Some Applications of the Sound Object Library

Victor E P Lazzarini
Department of Music
National University of Ireland, Maynooth
Maynooth Co. Kildare
Ireland
e-mail: Victor.Lazzarini@may.ie

Abstract

This paper describes a number applications of the Sound Object (SndObj) Library. The SndObj library is a cross-platform object-oriented library which provides classes for audio synthesis and processing with soundfile, text and realtime audio IO support. In particular, the newly-developed realtime input and output classes are discussed. The implications for the easy development of sound processing applications are assessed. Examples of MFC-based graphical user interface programs are shown. Some possibilities of integration with multi-platform frameworks are considered. As a complete example, a composition application is presented: the computer *instrument* of a piece for live performer and computer.

1. The Sound Object Library

The Sound Object (SndObj) Library is a C++ object-oriented audio programming library (Lazzarini & Accorsi, 1998). It was designed for use by programmers and composers to develop sound manipulation applications. The library is being designed both as framework and toolkit. Its classes encapsulate all the processes involved in sound creation, manipulation and storage. Another feature of the library is portability across UNIX and Windows platforms, allowing for platform-specific code where necessary (realtime IO is being currently implemented on certain operating systems). The library class hierarchy is founded on three base classes, corresponding to sound processing objects, maths function-tables and sound input/output objects.

Version 1.0 of the SndObj library includes several synthesis and processing objects and support for RIFF-Wave and AIFF soundfiles. Platform-specific objects provide realtime input and output, currently implemented on SGI workstations under IRIX 6.5, Windows95/98 and Linux/Unix with Open Sound System. Parts of this project have been developed on a number of platforms, Sun Sparc under Solaris, IBM RISC 2000 under AIX and SGI O2 under IRIX, as well as on Pentium PCs under Windows 95 and Linux. At the moment, the latest (beta) versions of the library have been built on Windows, Solaris and IRIX operating systems. Reference documentation, including sample applications is currently available in html format. Compiled library binaries (Windows, Irix, Linux and Solaris), C++ code, documentation and sample programs are available at the NUI Maynooth web site (http://www.may.ie/academic/music/musictec). An extended introduction to audio programming and the library can also be found in (Lazzarini, 2000).

2. Realtime input and output

The addition of realtime audio input and output has created a number of new applications for the library. Initially, realtime audio IO classes were developed for the SGI platform, based on the Irix Audio Library API. Support for Windows realtime audio, based on the Multimedia Extensions API, and Linux, based on the Open Sound System API, was added. These classes are all derived from the SndIO base class. The model adopted was that of separating these platform-specific code from the multi-platform classes. Furthermore, each platform has a specific set of three classes for realtime IO: an abstract base class, an input class and an output class.

Platform	Base class	Input	Output
SGI	SndSgiRT	SndSgiRTI	SndSgiRTO
Windows	SndWinRT	SndWinRTI	SndWinRTO
Linux/OSS	SndOssRT	SndOssRTI	SndOssRTO

Constructors for these classes have almost similar arguments, so their use in a program is very similar. This approach was considered more suitable than that of designing a single class with mixed code controlled by compile-time definitions for each platform. The user have to define exactly what platform is his target at the time of coding. Modifying the code for a different platform is very simple, just a matter of changing the class constructor. Their behaviour, including that of their public methods, is the same as any other SndIO-derived class. `Read()` and `Write()` methods are provided for reading and writing samples from/to the DAC/ADC. Similarly to other library classes, its use is very straightforward, enabling the quick development of audio processing software.

The following code excerpt shows the use of an realtime output object used in conjunction with a SndObj-derived object:

```
SndWinRTO output(1);              //mono 16-bit output
output.SetOutput(1, &oscillator);//sets the SndObj-derived
                                  //object oscillator
                                 //as the output to channel 1
(...)
for(int n = 0; n < 44100; n++){
// loop 44100 times
(...)
oscillator.DoProcess();           //oscillator processing
output.Write();                   //writes samples to DAC
(...)
}
```

Realtime MIDI control input is also being implemented on the platforms above. This is performed by the SndIO-derived SndMidiIn class. A number of SndObj-derived MIDI message manipulation classes complement the functionality of the system. These classes are going to be available in the next version of the library, adding new possibilities for gesture control of signal processing.

3. MFC-based examples

Some new possibilities presented by the realtime IO can be demonstrated by a number of stand-alone applications. These were developed using a Microsoft© C++ graphical user interface and application framework, MFC. Although this framework, including its application models, is not exactly designed for the typical music/audio applications, a simple dialog-based model can be used to provide basic graphic user interface. Four programming examples were developed showing the possibilities of the library in association with MFC. These are collectively called *SndObj*Tools, for lack of a better name. They include simple reverberation unit, WSchroeder; a drone synthesizer, WRisset; a multi-string resonator, WStreson; and a channel vocoder/bank of filters, Wcvoc (**fig.1**).

3.1 The graphic interface

As mentioned before, these programs are based on a simple dialog-based MFC application model. This provides a dialog box with all its custom controls, without a menu bar or a background canvas. Since the function of these programs is to get an audio input, process it and send the result to the output, this simple model is enough. An exit button is provided for killing the application and a process button is available for toggling processing on/off. A number of sliders, text boxes, lists and radio buttons are used to change application parameters.

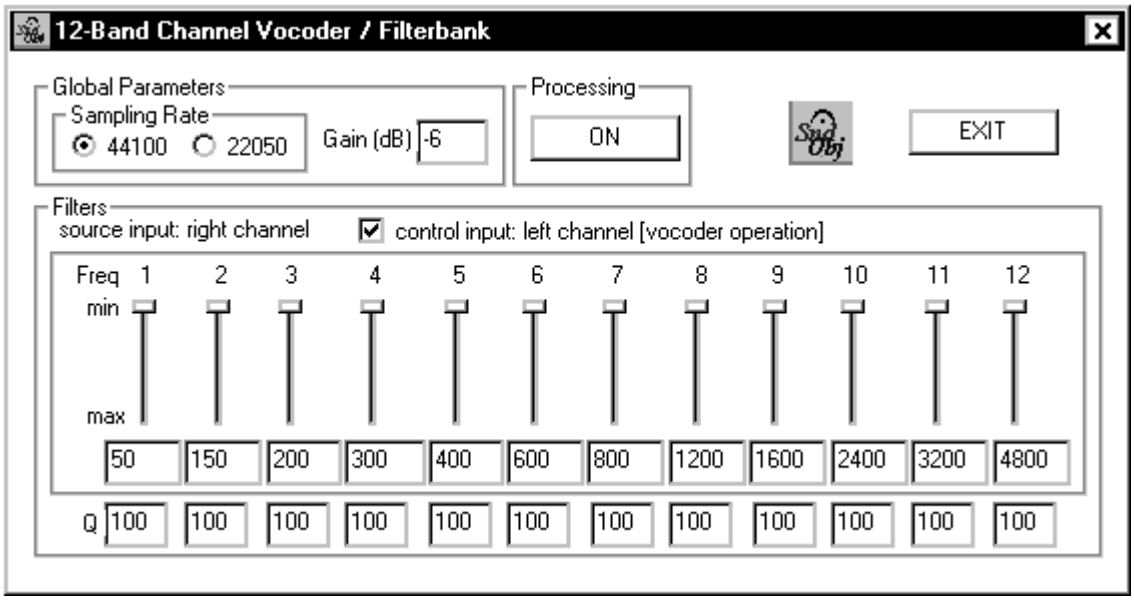

Figure 1

The dialog box application relies on an object derived from the MFC class `CDialog`. This contains pointers to SndObj- and SndIO-derived objects as class members. These objects are responsible for the sound processing algorithm. Ordered calls to the `DoProcess()` methods of the SndObj-derived objects are used in a processing loop to generate/process audio samples.

3.2 Multithreading

A multithreading model was used for these applications. This enables the immediate return of control to the application interface, so that some parameters can be altered on-the-fly. It also makes possible the on/off switching of the processing function. The program can behave in similar fashion to an outboard effects processor. For instance, in the case of the WCvoc program, shown in fig.1, the frequency sliders can be used to change the filter centre frequency in realtime.

The multithreading is achieved by creating an instance of the MFC class `CWinThread` and creating a function which will be executed in a secondary thread. This function contains the processing loop, which runs until a signal is sent to interrupt it. The secondary thread starts when the MFC function `AfxBeginThread()` is called. The name of the thread function and a pointer to the CDialog-derived class containing the processing objects are passed as arguments to `AfxBeginThread()`. The thread function contains the ordered calls to the `DoProcess()` methods of the processing objects. When this function returns, the secondary thread is finished and memory de-allocated. The start and termination of the processing thread is controlled by toggling on/off the process button in the user interface.

4. Other frameworks and multi-platform code

The examples mentioned above were developed for the Windows platform only. A more interesting situation would be to develop multi-platform GUI applications, taking advantage of the cross-platform features of the library. The use of a cross-platform framework will help develop such applications. Nevertheless, the multithreading support is somewhat different in UNIX and Windows, so some differences in coding are to be expected. In general, it is expected that a certain degree of similarity can be maintained between the code for different platforms. It should be noted that the use of a C++-based framework is also preferred to script-based GUIs, such as Tcl/Tk, or interpreted languages such as Java. The reason is that the C++-frameworks provide far better possibilities for the integration of code and increased flexibility. Also, because the final product is always an executable binary, there is no necessity of a run-time system. Some experiments with the V framework (Wampler, 199) suggested that it could be successfully employed to generate cross-platform applications. Since it is not a commercial software, its integration with the SndObj library is likely to be pursued further. It would enable the whole development process to be based on open-source software.

5. A computer *instrument*

As an example of a composition application, the computer *instrument* for a piece called *The Trane Thing*, composed by this author, is discussed. The piece was written for tenor saxophone and computer *instrument*. This *instrument* is a program developed for Windows, which does audio processing in realtime. Two basic ideas were behind this program: the first one was to set a number of virtual *strings* which would be excited by the incoming saxophone sounds; and the second, to sample sounds and play them back later. In terms of composition, the piece follows a very simple procedure, presents a certain number of musical ideas and comments upon them. These commentaries are not only written, but also improvised by the player, and here the possibility of using sampled elements is very important structurally. Most

of the motivic fragments used are based on the music of John Coltrane. Nevertheless, they are based on literal quotations, but come from the author's memory of Coltrane's recordings.

The program is an MFC-framework dialog-based application. It features four string resonators (Moore, 1990; Steiglitz, 1995), based on `StringFlt` classes, a number of gain controls, and also four sampling units, based on `SndLoop` classes. The string resonator frequencies can be set by a slider, and each of them work around a certain octave range (sometimes overlapping). The initial frequencies are tuned to specific pitches that are structurally important in the piece, so that the computer responds to them when they are played. Each string has a gain attenuation control, which can be used to regulate individual intensities. These are also slider-based, but in order to avoid unwanted clicks, a smoothing fade-in/fade-out was implemented. The fade in/out time is pre-set to of ½ second, on which the gain changes from the old setting to the new one.

The sampling units, as mentioned before, are based on a `SndLoop` object. This is a custom-developed class (which will appear in later editions of the library) that samples a signal and repeats this signal in a loop at a certain speed. The length of the loop in seconds and the pitch change (a multiplier) can be set by the user. There is also a method, `Resample()`, which can be used to trigger a new sampling operation. The controls on the program panel allow the user to choose between a number of pre-set pitches, set the gain attenuation and resample a signal, on each of the four sampling units. In addition, each of them also have a different sampling length from less than one to nine seconds. The mixed output from the sampling units is further modified by the use of two allpass filters which give a light (electronic-sounding) reverberation. The strings output is added to this signal and this sum is sent to the computer DAC (**Fig.2**). The program window is shown on **Fig.3.** Each one of the processing units can be turned on/off by a button. The program operation is also enabled by a general *processing on/off* button.

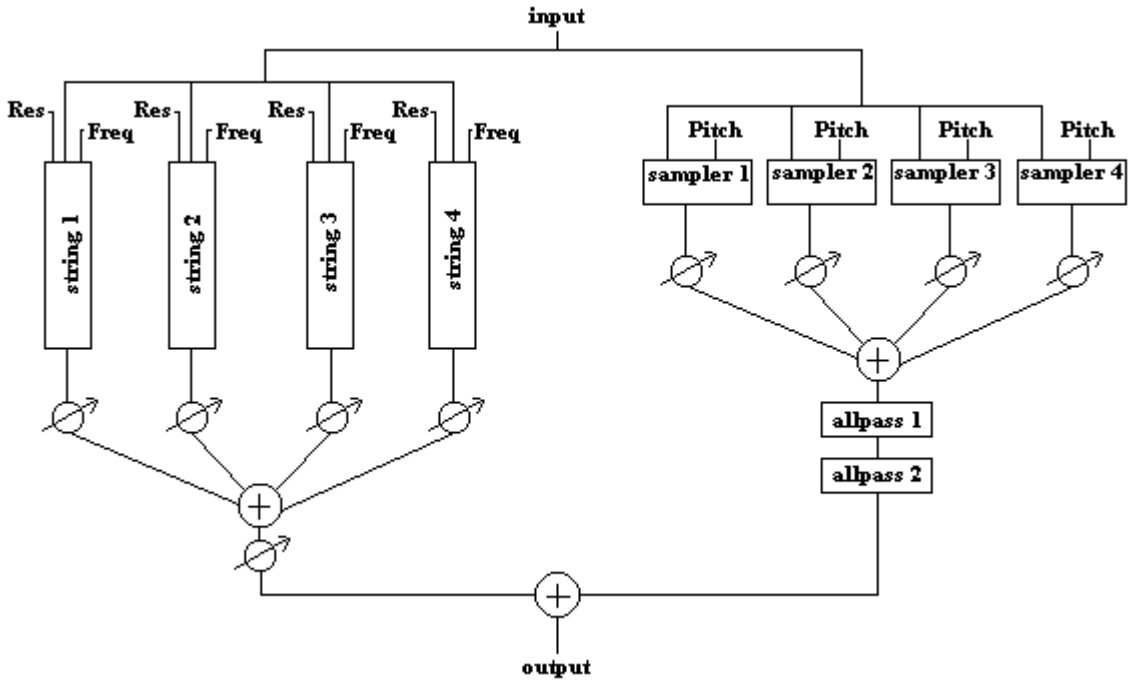

Figure 2

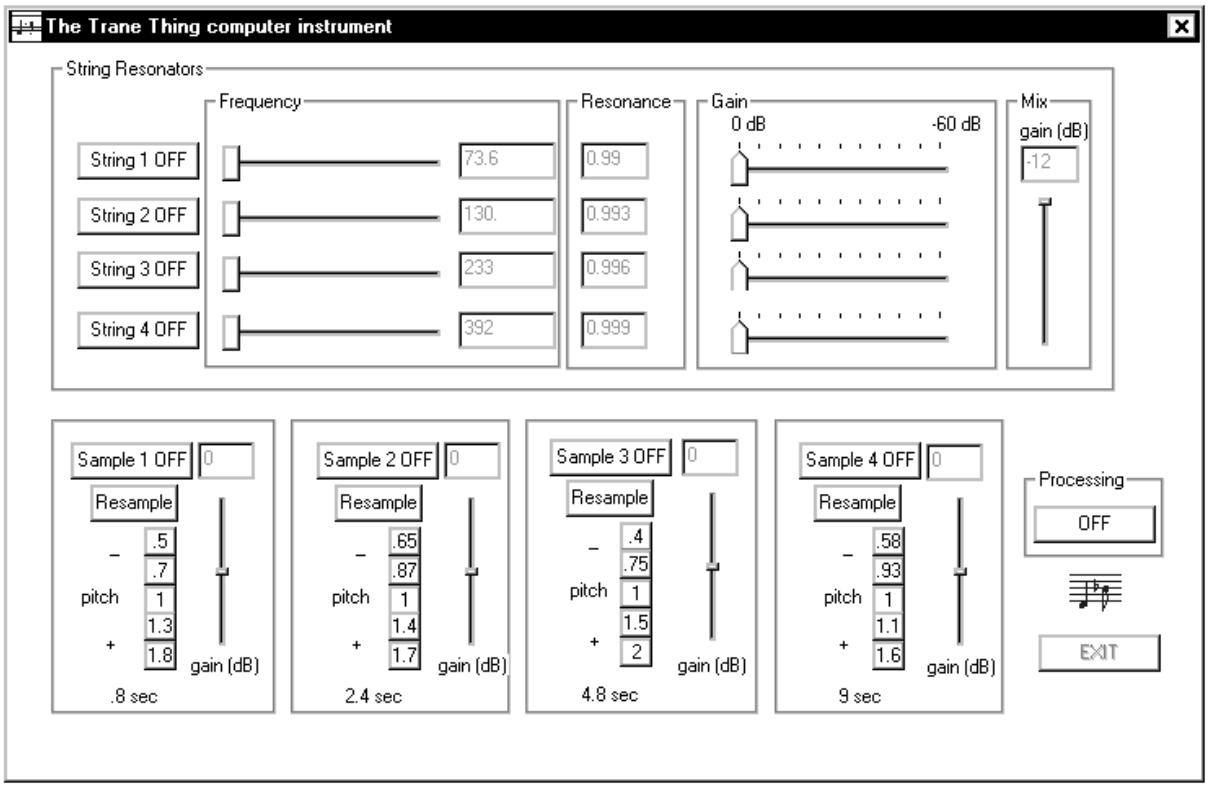

Figure 3

This software has a similarl designed to the previously discussed *SndObj*Tools programs. It relies on two threads, a control one, that takes input from the user and a processing thread. The latter is based on simple loop that calls the objects processing methods. The processing on these objects can be bypassed by the use of the `Disable()` method. The reverse action is made possible by a `Enable()` member function. All the calls to parameter-setting methods run on the control thread, so on-the-fly parameter change (slider movements, etc..) are possible and have an immediate effect on the processing.

The design of a program such as this one, to be used in a composition is one of the most important examples of the application of the SndObj library. It gives the composer the possibility to create his/her own design for the processing of audio and integrate that in the composition structure. The SndObj library makes it possible for many different algorithms to be implemented, including new classes to extend the framework (the development of the `SndLoop` class is an example of this). The result is also interesting in terms of performance logistics. Since the *instrument* is an executable binary, it does not require any runtime system or interpreter, apart from the operation system itself. So copies of the instrument can be easily distributed. The performance of the piece will only require a reasonably fast computer, running a particular OS, with good audio hardware. Also, because of the fact that the SndObj library is portable to the most commonly found OS's, it is possible to design instruments that are fully portable across a range of platforms (using, perhaps, portable application frameworks to make life easier).

7. Future Prospects

The SndObj library has become a very interesting tool for computer music composition and research. Future prospects include the addition of MIDI i/o, currently being developed, and support for other types of gesture control. In addition, the implementation of new processing classes will also increase the scope for its applications. Also, more composition applications, such as the one described above, will be developed. Finally, under research at the NUI Maynooth, in a joint project of the Music and Computer Science departments, is the implementation of a distributed processing system, based on and using the library. The target platform is, initially, the Beowulf (Sterling et al, 1999), but it should be portable to SMP machines. This system could provide a complete sound processing workbench on an affordable supercomputing facility.

6. Conclusion

This paper described, briefly, a number of applications developed using the SndObj library. A general description of the library was given and the new realtime input and output classes were introduced. These were shown to add a very important functionality to the library. A number of programs developed using the application framework Microsoft© Foundation Classes were discussed. Some other GUI possibilities were also considered. The use of multi-platform C++ frameworks was shown to be the most suitable way of adding graphical interfaces to SndObj-based applications. As a final example, a computer *instrument*, written for a particular musical use, was presented. In summary, the SndObj library can be used in a number of interesting applications, a very small number of which was shown in this article. It has the potential to be an interesting software tool for computer music research and composition.

Bibliography

Lazzarini, VEP (2000). “The SndObj Sound Object Library”. Organised Sound 5 (1). Cambridge Univ. Press, Cambridge.
Lazzarini, VEP & Accorsi, F (1998). "Designing a Sound Object Library". Proceedings of the V Brazilian Computer Music Symposium. Editora da UFMG, Belo Horizonte.
Moore, FR (1990). *Elements of Computer Music*. Prentice Hall, Englewood Cliffs, NJ.
Steiglitz, K (1995). *A Digital Signal Processing Primer.* Addison-Wesley Publ. Co., Menlo Park, CA.
Sterling, TL, Salmon, J, Becker, CJ & Savarese, DF (1999). *How to Build a Beowulf: A guide to the Implementation and Application of PC Clusters*. MIT Press, Cambridge, MS.
Wampler, B (1999). *V Reference Manual.* HTML document. http://www.objectcentral.com/vgui/vrefman/vrefman.html