STUDIA PHONOLOGICA XIX (1985)

The Human Interface of a Speech Work-Station

Alain de CHEVEIGNE, Minobu ABE and Shuji DOSHITA

Introduction

This paper describes an interactive speech input terminal, designed to make speech acquisition easier and more pleasant.

Such a tool is not NECESSARY, might one say—but then, interactive character terminals were not necessary when they first appeared, nor were word processors...

A speech editor is like a text editor. The speech waveform is seen AS IT IS INPUT. Then it can be manipulated, cut, labeled, and played back. There are many ways to make a text editor: some are simple, efficient and friendly, and others are complex, heavy and hard to learn. The same applies to speech editors: careful engineering of the user-machine interface can lead to a tool that is a pleasure to use.

The heart of the design lies in one idea: TWO modes are involved in speech scquisition, each with its own time scale. The first, that of natural speech, is fast and erratic, and closely related to a SUBJECTIVE time scale. The second time scale is that of the manipulation of the data.

Any design must respect both, and a key problem is to find a way of switching from one mode to the other.

A. Defining the Need

1 Is anything wrong?

The typical researcher in speech lives in a room full of computers so noisy he can scarsely hear himself think. Working all day with data files, poles and lpc coefficients, he may come to view these abstract objects as speech, and the sound itself, the gesture, their place in human life, as epiphenomena. Speech input then is a chore, to be done quickly and efficiently before "real" work can begin.

Within this context the important issues are mainly technical: memory size,

Alain de CHEVEIGNE: JSPS invited scholar, Department of Information Science, Faculty of Engineering, Kyoto University (also: LIMSI, Universite de Paris-Sud-Orsay, 91405 Orsay France).

Minobu ABE (安部美乃夫): Master course student, Department of Information Science, Faculty of Engineering, Kyoto University.

Shuji DOSHITA (堂下修司): Professor, Department of Information Science, Faculty of Engineering, Kyoto University.

direct memory access, disc access, real-time processing and so on. Compared to these difficult problems, "human interfaces" may appear an unnecessary luxury: nice if we had them, but not really worth the trouble.

However, one might argue that this outlook is itself the product of the available technology. New techniques could bring a new outlook. For example, before high level languages, interactive computer terminals, or text editors appeared, not one user in a hundred felt the need. So may it be with speech input.

2) The luxury

Now let's try another angle. Imagine a quiet room (maybe your sound booth, or your office, or perhaps the garden?) with a terminal, a microphone, a teapot, cups, and a few seats for guests. The terminal gives you full control over the whirring clanking beasts locked up in the computer room. While the guests are chatting, you watch the speech as it flows in. You see how much of your (limited) memory is occupied, and where each bit of speech is stored. You can blow up any portion to inspect it, or splice pieces together and play them forewards or backwards. You can label them and put them in your data bank, call out your processing programs, make sonagrams, extract pitch, match templates. The printer in the next room makes hard copies for you to autograph and give to your guests to take home.

3) Designing a speech input interface.

The processing capabilities of such a speech owrk-station are important, as is the interactive interface by which processing is initiated and the results displayed. However, before speech can be processed, it must be input, and this constitutes an interface of a quite different kind. Addressing this problem is the object of the work described here.

a) Two time scales

Speech is a part of human activity, and proceeds at its own rhythm. Saying to a speaker: "Please speak NOW" or "Sorry, the level was wrong, please repeat...NOW" forces an unnatural and unpleasant situation, and may lead to poor quality speech data. Speech input must therefore be continuous. This is the first time scale.

However memory space is limited. Even if it were infinite, there is not much point in gathering infinite quantities of data. A solution is to input continuously, writing over old data, thus keeping a constant length record of past speech.

Data input in this way is volatile. There must be some way of "freezing" it at times, so that it can be manipulated. This is the second time scale, that of data manipulation.

These two modes (called FLOW and FREEZE from now on) are exclusive: you cannot both have your cake and eat it. You cannot input new data and keep the old, at least not with the same memory. The best that can be done is to facilitate the transition between the two.

b) Visulaizing speech data.

Switching between FLOW and FREEZE, one can easily lose track of what memory actually countains. A solution is to make the memory content visible at all times on the screen.

It is important that the visual mapping be the same in both modes. In FREEZE mode it is relatively easy to define a mapping of data to screen.

In FLOW mode, mapping is much more difficult. Data conversion, storing, processing (for mapping) and display must be all done in real time. Also, if we wish the mapping to be continuous in time and homogenous with FREEZE mode, the displayed waveform must "float" accross the screen. This requires special control over the bitmap display.

e) Simplicity of manipulation.

The philosophy was: "Make it simple". When "simple to use" conflicted with "simple to make", we chose the former. We strived to make the options few and the commands intuitive.

B. THE EDITOR

1) Architecture

The editor is divided into two parts: the host and the terminal. The first part stays in the computer room, with it's analog-digital converter (ADC), memory, disks, computing power and appliaition programs. The second part is the one that you carry into the garden, with the microphone.

We did not wish our design to be constrained too closely to any given host/terminal pair. The host is likely to be an existing computer system, the terminal could be one of many available micro-computers (not all are adequate, see further on). The most common means of communications is the RS-232C serial line.

In our implementation, the host is a NEC PC–9801, chosen for the wide range of add-on boards (ADC, DAC, etc.) available, linked to a HITAC–240H mainframe computer. The termnial is a NEC PC–100, chosen for it's high re-

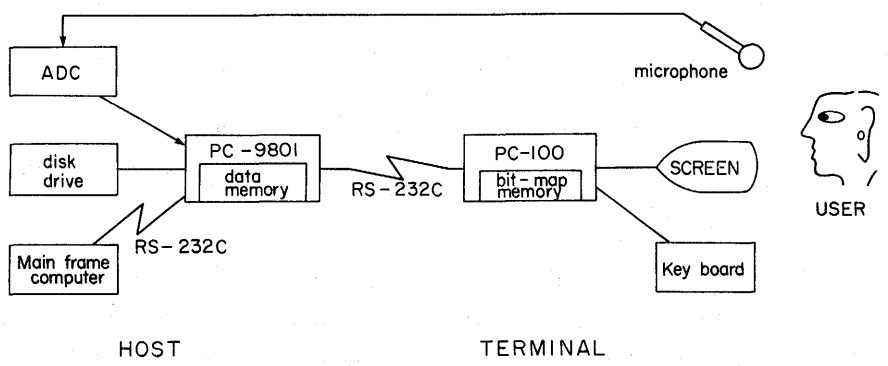

Fig. 1. Architecture of speech editor.

solution bit mapped graphic screen, and easy control of the bitmap. The two communicate over a 9600 bit/s RS–232C serial line (Fig. 1).

The user interacts with the PC–100, the PC–9801 is it's slave.

2) Mapping memory to screen.

a) Scaling

The host data memory contains 256k samples of data, but the terminal screen can only show 512 samples at a time. The data must therefore be reduced by a certain ratio (called SCALE). SCALE will take the value 1～512. To SCALE samples correspond one point (or vertical line) on the screen (Fig 2). SCALE= 512 maps the entire memory to the screen.

b) Anti-aliasing

Displaying one sample out of every SCALE would cause severe aliasing.

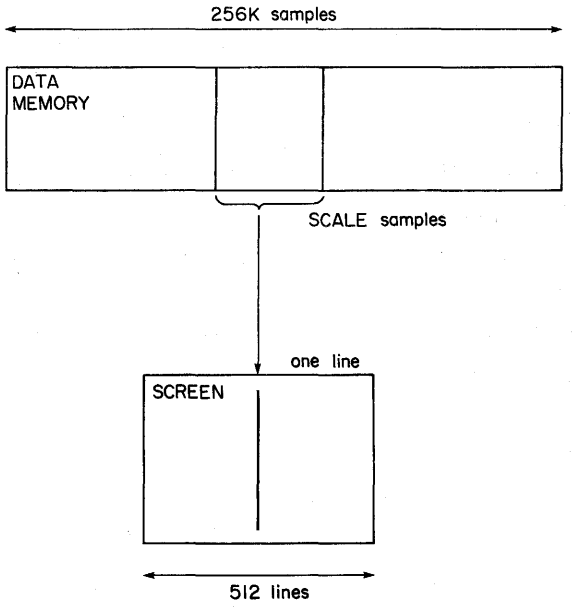

Fig. 2. Scaling all or part of the data memory to fit the screen.

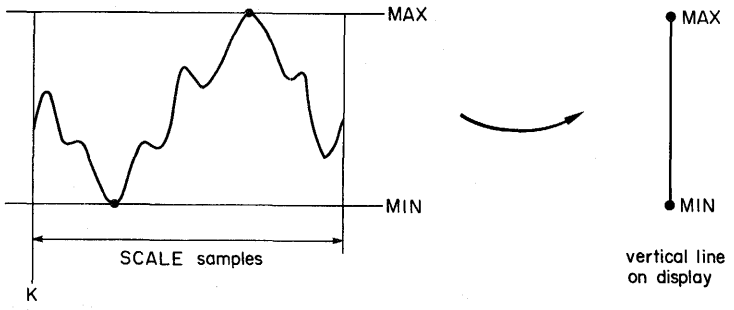

Fig. 3. Anti-aliasing.

Down-sampling is possible, but it is equivalent to severe low-pass filtering, and we do not particularly want to see low-pass filtered speech.

The solution adopted (Fig 3) is to calculate the maximum and minimum of every group of SCALE samples:

$$\left.\begin{aligned} \mathrm{MIN} &= \min_{i=0,\cdots \mathrm{SCALE}-1} (S_{k+i}) \\ \mathrm{MAX} &= \max_{i=0,\cdots \mathrm{SCALE}-1} (S_{k+i}) \end{aligned}\right\} \quad (1)$$

(where $S_{j,j=1,\ldots 215k}$ are the speech data samples and k is the index of the first sample of the group)

We then draw a vertical line from MIN to MAX on the screen.

For large values of SCALE, this guarantees that we see the envelope of the waveform. For samller values of SCALE we get more detail, and for SCALE=1 we see each original sample.

c) Visual continuity

For SCALE small, consecutive lines may not overlap, so there may be gaps in the displayed wave-form. To ensure a minimum overlap, the previous calculation is modified:

$$\left.\begin{aligned} \mathrm{MIN} &= \min_{i=0,\cdots \mathrm{SCALE}-1} (S_{k+i}, \text{ previous MAX}) \\ \mathrm{MAX} &= \max_{i=0,\cdots \mathrm{SCALE}-1} (S_{k+i}, \text{ previous MIN}) \end{aligned}\right\} \quad (2)$$

This guarantees the visual continuity of the display (Fig 4).

d) Data transfer

MAX and MIN are the only data that need transfering via the serial line. They are both represented by one byte, so it takes 0.83 seconds to transmit a screen of data (1024 bytes) on a 9600 bit/s line.

3) FLOW mode

The editor has a data acquisition mode, FLOW, and a data manipulation

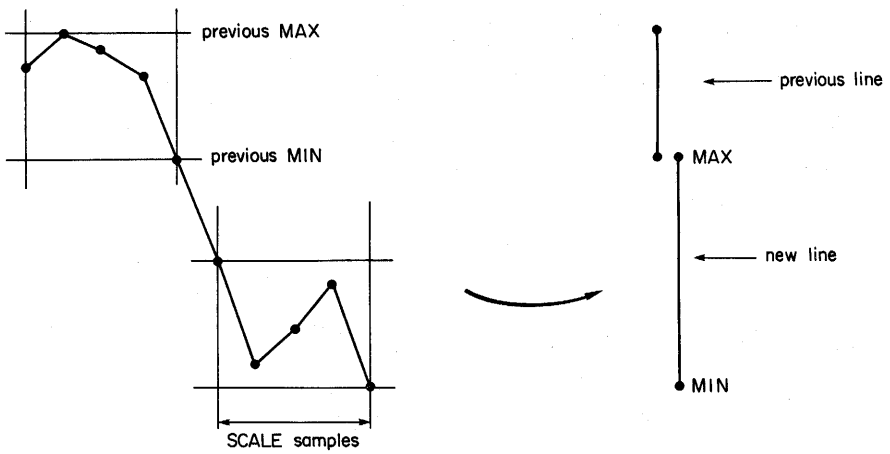

Fig. 4. Insuring visual continuity.

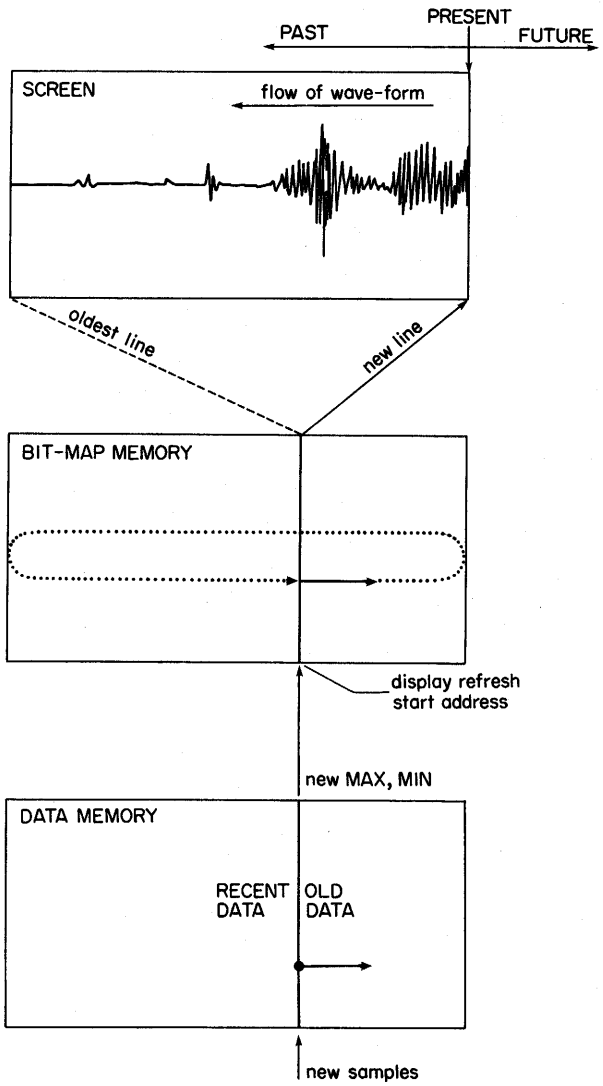

Fig. 5. Mapping data memory to screen in flow mode.

mode FREEZE.

In FLOW mode, the mapping discussed above is done in real time. Within the data acquisition loop, the following operations are performed for each asmple:

* sample the input signal,
* convert it to value Si,
* compare Si to the current MAX and MIN, update MAX or MIN if necessary.

Every SCALE samples the loop contains the following steps:

* send MAX and MIN to the RS–232C port,
* swap MAX and MIN (see formulae (2))

This loop is written in machine language for speed, and resides in the PC–9801. The PC 100 simultaneously performs a machine language loop that does the following:

* take MAX and MIN from the RS–232C port,
* draw a line from MIN to MAX,
* increment the bitmap refresh start address.

The last step is necessary to obtain a homogenous mapping of memory to screen: The screen is a picture of memory, in which the most recent samples map to the right hand edge. Each time a group of samples is over-written, the old line thta represented them "floats" off the left edge, and a new line is drawn to the right. The entire wave-form appears to flow leftwards, hence the name of the mode.

Udpating the refresh start address avoids copying the whole bitmap when the display shifts leftward (Fig 5).

A simpler alternative would have been to display left-to-right, erasing or over-writing the display each time it is full. However this produces an inhomogenous "saw-tooth" visual feedback, and we felt that it would have defeated our purpose.

The display loop in the PC–100 does one more thing: test the key-board. An "E" (expand) or an "S" (shrink) changes the value of SCALE (values of 128, 256 and 512 are available in FLOW mode). An "F" (FLOW/FREEZE) transfers to FREEZE mode.

4) FREEZE mode.

FREEZE is basically simpler than FLOW: there is no input, all one does is browse through the frozen speech data.

The display defines a window opened on data memory, that can be centered anywhere, with any scale (between 1 and 512, in powers of 2). SCALE is modified using the same keys "E" and "S" as in FLOW mode.

A cursor on the display can move left or right, in steps of 1 or 10 points. The cursor points to a unique sample in data memory. When the scale is modified, the cursor defines the fixed point around which the waveform expands or shrinks. This fixed point can be brought to the center of the display by the command "C" (this shifts the window position in data memory).

Ten markers are available and can be set by placing the cursor at a sample and pressing any key between "0" and "9". If the marked samples are within the window, the markers show up as red lines. In addition, the position in time of each marker relative to marker 0 is calculated and displayed.

The markers define cutting points for editing or playback (not yet implemented). For example, to file a portion of data on diskette one can place a marker at each end, press the key "M", input the file name, and input the marker numbers. By using more than two markers one can splice portions together. Inverting the markers stores the portion backwards.

Finally, by pressing "F" one can toggle back to FLOW mode.

5) Additional remarks

The input speech is sampled at 32 KHz and converted to 12 bits, stored as 16 bit words.

Both micro-computers use 16 bit 8086 processors. The main programs and subroutines were written in lattice C over MS- DOS. The FLOW subroutines (some of which are used by the FREEZE mode also) were written in assembly language for speed.

The reader wishing to implement a similar system must make sure that the machine that he intends to use as a terminal supports bit-by-bit refresh address incrementation (smooth horizontal scroll). This is essential for the FLOW mode. Many micro-computers olny support 16 bit word-by-word incrementation, and this produces a jerky movement instead of a smooth one.

The limited rate of the 9600 bit/s serial line introduces a small but perceptible lag in the redrawing of the screen (0.83 seconds), and limits the value of SCALE to 128 in FLOW mode. If the available rate is lower than 2400 bits/s, it is probably not worth trying to impelement an editor of this type.

If the host and terminal wrer the same machine we would avoid the communication problem. However, if only one processor is available, the data input and drawing loops must be interleaved in real time.

C. Perspectives

The FLOW mode, and the visual continuity of the FLOW/FREEZE transition are the main featurse of our editor. They form a firm basis for the speaker-machine-user interface.

Once the data is input, more sophisticated manipulation, processing and visualisation are possible, and a natural step would be to extend the editor in that direction.

A user may wish to access in quick succession different portions of a signal, or the same portion at diffreent scales, or several different signals, or the result of processing (spectrograms, pitch curves, phonetic labelling, etc). Even with the data reduction involved in scaling, the amount of potential visual information is enormous. Smalltalk-like mouse-driven windows would provide an efficient means of managing this information.

For processing, a number of environments have been proposed. Kopec (1984a, b) proposed a system built over a Lisp-machnie workstation and incorporating a signal representation language SRL (Kopec 1983), ISP environment and interactive user interface. Johnson (1984) proposed a software system based on UNIX, that enables signal processing "tools" to be assembled as blocks within a data-flow language (each block is executed when it's input data is ready). The tools can be written in any language. A "scope" tool manages a multi-window

display. Gong and Haton (1985) proposed a UNIX-based environment that combines interactive features similar to Kopec (1984a, b) with richer processing and data base managing possibilities.

The ideal would be to integrate a FLOW mode into one of these environments. This is possible only if the system that the environment is built upon allows real time data input and display, and in particular if the bitmap refresh address can be updated. If this turns out impossible, a solution might be to use multiple displays (as in Kopec's system), one being dedicated to input monitoring.

The real-time input niterface may seem a minor element in a sophisticated signal processing environment. This is certainly true if the emphasis is on interactive processing or analysis of stored data. However if, as it should, the interaction is to include speech input (and not just key-punching and mouse shaking), then the ideas expressed here may prove useful.

CONCLUSION

We described our efforts to design a smooth speaker-machine-user interface for a speech work-station, and the motivations behind these efforts.

The editor we built is an easy-to-use and useful tool for monitoring speech input. In addition to speech, it should be ideal for monitoring the input of continuous slowly varying signals such as biophysical parameters (brainstem potentials, nerve action potentials, EEC, ECG, etc.) or speech parameters such as pitch and intensity (de Cheveigne 1982).

It is our hope that these ideas can successfully be integrated into future speech and signal processing environments.

ACKNOWLEDGEMENTS

The first author is supported by a fellowship granted by the Japan Society for the Promotion of Science. He wishes to thank the second author for doing most of the work, Dr Ariki for trying some of these ideas in his real-time speech analysis-synthesis system, and the members of prof. Doshita's lab for their help.

REFERENOES

de Cheveigne, A. (1982) "Memoire digitale de transitoires, contribution a la phonetique instrumentale" These de 3e cycle, Universite Paris 7.

Cong, Y. and Haton, J. P. (1895) "Assia, un editeur "intelligent" pour la manipulation et l'analyse du signal vocal" Actes des 14e JEP du GALF.

Johnson, D. H. (1984) "Signal processing software tools" Proc. IEEE ICASSP-84, 8.6. 1–3.

Kopec, G. E. (1984a) "The integrated signal processing system ISP" Proc. IEEE ICASSP-84, 8.1. 1–4.

Kopec, G. E. (1984b) "The integrated signal processing system ISP" IEEE trans ASSP-32, 842–851.

(Aug. 31, 1985, received)