative to the last position of the mouse; there is no sense of absolute position or correspondence of the mouse position to the on–screen cursor position. Furthermore, there may be no meaningful mapping between the on–screen position of items and the underlying information represented by the visual metaphor anyway.

Despite recently enacted governmental legislation requiring computer vendors to demonstrate that their equipment and software can be made accessible [Lad88] to be eligible for Federal procurement contracts (Title 508 of the Rehabilitation Act of 1986), and requiring employers to purchase equipment and software to make their systems accessible according to the needs of their users (the Americans with Disabilities Act), reasonable access to GUIs remains an unsolved problem. There is substantial fear among blind computer users of losing jobs as more companies move to computer systems with GUIs.

STRATEGIES FOR PROVIDING ACCESS

We feel that the fundamental problem with access to GUIs is that often the visual representation of interface metaphors used in GUIs is not intuitive for someone who cannot see. Many aspects of GUIs are artifacts of a limited, two dimensional screen. Occluded windows and icons are the most obvious examples. If workstations had an infinite screen, there would be no reason to occlude one window with another. Users also often iconify objects simply to provide more screen space. A second important problem with access to GUIs, is that by the time information reaches the computer screen it has been converted into simple pixel values. Even textual output is converted into pixels which form letters. Therefore, it is no longer possible to translate the contents of the screen in any meaningful way by simply looking into the workstation's framebuffer.

There has been some research that has attempted to deal with the second problem of providing straightforward (direct mapping) access to graphical–based computer systems. Like the solutions which were developed for the older character–based systems, these

access systems for GUIs largely rely on auditory feedback to convey information to the user. Essentially these interfaces attempt to capture information enroute to the screen and convert it into an accessible format. [BBV91, Van89, Van, Fox91, KY87]. Most of this research has resulted in prototype systems that allow users with visual disabilities some degree of access to a computer with a GUI.

A characteristic of most of these research prototypes is that they are still based on the underlying "model" of the visual interface. For example, many systems provide the ability for icons or windows to speak their names when the on–screen pointer moves over them. Such systems are merely augmenting the graphical information presented on the screen with auditory feedback: a user who has no concept of icons or the arrangement of windows on the screen would be hard–pressed to be productive in such a system. Most interaction strategies used in conjunction with GUIs rely on visual information. For example, pop-up windows for error messages are very awkward to access in current prototype systems. If a bell signals an error message, then the blind user must search the screen for the message. If the message is not accompanied with an auditory cue, it will most likely go unnoticed.

We feel that a more promising approach would be to "step back" to an earlier point in the interface design process. GUIs have certain innate characteristics which makes them highly usable by sighted We need to identify the underlying principles that make such interfaces highly powerful (and even enjoyable) to the sighted users, and model these principles in an auditory interface. A simplistic mapping of the visual representation on the interface metaphor into an auditory representation is insufficient. Rather than simply assigning auditory attributes to the existing visual representation of the GUI, a new nonvisual interface is called for that does not rely on an underlying visual model.

Our platform for this project is high–end computer workstations running the Unix operating system. On this platform, the X Window System from MIT is the standard GUI. There are a large number of X applications available for such diverse tasks as financial analysis, electronic engineering, software development, and word processing. So far, very little work has been done on making this class of machines accessible. For computer users who are visually disabled, the availability of accessible Unix workstations will open new doors to employment, financial independence, and personal satisfaction.

THE MERCATOR ENVIRONMENT

Our objective is to produce a computing environment which will give blind users capabilities equal or superior to those provided to sighted users of computer workstations.

Currently, we are working on a prototype system will provide the same functionality as contemporary "desktop" graphical user interfaces. Users will be able to identify and manipulate objects (such as files, databases, and application programs) in the computing environment. They will be able to organize those objects into meaningful associations, transfer information between objects, and move objects around in the workspace. They will be able to create, destroy, modify, and duplicate objects. Objects will be identified and located by distinctive sound patterns.

The user will interact with our system through a combination of voice, tactile, and keyboard input. These input means will allow users to select and manipulate objects in an apparent space created by stereo and other audio effects. Output will be accomplished through the creation of this 3-D audio space and through speech synthesis.

An important characteristic of this interface will be the ability to transparently map graphics and textual output into audio and tactile output. This will enable existing applications to work within the Mercator environment fostering collaborative work between sighted and nonsighted users.

In addition to the high-level interface, we will be targeting key applications which may be rewritten to make the Unix workstation environment easier to access. For example, interactive help will be provided by an on-line hyper-audio system combining the functions of the on-line manual and the shell for locating, identifying, and describing objects. We will identify and replace these key applications as needed to enable blind users to more effectively use the system.

We will then develop the necessary extensions to X to support audio resources and operations. Along with it we will assemble a toolkit for the support of audio application development. We will then be able to build a complete non-visual computing environment based on a Unix workstation with X which could coexist in a network of other visual and non-visual X workstations.

This section delineates the design of our system and some of our implementation techniques. We will discuss the principles which have influenced our design, as well as the design itself. We shall focus on the basic components of our design (audio rooms, audio objects, and navigation through the system) as well as our justifications for our design decisions.

Design Principles and Techniques

In this section, we will describe the principles which have guided our initial design for the Mercator environment. We believe that these principles are important for developing a system which is easy to use, powerful, flexible, and portable to other platforms. Next we will describe the techniques we are using to produce an intuitive and powerful interface.

We began with these principles:

- The system should not simply retrofit visual interfaces with auditory information. Producing a completely new implementation of a suitable metaphor that relies solely on audition can produce a far more powerful interface for users with visual disabilities.
- Existing applications must be able to function within this new environment. An environment built from the ground up to support those who are blind will not be very useful if the users of the system cannot access the applications used by their sighted coworkers.
- The system must be "open–ended," so as to ensure that new applications which conform to industry standards will be able to function in the environment. This is the only way to ensure that sighted and blind workers will be able to collaborate on projects without having to "reimplement the wheel" whenever a new piece of software comes on the market.
- Wherever possible, the system must rely on software solutions rather than expensive hardware solutions. Software–based solutions decrease overall cost and increase portability. For example, 3D sound generation and speech recognition will be primarily implemented in software.
- The system must rely on industry standards wherever possible. The use of Unix and the X Window System ensure that Mercator will be portable to any vendor's workstation platform.
- The system must provide support for existing interface devices and hardware with which users are familiar. Examples include braille keypads and trackball devices.
- Users must be able to configure the system to suit their preferences. There are many variables in a system such as Mercator, and in many instances there is no "right" setting. Allowing users to customize their environments ensures user satisfaction and usability over a wide user base. This means that there will be several ways to accomplish the same ends for various operations in the system.
- The system must support learning through the use of a transition path for users as they gain experience and become familiar with the operation and capabilities of the system. Likewise the system should permit "short-cuts" for experienced users.
- Online, context–sensitive help is essential.

We feel that these principles are essential to the creation of a usable, effective interface which approaches the richness and functionality of existing visual interfaces.

Audio Rooms

A simplistic approach to designing a nonvisual environment might be to have all the user's applications, documents, and files arrayed around him or her in this synthetic audio environment. Obviously as the number of objects in the environment increases there is sensory overload: too many things are happening at once and it is difficult to cull the desired information out. The analog to this in the GUI realm is screen clutter: users have too many open windows and icons on the screen and it becomes difficult to find information quickly. In the audio domain, this clutter is called noise.

Research initiated at the Xerox Palo Alto Research Center (Xerox PARC) has resulted in a solution for visual window clutter [DAHC86, CAH87, Cla91]. Researchers at Xerox PARC found that for a given set of windows on a user's display, that users tended to follow a certain pattern of window accesses. For example, a given user may go back and forth many times between a word processor and a spell checker, then he or she may enter a mode where a spreadsheet and a database are accessed for a while. From either of these working modes the user may periodically go to an electronic mail application.

Xerox PARC researchers formalized the notion of these patterns of access of windows, and created the concept of "rooms." Rooms are basically a set of windows which have been grouped together because the user tends to access them that way. Users can create rooms and install applications and documents in them. For example, a user may create a "mail room" which has a mail reader, mail composition tool, a text editor, and so forth. A user may also have a room associated with a certain project that contains all the documents and files related with that project

as well as any applications useful in work related to the project.

In this way, users can create the layout of windows that suits them best. Rather than having one screen cluttered with all the windows and icons that a user may access throughout the day, the user toggles between these virtual screens and has easy access to the information within them.

In the Xerox PARC Rooms model, rooms are connected by doors. Users can create doors to rooms from within a given room. In this way the system of rooms and doors begins to resemble a network. The linkages of rooms and doors resembles the transition patterns that the users follow in their daily work.

The visual medium has much more bandwidth for communication than the auditory medium [Yor87]. Therefore it is even more important in an auditory–based environment to have some sort of mechanism to sort and segregate applications and documents by functionality and patterns of access. Users need to have complete control to layout their environments into easy–to–manage chunks (rooms) and then within those rooms, layout the applications within them in whatever way they feel is meaningful (just as a sighted user will arrange icons in a folder or windows on a screen according to his or her work priorities and sense of aesthetics).

Likewise, in the Mercator environment, a user creates a system of rooms and arranges their files and applications within these rooms. This is analogous to creating directories and moving files into them on systems such as Unix or MS–DOS. The system maintains the concept of a current room (much like a current directory in other systems). While a user is in a given room the computer presents spatialized sound to inform the user of the contents of the room and the locations of any objects in the room.

Audio Objects

What are the "contents" of such a virtual auditory room? Essentially, rooms can contain three basic types of objects: applications, files (non–executable data), and doors. Each object has a default sound which identifies its type. Users may choose unique sounds for objects to aid them in quickly identifying and locating objects. [Gav89].

Just as with light, sound has many different dimensions in which it can be perceived. Visual perception distinguishes such dimensions as color, saturation, luminescence, and texture. Audition has an equally rich space in which human beings can perceive difference: pitch, timbre, and amplitude. There are also much more complex, "higher level" dimensions, such as reverberance, locality, phase modulation, and so forth. Humans have a remarkable ability to detect and process minute changes in a sound along any one of these dimensions.[Ros90]

GUI designers have long relied on the richness of the visual medium to convey a multitude of information about objects in their interface: a group of icons can change color to indicate that they have been selected, a warning message is displayed in a brighter (more saturated) color, and so on. We plan to rely on (and experiment with) the human perception of multidimensional sound.

As an example, take the case of selecting an object. Naively, one may consider it sufficient to simply present the user with "the selection sound" (perhaps a tapping noise) whenever a selection is made. It is possible, however, to convey much more information by varying "the selection sound" along one or more dimensions. For example, the pitch of the sound could vary in indirect proportion to the size of the object selected. Thus, large ("heavy") files would make a deeper tap than smaller files. Further, one could imagine varying the tapping sound along the timbre dimension, so that applications sound like metal being tapped and files sound like wood being tapped. In each case, the user recognizes that the sound is a tap (that is, the user correctly identifies that the sound represents feedback from a selection operation), but the extra information carried along with the tap tells the user a great deal about the object which has been selected, yet does not require a tedious machine–generated voice to enumerate the qualities of that object.[Gav89]

By exploiting the dimensional properties of sound (in addition to the synthetic spatialization of sound), we feel that we can greatly enhance the user's experience with Mercator.

Navigation

Users can rely on the auditory qualities of objects in their environments to guide them as they "navigate" through the system. Within a room, users select objects via some control device. Currently, we are investigating the use of a trackball, a touchpad, and an analog joystick for control. Users select objects by moving to their apparent positions by using the input device. In essence, all rooms may be thought of as circular containers. That is, all objects in the room are equidistant from the center of the room (the user's initial position), but perhaps not equidistant from each other.

As the user "moves" his or her apparent position, the sounds representing the contents of the room shift

in relation to the motion. Thus, the behavior is similar to what is expected in a "real" room. As the user approaches an object at the perimeter of the room, the sound grows nearer. When the user is "on top of" the object, some special auditory cue is presented signifying that the object has been selected (for instance, the object may speak its name). Users can maneuver around the perimeter of the room via a left–right motion on the input device. Use of the input device in this manner will result in the user being presented with a list of the objects in the room as the user's apparent position changes.

Interacting with Objects

Once an object has been selected it may be operated on, just as an object in a GUI may be dragged, deleted, copied, and so forth. In the case of applications, they may be run after being selected. In the case of doors to other rooms, they may be "opened," putting the user in the new room. Mercator keeps a list of applications associated with each data file. By selecting a data file the default application for that type of data file may be run on it.

Once a user is "inside" an application the system switches modes. The user is now preoccupied with performing work inside the application and the navigation functions of the system go into the background. Users can, of course, rapidly get back to the rooms environment to get to other applications or files. The system exploits the multitasking in Unix by allowing users to switch rapidly between active applications. The application (if any) which is currently the "front" application is referred to as the foreground application. All other running applications are referred to as background applications.

Mercator can notify users of exceptional events while an application is in the foreground. Such events may occur as a result of background applications needing to communicate with the user, a change of state in the system, or the Mercator environment itself needing to communicate with the user. For example, if a compiler is running as a background process and the job exits because of errors, the system can inform the user of that fact.

IMPLEMENTATION STRATEGIES

The implementation of the Mercator environments depends on our ability to capture information enroute to the graphics server and to present a rich, three dimensional auditory environment. This section will discuss these problems in greater detail.

Capturing Information

Two characteristics of the X Window System lend themselves to implementations of accessible user interfaces. First, X was designed from the ground up as a network–oriented windowing system. Second, by convention, X applications are structured in such a way that it is possible to retrieve quite a deal of semantic information from them. Both of these characteristics together make possible an interface which can transparently interact with both the X Window system and its applications. We shall now investigate these properties in more detail.

X Client-Server Architecture. One of the characteristics of modern graphical windowing environments (such as X Windows or OS/2 Presentation Manager) is that there is often a separate process running on the computer whose sole responsibility is to control the windowing functions on the computer. This process (called the "server" in X parlance) regulates access to the computer's display and performs drawing and windowing operations at the request of applications (called "clients" by X, since they request services from the X server). In the case of X, the server and any clients do not even have to be on the same machine: they communicate with one another via interprocess communication[SG86].

This rigid dichotomy between client and server is a great boon for implementors of systems such as Mercator. Since the server alone has sole access to the physical display hardware of the workstation, it is impossible for individual programs to circumvent the server and render directly onto the display hardware. Furthermore, since the client applications must communicate with the server to request graphical output and windowing operations, it is possible to build a layer of software that intercepts requests from applications to the X server, operates on these requests itself, and (possibly) pass requests on to the X server.

To the client applications, this layer of software "looks" just like a normal X server; they "believe" that they are sending requests to an X server to create windows, move windows, draw lines, etc. This layer of software can then interpret the requests it receives, decide to act upon those it wishes to act upon, and discard those that are irrelevant[Sch87].

This is how we intend to allow existing applications to function in our environment. Since we can intercept virtually every graphics output request an application can make, we have full control over what the user sees (or in our case, hears) while the application is running.

For example, if an application wishes to pop up a dialog box, it sends a request to the X server telling it

that the application wants to display a new window at a given location, with the following text and buttons in it. The intercepting software layer can "catch" this request, route the text to a speech synthesizer, and wait for input from the user. Upon proper input, the software compatibility layer can synthesize a mouse click on the appropriate button. To the application it appears as if the user has located the dialog box (visually) on the screen and clicked the button by hand.

X Widgets and Resources. In the X world, most applications are built out of user interface objects, called "widgets." Widgets are the on–screen visual components of a graphical user interface. Scrollbars, buttons, and text fields are all examples of widgets. Each widget has a certain name and a class (or type) associated with it. Further, each type of widget has certain data associated with it in a list called a resource list. The data in the resource list controls virtually all aspects of the widget: what text is displayed, whether or not the widget is accepting input, etc. [SA87] It is possible to change or examine the values of individual resources. Thus, for a text displayer widget, it is possible to query the resources associated with the text displayer to determine what text is actually being displayed currently, what text is highlighted or selected, what are the properties of the displayed text, and so forth[Pet91].

By using resources, it is possible to query any client application and retrieve complete information about the structure of that application. For example, if the user asks for the various menu labels for an application, Mercator can query the application to find all the widgets of class MenuLabel, and then retrieve the resources for each of those to get the text associated with each menu label. Thus, upon request, the system can speak the list of menu labels associated with the application: "file, edit, font, help," and so on. Once the user selects a menu label, the system can then find the various menu subitems under this label. For instance, if the user selects the "edit" menu item, the system can respond with, "Menu subitems are: cut, copy, paste."

Since the widgets in an application reflect the structure of the application, the Mercator environment can have a wealth of semantic information available to it about the functionality and operation of any component of the application. By retrieving the resource values associated with any widget it is possible to determine all the various characteristics of that widget.

Generation of a 3D Audio Space

The Mercator user interface requires a sophisticated three dimensional audio environment which we will create through the use of modern digital signal processing techniques and psychoacoustics. In such an environment, sounds not only have their usual qualities of timbre and loudness but also they appear to be located at a point in space around the user's head. In nature, people localize (place in space) a sound by determining the difference between the sound that reaches their right ear versus the sound reaching their left ear. The Mercator environment will synthesize these differences with a computer to give the effect of sounds coming from a certain locale.

The differences in the sounds reaching each ear are very slight, and the human brain's ability to recognize fine degrees is great. Additionally the differences vary with the angle of the incident sound and the distance to the sound source. For this reason, significant amounts of computer resources must be spent generating these sounds to produce a noticeable effect accurately. We will use a digital signal processing program which will be given a single sound and a location as its input and will produce two sounds, one for each ear, which when played in stereo headphones give the listener the effect of the original sound at the correct location. [Ant79, Bla73, Opp83, OS75, Wol88]

STATUS AND FUTURE DIRECTIONS

We are currently in the advanced design stage and early implementation stage of a Mercator prototype for the NASA Marshall Space Flight Center.

This work represents the collaboration of researchers from the College of Computing and the Center for Rehabilitation Technology at Georgia Tech. The group also maintains an advisory board with representatives from Federal agencies devoted to the needs of the visually disabled and from the Unix and HCI communities.

The Mercator environment will provide a rich platform for HCI experimentation. We plan to explore issues such as how to provide feedback for navigating in an auditory environment, how to design auditory icons with multidimensional audio information, how to translate common visual actions such as skimming information, and cut-and-paste editing, and how to represent common graphical information such as bar charts, line drawings and so on.

Although this work is being done to develop an environment for computer users who are blind, we feel that the results from our efforts will be applicable to interface design in general. As the visual bandwidth

becomes more and more cluttered, designers will be forced to take advantage of other sensory channels. We believe the successful use of auditory information will result in a much richer and effective communication between humans and computers.

References

[Ant79] Andreas Antonious. *Digital Filters: Analysis and Design.* McGraw-Hill, 1979.

[BBV90] L.H. Boyd, W.L. Boyd, and G.C. Vanderheiden. The graphical user interface: Crisis, danger and opportunity. *Journal of Visual Impairment and Blindness*, pages 496–502, December 1990.

[BBV91] L.H. Boyd, W.L. Boyd, and G.C. Vanderheiden. Graphics-bases computers and the blind: Riding the tides of change. In *Technology and Persons with Disabilities*, 1991.

[Bla73] Jens Blauert. *Spatial Hearing.* MIT Press, 1973.

[Bux86] William Buxton. Human interface design and the handicapped user. In *CHI '86 Conference Proceedings*, pages 291–297, 1986.

[CAH87] Stuart K. Card and Jr. Austin Henderson. A multiple, virtual-workspace interface to support user task switching. In *CHI '87 Conference Proceedings*, pages 53–59, 1987.

[Cla91] Mark A. Clarkson. An easier interface. *BYTE*, pages 277–282, February 1991.

[DAHC86] Jr. D. Austin Henderson and Stuart K. Card. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics*, pages 211–243, July 1986.

[Fox91] Jackie Fox. Unlocking the door: Pcs and people with disabilities. *PCToday*, pages 43–51, March 1991.

[Gav89] William W. Gaver. The sonicfinder: An interface that uses auditory icons. *Human Computer Interaction*, 4:67–94, 1989.

[KY87] Richard M. Kane and Matthew Yuschik. A case example of human factors in product definition: Needs finding for a voice output workstation for the blind. In *CHI '87 Conference Proceedings*, pages 69–73, 1987.

[Lad88] Richard E. Ladner. Public law 99–506, section 508, electronic equipment accessibility for disabled workers. In *CHI '88 Conference Proceedings*, pages 219–222, 1988.

[Opp83] Alan Oppenhiem. *Signals and Systems.* Prentice-Hall, 1983.

[OS75] Alan Oppenhiem and Schafer. *Digital Signal Processing.* Prentice-Hall, 1975.

[Pet91] Chris D. Peterson. Editres–a graphical resource editor for x toolkit applications. In *Conference Proceedings, Fifth Annual X Technical Conference, Boston, Massachussets*, January 1991.

[Ros90] Thomas D. Rossing. *The Science of Sound.* Addison-Wesley, 1990.

[SA87] Ralph R. Swick and M.S. Ackerman. The x toolkit: More bricks for building user interfaces, or, widgets for hire. In *Conference Proceedings, Usenix, Winter*, 1987.

[Sch87] Robert W. Scheifler. X window system protocol specification, version 11. Massachusetts Institute of Technology, Cambridge, Massachusetts, and Digital Equipment Corporation, Maynard, Massachusetts, 1987.

[SG86] Robert W. Scheifler and J. Gettys. The x window system. *ACM Transactions on Graphics*, (2), April 1986.

[Van] G.C. Vanderheiden. Graphic user interfaces: A tough problem with a net gain for users who are blind. Prepared for Perspectives section of Technology and Disability.

[Van89] G.C. Vanderheiden. Nonvisual alternative display techniques for output from graphics-based computers. *Journal of Visual Impairment and Blindness*, 1989.

[Wol88] Stephen Wolfram. *Mathematica.* Addison-Wesley, 1988.

[Yor87] Bryant York. Pbe programming by ear (a programming environment for the visually handicapped). Technical Report BUCS Technical Report No.87-009, Boston University, September 1987.