SIGGRAPH '90 Workshop Report: Software Architectures and Metaphors for Non-WIMP User Interfaces

Mark Green
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada

Robert Jacob
Human Computer Interaction Lab
Naval Research Laboratory
Washington, D.C., 20375

Workshop participants
Peter Broadwell, Silicon Graphics Inc.
Stuart Card, Xerox Palo Alto Research Center
Rui Casteleiro, Instituto de Engenharia de Sistemas e Computadores, Portugal
Steve Feiner, Columbia University
Lisa Koelewyn, Boeing Computer Services
Bryan Lewis, IBM T. J. Watson Research Center
James S. Lipscomb, IBM T. J. Watson Research Center
Jock Mackinlay, Xerox Palo Alto Research Center
Jon Meads, Bell Northern Research
Roman Ormandy, Octree Software
Randy Pausch, University of Virginia
Michael E. Pique, Research Institute of Scripps Clinic
Jim Rhyne, IBM T. J. Watson Research Center
Luis Serra, Institute of Systems Science, Singapore
Chris Shaw, University of Alberta
Tim Skelly, Incredible Technologies

1. Workshop motivation

In the 1980's user interface design was dominated by the desk top metaphor. In this metaphor the display screen is viewed as the surface of a desk and windows are viewed as paper documents. A mouse is used to select items in windows, make choices from menus, and fill in forms. This style of interaction is well suited to document processing applications and is considerably better than the command line interfaces that were popular in the 1970's. The main problem with the dominance of the desk top metaphor is that it has blinded user interface designers and researchers to other interaction styles. It has been almost universally assumed that a desk top direct manipulation user interface is the best style of user interface for all applications. This point is supported by the facts that almost all user interface design and implementation tools support only this style of interaction, and that existing user interface style standards only address desk top user interfaces.

The main purposes of this workshop are to remind people that there are other styles of user interfaces, and to motivate research in appropriate tools to support the design and implementation of these user interface styles. It is not the purpose of this workshop to downgrade the desk top metaphor, since it has already shown its usefulness for a certain range of applications.

This report is divided into three sections. The first section deals with the nature of non-WIMP user interfaces and describes what we feel are their most important characteristics. The second section describes the hardware and software technologies that are required for the cost effective production of good non-WIMP user interfaces. The final section describes what we feel are some of the more important research problems in this area. There is some overlap between the different sections of this report, since some of the requirements for good non-WIMP user interfaces are also open research problems.

2. What are non-WIMP user interfaces?

The acronym, WIMP, stands for Windows, Icons, Mice and Pointing, and it is used to refer to the desk top, direct manipulation style of user interface. Thus, the term non-WIMP has been used to refer to any user interface style that is not based on the desk top metaphor. This definition is too broad for the purpose of this workshop, since it also covers command line user interfaces, which are not of interest to us. In addition, we wanted a term that characterizes this interaction style in a positive way. The term non-WIMP seems to be too negative in the sense that it doesn't capture the important properties of these user interfaces. Similarly, post-WIMP, even though it does exclude command line interfaces, still doesn't really describe the user interface styles we are interested in. Thus, our first task was to determine exactly what we meant by non-WIMP and then attempt to come up with a new term that better captures the types of user interfaces that interested us.

One way of characterizing these user interfaces is to identify the user interface styles that we consider to be non-WIMP. This process produced four non-WIMP styles, which are:

1) Virtual Reality - In many ways this is the classical non-WIMP style of user interface. After all, what could be further from the desk top than wearing a head-mounted display and a DataGlove?

2) Embedded Interfaces - This is the style of user interface that is used with computers that are embedded in other machines. The typical example of this style is the user interface to a photocopier. While we rarely think of this as a user interface, and it definitely doesn't use a mouse, it is becoming one of the dominant user interface styles. As the equipment becomes more sophisticated, and networked, the importance of this style of user interface will grow. Imagine the possibilities when all of your kitchen appliances are networked.

3) Notebook - The classical example of this style of user interface is Alan Kay's Dynabook. For this workshop the important characteristics of this style are the use of handwriting and gestures for input.

4) Hypermedia - This style of user interface was somewhat controversial, but was eventually added to our list of key styles. For this workshop, we were not interested in hypertext, but a full hypermedia system that included video, sound, graphics, and animation.

These four user interface styles are used to determine the important characteristics of non-WIMP user interfaces. After much discussion, the following five characteristics were considered to be important features of non-WIMP user interfaces.

1) High Bandwidth Input and Output - Most of the non-WIMP styles require large amounts of input and output bandwidth. In the case of a virtual reality user interfaces, the images presented to the user must be updated at least 10 times per second. In the case of handwriting the position of the pen and possibly the pen pressure must be measured many times per second. Handling large

amounts of input, and responding to it in an appropriate manner, is a problem that is never encountered in WIMP user interfaces, where there is rarely more than a few simple input events generated in a second.

2) Many Degrees of Freedom - Non-WIMP user interfaces have a large number of degrees of freedom in both the underlying application and the interaction devices. For example, in a molecular docking application the user must be able to control the position and orientation of the molecule in a three dimensional space. Similarly, a standard DataGlove has 16 degrees of freedom, each of which can be sampled up to 60 times per second. There is a problem mapping the degrees of freedom in the input devices onto the degrees of freedom in the application in such a way that the user can easily control the application.

3) Real-Time Response - All of the non-WIMP styles rely on real-time response to the user's actions. In the case of a head-mounted display, there must be very little delay between the time when the user moves his head and a new set of images are generated. If this delay is over 0.4 seconds the user will quickly become disoriented (possibly suffering simulator sickness) and will lose the illusion of interacting with a three dimensional environment. Similarly in handwriting the ink trail and other feedback must be produced as the user moves the pen, otherwise his writing ability will be impaired. This does not mean that a large amount of computing power is required, just that the cycles are there when the user needs them.

4) Continuous Response and Feedback - A non-WIMP user interface must be continually responding to the user's actions, since there isn't the well defined notion of a command that occurs in WIMP user interfaces. In classical WIMP user interfaces the user's actions are collected (with simple lexical feedback) until the end of the command is reached. At this point the command is passed to the application that generates the command's response. In a non-WIMP user interface there is a constant interaction between the user and the application. There are two aspects of this continuous interaction. First, most of the user input cannot be viewed as discrete events. It is continuous in the sense that it evolves over time, with no clearly defined beginning and ending. Second, the user interface must provide continuous feedback to the user's actions. The user interface cannot wait until the user has completed a motion before it provides feedback. In a virtual reality application, as the user moves through the environment the underlying model must be updated to account for the user's new position and any potential interactions with the objects in the environment. Similarly, as the user manipulates a molecule in a molecular docking application, the molecule must be redrawn in its new position, plus the energy of the current configuration must be computed.

5) Probabilistic Input - Since immediate response is very important, the user interface may need to guess what the user is trying to achieve in order to produce timely feedback. Occasionally these guesses will be wrong, and the user interface will need to correct its mistakes. Also, in the case of gesture or speech recognition, the recognition algorithms quite often produce a vector of possible responses along with their probabilities. For example, in a handwriting application, the user interface will echo the characters and words as they are entered, but the recognition algorithm may report several possibilities for each charater entered by the user. To provide immediate feedback the user interface must choose the most probable character. Once the complete word has been accumulated, it might be discovered that the most probable letter is not the correct choice; therefore, the user interface may need to correct its original guess. As more information is entered the user interface will discover that it has incorrectly recognized some letters, or possibly words, and must correct previous feedback. We have called this probabilistic input, and it is an important characteristic of non-WIMP user interfaces. WIMP user interface have well-defined digital inputs (mouse position, mouse button status, and keyboard strings), but non-WIMP ones often have fuzzier ranges. For example, the fact that the user's hand is close to the limit of a recognizable gesture, but not yet in or out of it, may be significant in the user interface. Speech recognizers often report fuzzy results like a 70% confidence of recognizing a word. One of the participants described an application in which floating-point precision was gladly sacrificed to pick up speed. In video games, precision is very low on the developer's priority list. Fuzzy input works well if the user is presented with low-latency high-rate continuous feedback. The user will automatically compensate, that is adjust his or her interactions, even if the imprecision in input crosses some qualitative interpretation boundary. (A supporting comment was made later in Brooks et al.'s [Brooks Jr.1990] SIGGRAPH paper on a manipulator input device: "Haptic senses are easily fooled; transducer error is compensated by the user.")

This list of characteristics suggests that non-WIMP user interfaces could be called highly interactive user interfaces. This led to an active discussion of what interactive means in this context.

Non-WIMP applications will need to be "truly interactive." This phrase has also been discussed in the context of WIMP applications, but here the meaning is: The user's set of valid interaction techniques should not be restricted, no matter what state the application is in. If the user has issued a command that will take several seconds for the system to complete, the user should be able to continue to perform any interaction not logically excluded. (An oft-quoted example is the Aspen Movie Map [Lippman1980], where even though you just issued a command to turn right at the next intersection, you can still choose to do anything in the meantime like enter a building.)

An example of this quality in desktop applications is the ability to start a long-running command in one window, then switch focus to a different window. The overall system remains interactive although any particular window could freeze. Achieving even this level of interactivity has already caused a minor software revolution (window managers, object-oriented construction, multi-tasking and multi-threading). Interactivity is much more important for non-WIMP user interfaces because we will have multiple input paths. For example, the user's left hand (or head or voice) might initiate a complex task while the right hand is writing. And the consequences will be greater; we certainly don't want the entire world to freeze, and we may not be able to take advantage of a simplistic partitioning of the world into windows.

Another definition for "interactive" is: not having to make a conscious decision between each step. If you need to focus your mental attention only once, it feels interactive even if it requires several physical steps to reach your goal. This forms the basis for the distinction between interactive and continuous interaction. In the case of interactive system there is a conscious set of steps, that is there is the notion of fixed commands and interactions. In a continuous system there is a smooth flow from one interaction to the next, and the user is not aware of command boundaries.

There was general agreement that the distinction between continuous and interactive work needs to be drawn in the system dialog design, but that the distinction ultimately lies not within the system, but within the user. Stu Card gave a rule of thumb that the dividing line between continuous and interactive usually lies at about 500 msec. for user actions and 100 msec. for display. For graphics interaction, which involves both, this leaves the range between 500 msec. and 100 msec. where the issue is most problematical.

Instead of traditional arguments on perception of smoothness, the distinction can be made on behavior. If a user continues input smoothly or repeatedly between infrequent screen responses, then the work is continuous. If, however, the user stops input, waits for a response, and then decides whether or not to give another input, then the work is interactive. This means that on the same system, one user could work interactively and one continuously. Consequently, on the border between the two types of input, it is not within the power of the system to mandate which is which.

In conclusion, we really don't have a term that can be used to replace non-WIMP. The term highly interactive user interface is somewhat indicative of the type of user interface that we are interested in, but it doesn't seem to be good enough to replace

non-WIMP. The discussion did generate a list of five important characteristics of non-WIMP user interfaces and an attempt to distinguish between interactive and continuous user interfaces.

3. Hardware/software support for non-WIMP user interfaces

There are a number of hardware and software technologies that are required for the cost effective production of good non-WIMP user interfaces. In some cases these technologies are fairly well understood and it's just a matter of equipment manufacturers offering them with their graphics workstations. In other cases, the technologies are not readily available and a considerable amount of research may be required before they will be available for general use. In this section we review the technologies that will be required for non-WIMP user interfaces.

3.1. Real-time operating systems

We see a common need in all non-WIMP applications for good performance, particularly low latency between input and feedback, and a consistent rate of output. When the user interacts with adevice, the results of this interaction must be communicated to the relevant parts of the user interface as quickly as possible. The operating system shouldn't buffer input or perform any other processing which is not absolutely necessary. It is better to sacrifice display quality when necessary than it is to allow the display (or other output) update rate to vary. These requirements are those best addressed by a real-time operating system with deadline scheduling. We need to know when the display update deadline is about to be overrun, so we can choose a faster representation or skip the update. We also need to know the real-time spent in display update, so the display routines can be adapted to the available processing time on subsequent update cycles. Note that real-time is important, not just the CPU time required by the computations. UNIX operating systems usually do not support the features outlined above. It should be noted that a real-time system is a universal characteristic of video games.

"Real-time" performance is frequently confused with "fast." Fast is important, but a tight predictability of interaction — low latency of input and smooth output rate — is equally important.

3.2. Performance monitoring tools

Non-WIMP applications will have in common a serious need for performance-monitoring tools. We have found, even in relatively simple virtual world applications, that alot of effort is expended on such tasks as time-stamping inter-process events; recording the event streams for postponed performance analysis, in order to determine where the time is being spent (network/display/simulation/inputs); and finally tuning. The fact that latency is as important as raw speed makes the analysis more tedious. Preferably there will be support for such tools built into the operating system, especially if the system is purported to be "real-time."

3.3. Stereo

Stereo graphics is an important part of many non-WIMP user interfaces. Stereo is one of the more important cues that are used in depth perception, therefore, it can play an important role in three dimensional user interfaces. In order for this technique to be effective it must be implemented properly, otherwise the eyes are not capable of focusing on the stereo image.

Incorrect perspective stereo display is done by rotating the viewed object about a central point, or by the equivalent action of angling the two virtual cameras inward. This produces a small vertical misalignment of the images that is impossible for the eye to fuse, because one image is taller than the other in one part of the picture and the other is taller in another part of the picture. Correct perspective display requires the cameras to be pointed parallel to each other [Lipton1990] with the clipping pyramids sheared to avoid visual rivalry on the left and right edges [Williams1990].

Fast display, perhaps twice as fast as perspective stereoscopic display, is the chief attraction of orthographic (non-perspective) stereo, which otherwise looks a little strange, because the eye expects things farther away to be drawn smaller. Graphics displays have been limited for the past few years by coordinate transformation speed, not by rendering speed, and perspective stereo is usually done with two 4x4 coordinate transformations with perspective division. Orthographic transformation for one eye view can be done with a faster 3x3 transformation with no perspective division, and then these transformed coordinates can be sheared [Lipscomb1989] at the cost of one multiplication and one addition per point to produce the other eye view. If the transformed coordinates are unavailable, only a 3x3 sheared rotation of the untransformed coordinates is required.

3.4. Multiple I/O paths

In highly interactive interfaces the user often interacts with multiple devices simultaneously. For example, automobile drivers operate the steering wheel, and the gas or brake pedal at the same time. This type of behavior is rarely seen in computer based user interfaces. Although some desktop and other current computer interface styles use multiple input devices, their dialogue design makes use of the devices logically sequential, not parallel. The user moves the mouse and then types on the keyboard; the user almost never types a character while moving the mouse, even though it is technically possible.

Current UIMS technology typically handles concurrent multiple input devices by serializing all their inputs into one common stream. This is well suited to serial dialogue styles, but is not appropriate for styles where the inputs are logically parallel (that is, where the user thinks of what he is doing as two simultaneous actions). Such parallel dialogues could of course be programmed within the old model, but it would be preferable to be able to describe and program them in terms of logically concurrent inputs, rather than a single serial token stream.

However, the parallel input streams are not entirely independent either. Simultaneous or near-simultaneous inputs from different devices may need to be interpreted with reference to one another. The user might say, "delete that object" while pointing or looking at something. To understand the spoken input, the simultaneous gesture input must be processed. A brief time may have elapsed from when the user spoke the phrase until the system received and processed it. This means that the spoken "that" in the phrase must be interpreted in the context of the gesture the user was making at the time he said "that," not at the time the speech was finally processed. Inputs from each of the concurrent channels must thus be time-stamped and then later interpreted with respect to inputs on the other channels that occurred at the same time.

A second problem is that of data volume. Handling a collection of high-bandwidth, parallel inputs constitutes a significant processing load in itself. Since accurate time-stamping of the concurrent inputs is important for interpretation, this implies a considerable increase over current systems in the proportion of processing power devoted to I/O operations. Interpreting and dispatching the inputs will also be a significant load, as discussed in the next section.

3.5. Input data reduction

User interaction in non-WIMP interfaces will be characterized by devices that are more closely matched to the user's characteristics than the computer's. Rather than training a user to operate a keyboard of switches that generate symbols that are easy for a computer to interpret, the user will be allowed to use his existing, natural communicative abilities and the computer will have to perceive and interpret his actions. For example, the user might be able to make arbitrary gestures or hand motions as input. These would be sensed by 3D trackers, DataGloves, or perhaps an array of cameras. The computer will then process the raw input from these devices to produce higher-level actions meaningful to the dialogue. Similar computer processing is required by speech, eye movements, handwriting, and other such input media. The translation of raw keyboard input into meaningful tokens is so trivial (perhaps just consisting of translating shift key combinations into single characters) that this aspect of user input processing has been neglected by user interface designers. However, with more powerful input devices, translation of the raw device inputs into dialogue tokens becomes a complex and significant problem.

In many cases, such as speech, it is an unsolved problem. Input data reduction can now occupy a large fraction of the available computer processing power. For example, Bryan Lewis reported that his virtual reality system is implemented as a distributed system in which multiple workstations are linked together to provide the needed processing power. Several of the workstations are entirely dedicated to input processing.

One simple form of input processing that is often desirable is filtering of the data stream. Low-pass temporal filtering of inputs such as jerky hand movements or gestures is often helpful. An extreme case of a filter is the restriction of multiple degrees of freedom down to one or two degrees, which sometimes will provide the easiest interaction. However, temporal filtering will conflict with high performance objectives, not so much because of its computational requirements but because low-pass filtering inherently reduces responsiveness to rapid or sudden inputs.

3.6. Sound

Sound is an important, and often neglected, part of our world. Sound can provide us with another high bandwidth input and output channel, in addition to the visual channel. If bothhands are occupied with an important manipulation, and the eyes are focused on this manipulation, the sound channel provides additional feedback and another means of entering commands. The sound channel can be divided into two parts depending upon whether speech is being used. For input, speech is the important aspect of the sound channel. For most non-WIMP applications the speech recognition component must have a high recognition rate and require a minimal amount of training. For most of these applications a large recognition vocabulary is not required, and the recognition vocabulary can depend upon the context of the interaction, which also reduces the size of the recognition vocabulary.

For output, both speech and non-speech sound is required. Good quality speech can be used to provide feedback, along with help and instructions on how to use the system. Non-speech sound is a useful feedback mechanism for a number of three dimensional interactions. For example, a different sound can be associated with each DataGlove gesture, when the user makes the gesture the sound is produced to indicate that the system has correctly recognized the gesture. Similarly, sound can be used to indicate proximity to an object in three dimensional space. The pitch of the sound could be a function of the distance between the user and the object, and a special sound can be used to indicate when an object has been grasped or collided with.

Another issue with sound as an output medium is positioning the sound source within the three dimensional volume occupied by the user. This is particularly important in virtual reality user interfaces where the sound source has a visual representation that is located in the user's three dimensional space. The illusion of reality provided by the interface will quickly be broken if the sound doesn't come from the same position as the object that produces it.

3.7. Distributed software

A wide range of non-WIMP user interfaces use several cooperating processors. This is due to the continuous nature of the input provided by the user. The user is producing a stream of input information that must be processed in real-time, and this processing usually entails expensive operations, such as filtering and recognition. With current hardware technology, the only cost effective way of handling multiple input devices of this nature is to use a separate processor for each device. Also, producing the two images for a head mounted display is usually accomplished by using two graphics displays. The use of distributed computing provides the hardware resources required to implement these non-WIMP user interfaces, but this approach introduces a whole new set of problems.

The construction of distributed applications is more difficult than the construction of sequential applications running on a single processor. We need to have software tools that facilitate the construction of distributed applications. In addition, distributed applications introduce another source of timing problems. As outlined in section 2, the lag between user input and response is an important factor in the quality of the user interface. With several processors involved with a single interaction, it is difficult to determine the source of lags. One suggested approach to this problem is to time stamp all the data as it is processed. In a distributed system this is not as easy as it sounds, since the clocks on all the processors must be synchronized in order for the time stamps to be meaningful. Therefore, in addition to program construction tools, we need tools that enable us to monitor the performance of distributed systems and determine the sources of time lags.

3.8. Video

There are two sources of video that could be used in non-WIMP user interfaces, which are live and pre-recorded. Live video could be used in remote manipulation applications and in CSCW applications where each of the participants in the meeting sends a live video feed to the other participants. Pre-recorded video could be used in a wide range of instructional applications, and in artistic applications where interacting with video sequences could be one form of artistic expression.

The ability to display standard video signals is now available on some micro-computers and workstations, but it is still not a common feature. In order to be useful in non-WIMP applications, the video display must be treated in the same way as any other form of graphics, it cannot be restricted to a fixed window on the screen. For example, the video could be projected onto the wall of a room that is part of a model that the user is walking through. In order to do this the video signal must be treated as a raster that can be mapped onto any of the polygons in the user interface.

4. Future research directions

This section outlines some of the areas where further research would be fruitful. This section is divided into sub-sections based on major research areas, and each sub-section describes a range of research projects.

4.1. Input transduction

A wide range of input devices are used in non-WIMP user interfaces, and as outlined in previous sections these devices can generate large volumes of data that cannot easily be divided into discrete chunks. The properties of these devices are radically different from those of the devices used in WIMP user interfaces, which leads to a range of research topics at the input device level.

Since there is wide range of input devices, and a user interface may utilize one or more of these devices depending upon the hardware configuration it encounters, the user interface needs some way of determining the current device configuration. In WIMP user interfaces this is not a major problem, since these user interfaces always assume that a mouse and keyboard are the only available input devices. But, in a non-WIMP user interface, a variety of joysticks, Polhemus or hand digitizers can be used for essentially the same purpose. Obviously, the user interface needs to know which of these devices it is dealing with so it can use the appropriate driver software, and more importantly modify its interaction techniques to account for the properties of the available devices. No vendor of input devices or graphics hardware has produced a general scheme for input device identification.

There are several ways of approaching this problem. One approach is for the device to give only its name, with the software figuring out what the name means. One problem is that the inventor of a new device must make its name known to all computers that might use it. Of course, the new device could pretend that it is a device already known to the system, but such coercion may not be appropriate or possible.

Another option is for the device to say its type using a common vocabulary (e.g. remain-in-position, spring-return, isometric, body-mounted, or held). But invention knows no bound. New devices with new characteristics would have to identify themselves with new words that may not be widely known outside themselves. This may lead the application program to fail to make use of

all that these devices can do or to assign duties to them for which they are not suited.

A third approach is to use some form of device taxonomy or specification language for input devices [Mackinlay1990]. In this approach the developer of a new device would produce a high level specification of the device's functionality, which could then be added to a database describing all the locally available devices. Again, problems will arise if a radically new device is produced that can't be described using the existing specification language.

There are a number of low level device problems that need to be investigated. For example, some six dimensional devices suffer from inadvertent axis activation. When a user pushes to translate there may be inadvertent rotation, which in turn tilts the translation axis askew from the desired direction. Similarly, when the user rotates the device there is a good chance that he or she will also translate it a small distance. One solution is to use large thresholds before response occurs, which requires manhandling the device. Another solution is to put translation and rotation on separate devices, which can leave no hand free for something else. One can instead use a joystick constructed mechanically to isolate axes while still keeping them within reach by fingers alone, which severely limits the number of degrees of freedom. A mode switch to turn degrees of freedom on and off is another weak alternative. Each solution has its problems.

Another low level device problem is noise and lag. Some of these devices report a continuous position and orientation in space. Measurement errors and noise within the analogue electronics results in a certain noise level in the device readings. For some applications, this noise level is okay, particularly when the user has enough feedback to easily correct for the noise level. For other applications, such as measuring head position for a head-mounted display, small amounts of noise are not acceptable. The typical solution to this problem is to filter the data coming from this device, but this can introduce a lag into the device's response, which can be just as serious as the noise problem. We need to be able to characterize the noise that occurs in these readings and develop filtering schemes that remove as much noise as possible without increasing lag. Predictive filters are worth investigating for these applications. Some commercial devices already perform some filtering of their data, but it is not clear that the appropriate filters are being used and none of them attempt predictive filtering. These devices should either not filter the data at all (thus allowing the user to construct appropriate filters), or provide a mechanism for downloading user defined filters.

To achieve good performance and high interactivity, we will not often be able to afford a blocking, synchronous style of messages and acknowledgments. In fact it will often be impossible to guarantee accurate interpretation. That is, context information will be coming in from multiple devices, and the application will be making its best guess at an interpretation as quickly as possible, for the sake of fast continuous feedback. Thus it will often be desirable to undo an erroneous interpretation, asynchronously and preferably without sacrificing interactivity, not an easy task. Expanded error-reporting services will probably be needed in the operating system.

4.2. Time management

In several ways, time (as opposed to the sequence of device interactions) has not been treated as a "first class citizen" in user interface specification or implementation. Better support for handling time will be needed for more highly interactive interfaces.

There are several facets to this problem. One is handling the specification of a user-computer dialogue in which the times at which inputs and outputs occur are significant. Existing user interface description languages (UIDLs) do not handle this problem. Recent research has investigated the use of temporal logic in user interface specification, but this is only a first step since temporal logic is still mainly concerned with sequencing and not the actual times at which interactions occur. UIMS's rarely make control or use of time available to the interface designer in any serious way. Syntax is considered to be the sequence of inputs and outputs, irrespective of their actual times. Notations and concepts for handling specific times in a user interface design and UIMS implementation will thus be needed.

Another aspect of time in the UIMS, that of support for real-time, deadline-based computations, has been discussed in several sections of this report. This requires good support for real-time operations from the underlying hardware and operating system. In fact, a theme that emerged frequently throughout the workshop was that new, highly interactive interfaces are more like real-time application programs than traditional user interface programs and will require similar real time support from the operating system. This type of support is lacking in existing user interface programming environments.

In a highly interactive interface, it is often necessary to produce an output before some deadline (such as the next screen update). Having a slightly "poorer" output on time is preferable to having a better output, but too late. One strategy is to use conservative algorithms, which always produce the results on time. It would be better to have support for fully exploiting the available time. This requires the ability to choose in real time between alternative algorithms, based on the time available to the next deadline. A way of predicting the performance of alternative algorithms, in real time, based on the data to be presented to them, would help in this endeavor. Also useful would be development of algorithms that inherently permit graceful degradation or improvement. That is, an algorithm that can continuously adapt its "quality" to the available time, making optimal use of whatever time is available to improve the output, but reliably completing within its deadline. Incremental rendering techniques are one approach to this problem [Bergman 1986]. These techniques could be extended for highly interactive user interfaces, or form the motivation for other approaches to this problem.

4.3. Design support

As we have seen, highly interactive interfaces are characterized by user input that may be continuous, concurrent, and probabilistic. Existing user interface description languages (UIDLs) do not handle such inputs; they almost universally assume that the input is a serial stream of discrete deterministic tokens or events for which only sequence, not actual time, is significant. New UIDLs are needed to describe dialogues whose inputs and possibly outputs are continuous, concurrent, and probabilistic. As noted, the concurrent inputs may be interrelated, so the UIDL must be able to specify interpretations of inputs that depend on the states of other inputs at the same point in time. Also, as noted, better representations for real-time, rather than just sequence, will be needed in the UIDL.

A principal component of design support for highly interactive interfaces is, then, a UIDL for continuous, parallel, probabilistic inputs. While the need for such languages and the requirements that they must address were discussed in the workshop, the shape they might take was not discussed. The workshop concentrated on a higher-level description of what kinds of requirements such a language would have to support, but not how to fulfill them. The latter will provide an important and fruitful area of research for the next generation of user interface software.

Non-WIMP user interfaces will require a new set of interaction techniques. The interaction techniques that we have developed so far are mainly two dimensional and they may not transfer to the three dimensional domain that is used by many non-WIMP user interfaces. Similarly, we will need a new set of guide-lines and metaphors for the development of non-WIMP user interfaces. At the present time there are only a few examples of good non-WIMP user interfaces. We need to generalize the experience gained in developing this kind of user interface to produce a design methodology for non-WIMP user interfaces.

4.4. Implementation support

User interface management systems (UIMS) have the potential to be very useful in non-WIMP applications. But we need to be more precise about the meaning of "UIMS." In the WIMP world "UIMS" often means a window-manager-like toolkit, and application logic is often closely bound to the characteristics of

the desktop display. The other definition of UIMS in the literature is an event-handling executive, independent of display characteristics, serving as a switchboard to coordinate multiple objects/processes/machines. This is the fruitful way to think of UIMS's for non-WIMP applications.

UIMS's and other implementation tools for non-WIMP user interfaces must be broader in scope than the tools that have been produced for WIMP user interfaces. In the case of WIMP user interfaces, the implementation tools have produced code that runs in a relatively restricted environment. The implementation tools for this new generation of user interfaces must deal with a wide range of problems. As shown in previous sections, the implementation on a non-WIMP user interface is sometimes distributed over several processors. The implementation tools must be able to generate code that can deal with a distributed environment. Similarly, they must be able to deal with multiple concurrent activities, which has never been a problem for the WIMP implementation tools. Thus, we may need to broaden our view of what a UIMS consists of.

References

Bergman 1986.
L. Bergman, H. Fuchs, E. Grant and S. Spack, Image Rendering by Adaptive Refinement, *SIGGRAPH'86 Proceedings*, 1986, 29-37.

Brooks Jr. 1990.
F. P. Brooks Jr., M. Ouh-Young, J. J. Batter and P. Jerome, Project GROPE - Haptic Displays for Scientific Visualization, *SIGGRAPH'90 Proceedings*, 1990, 177-185.

Lippman, 1980.
A. Lippman, Movie-Maps: An Application of the Optical Videodisc to Computer Graphics, *SIGGRAPH'80 Proceedings*, 1980, 32-42.

Lipscomb 1989.
J.S. Lipscomb, Experience with stereoscopic display devices and output algorithms, *Proc. SPIE 1083*, (1989), 28-34.

Lipton 1990.
L. Lipton, *StereoGraphics Corp. programmer's guide*, StereoGraphics Corp., 1990.

Mackinlay 1990.
J. Mackinlay, S.K. Card and G.G. Robertson, A Semantic Analysis of the Design Space of Input Devices, *Human-Computer Interaction 5, 2&3 (1990), 145-190.*

Williams 1990.
S.P. Williams and R.V. Parrish, New computational control techniques and increased understanding of the depth-viewing volume of stereo 3-D displays, *Proc. SPIE 1256*, (1990).

Reading list

1. M. M. Blattner, D.A. Sumikawa and R.M. Greenberg, Earcons and Icons: Their Structure and Common Design Principles, *Human-computer Interaction 4*, (1989), 11-44.
2. R. A. Bolt, "Put-That-There": Voice and Gesture at the Graphics Interface, *Proceedings of SIGGRAPH'80*, 1980, 262-270.
3. R. A. Bolt, Gaze-Orchestrated Dynamic Windows, *Proceedings of SIGGRAPH'81*, 1981, 109-119.
4. E. G. Britton, J.S. Lipscomb and M.E. Pique, Making Nested Rotations Convenient for the User, *Computer Graphics 12*, 3 (Aug. 1978), 222-227.
5. F. P. Brooks Jr., Walkthrough - A Dynamic Graphics System for Simulating Virtual Buildings, *Proceedings 1986 Workshop on Interactive 3D Graphics*, 1986, 9-21.
6. W. A. Buxton, R. Sniderman, W. Reeves, S. Patel and R. Baecker, The Evolution of the SSP Score Editing Tools, *Comp. Music J., 3*, 4 (1979), 14-25.
7. W. A. S. Buxton, E. Fiume, R. Hill, A. Lee and C. Woo, Continuous Hand-gesture Driven Input, *Proc. Graph. Interface '83*, 1983, 191-195.
8. W. A. S. Buxton, Lexical and Pragmatic Considerations of Input Structures, *Computer Graphics*, Jan. 1983, 31-37.
9. W. A. S. Buxton and G. Kurtenbach, Editing by Contiguous Gestures: A Toy Test Bed, *ACM Chi'87 poster*, 1987, 1-5.
10. W. Buxton, Introduction to This Special Issue on Nonspeech Audio, *Human-computer Interaction 4*, (1989), 1-9.
11. S.K. Card, J. Mackinlay and G. G. Robertson, The Design Space of Input Devices, *ACM CHI'90 Proc.*, Apr. 1990, 117-124.
12. J. M. Carroll and R. L. Mack, Metaphor, Computing Systems, and Active Learning, *Int. J. of Man-Machine Studies 22*, (1985), 39-57.
13. M. Coleman, Text Editing on a Graphic Display Device Using Hand-drawn Proofreader's Symbols, in *Pertinent Computing Concepts in Computer Graphics*, M. Faiman and J. Nievergelt (ed.), 1969, 282-290.
14. D.C. Engelbart and W. K. English, A Research Center for Augmenting Human Intellect, *Fall Joint Computer Conference*, 1968, 395-410.
15. S. Feiner and C. Beshers, Worlds within Worlds: Metaphors for Exploring n-Dimensional Virtual Worlds, *UIST'90 Proceedings*, 1990, 76-83.
16. S. S. Fisher, M. McGreevy, J. Humphries and W. Robinett, Virtual Environment Display System, *Proceedings of ACM 1986 Workshop on Interactive 3D Graphics*, 1986, 77-87.
17. J.D. Foley, Interfaces for Advanced Computing, *Scientific American*, October 1987, 126-135.
18. H. Fuchs, Z. Kedem and B. Naylor, On Visible Surface Generation by A Priori Tree Structures, *SIGGRAPH'80 Proceedings*, 1980, 124-133.
19. H. Fuchs, G. Abram and E. Grant, Near Real-Time Shaded Display of Rigid Objects, *SIGGRAPH'83 Proceedings*, 1983, 65-69.
20. W. W. Gaver, Auditory Icons: Using Sound in Computer Interfaces, *Human-Computer Interaction 2*, (1986), 167-177.
21. J. J. Gibson, *The Ecological Approach to Visual Perception*, Houghton Mifflin Company, Boston, 1979.
22. J. N. Gitlin and S. O. Trerotola, Beta Test of Radiology Reporting System Using Speech Recognition, in *SCAR90: Computer Applications to Assist Radiology*, R. L. Arenson and R. M. Friedenberg (ed.), California: Symposia Foundation, 1990, 151-157.
23. M. Green, Artificial Reality: A New Metaphor for User Interfaces: *International Conference on CAD & CG, Beijing, China*, 1989
24. M. Green and C. Shaw, The Datapaper: Living in the Virtual World, *Graphics Interface'90 Proceedings*, 1990, 123-130.
25. M. Green, Virtual Reality User Interfaces: Tools and Techniques, *Computer Graphics International'90*, Singapore, 1990, 51-68.
26. A. G. Hauptman, Speech and Gestures for Graphic Image Manipulation, *Conference on Human Factors in Computing Systems - CHI 89 Conference Proceedings*, Austin, Texas, April-May 1989, 241-245.
27. J. E. Holmgren, Toward Bell System Applications of Automatic Speech Recognition, *The Bell System Technical Journal 62*, 6 (July-August 1983), 1865-1880.
28. R. J. K. Jacob, What You Look At is What You Get: Eye Movement-Based Interaction Techniques, *Proc. ACM CHI'90 Human Factors in Computing Systems Conference*, 1990, 11-18.
29. J. Kim, On-line Gesture Recognition by Feature Analysis, *Proc. Vision Interface'88*, June 1988, 51-55.
30. M. W. Krueger, *Artificial Reality*, Addison Wesley, Reading, MA, 1983.
31. M. Krueger, VIDEOPLACE — An Artificial Reality, *CHI'85 Proceedings*, 1985, 35-40.
32. G. Lakoff and M. Johnson, *Metaphors We Live By*, The University of Chicago Press, 1980.

33. C. Y. Laporte and J. Houle, A Voice-Interactive Evaluation System for Command Functions in Military Aircraft, *Microcomputer Applications 8, 1(1989) 20-26.*
34. J. Lewis, L. Koved and D. Ling, Dialogue Structures for Virtual Worlds *(CHI'91 Proceedings (in press)*, 1991.
35. J. Ligget and G. Williams, An Empirical Investigation of Voice as an Input Modality for Computer Programming, *International Journal of Man-Machine Studies 21,* (1984), 493-520.
36. J. S. Lipscomb, F. P. Brooks Jr. and M. E. Pique, The GRIP-75 Man-machine Interface, *ACM SIGGRAPH Video Review 4,* (Aug. 1981).
37. J. S. Lipscomb, A Trainable Gesture Recognizer, *Pattern Recognition,* in press.
38. J. Mackinlay, S. K. Card and G. G. Robertson, A Semantic Analysis of the Design Space of Input Devices, *Human-Computer Interaction 5,* 2&3 (1990), 145-190.
39. R. Makkuni, A Gestural Representation of the Process of Composing Chinese Temples, *IEEE Comp. Graph. & Appl. 7,* 12 (December 1987), 45-61.
40. G. L. Martin, The Utility of Speech Input in User-computer Interfaces, *International Journal of Man-Machine Studies 30,* (1989), 355-375.
41. M. Minsky, M. Ouh-young, O. Steele, F. P. Brooks and M. Behensky, Feeling and Seeing: Issues in Force Display, *Computer Graphics 24,* 2 (1990), 235-244.
42. L. J. Moss and R. M. Friedenberg, Initial Experiences with an Online Speech Recognizer, in *SCAR90: Computer Applications to Assist Radiology,* R. L. Arenson (ed.), California: Symposia Foundation, 1990, 147-150.
43. R. Nakatsu, Anser: An Application of Speech Technology to the Japanese Banking Industry, *Computer Journal of the IEEE,* August, 1990, 43-48.
44. R. Pausch and J. H. Leatherby, A Study Comparing Mouse-Only Input vs. Mouse-Plus-Voice for a Graphical Editor, *University of Virginia, Computer Science Report # Tech. Rep.-90-17,* July 1990.
45. J. R. Rhyne and C. G. Wolf, Gestural Interfaces for Information Processing Applications, *RC 12179, IBM Research,* Sept. 1986.
46. J. Rhyne, Dialog Management for Gestural Interfaces, *ACM Computer Graphics 21,* 2 (1987), 137-142.
47. A. H. Robbins, D. M. Horowitz, M. K. Srinivasan, M. E. Vincent, K. Shaffer, N. L. Sadowsky and M. Sonnenfeld, Speech-controlled Generation of Radiology Reports, *Radiology 164,* 2 (1987), 569-573.
48. G. Robertson, S. Card and J. Mackinlay, The Cognitive Coprocessor Architecture for Interactive User Interfaces, *UIST'89 Proceedings,* 1989, 10-18.
49. C. Schmandt, M. S. Ackerman and D. Hindus, Augmenting a Window System with Speech Input, *Computer Journal of the IEEE,* August 1990, 50-56.
50. K. Shoemake, Animating Rotation with Quaternion Curves, *SIGGRAPH'85 Proceedings,* 1985, 245-254.
51. A. Singh, J. Schaeffer and M. Green, Structuring Distributed Algorithms in a Workstation Environment: The FrameWorks Approach, *International Conference on Parallel Processing,* August 1989.
52. R. B. Smith, The Alternate Reality Kit: An Animated Environment for Creating Interaction Simulations, *Proceedings of 1986 IEEE Computer Society Workshop on Visual Languages,* 1986, 99-106.
53. R. B. Smith, Experiences with The Alternate Reality Kit: An Example of the Tension Between Literalism and Magic *Proceedings of CHI and Graphics Interfaces 1987,* 1987, 61-67.
54. D. J. Sturman, D. Zeltzer and S. Pieper, Hands-on Interaction with Virtual Environments, *Proc. ACM SIGGRAPH Symposium on User Interface Software and Technology,* Williamsburg, Va., 1989, 19-24.
55. I.E. Sutherland, Head-Mounted Three-Dimensional Display, *Proceedings of the Fall Joint Computer Conference, vol. 33,* 1968, 757-764.
56. C. C. Tappert, C. Y. Suen and T. Wakahara, The State of the Art in On-line Handwriting Recognition, *IEEE Trans. on Pat. Anal. and Mach. Intel. 12, 4,* (Aug. 1990), 787-808.
57. D. Visick, P. Johnson and J. Long, The Use of Simple Speech Recognizers in Industrial Applications, *Proceedings of INTERACT 84, First IFIP Conference on Human-Computer Interaction,* 1984.
58. C. Wang, L. Koved and S. Dukach, Design for Interactive Performance in a Virtual Laboratory, *Proceedings of 1990 Symposium on Interactive 3D Graphics,* 1990, 39-40.
59. D. Weimer and S. K. Ganapathy, A Synthetic Visual Environment with Hand Gesturing and Voice Input, *Conference on Human Factors in Computing Systems - CHI 89 Conference Proceedings,* Austin, Texas, April-May 1989, 235-240.
60. C. G. Wolf, Can People Use Gesture Commands, *ACM SIGCHI Bulletin,* Oct. 1986.
61. C. Wolf, J. Rhyne and H. Ellozy, The Paper-Like Interface, in *Designing and Using Human-Computer Interfaces and Knowledge Based Systems,* G. Salvendy and M. Smith (ed.), Elsevier, Amsterdam, 1989.
62. D. Zeltzer, S. Pieper and D. Sturman, An Integrated Graphical Simulation Platform, *Proceedings of Graphics Interface'89,* 1989.
63. T. G. Zimmerman and J. Lanier, A Hand Gesture Interface Device, *Proceedings of CHI and Graphics Interface 1987,* 1987, 189-192.