Integrating Eye Movement and Haptic Interaction Data for Comparing Multimedia Interfaces

N. Hari Narayanan

Intelligent & Interactive Systems Laboratory

Computer Science & Software Engineering Department

Auburn University, Auburn, AL 36849, USA

narayan@eng.auburn.edu

Eric C. Crowe

Computer Information Systems Department

DeVry Institute of Technology

Fremont, CA 94555, USA

ecrowe@fre.devry.edu

With the development of novel multimedia interfaces, new approaches are needed to analyze the process of interaction with such interfaces and evaluate them at a fine grain of detail. Eye tracking is a technology that can provide detailed data on the allocation and shifts of users' visual attention across Graphical User Interfaces of interactive systems. However, integrating, analyzing and visualizing multimodal data on user interactions remain difficult tasks. In this paper we report on steps toward developing a suite of software tools to facilitate these tasks. We designed and implemented an Eye Tracking Analysis System which generates a combined gaze and action visualization from eye movement data and interaction logs. This new visualization allows an experimenter to see the visual attention shifts of users interleaved with their actions on each screen of a multi-screen multimedia interface. An experiment on comparing two interfaces – a traditional keyboard and mouse interface and a speech-controlled interface – to an educational multimedia application was carried out to demonstrate the utility of our approach.

Keywords: Human-Computer Interaction Analysis, Multimedia Interfaces, Eye Tracking.

1. Introduction

In order to evaluate the usability and usefulness of multimedia interfaces, one needs tools to collect and analyze richly detailed data pertaining to both the process and outcomes of user interaction. Eye movements of computer users, when combined with data from other input modalities such as spoken commands and haptic actions with the keyboard and the mouse, provide a rich data set for Human-Computer Interaction (HCI) analysis. Eye tracking has already shed light on how users execute specific interface actions such viewing menus [Aaltonen et al. (1998), Byrne et al. (1998)] and eye typing [Salvucci (1999)]. This paper reports on a research project that takes this idea a step further by developing automated software support for analyzing and visualizing data resulting from tracking the eyes and actions of users while they interact with a multimedia system.

A premise of this research is that insights into the usability and effectiveness of an interface can be developed from knowing not only what users do with a Graphical User Interface (GUI), but also where they look. A comprehensive picture of the interaction process with a multimedia application includes not only what navigation and control actions users undertake but also how they distribute their visual attention among multiple visual entities present on various screens (or pages) of the interface. In order to obtain such a picture, we developed a novel data visualization technique called an Attention-Action Timeline. From this visualization, an experimenter can gain a comprehensive picture of an entire interactive session, and thus can discover what actions users performed and where users looked between those actions. The utility of this approach is demonstrated through a pilot experiment in which a speech-controlled interface and a traditional interface to a multimedia educational application are compared.

This paper has three aims. The first is to outline an approach to comparative analysis of multimedia interfaces. The second is to describe software for combining and visualizing users' eye movement and action data that we have designed and implemented as part of this approach. The third is to demonstrate the utility of this approach by discussing results of an interface comparison experiment. The rest of the paper is structured as follows. In Section 2, we present a framework for comparing interfaces. Section 3 describes our approach toward collecting, combining and visualizing interaction logs and eye tracking data. The software system implementing this approach and the visualization it generates are described. This system and its capabilities are compared with related work in Section 4. The following section describes HalVis — the educational multimedia system that was used in the experiment — and explains how its original interface was retrofitted with speech control. Experimental procedure and results are presented next, in Section 6. This experiment demonstrated the utility of combining interaction logs and raw eye movement data to analyze users' interactions with interfaces. Section 7 concludes the paper with a summary of the contributions of this research and its future.

2. A Framework for Interface Comparison

The primary objective of an interface is to help a user effectively carry out various tasks by interacting with an underlying system. For instance, in the domain of educational multimedia, an effective interface is one that is successful in enabling the learner to comprehend complex information on various topics provided by the system. It is irrelevant whether an ineffective interface is usable or not. On the other hand, it does matter if an effective interface has poor usability. Therefore, the first question to ask about an interface is whether it produces the desired outcome. In the educational domain, this translates to whether the interface significantly improves the knowledge of users. This can be empirically tested by measuring the prior knowledge of a representative sample of users in a pre-test, having them work with the interface, and then measuring their knowledge improvement in a post-test.

Then a second question needs to be asked: does the interface provide a significantly better means of producing the desired outcome than other possible interfaces to the same system? This can be empirically tested by a comparative study in which multiple groups of users work with the interface in question as well as other competing interfaces. Data on the desired outcomes (e.g. comprehension), task times and usability measures such as satisfaction can then be compared to see if the interface in question is indeed better.

If the interface is found to be better, a third question can be asked: why is it better? While the previous two questions dealt with *outcomes* of interacting with an interface, the present question addresses the *process* of interaction. As a multimedia interface can contain several entities such as text, audio, diagrams, animations, and video, which of these components contribute to its effectiveness is an issue worth pursuing. If a component is found to be ineffective, removing it can benefit both the interface design (by making it more compact) and the user (by reducing the number of possible actions). One way to answer this question is to design and conduct a series of *ablation studies* in which a complete version of the interface is compared with versions in which one (or a group of similar) components have been elided. Then statistical comparisons of the collected data will throw light on the differential contributions of these components. Unfortunately, this is a time and effort intensive approach.

Another way to address this question is to study the microstructure of interaction by collecting and analyzing logs of user interactions with the interface. Any screen of an interface to a multimedia system will typically contain several components, some of which will be interaction objects like buttons while others will be information carrying entities like text or pictures. So collecting only interaction logs (logs of haptic actions such as mouse clicks and key presses) will provide only an incomplete picture. For instance, suppose when the user clicks on a hyperlink, a page with a bar chart, a photograph and a textual explanation appears. While the interaction log will indicate that the user moved to this new page, it will not show whether he or she attended to the chart, the photo or the text. Thus, there is a need to track the visual attention of users. Eye movement data from an eye tracker, when combined with interaction logs, can provide a complete picture of the interaction process, from which conclusions about the utility of specific components of an interface can be drawn. Eye tracking can also reveal clues about usability [Benel et al. (1991)]. Unexpected or unusual scanning patterns (such as not attending to some components of a screen with many visual objects) and fixation patterns (such as staring at a component for a long time) can indicate potential trouble spots in an interface.

A significant advantage in using eye movements as a source of data is that these are automatic, giving an accurate measure of where one's visual attention is directed. This is information that cannot be accurately obtained by asking a user what he or she is looking at. Interpretation of eye movement data in interaction tasks can be based on the empirically validated *eye-mind assumption* [Just & Carpenter (1976)] that when a person is performing a cognitive task while looking at a visual display, the location of his/her gaze on the display corresponds to the symbol that he or she is currently processing in working memory. The ability of eye tracking to reveal otherwise unobtainable information about visual interaction has lately generated increasing interest in using this technology for evaluation of computer interfaces [Goldberg & Kotval (1998)].

3. Integrating Eye Movement Data and Haptic Interaction Logs

Data collected in typical HCI experiments include process data (e.g. task completion times, verbal protocols, error rates, experimenter observations, user interaction logs), outcome data (such as pre/post-tests to measure change in a user's knowledge or skills) and survey data (e.g. interviews, user satisfaction questionnaires). All these kinds of data, however, miss one crucial aspect of interaction. When there are multiple information carrying entities simultaneously present on a screen, how do users allocate and shift their visual attention among these? This information can be important in understanding both the process and outcomes of interaction. An eye tracker can be used to obtain Points of Regard (POR) data, which indicate the location of visual attention of users in terms of screen X and Y coordinates.

An eye tracking device tracks two types of eye movements: saccades and fixations. Saccades are rapid eye movements that allow the fovea to view different portions of the display. Often a saccade is followed by one or more fixations when objects on a scene are viewed. Just like interaction logs (a time-stamped list of all haptic user actions using the keyboard and the mouse), saccade and fixation data provided by an eye tracker are also too low level and voluminous to be directly useful for interface comparison. This problem compounds when these two data streams are combined. To address this problem, we designed and implemented an Eye Tracking Analysis System (ETAS) which generates high level gaze and action visualizations from eye movement data and interaction logs. The eye movement data is collected by an eye tracker, and the interaction logs are automatically recorded by instrumenting the interface. The original contribution of ETAS is in integrating information on users' haptic and visual actions, reducing the resulting voluminous data by aggregation and abstraction, and generating a screen-by-screen visualization of what users watch and do with an interface. This visualization of the attention and actions of a user is much more useful to an designer than the raw data itself. Besides generating this visualization, ETAS also computes and provides quantitative information such as the amount and percentage of time a user gazed at each object on each screen, and the frequencies and percentages of a user's gaze shifts between any pair of objects on any screen. The computations are done by a module called GRIP and the visualizations are generated by a module called AAT (see Figure 1).

GRIP (implemented in C++) allows one to a priori define regions of interest on different pages or screens of an interface. These are defined as bounding boxes enclosing visual entities such as pictures, buttons, etc. From the eye movement data GRIP then computes a user's gaze patterns across all such visual entities. It can be used as a stand-alone program to aggregate POR and fixation data in several ways. The full capabilities of GRIP are described elsewhere [Narayanan & Schrimpsher (2000)]. Here we discuss only those analysis steps that ETAS uses for creating its visualization.

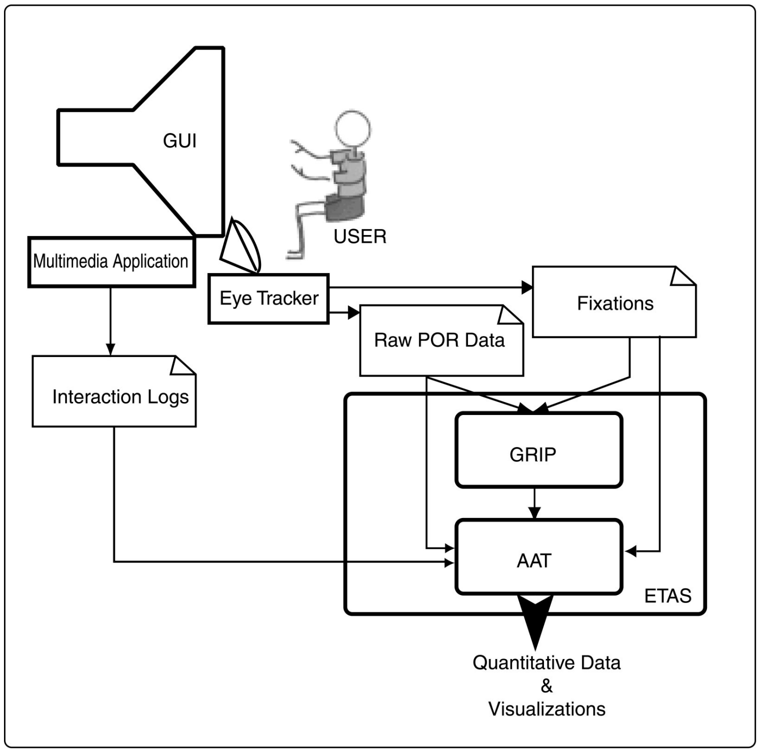


Figure 1. ETAS Architecture

GRIP incorporates two advances in eye movement data analysis. The first advance is that it extends POR data analysis into the realm of dynamic multi-screen and multimedia interfaces. The second advance is that it aggregates and reduces the voluminous raw data produced by an eye tracker. GRIP processes eye movement data in five steps: (1) process POR data to compute occurrence times; (2) compute fixations and associate them with visible components of the interface; (3) aggregate fixations into gazes; (4) compute gaze shifts; and (5) aggregate gaze shifts. First, it partitions POR data into sections corresponding to each screen or page of the interface and computes the occurrence time of each POR. This produces a segmented and time-stamped chronological ordering of the POR data along with a list of interface components visible in each segment. The second step is reducing this data by aggregating POR into fixations. GRIP uses the eye tracker's software to compute fixations from POR data. Each fixation is associated with a visible interface component. Third, consecutive fixations on the same component are aggregated into gazes and gaze durations are computed. Thus a chronological list of the user's gazes on the various visual components of the interface is produced. The next step is to generate a chronological list of gaze shifts among different components. In the fifth step GRIP computes, for each component of the interface, all other components to which the user's gaze shifted from that component.

The second module, called AAT, is implemented in Visual C++. It takes as input the output of GRIP, the interaction logs recorded by the interface, and the POR and fixation data computed by the eye tracker. These data sets are combined to produce a screen-by- screen visualization of how each user navigated through the system, which visual entities were attended to on each screen, and what actions were taken. Figure 2 provides a sample of this visualization, which we call the Attention-Action Timeline. In this figure, each row represents a region of interest that the designer defined. Typically, each region of interest will encompass an interaction or presentation component of the interface. The rows are labeled with names of the corresponding components, provided by the designer at the time regions of interest are defined. "No Entity" and "Off Screen" are special labels used to signify a user looking at a blank area of the screen and a loss of track, respectively. The thin horizontal line

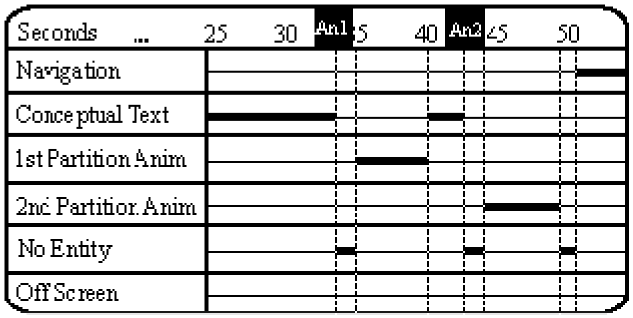


Figure 2 Attention-Action Timeline

to the right of each label denotes a timeline. A thickening of a timeline indicates the duration for which a user's gaze stayed on the corresponding object. The clock starts at 0 when the user begins working with the multimedia system, and keeps track of the seconds elapsed. When the user's visual attention shifts to another object, its timeline becomes thicker. Therefore, a left-to-right scan of this visualization provides the designer with a sense of both how long a component was attended to and how the user's visual attention shifted back and forth among the many visual components simultaneously present on the screen. A visualization is produced for each screen of the interface being analyzed.

User's actions are overlaid on this visualization. Action information boxes appear in the topmost row above the timelines, indicating when a user performed actions. Each action appears as a labeled box that can be clicked on to obtain more information about it. Information about what the actions were and when they were executed is extracted automatically from the interaction log of each user. For example, in Figure 2, "An1" and "An2" represent actions made just before looking at the first and second animations respectively. The action boxes and times at which each gaze shift from one visual component to another occurred are indicated on the top row of the visualization, above vertical dashed lines.

ETAS is designed to be used with interactive multimedia systems built with any application and running on any computer. There are only three requirements. The first is that the designer must create a component description file specifying labels and spatial extent (in terms of the screen coordinates of the bounding box) of every visual component of the system's interface. Software that comes with many eye trackers allows this to be done interactively. Second, the computer running the interactive multimedia system should send a code, corresponding to each component when it becomes visible/active, to the eye tracking computer through a serial cable. This code is simply an integer corresponding to the position of the component in the component description file (e.g. the code of the first component is 1). The eye tracking computer automatically adds this code to the raw POR data it generates. Third, the designer must instrument the interface so that each time a user executes a haptic action (e.g. press a button, select a menu item) a record of it is created with an action label and a time stamp in an ASCII interaction log file. The latter two instrumentation steps are easy to accomplish in most multimedia authoring platforms (such as the Macromedia Director^{TM}) by adding the corresponding instructions to appropriate event handlers associated with interaction objects of the interface. These requirements do however impose a front-end workload on the designer before ETAS can be put to use.

Though the rest of this paper describes the use of ETAS in an experiment comparing keyboard/mouse and speech control of a particular GUI, it should be noted that this tool has wider applicability. The screen-by-screen eye movement and action visualization that ETAS can generate from raw eye tracking data and interaction logs is useful for different analyses. For example, one can compare how users allocate their visual attention to a single object that appears in multiple screens (such as a banner advertisement that appears on all pages of a commercial web site) across different interface designs. The overlaying of user actions on this visualization lets the designer identify gaze patterns that precede and follow particular actions, facilitating an analysis of cause and effect relationships.

4. Related Work on Eye Movement Data Analysis

The use of eye tracking for HCI research has a long, but sporadic, history. Eye tracking has been used both to synthesize and to analyze interactions. In the synthesis mode, the eye tracker functions as an input device, tracking where the user is looking at and using this information in real-time to generate appropriate system responses. An example is the "eye typing" task, in which a user can look at the image of a keyboard on the monitor and select letters to be typed merely by looking at them [Salvucci (1999)]. In the analysis mode, eye movement data collected from users while they interact with an interface is used to analyze the process of interaction. Here we discuss a selected sample of recent work related to analysis as it is the focus of research reported in this paper.

One HCI task that has been analyzed in depth using eye movement data is the visual search of menu items. In one study [Aaltonen et al. (1998)] eye movement data showed that a visual search behavior called "sweeps" could possibly account for the systematic and random search strategies previously reported for menu search. Another study employed eye tracking to assess the validity of two cognitive models of menu search [Byrne at al. (1998)]. Both used data analysis programs tailor-made for the menu search task, which calculated fixations, scan paths, sweeps and selection times from mouse and eye movement data. It is not obvious whether these analysis programs will work for general interactions with a GUI.

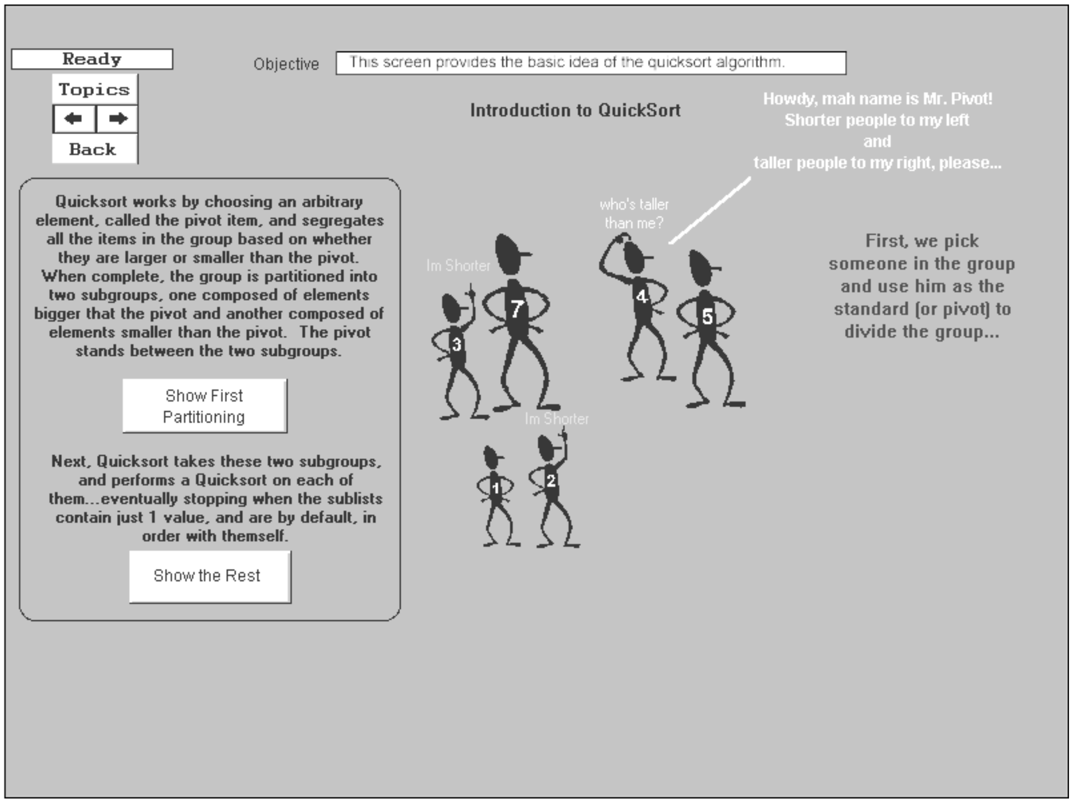
Salvucci describes the use of Hidden Markov Models (HMM) to convert raw eye movement data into a sequence of fixations and to map the fixations onto predictions of an HMM-based process model [Salvucci (1999)]. This results in an interpretation of the eye movements in the context of a specific task like eye typing. While this provides a powerful method for predicting and interpreting eye movements, it is limited to well-defined tasks for which one can a priori model the expected eye movements. General interactions with a GUI are too open-ended to be amenable to such modeling.

One of the first studies to track eye movements of subjects viewing a multimedia information presentation [Faraday & Sutcliffe (1997)] involved an experiment with six subjects viewing an 18-second presentation from a CD-ROM. Based on this experiment, the authors developed several design guidelines for multimedia. For analysis purposes, raw eye movement data was converted to scan paths consisting of saccades and fixations over 5-second intervals, nearby fixations were clustered into an ordered sequence, and overlaid on a frame of the presentation during each interval to produce a visualization of the data. As data analysis was not the focus of this paper, it is not clear how much of this analysis was done manually. ETAS not only automates this process, but also takes data analysis to a higher level of aggregation and visualization. This allows the analysis of complex multimedia presentations over longer periods of time (for example, a session with HalVis typically takes about 45 minutes).

In summary, the long term goal of our research is the development of a general aggregation, analysis and visualization tool that will integrate and then distill haptic (keyboard/mouse) and visual (eye movement) data to provide designers with useful descriptions and depictions of users' interactions with interactive multimedia systems. This is in contrast to the data analysis techniques described in most current literature on the use of eye tracking in HCI, where either the focus is not on analysis, or the analyses are highly task-specific and/or done manually.

5. Two Interfaces of an Educational Multimedia Application

In order to test the utility of ETAS in allowing an interface designer to experimentally compare different interfaces we chose HalVis (Hypermedia Algorithm Visualization), an interactive multimedia system that provides animated visualizations of several fundamental


Ready
Topics
Back
Objective
Introduction to QuickSort
Howdy, mah name is Mr. Pivot! Shorter people to my left and taller people to my right, please...
who's taller than me?
Im Shorter
Im Shorter
Show First Partitioning
Show the Rest

This screen provides the basic idea of the quicksort algorithm.

Quicksort works by choosing an arbitrary element, called the pivot item, and segregates all the items in the group based on whether they are larger or smaller than the pivot. When complete, the group is partitioned into two subgroups, one composed of elements bigger that the pivot and another composed of elements smaller than the pivot. The pivot stands between the two subgroups.

Next, Quicksort takes these two subgroups, and performs a Quicksort on each of them...eventually stopping when the sublists contain just 1 value, and are by default, in order with themself.

First, we pick someone in the group and use him as the standard (or pivot) to divide the group...

Figure 3. Conceptual View

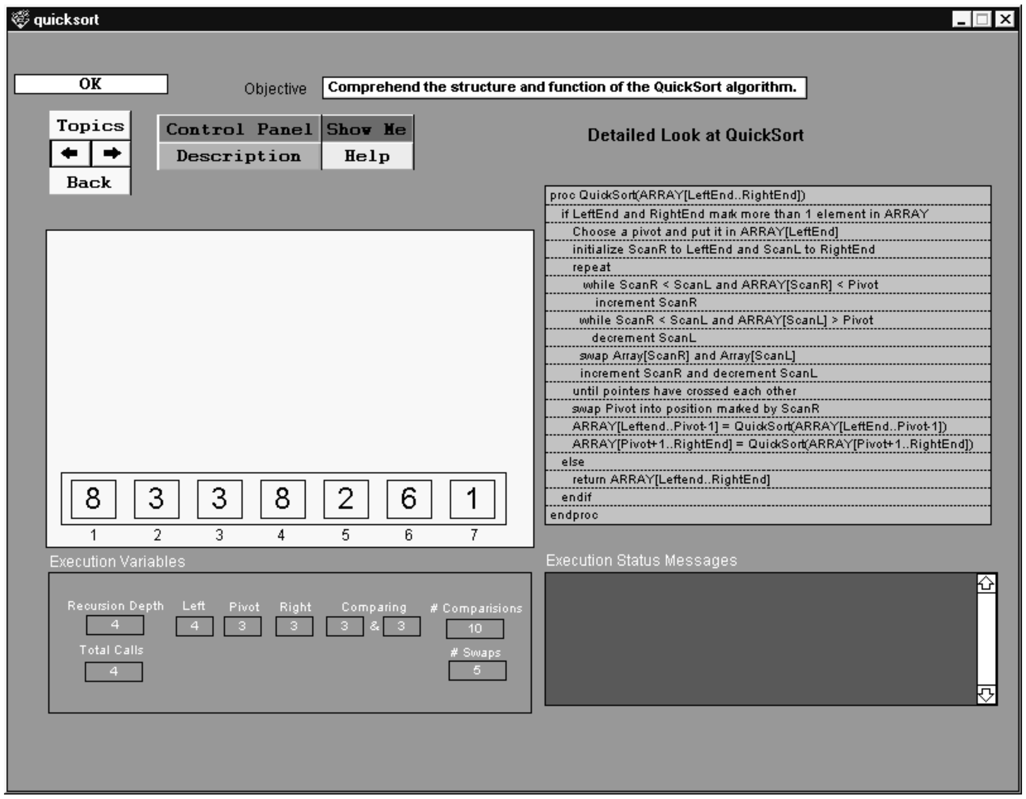
computer algorithms, as a test-bed. HalVis was designed as a self-learning aid for undergraduate computer science students, and was proven to be effective in several empirical studies [Hansen et al. (2000)]. Thus, we started with an effective educational multimedia system and its traditional keyboard and mouse (haptic) GUI, then created a new speech-controlled version of its interface.

The GUI of HalVis provides three main views of an algorithm and a question section (Figures 3-6 show grayscale pictures of these views that are actually implemented in color). The first view, called the Conceptual View, consists of a screen that shows a familiar analogy that the student can interact with and animate, with accompanying text (see Figure 3). The quicksort analogy shows a line of people partitioning and arranging themselves in the order of increasing height, based on the height of one member called the "pivot". This process is illustrated using a series of moves shown with smooth animations and textual explanations, which capture the essence of the quicksort algorithm in terms of its fundamental operations.

The second view, called the Detailed View, presents a visualization of the algorithm in four different panes simultaneously. One pane shows a smooth graphical animation of the steps of the algorithm (Figure 4, left side), the second shows pseudocode (a step-wise description) in which the steps being animated are concurrently highlighted (Figure 4, right side), the third provides values of important variables used by the algorithm (Figure 4, lower left), and explanatory messages about the events occurring in the animation appear in the fourth pane (Figure 4, lower right). The algorithm animation in this view provides a micro-level (fine-grained) picture of the algorithm's operation on a small data set.

The third view, called the Populated View, presents a macro-level animated view of the algorithm's behavior on a large data set (see Figure 5). The questions section contains a series of questions about the algorithm, which the user may answer and receive feedback. There are two types of questions: one that requires the student to reorder steps of the algorithm correctly (see Figure 6), and another with multiple choice questions. All of these views can be visited in any order and repeated any number of times. HalVis was originally developed in Asymmetrix Toolbook. The quicksort visualization used in our experiment was implemented with Macromedia Director.

All interface screens of HalVis have been designed for traditional haptic control with a keyboard and a pointing device such as a mouse. Most commands are entered by mouse clicks on buttons or menu items, and most data entry (such as changing the data input to an animation) is done through the keyboard.



Objective: Comprehend the structure and function of the QuickSort algorithm.

Detailed Look at QuickSort

```
proc QuickSort(ARRAY[LeftEnd..RightEnd])
  if LeftEnd and RightEnd mark more than 1 element in ARRAY
    Choose a pivot and put it in ARRAY[LeftEnd]
    initialize ScanR to LeftEnd and ScanL to RightEnd
    repeat
      while ScanR < ScanL and ARRAY[ScanR] < Pivot
        increment ScanR
      while ScanR < ScanL and ARRAY[ScanL] > Pivot
        decrement ScanL
      swap Array[ScanR] and Array[ScanL]
      increment ScanR and decrement ScanL
    until pointers have crossed each other
    swap Pivot into position marked by ScanR
    ARRAY[Leftend..Pivot-1] = QuickSort(ARRAY[LeftEnd..Pivot-1])
    ARRAY[Pivot+1..RightEnd] = QuickSort(ARRAY[Pivot+1..RightEnd])
  else
    return ARRAY[Leftend..RightEnd]
  endif
endproc
```

8	3	3	8	2	6	1
1	2	3	4	5	6	7

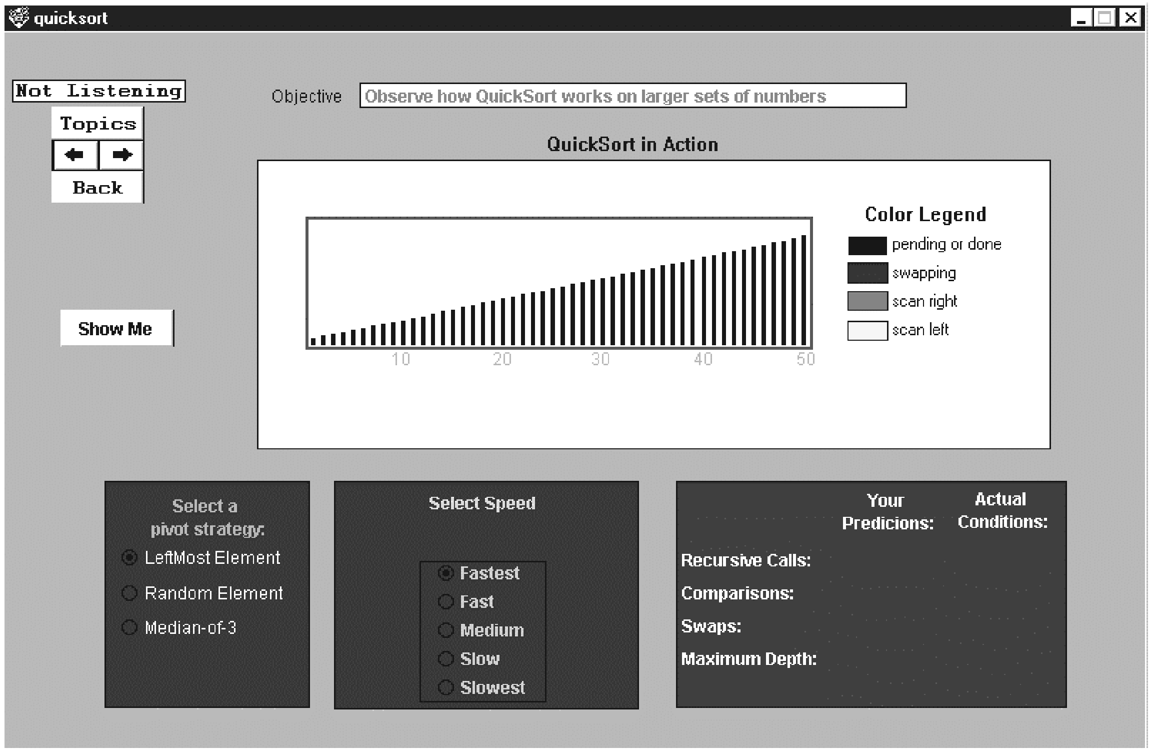
Recursion Depth: 4 | Left: 4 | Pivot: 3 | Right: 3 | Comparing: 3 & 3 | # Comparisions: 10
Total Calls: 4 | # Swaps: 5

Figure 4. Detailed View

A second version of the quicksort visualization was built with the original visual interface retro-fitted with a speech recognition engine. The speech controlled interface was made as close to its keyboard-mouse counterpart as possible, designing in such a way that a student can interact with and navigate through the quicksort visualization using *only* spoken commands. In this speech-controlled version, every button, menu, or keyboard/mouse command in the original interface can be executed through speech. There are three types of speech commands: navigation, control and data entry. The navigation commands include names of the topics one can visit (label of any menu item from the hierarchical pull-down menu "Topics") and commands "previous", "next", and "back". Also, any hyperlinked words and phrases appearing in any screen of the interface can be spoken to activate the links (instead of clicking with the mouse). Control commands include labels of all control buttons (e.g., the "Show Me" button to start an animation in Figure 4) and dialog boxes that start sequences of actions (e.g., the dialog box that asks for student predictions of algorithm performance in the Populated View, which will then be displayed—see Figure 5, lower right). One can interact with dialog boxes by speaking the commands on the dialog box buttons (such as "ok" or "cancel"). Data entry dialog boxes enable users to input numbers by speaking the numbers.

We used a speech recognition engine from Microsoft. The speech-enabled interface includes a speech recognition feedback mechanism shown as a white box above the navigation commands (feedback examples can be seen in Figures 3-6 inside white boxes above the "Topics" menu; these figures show the speech-enabled GUI of HalVis). Whenever a user speaks a command and the recognition engine understands it, that phrase would be placed in the white feedback box. If the speech engine does not recognize a word then "???" will be displayed followed after a short delay by "Please Say Again". Buttons also continue to perform a push down behavior when their respective labels are spoken, in order to provide an additional feedback mechanism confirming correct interpretation of the command.

A third difference between the original and speech-enabled interfaces is that in the latter button and menu labels have been made shorter. Since a user has to vocalize the labels



Objective: Observe how QuickSort works on larger sets of numbers

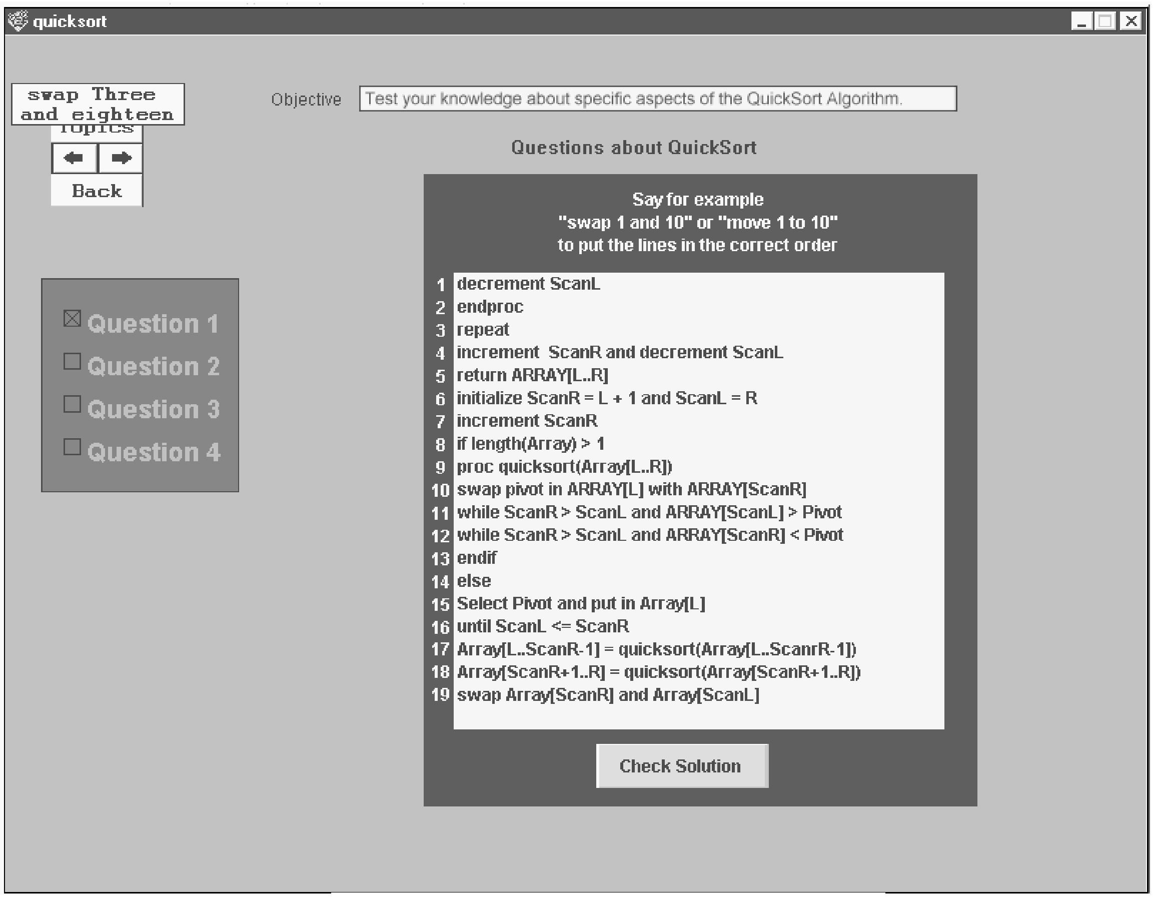
Select a pivot strategy: [✓ LeftMost Element] Random Element Median-of-3

Select Speed: [✓ Fastest] Fast Medium Slow Slowest

Figure 5. Populated View

to actuate commands, these were made as brief as possible to reduce the number of disfluencies. Finally, we provided a "What can I say?" command that is available at all times, and brings up a window with all spoken commands available in the current context.

In the question section, the first question underwent significant redesign to speech-enable it. This is the pseudocode reordering question in which the steps of the algorithm are presented to the student in random order to be reordered correctly. With a traditional haptic interface, students would answer by clicking and dragging lines where they need to go. These are actions that can't be performed only with speech. Though one could still use a keyboard and a mouse with HalVis's speech-controlled interface, we wanted to compare haptic control with speech-only control. Therefore, a way to use speech control with this question had to be devised. We solved the problem by placing a line number next to each line of pseudocode (see Figure 6). Users could then say "Swap 1 and 10" or "Move 1 to 10" to place the lines in the correct order. Both swap and move commands reordered the pseudocode differently. With



Objective Test your knowledge about specific aspects of the QuickSort Algorithm.

Questions about QuickSort

Say for example "swap 1 and 10" or "move 1 to 10" to put the lines in the correct order

```
1  decrement ScanL
2  endproc
3  repeat
4  increment  ScanR and decrement ScanL
5  return ARRAY[L..R]
6  initialize ScanR = L + 1 and ScanL = R
7  increment ScanR
8  if length(Array) > 1
9  proc quicksort(Array[L..R])
10 swap pivot in ARRAY[L] with ARRAY[ScanR]
11 while ScanR > ScanL and ARRAY[ScanL] > Pivot
12 while ScanR > ScanL and ARRAY[ScanR] < Pivot
13 endif
14 else
15 Select Pivot and put in Array[L]
16 until ScanL <= ScanR
17 Array[L..ScanR-1] = quicksort(Array[L..ScanrR-1])
18 Array[ScanR+1..R] = quicksort(Array[ScanR+1..R])
19 swap Array[ScanR] and Array[ScanL]
```

Figure 6. Pseudocode Reordering Question

the swap command, the lines of pseudocode are simply interchanged, whereas the move command moves one line to a stated position and all subsequent lines propagate up or down depending on the direction of the move. With the multiple choice questions, the users could simply state the letter or number corresponding to their choice of answer.

6. Comparing the Two Interfaces

In order to test the utility of ETAS in allowing an interface designer to experimentally compare different interfaces, the process of how two groups of students viewed, interacted with and navigated the haptic and speech-controlled interfaces to HalVis was experimentally analyzed. This experiment had two goals. The primary goal was confirming the utility of our approach, implemented as ETAS, to combining, analyzing and visualizing interaction logs and eye movement data. A secondary goal was generating design heuristics for speech-enabling interfaces by comparing how users interacted with the same multimedia system using a speech-controlled or a traditional haptic interface. The following subsections describe the experimental procedure, data collection and results.

6.1 Experimental Procedure

Thirty-four students from an upper level undergraduate course on data structures and algorithms at Auburn University volunteered to participate in the experiment. Participants were divided into two matched groups, the keyboard-mouse (KM) group and the speech-control (SC) group. This matching was done by ranking students according to their GPA, their grade in a previous computer science course, and their ACT or SAT test scores. For each pair of students in this ranking, one was assigned to the KM group and the other to the SC group randomly. Since students had their eyes tracked, they were tested individually.

The experiment room was setup with HalVis running on a 200MHz Pentium computer with 128MB RAM, and a 21" monitor raised slightly above eye level and attached to the back wall. We used an ISCAN ETL-400 eye tracker that generated POR coordinates at 60 Hertz. The system uses an infrared light source that shines a beam on one eye and an infrared camera that processes the resulting corneal and pupil reflections to compute POR coordinates. The source and the camera were mounted on a pan/tilt assembly and placed at a slightly upward angle on a table below the computer monitor that subjects looked at. The eye tracker required a brief calibration step with each user in order for it to be able to compute the POR data and fixations in screen coordinates. We defined a fixation as the user looking within a 20x20-pixel square for at least 100ms.

Subjects sat on a chair with a high headrest, with an adjustable table in front to place a keyboard and mouse (for the KM group only). Located to the right of the subject were the experimenter's chair and controls for the eye tracking system. These controls included two monitors. One displayed the subject's eye as seen by the camera (the eye monitor), and the other showed the same display as the subject's monitor with a pair of cross hairs indicating the subject's POR superimposed on it (the scene monitor). The eye tracker can lose track of the subject's eye if the subject closes his/her eye for some time or turns his/her head rapidly. When this happened, the experimenter detected it on the eye monitor and controlled the camera through the eye tracker's computer to recapture the eye.

Each participant took a pre-test designed to measure his/her prior knowledge on the quicksort algorithm. A brief training session to acquaint the participant with the algorithm visualization interface, and for adjusting and calibrating the eye tracking equipment, followed. The participant was then instructed to work through the visualization well enough to be able to answer a post-test similar to the pre-test. The KM group interacted with the system using a keyboard and mouse. The SC group was not provided with a keyboard or a mouse. They had to interact with the system using *only* spoken commands. When a participant finished working with HalVis, a post-test was given. An entire session lasted from one to two hours. All students in the SC group were unfamiliar with speech recognition and had never before used speech to interact with a computer.

6.2 Data Collection

We collected four different kinds of data in this experiment. The first is quantitative data on student learning, which was collected by having all participants take a pre-test before the experiment and a post-test afterwards. These tests examined the extent of conceptual and procedural knowledge of students on the quicksort algorithm. Students participated in the experiment one-by-one in the presence of the experimenter. His written observations on interesting events that happened constitute the second kind of data. The HalVis visualization was instrumented so that every command that each participant executed, via speech or keyboard/mouse, was time stamped and logged automatically. These logs, the third kind of data, provided detailed information on the interaction process. The fourth kind was eye movement data collected from each participant. It consisted of Points of Regard (POR), or the location of visual attention of users, as screen X and Y coordinates.

6.3 Qualitative Results

In this subsection we discuss qualitative results derived from the experimenter's observations and from the Attention-Action Timeline (AAT) visualizations generated by ETAS. With each result, we suggest a corresponding heuristic regarding the design of speech-enabled graphical user interfaces.

In analyzing navigation patterns using AAT, we discovered that the KM group repeatedly re-visited views much more than the SC group. Observations made by the experimenter revealed an explanation for this. The SC group had difficulty going back to previously visited views. The commands to go back and forward are represented in the visual interface as buttons with arrow icons. The corresponding spoken commands are “previous” and “next”. Unless a subject remembered from the training session which words to speak to activate these buttons, it would not be obvious to him or her that other similar sounding words (e.g., “backward”, “forward”) will not work. Members of the SC group had difficulty remembering the “previous” command. The “next” command did not turn out to be as problematic, possibly because subjects had to use it more in the training session. The corresponding design heuristic is that providing textual labels indicating spoken commands on interaction objects may be more beneficial (by improving learnability) in a speech-controlled interface than doing so in a traditional interface.

The AAT also showed that several students in the SC group skipped some pages of the algorithm visualization. Observations made by the experimenter indicated that this occurred mostly because the speech recognition engine mistakenly concluded that a user spoke the name of a page that is two or three screens ahead and displayed that page. If the user did not look at the speech recognition feedback box and realize this, or otherwise catch this error, he/she would not only miss important information but also be unaware that it happened. This matches the finding in a text typing task study that, in the speech-input mode, subjects were not immediately aware of misrecognitions [Karat et al. (1999)]. This illustrates a potential weakness of a speech-controlled interface when compared to a traditional one: an increased likelihood of misrecognitions matching unintended system commands. A corresponding heuristic is that it may be better not to have active spoken commands that the user is not visually aware of. An alternate way to alleviate this problem is to provide an indication of the user’s location within the system on each screen that would make any skipped sections more obvious. For a multimedia application that is sequentially structured as pages, this may be as simple as displaying the current page as “page n of m” where m is the total number of pages. If the application is non-linear (e.g. hypermedia), a map that explicates both “you are here” and “you were there” information is preferable. While these are useful heuristics for any navigational interface, we believe they become even more important when voice-based navigation is included.

Dialog boxes are not very natural because it is not always obvious to users that they must respond to the dialog box before any other interaction can be initiated. A speech-controlled interface appears to make this problem worse. AAT visualizations revealed that participants in the SC group often issued spoken commands meant for other interaction objects while a dialog box was visible. They did not realize that they had to interact with the dialog box first with speech to dismiss it, saying “ok” or “cancel”, even though the corresponding dialog box buttons were visible and they knew that buttons could be activated by speaking the button labels. Such pre-emptive dialog boxes are generally considered to be examples of poor interface design. The difficulty these cause to users appears to have been exacerbated in the speech-controlled interface. A corresponding heuristic is to avoid dialog boxes in a speech-controlled interface. If dialog boxes that respond to spoken input have to be included, the production of spoken output by the dialog boxes, drawing the user’s attention to what he or she needs to say to respond, can help alleviate this problem.

6.4 Quantitative Results

Table 1 Test Scores

	Pre-test scores	Post-test scores	Score improvement	Statistical test
KM group	21.531 (14.08)	44.094 (14.136)	22.562 (11.531)	Paired t-test; $p<0.0001$
SC group	25.312 (15.323)	42.625 (14.236)	17.312 (10.157)	Paired t-test; $p<0.0001$

Table 1 summarizes test scores of the two groups. The mean scores (with standard deviations in parentheses) out of a maximum of 80 points are shown. Each group exhibited a statistically significant increase in learning after interacting with HalVis (see paired t-test results in Table 1). The mean improvement score (difference between post- and pre-test scores) of the SC group was less than that of the KM group. However, a statistical comparison of the improvement scores of the two groups revealed only a marginal significance (unpaired t-test, p = 0.1819). So, despite the observed limitations of the speech-controlled interface and the somewhat lower absolute scores of the SC group, it is safe to assume that the speech-controlled interface did not significantly affect the system's effectiveness in comparison with the traditional interface.

A previous study comparing keyboard/mouse and speech interfaces showed that the subjects using speech spent twice as long with the application [Buskirk & LaLomia (1995)]. We did not find this to be the case. ETAS showed that the mean time spent with HalVis was 26.763 minutes for the KM group (standard deviation 12.299) and 28.687 minutes for the SC group (standard deviation 15.186). However, the percentage of time participants spent viewing the navigation control panel (upper left part of Figures 3-6) increased significantly (unpaired t-test, p = 0.0077) for the SC group. This panel contains the speech recognition feedback box for the SC group. This indicated that subjects did pay attention to visual feedback of spoken command recognition. The corresponding heuristic is that error detection in a speech-controlled interface may benefit from providing explicit visual feedback of speech recognition results. As users seem to pay attention when such feedback is provided, it has the potential to improve error detection and recovery. Such feedback is not usually provided in a traditional interface.

7. Conclusion

This paper described a novel approach to combining and visualizing eye movement and haptic interaction data in the service of experimentally evaluating and comparing interfaces to multimedia systems. The tool that was developed, ETAS, combines users' visual attention data with action data to create an integrated visualization of an entire interactive session. This visualization makes explicit what actions users took and what visual stimuli were of interest to them between those actions. As ETAS can combine user actions executed by speech or touch with user gazes, it is a generally useful tool for visualizing and analyzing multimodal interactions. The technology for combining and analyzing eye tracking data in conjunction with other interaction data holds considerable promise as a powerful tool for analyzing advanced visual interfaces.

Only recently have compilations of research on how to create usable applications with visual interfaces controlled by speech started to appear [Gardner-Bonneau (1999)]. This research has not kept pace with the increasing commercialization of speech technology. In this context, the design heuristics derived from using ETAS to compare a traditional keyboard/mouse-controlled graphical user interface with its speech-controlled counterpart will be useful to interface designers interested in speech-enabling existing graphical user interfaces. These heuristics may also be viewed as starting points for further investigating the design of voice interaction integrated with haptic input devices such as the pen, keyboard and mouse. This is an avenue for future research. Another avenue that we expect to pursue in future is to investigate using the eye tracker as an input device (e.g. see [Sibert & Jacob (2000)]) together with speech. We also plan to further develop the technology for combining, analyzing and visualizing eye tracking data and interaction logs.

Acknowledgments: This research was supported by contract N00014-96-11187 and an equipment grant from the Office of Naval Research. The development of HalVis was supported by grants CDA-9616513 and REC-9815016 from the National Science Foundation. Thanks go to Dan Schrimpsher for GRIP and helping to conduct the experiment.

8. References

Aaltonen, A., Hyrskykari, A., and Räihä, K-J. (1998). 101 Spots, or How Do Users Read Menus? in *Proceedings of Human Factors in Computing Systems Conference (CHI'98).* ACM Press, New York, 132-139.

Benel, D. C. R., Ottens, D., and Horst, R. (1991). Use of an Eyetracking System in the Usability Laboratory, in *Proceedings of the Human Factors and Ergonomics Society 35th Annual Meeting*. HFES, 461-465.

Buskirk, R. V. and LaLomia, M. (1995). A Comparison of Speech and Mouse/Keyboard GUI Navigation, in *Proceedings of Human Factors in Computing Systems Conference (CHI'95).* ACM Press, New York, 96.

Byrne, M. D., Anderson, J. R., Douglas, S., and Matessa, M. (1998). Eye Tracking the Visual Search of Click-Down Menus, in *Proceedings of Human Factors in Computing Systems Conference (CHI'98).* ACM Press, New York, 402-409.

Faraday, P. and Sutcliffe, A. (1997). Designing Effective Multimedia Presentations, in *Proceedings of Human Factors in Computing Systems Conference (CHI'97).* ACM Press, New York, 272-278.

Gardner-Bonneau, D. (ed.). (1999). *Human Factors and Voice Interactive Systems.* Kluwer Academic Publishers, Dordrecht, Netherlands.

Goldberg, J. H. and Kotval, X. P. (1998). Eye Movement-Based Evaluation of the Computer Interface, in *Advances in Occupational Ergonomics and Safety*, (S. K. Kumar, ed.). IOS Press, Amsterdam, 529-532.

Hansen, S. R., Narayanan, N. H., and Schrimpsher, D. (2000). Helping Learners Visualize and Comprehend Algorithms. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning* vol. 2(1), available at http://imej.wfu.edu.

Just, M. A. and Carpenter, P. A. (1976). Eye Fixations and Cognitive Processes. *Cognitive Psychology* vol. 8, 441-480.

Karat, C-M., Halverson, C., Horn, D., and Karat, J. (1999). Patterns of Entry and Correction in Large Vocabulary Continuous Speech Recognition Systems, in *Proceedings of Human Factors in Computing Systems Conference (CHI'99).* ACM Press, New York, 568-575.

Narayanan, N. H. and Schrimpsher, D. (2000). Extending Eye Tracking to Analyze Interactions with Multimedia Information Presentations, in *People and Computers XIV – Usability or Else!*, (S. McDonald, Y. Waern, and G. Cockton, eds.). Springer-Verlag, Berlin, 271-286.

Salvucci, D. D. (1999). Inferring Intent in Eye-Based Interfaces: Tracing Eye Movements with Process Models, in *Proceedings of Human Factors in Computing Systems Conference (CHI'99).* ACM Press, New York, 254-261.

Sibert, L. E. and Jacob, R. J. K. (2000). Evaluation of Eye Gaze Interaction, in *Proceedings of Human Factors in Computing Systems Conference (CHI'00).* ACM Press, New York, 282-288.