A proposed design for an audio processing system

VICTOR E. P. LAZZARINI

Núcleo de Música Contemporânea, Departamento de Artes, Universidade Estadual de Londrina, Cx. Postal 6001, 86051-970 Londrina Pr, Brazil
E-mail: Victor.Lazzarini@lda.palm.com.br

The MUSIC *N*-family of sound compilers offers a flexible but complex set of tools for computer music composition. These packages are, in a general sense, difficult to grasp by the beginner. Their design is also based on an outdated paradigm of *scores* and *orchestras*. New alternatives to such systems are a current necessity. This paper presents the software Audio Workshop as a proposed design for a sound synthesis and processing system. This program has been developed for the Win32 platform, offering many synthesis and processing options in a simpler and intuitive way. The paper describes the elements of the program and discusses its design and applications.

1. THE MUSIC *N*-FAMILY OF AUDIO PROCESSING SYSTEMS: TWO DIFFICULTIES

Systems for sound synthesis and processing, such as Music V (Matthews 1969), Cmusic (Moore 1990) and Csound (Vercoe 1992), are basic tools of computer music practice. They provide the means for the creation of sounds that is part of the activity of the composer involved with electroacoustic music. Most of these systems were developed from a family of audio processing programs that are situated at the basis of the evolution of computer music (Dodge and Jerse 1985). This family dates back to 1957 with the MUSIC I program, written by Max Matthews (Manning 1993). These packages are usually called sound compilers, because they involve processes that are similar to the coding and compilation of a program. They inherited many design features of the first packages, the most notable being the interface with the user and the *orchestra–score* paradigm. Although they are a very powerful means of sound manipulation, these packages pose two primary difficulties. The first is the effort an inexperienced user, for instance a composer introducing himself to computer music, has to put into the process to achieve some results. The second is that the *orchestra–score* paradigm is somewhat outdated and misleading, with respect to a broader view of music brought on by the electroacoustic practice.

The initial difficulty with the MUSIC *N*-family of sound synthesis 'languages' is presented when employed by musicians willing to practise computer music, but with little understanding of programming. It can be shown by tracing the steps taken in the process of synthesising sounds using a sound compiler. The first step when dealing with such packages is to define a code for the signal flow of the *instrument* used in the synthesis or processing. This is done using the proper syntax of the particular 'language' being employed. In Csound, the most popular of such systems, the syntax includes several kinds of programming statements, variables and constants of three different rates, opcodes and function generators (Pope 1993). A simple sinewave instrument definition, including the *orchestra* header, written using Csound syntax would be:

```
      sr=44100
      kr=100
      ksmps=441
      nchnls=1

instr 1

a1    oscil  p4, p5, 1

      out    a1

endin
```

In order to make this 'instrument' synthesise a sinewave of some arbitrary frequency and amplitude, a *score* would have to be defined in a separate file. This is an example of a Csound *score* code that uses the above to create a one-second 440 Hz sinewave sound:

```
f 1 0 1024 10 1
i 1 0 1  10000 440
```

The third step is to save the code in two separate files (orchestra and score). To compile the sound, which will be written in a soundfile, a Csound command line should be employed:

```
csound sine.orc sine.sco -W -osine.wav
```

The designing, coding and compiling of such an instrument require prior knowledge of several concepts on the part of the user. For an inexperienced

Organised Sound 3(1): 77–84 © 1998 Cambridge University Press. Printed in the United Kingdom.

user, the results of the example above could be deceivingly uninteresting, considering the effort that he puts into the process. A situation arises where the computer music novice has to understand concepts that are not necessarily musical in order to carry out his/her first software synthesis experiments. The whole process involves some problems of computer programming that are sometimes difficult to grasp by the inexperienced user. These problems could be introduced (or not at all) to the musician at a later time in the development of his/her electroacoustic and computer music practice. To the novice, though, it seems that the contact with the production and manipulation of sounds is of primary importance. Therefore, it must be offered in a more immediate fashion, without the necessity of being acquainted with a particular sound compilier syntax.

The paradigm of a list of statements that controls coded *instruments* was installed by the MUSIC *N*-family of programs. The structures of their syntax favour their use in a more traditional way, where *orchestras* of *instruments* play *notes* assigned to them by a *score*. Composers are often biased to create music that is note-oriented in nature when dealing with such systems. Instead of encouraging a new practice, on a new medium, they model themselves after the tradition of concert music. The paradigm of sound compilers is outdated, with respect to the development of a broader conception of music, brought on by electroacoustic practice. Sound compilers offer, nevertheless, some flexibility to accommodate different uses. Composers often look for ways to use these programs in a more creative manner, sometimes subversive to their original design.

It would be interesting, when considering some ideas for the design of a sound processing system, to bear in mind these two difficulties. A flexible and powerful, yet intuitive and simple, tool for computer music is a real necessity for composers of electroacoustic music (Caesar 1997). The tool would have, among other characteristics, more immediate access to the manipulation of sounds and a better user interface. It would also be designed in such a way that the user would be encouraged to create music free from the determinism of the *orchestra–score* paradigm. Such a program would provide a smoother contact with sound synthesis and processing than the more complex sound compilers, as well as act as an introduction to their use.

2. A PROPOSED DESIGN: AUDIO WORKSHOP

In consideration of these questions, Audio Workshop (Lazzarini 1997), a sound synthesis and processing program, is proposed as an example of an alternative design. The necessity for the development of tools directed to introduce computer music practice to noninitiated composers and students was the motivation behind the initial versions of the program. As there was little point to research that led to a sound compiler clone, a new design was put to the test. Using a graphic interface to provide the outer shell and the communication with the user, the program provides a platform where the beginner can work in a more intuitive way.

The design of Audio Workshop is directed to the treatment of sounds (and not *notes*), stored in soundfiles, as the basic elements of computer music practice. The elementary unity manipulated by the program is the soundfile, either as a product of direct synthesis or as an input and output of a signal processing operation. The soundfile is treated as a virtual tape cutting that would be used in the compositional process.

The functionality of Audio Workshop is divided into two main areas: synthesis and processing (modification), which are accessed by the respective menus or toolbar buttons. The user can experiment separately with different sound synthesis strategies, and then process the sound stored in soundfiles. Immediately after the synthesis or processing is done, the user can view the resulting waveform and listen to the sound created (figure 1). The steps taken in a typical user action are much simpler, when compared to the use of a sound compiler, as shown before.

Audio Workshop has been successfully used as a means of teaching Music Technology to noninitiated musicians (Lazzarini 1997). The emphasis on its use as a training and introductory tool has been the main concern of the development process. Nevertheless, the program can be useful for experienced users, offering good processing capabilities in a neat and fast package, as will be shown.

3. PROGRAM DESCRIPTION

Audio Workshop is being developed for the Win32 platform in C++ under Microsoft© Visual C++, and

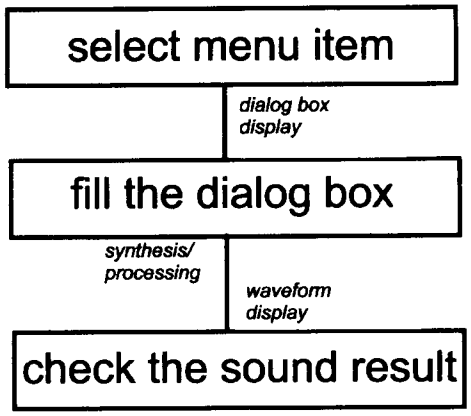

Figure 1. Typical user steps in Audio Workshop.

https://doi.org/10.1017/S1355771898009194 Published online by Cambridge University Press

it uses the MFC class library to supply some of the user interface features. The program is organised into three main sections, which are: synthesis, processing and soundfile utilities. There are two main synthesis and fifteen processing options available. The soundfile utilities include waveform display, playback controls and properties display. A help file is available. The program reads and writes 16-bit RIFF-Wave format soundfiles (Pope and Van Rossum 1995), the most common audio format for the Windows-PC platform. A proprietary parameters file is also read and written by the program, storing parametrical information for the synthesis routines.

All the synthesis and processing functions are accessed via menu commands which display dialogue boxes. These contain fields which correspond to parameters which will supply the processing routines. The *soundfile* field is common to all dialogue boxes, where the user will supply the names of the output and input (when applicable) files of audio data. An example of a dialogue box of the processing section is shown in figure 2. In this case, the user has to select the type of filter, fill in the fields reserved to input and output soundfiles, gain, centre frequency envelope and bandwidth (if necessary). As described before, the program will generate a new (filtered) soundfile and display its waveform. The same action will be required to use any of the other processing and synthesis options.

3.1. The synthesis section

The synthesis section provides two main options for sound creation: wavetable/FM and additive synthesis. The *synthesise* menu calls the appropriate parameter dialogue boxes for either type of synthesis. When displayed for the first time, the boxes are shown in their default configuration, with some items disabled. This is the simplest of their forms. In the case of the wavetable/FM dialogue, all the envelopes and frequency modulation are switched off, causing the system to behave like a single oscillator with fixed frequency and amplitude. As the user gets acquainted with the synthesis processes, he/she will be incorporating other features. Envelopes, modulation, panning and noise generation can be switched on or off by means of check boxes.

The signal flow of the wavetable/FM synthesizer is shown in figure 3. The diagram shows the default configuration of the dialogue box, with all the checkboxes, represented by the switches, unchecked. For the functions stored in the tables (f1 and f2 in the diagram), the user can optionally have saw, square or buzz (pulse) waveshapes, with up to 99 harmonics (care must be taken to avoid aliasing). For each parameter in the diagram there is a corresponding dialogue box item for the user to fill in. Other dialogue box items not included in the diagram are soundfile name, duration, sample rate and panning control (in the case of stereo soundfiles). Amplitudes are given in dB, and frequency in Hz.

The size of the table used is 16,384 and a truncation method is used by the oscillator to read the values stored in the table. Interpolating oscillators will be available in the near future. Nevertheless, the quality of the truncating oscillator output has been musically satisfactory, both for simple and FM synthesis. The time-lapse figures for synthesis (table 1) show a better than realtime performance of the synthesizer, which makes possible the implementation of realtime synthesis. There are plans to add realtime processing to the program in forthcoming versions.

The envelope generators are of the four-stage ADSR type, where the release period is calculated

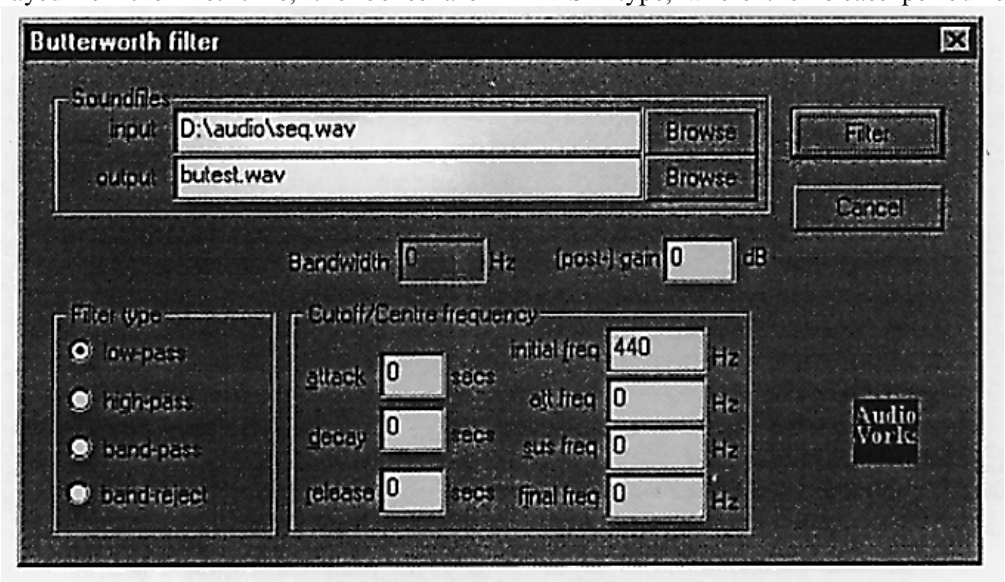

Figure 2. Butterworth filter dialogue box.

https://doi.org/10.1017/S1355771898009194 Published online by Cambridge University Press

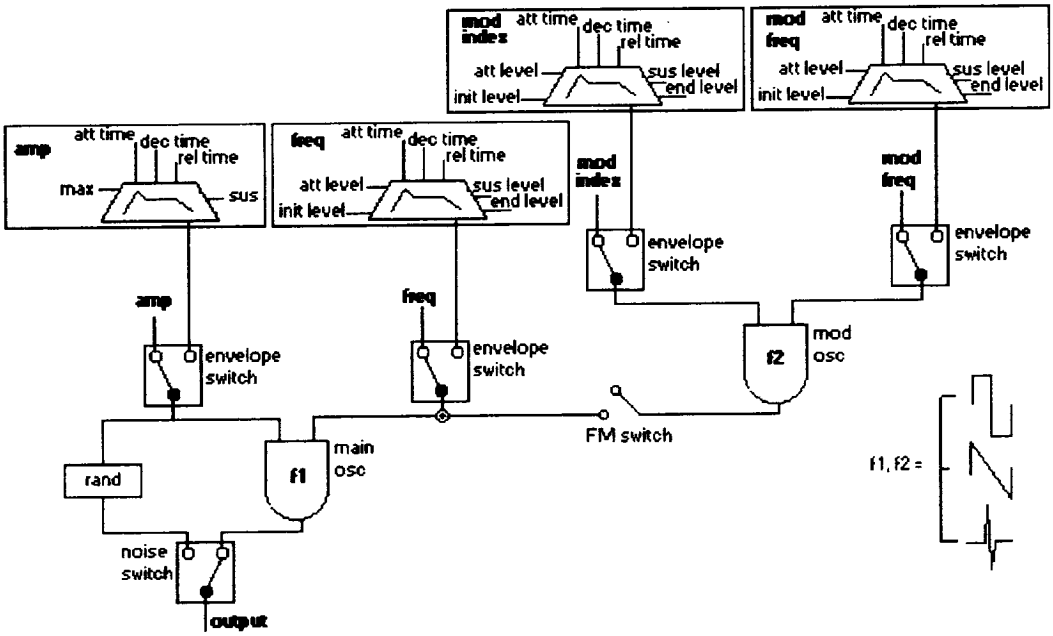

Figure 3. The signal flow of the wavetable/FM synthesizer function.

backwards from the end of the file. They are more flexible than the LINEN (Dodge and Jerse 1985) envelope generators offered by the sound compilers. A check box switches on the rand generator, which is a simple yet efficient noise generator, with amplitude envelope control. The panning control is switched on by the stereo button, with which the user can apply a time-varying pan on the synthesised sound.

The additive synthesis routine employs up to twenty-five parallel oscillators with sinusoidal output, each with time-varying amplitude and envelope. The interface with the user is via a property-page style window, a tabbed dialogue box with a number of pages. An additional setup dialogue box can be invoked when the user wishes to change the number of oscillators employed. The program will then display the additive synthesis window with the specified number of components. Each oscillator has its corresponding page, where the user can set its amplitude and frequency values. There is also a page where the user sets the global attributes such as soundfile name, duration, sampling rate and overall amplitude.

Table 1. Time-lapse figures for 10-second 44.1 kHz soundfiles synthesised by the program on a 166 MHz Pentium computer.

Simple oscillator synthesis (fixed freq./fixed amp.)	Mono FM without envelopes	Stereo FM with all envelopes
2.04 s	2.53 s	5.66 s

Table 2. Time-lapse figures for 10-second 44.1 kHz mono soundfiles created by the additive synthesis routine on a 166 MHz Pentium computer.

6 oscillators	12 oscillators	25 oscillators
8.9 s	15.5 s	30.0 s

The frequency and amplitude of each individual oscillator is controlled by a four-stage envelope. In order to scale the individual component amplitudes, the output of each oscillator is multiplied by

$$A_{\text{overall}}/(A_{\text{osc}\,1} + A_{\text{osc}\,2} + \cdots + A_{\text{osc}\,N}),$$

where A_{overall} is the overall amplitude and $A_{\text{osc}\,N}$ is the individual oscillator amplitude. This mechanism provides better control over the synthesis process, avoiding clipping distortion. The time lapse figures for additive synthesis show a better than realtime performance for six components and an acceptable performance for more components. Some strategies to improve the computational efficiency of the additive synthesis routines are currently being studied.

Given that both the additive and the wavetable/FM synthesis functions employ a great number of parameters, a suggestion was made to create some means of storing and retrieving these parameters. The *save* command, of the *synthesise* menu, saves all the current parameter data supplied to both synthesis functions in a binary file. When retrieving this data, there are two commands, one for additive parameters and the other for wavetable/FM parameters. They load the specified parameters into the program and display the respective synthesis window. There are plans to implement converters to transform the binary parameter files into meaningful ASCII text information so that users could manipulate the synthesis data in a text editor. The program would then convert this information back into binary form. This

https://doi.org/10.1017/S1355771898009194 Published online by Cambridge University Press

would be useful for additive synthesis, because of the sheer number of parameters involved.

3.2. The processing section

The routines implemented for soundfile processing can be divided into four main groups: amplitude transformation, mixing and panning, channel extraction and interleaving, filtering and delay lines. The first group is composed of seven options: envelope, mixer, splice/crossfade, loop, reverse and pitch shifter. The second has channel extraction, interleave and panning routines. The filtering functions are: filterbank, reson and Butterworth filters. The delay group has all-pass and comb filters, flanger and a general multidelay.

The envelope shaper applies a four-stage envelope to an input soundfile. It works by normalising the input and applying the chosen shape to it. The parameters prompted by the dialogue box are the envelope breakpoints and the respective amplitudes in dB. The envelope shaper can be easily used in consecutive passes, for optimum attenuation or boost of selected sections. The program overwrites input soundfiles, so that consecutive processing can be done without the use of additional storage, if the user so wishes. The simple mixer works with two input soundfiles, each with separate gain control (in dB). Two soundfiles can also be spliced together by the program, with user-defined crossfade time. Looping is done by selecting a section of the soundfile to be repeated a number of times, with each repetition being consecutively spliced after the initial portion of the soundfile. A reversing option that writes a soundfile backwards is available. To complete this group of processing routines, the pitch shifter transposes the soundfile a specified interval (expressed as a multiplier), as in a varispeed tape control. This function uses linear interpolation for nonintegral sample indexes, resulting in a satisfactory output.

The channel utilities are offered mainly as a means of converting multichannel files to be processed by routines that accept only mono files. The extract option extracts multichannel files with up to four channels, and the program can interleave two mono files into a stereo one. The pan routine moves the input soundfile between two stereo channels, according to the time breakpoints and positions specified in the corresponding dialogue box.

The filters offered by the program are of two main designs: two-pole general-purpose IIR resonator, of the form

$$y(n) = a_0 x(n) - b_1 y(n-1) - b_2 y(n-2),$$

and the recursive filter

$$y(n) = a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) - b_1 y(n-1) - b_2 y(n-2),$$

Table 3. Time-lapse figures for some processing routines applied to a 10-second 44.1 kHz mono input soundfile on a 166 MHz Pentium computer.

All-pass	Butterworth (low-pass)	Envelope	Filterbank (10 filters)	Reson	Pitch shifter
1.97 s	3.73 s	2.09 s	4.94 s	2.86 s	1.64 s

with coefficients set to give a Butterworth response.

The resonator is presented as a stand-alone filter, with time-variable centre frequency and as a component in a bank of parallel filters. The filterbank can employ up to ten of these resonators, with individual frequency, half-power bandwidth and gain. The reson option opens a dialogue box with bandwidth, gain and centre frequency envelope controls. The Butterworth filter is a design that has a maximally flat pass-band (Dodge and Jerse 1985), and is offered in four options: low-, high-, band-pass and band-reject. It has controls for centre frequency, gain and bandwidth, as shown in figure 1.

The all-pass and comb routines are fixed delay configurations that can be used to simulate complex reverberation. The comb filter is a simple recirculating delay line, with feedback gain control, which is a simple multiplier. This configuration colours the input sound, as it boosts certain frequencies and attenuates others, in a shape resembling the teeth of a comb. In contrast to it, the all-pass filter passes equally all the frequencies, in its steady state. It has a somewhat more complex configuration, shown in figure 4. The delay function works with a user-specified number of delay lines, whose delay times are integer multiples of the first delay. Its output is similar to that of a multitap delay. A variable delay, implemented using linear interpolation, is at the centre of the flanger routine, which can simulate Doppler effects and a number of flanging effects.

The computational efficiency of the processing routines is satisfactory. The time lapse figures for some of them are shown in table 3. As with some of the synthesis functions, there are plans to add real-time capability to some of the processing units, as the program performance is compatible.

3.3. Soundfile utilities

Complementing its functionality, soundfile waveform display and playback are performed by the program. After synthesis or processing, the output soundfile waveform is displayed and the user can play back the sound using the controls situated at the bottom of the program main window. These include play/pause, stop, forward and rewind. The play button serves either as play, pause or resume. The stop button returns the current play position to the beginning of the file and the rewind and forward buttons move the position 100 ms backwards or forwards. A third button opens a dialogue box for the user to specify a

https://doi.org/10.1017/S1355771898009194 Published online by Cambridge University Press

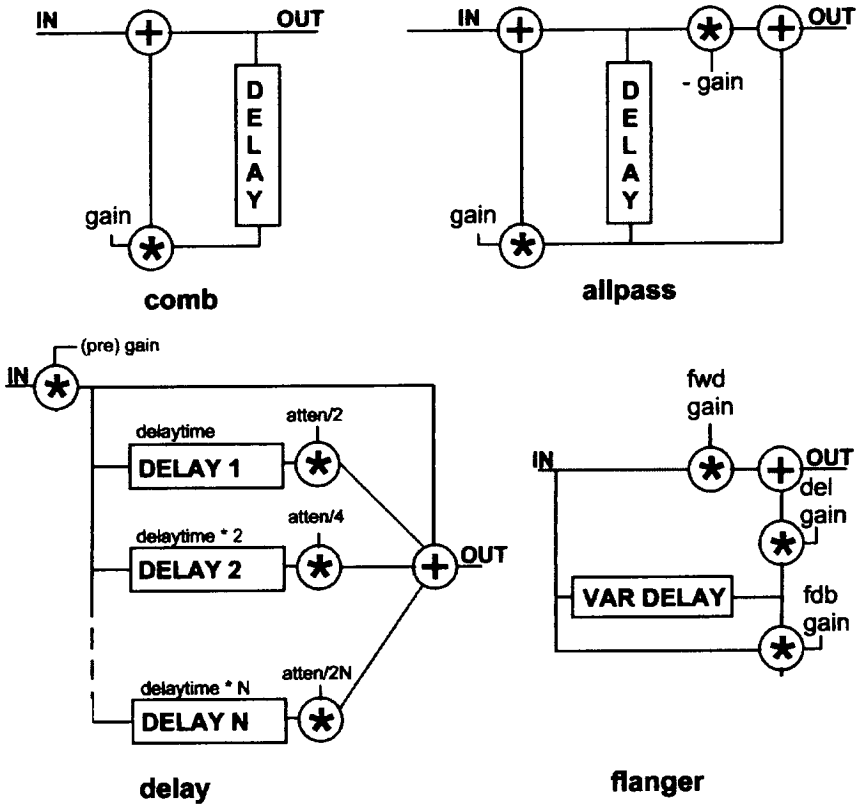

Figure 4. Audio Workshop delay units.

soundfile to be opened by the program. The program will then display the chosen file, and the user can use the playback buttons to perform it.

The drawing routine plots the soundfile with a user-specified zoom factor. It was implemented as a secondary thread, so that it does not monopolise the program resources. When the invoked synthesis or processing routine is over, the program starts a second thread that performs the drawing and returns control immediately. The user does not need to wait for the drawing of the waveform to finish in order to play back the soundfile or to access the program menus. As drawing a waveform can be a slow process, this is a very important feature of the design of Audio Workshop. Nevertheless, optimising the waveform display is still a concern for the developing process. The main program window, including a waveform display, is shown in figure 5.

The user can also obtain some information on soundfiles, such as duration, sample rate, channels and size in bytes. This is done by the command *properties* in the *info* menu. The soundfile utilities give the user a more interactive and immediate contact with the result of their work.

4. APPLICATIONS

At a cost of some flexibility, the program offers a simpler approach to sound synthesis and processing. The fact that the audio and control paths of Audio Workshop are hard-wired, although simplifying its use, limits the synthesis possibilities of the program. Therefore, Audio Workshop is not designed for tasks where the power of a sound compiler (such as Csound) would be more suitable. Nevertheless, the integration of the graphic interface, playback capabilities and processing functions makes the package an easy to use, fast and reliable tool for straightforward synthesis and transformation.

Audio Workshop has been used by music students attending the Music Technology course at the Universidade Estadual de Londrina, Brazil. The students in general had little or no previous knowledge of computer music. During the course, they have been practising the basics of software synthesis and signal processing with a direct approach to compositional applications. Their difficulties and successes have been taken into consideration, helping to shape the development of the program. It is expected that some students would employ Audio Workshop as a stepping stone on their learning process. After learning the basics, they would take up the use of more complex systems.

The lack of more advanced editing capabilities in the program has led users to employ Audio Workshop in conjunction with commercial soundfile editors. This is an underdeveloped area of the program that needs some attention. Although some graphical

https://doi.org/10.1017/S1355771898009194 Published online by Cambridge University Press

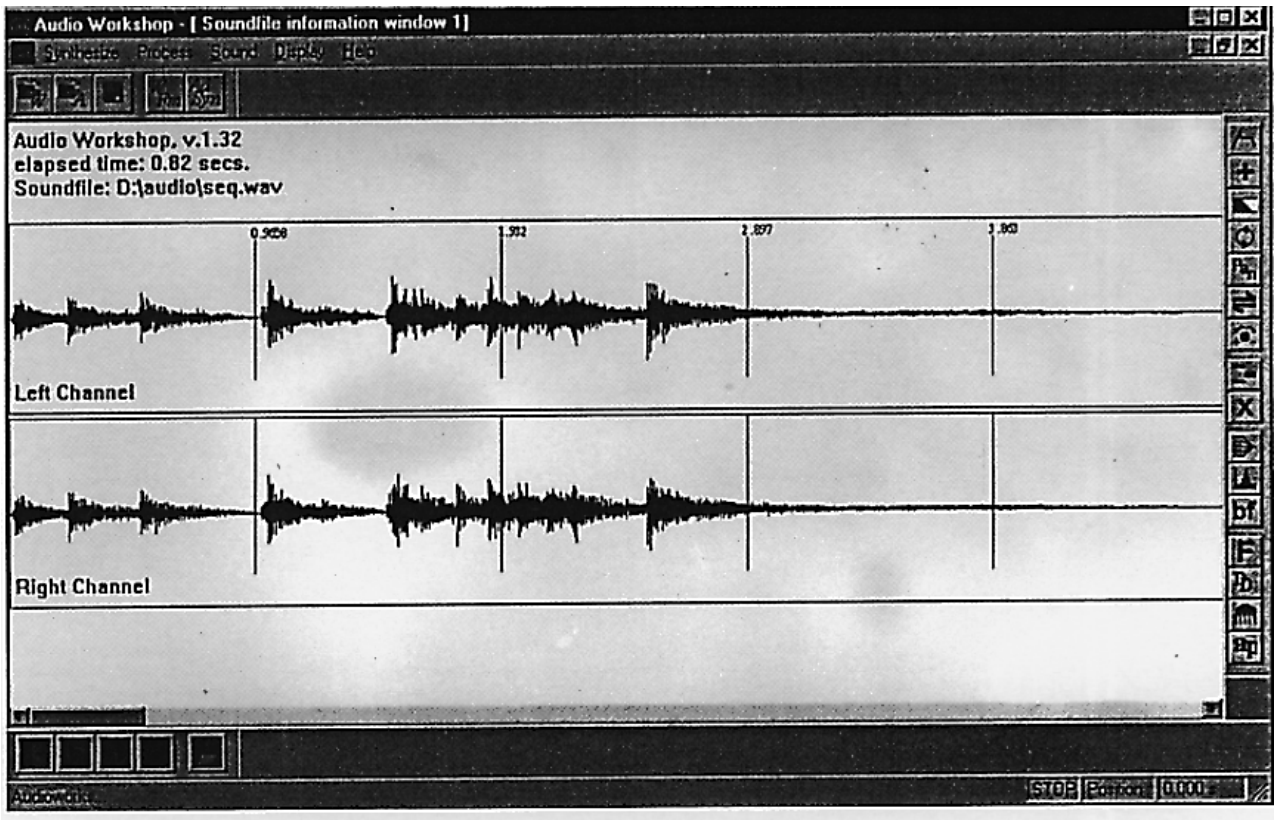

Figure 5. Audio Workshop main program window.

editing will be added to the program, Audio Workshop is not designed to be a soundfile editor, a task which is already very well performed by commercial programs. The association of editor programs with Audio Workshop in the compositional process is something to be pursued, rather than avoided. The program is designed to complement the functionality of the software already available.

Audio Workshop is not a finished tool. Some important functions are yet to be added to the program. Apart from graphical editing of soundfiles, frequency analysis, together with spectrum manipulation and resynthesis, will complete the program. They will give the user more powerful control over sound manipulation. These functions, when introduced into the program, will receive a graphic interface, where the processes involved can be handled in a metaphoric way. The design of this section of the program is not yet developed, but ideas are already being considered. The main concern is to present the very complex world of spectrum manipulation in an intuitive manner.

5. CONCLUSION

At present, Audio Workshop provides quick access to classic synthesis techniques and processing routines. It provides a good alternative to sound compilers for basic computer music tasks. Its design is more intuitive and free of the *orchestra–score* paradigm that prevails in the MUSIC *N*-family of software. The program has been successfully used as a learning tool for students being introduced to music technology. New functions will be added to Audio Workshop in the future, including spectral analysis and resynthesis. These will be incorporated in a new intuitive graphic interface, where the user will deal with the processes involved in metaphoric way. The program is available for downloading from http://www.ccc.nottingham.ac.uk/~amxvl/obras.html#software

REFERENCES

Caesar, R. 1997. Novas interfaces e a produçao eletroacustica. In A. Arcela (ed.) *Proc. of the IV Brazilian Symp. on Computer Music.* Brasılia: CIC, Universidade de Brasılia.

Dodge, C., and Jerse, T. 1985. *Computer Music: Synthesis, Composition and Performance.* New York: Schirmer Books.

Lazzarini, V. 1997. Audio Workshop, a program for audio synthesis and processing. In A. Arcela (ed.) *Proc. of the IV Brazilian Symp. on Computer Music.* Brasılia: CIC, Universidade de Brasılia.

Manning, P. 1993. *Electronic and Computer Music*. Oxford: Oxford University Press.

Mathews, M. 1969. *The Technology of Computer Music*. Cambridge, MA: MIT Press.

https://doi.org/10.1017/S1355771898009194 Published online by Cambridge University Press

Moore, F. 1990. *Elements of Computer Music*. Englewood Cliffs, NJ: Prentice-Hall.

Pope, S. T. 1993. Machine Tongues XVIII: a child's garden of sound file formats. *Computer Music Journal* **19**(1): 25–63.

Pope, S. T., and Van Rossum, G. 1995. Machine Tongues XV: three packages for software sound synthesis. *Computer Music Journal* **17**(2): 23–54.

Vercoe, B. 1992. *Csound*, *a Manual for the Audio Processing System*. Cambridge, MA: MIT Press.

https://doi.org/10.1017/S1355771898009194 Published online by Cambridge University Press