SmsPerformer:
A Real-Time Synthesis Interface for SMS

Alex Loscos, Eduard Resina
Audiovisual Institute, Pompeu Fabra University
Rambla 31, 08002 Barcelona, Spain
{aloscos, eduard}@iua.upf.es http://www.iua.upf.es

Abstract

SmsPerformer is a graphical interface for the real-time SMS synthesis engine. The application works from analyzed sounds and it has been designed to be used both as a composition and a performance tool. The program includes programmable time-varying transformations, MIDI control for the synthesis parameters, and performance loading and saving options.

1 Introduction

The SMS synthesis is based on the combination of additive and subtractive synthesis [1]. The SMS analysis output is (1) a collection a frequency and amplitude values that represent the partials of the sound, (2) either filter coefficients and a gain value or spectral magnitudes and phases representing the residual sound and (3) a set of envelopes that represent high level attributes of the sound. From this representation an efficient synthesis can be implemented that offers many transformation possibilities.

The SmsPerformer application is a continuation of the software implementation of SMS started by a C code program with a command line interface developed by X. Serra and followed by a Visual C++ code with graphical interface implemented by J. Bonada named SmsTools [2]. SmsPerformer brings new possibilities to the SMS synthesis with real-time transformations and a graphical interface.

With the power of current general purpose processors it has become feasible to have software real-time implementations of additive synthesis and other musical research teams are developing similar programs [3][4][5].

2 Real-time SMS

Recent changes and optimizations to the SMS software have led to the current real-time implementation.

One of the changes concerns disc access. Loading the analyzed sound file into cache memory rather than reading it from disc speeds up frame fetching.

Interpolation between frames was one of the most stringent processes. Without frame interpolation, just taking the nearest, solves possible clicks in the output device buffer when morphing or time stretching. These problems are produced if the CPU spends more time in computing, synthesizing a given frame and writing it into the buffer, than the actual frame duration.

The number of partials to synthesize is another stringent requirement for the system to run in real-time as is the synthesis of the residual. In the analysis we can choose the residual of a frame to be represented as a magnitude spectral envelope, a complete complex spectrum, or a waveform. When kept as a magnitude spectral envelope, or a spectrum, the residual is deconvolved and added to the partials in the frequency domain. Once the spectrum of the current frame is filled, the IFFT is used to synthesize the waveform. When keeping the residual as a waveform, this is added in the time domain to the already synthesized partials, which is faster, but less flexible, than when keeping the residual in the frequency domain.

There is another synthesis process that presents a trade-off between computation time and flexibility. This the high-level attributes to be transformed and added back to the low level SMS representation [6]. More powerful and intuitive musical transformations can be achieved by controlling the high-level attributes at the expense of more operations to be carried in the synthesis.

3 Real-time under Windows

SmsPerformer has been programmed under Windows NT with Visual C++ 5.0 and using the SMS class library.

Because of the interactivity requirement we had to use two threads, one for the main synthesis process and the other for the graphical interface. The main window process creates the synthesis thread and sends new synthesis and transformation parameters to the thread each time a slider is scrolled.

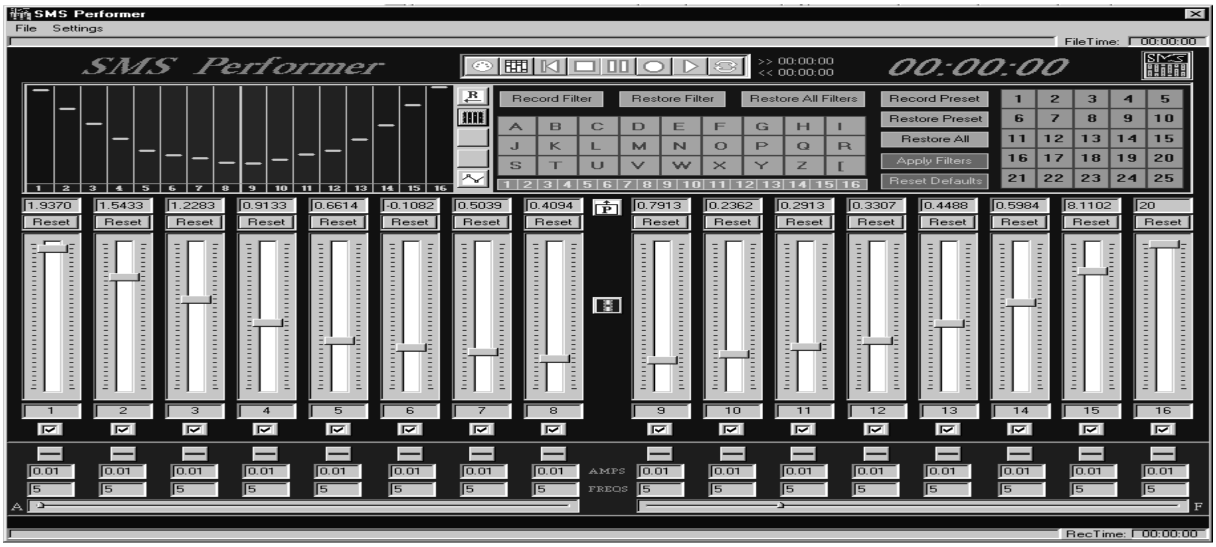

Figure 1. SmsPerformer main window.

When threading a sound application, setting priorities is not very reliable. The system uses the base priority level of all executable threads to determine which thread gets the next slice of CPU time. When setting the synthesis thread to a higher priority, we found the slider-scroll update rate was too slow, so we had both threads running with the same priority level.

SmsPerformer can run the synthesis and all available transformations in parallel and in real-time in a Pentium Processor 200 MHz, 32Mb RAM, under Windows 98 or NT, with 40 sinusoids, waveform residual, 85% of the system resources free. In terms of latency, using ISA (Sound Blaster) and PCI (Aztec and Event) sound cards we obtained transformation response delays under 40 milliseconds. Different framework conditions have different limitations.

3.1 Audio playback

SmsPerformer has been implemented using both Microsoft Win32 application programming interface low-level API and DirectX DirectSound [7][8][9].

The API option presents some tricks once you have sent the sound data to the sound card (filling the lpData field of the WAVEHDR structure and using the waveOutPrepareHeader and waveOutWrite functions) and you have to unprepare the WAVEHDR making sure all sound data has already been written in the Output Audio Device buffer to reuse it (using the waveOutUnprepareHeader function).

The waveOutGetPosition function retrieves the current playback position of the given waveform-audio output device but it does it very inaccurately. This forced us to work with an error margin. On the other hand the WHDR_DONE flag of the dwFlags field of the WAVEHDR structure is set by the device driver to indicate that it is finished with the buffer and is returning it to the application, but this setting has an important delay. So in both cases we need to increase the number of wave headers structures (SmsPerformer uses short sound data buffers).

DirectSound provides low-latency mixing, hardware acceleration, and direct access to the sound device. Its circular view enables infinite streaming buffers; as the front of the buffer is being consumed, the rear of the buffer can be refilled [10][11].

The write cursor indicates the position at which it is safe to write new data to the buffer. The write cursor always leads the play cursor, typically by about 15 milliseconds worth of audio data (shown in *Figure 2)*.

SmsPerformer implementation uses a streaming secondary circular buffer of 16 frames, located in

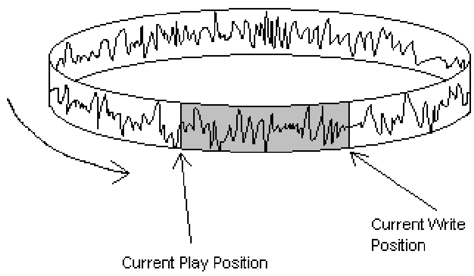

Figure 2. Secondary buffer with current play and write

system memory. DirectSound not only has a lower playback latency but also needs less sound buffers.

4 SmsPerformer features

SmsPerformer has many ways to control the synthesis-transformation of the sound: the main window sliders, a score file, a MIDI controller, or a set of graphical envelopes.

4.1 Main window sliders

The most immediate way to use the program is scrolling the main window sliders. You can configure which synthesis-transformation parameter controls each slider and set the maximum and minimum values of the parameter slider in a dialog window.

You can also modify the sliders value by drawing with the mouse the configuration you want the sliders to have. This can done in the up-left panel of the main window shown in *Figure 1*.

The sliders values can be modulated by a sinusoidal signal and also by a uniform random noise. When sinusoidal modulation is chosen you can control amplitude and frequency of the modulation with the sliders placed above. When noise modulation is chosen you can only control the amplitude of the modulation.

4.2 External MIDI controls

The application accepts MIDI messages for changing the transformation parameters from an external MIDI controller. In this way you can get a better physical feeling (like playing an instrument) than using the main window sliders. It is also possible to change more than one parameter at a time. For this purpose we have used a Peavey MIDI Controller PC1600 (shown in *Figure 3).*

The MIDI controller sends channel pitch bend messages to the application. The status bit is used to specify the slider that has to receive the message and it is followed by data bytes that specify the position of the slider between the maximum and the minimum defined. The message is like

First byte	Second byte	Third byte
0xEn	LSB	MSB

where n is the number of slider (channel) to control, 0 for slider 1 and F for slider 16, and then data is specified with a 14-bit number, using two 7-bit bytes, least significant byte first [12].

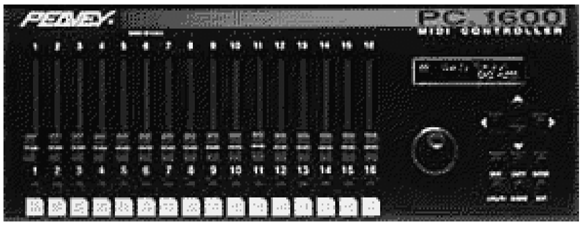

Figure 3. Peavey MIDI Controller PC1600.

A dialog window helps you specify the MIDI device, the number of data bytes of the messages, and the update message rate. This message rate indicates how often the sliders values are updated (pitch bend MIDI messages between time lapses are dismissed).

4.3 MIDI files

You can save performances as MIDI files and play already saved performances by loading them. The MIDI file saves the value, the slider (channel) and the time of a value change. When playing a MIDI file you can act on it modifying some parameter values at the time you hear the saved performance.

4.4 Score files

SmsPerformer can also read sms score files: synthesis files (*.syn) and hybridization files (*.hyb). These files are text files in which you can specify all sms synthesis and transformation parameters [2].

SmsPerformer can hybridize two sounds and modify hybridization parameters while the sound is being played. To do so, once the two sounds have been analyzed and saved as sms files, you need a hybridization file in which the two sms files have been specified. This is an example of hybridization file:

```
InputSmsFile violin.sms
InputSmsHybridFile trumpet.sms
Hybridize 1
```

4.5 Filters and presets

There is a bank of configurable filters and presets. Filters refer to which sliders are active and which are not. Presets refer to the value of each slider. You can also disactivate a slider by clicking the checkbox above it.

There are a number of preset options but you can create your own filters and presets and save them. You can see the filter and preset windows in the right upper side of the main application window (shown in *Figure 1*).

4.6 Envelope

You can have a performance by defining an envelope. This envelope is defined in a x-time axis and y-preset axis as shown in *Figure 4*.

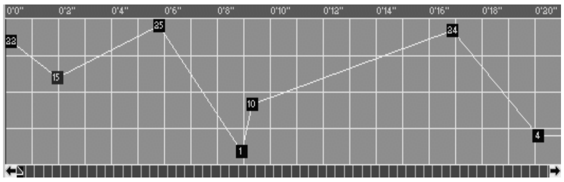

Figure 4. Envelope window.

You can define the presets you need and place them in the time axis. The values of the sliders are interpolated from one point to the next. You can modify both time and preset values of an envelope point while the sound is playing.

4.7 Play modes

The play modes in which you can work when performing with a given envelope, or with saved performances, are Normal, Loop and No-End.

Normal play mode plays the sound only once. If the transformations defined, or saved, make reference to a time outside the actual sound duration, it does not use them.

Loop play mode loops the sound until you click stop. This mode also loops transformations saved or defined whatever time duration they have.

No-End play mode loops the sound but not the transformations. So the sound is modified until the last saved or defined transformation time and then keeps looping maintaining the last parameters values. Clicking stop breaks the loop.

5 Conclusions

SmsPerformer is the first application that makes use of the real-time possibilities of the SMS software. Despite the problems of the Windows operating system to support real-time audio applications, SmsPerformer has shown to be useful for music applications. It is the first attempt of a performance tool and as such there is room for many improvements.

6 Acknowledgements

We would like to acknowledge the contribution to this research of the other members of the Music Technology Group of the Audiovisual Institute.

References

[1] X. Serra. “Musical Sound Modeling with Sinusoids plus Noise”, in G. D. Poli, A. Picialli, S. T. Pope, and C. Roads, editors. *Musical Signal Processing*. Swets & Zeitlinger Publishers. 1996.

[2] Music Technology Group. *SMS Web site*. URL: http://www.iua.upf.es/~sms.

[3] M. Wright, D. Wessel and A. Freed. *"New Musical Control Structures from Standard Gestural Controllers"* Proceedings of the ICMC. 1997. [Available at http://cnmat.CNMAT.Berkeley.EDU/ICMC97/papers-html/Tablet.html]

[4] IRCAM Sound Analysis/Synthesis Group. *Web site*. URL: http://www.ircam.fr/activites/recherche/ana-syn-e.html

[5] Bradford Enhanced Synthesis Technology Group *Web Site*. URL: http://www.comp.brad.ac.uk/research/music/simulator.html

[6] X. Serra and J. Bonada. “Sound Transformations based on the SMS High Level Attributes”. *Proceedings of the Digital Audio Effects Workshop*. 1998.

[7] P. L. Childs. *Audio Latency Analysis for Windows-based Systems*. URL:http://telephony.cornell.edu/Latency.html, 1997.

[8] A. Freed, A. Chaudhary, and B. Davila. “Operating Systems Latency Measurement and Analysis for Sound Synthesis and Processing Applications”. *Proceedings of the ICMC,* 1997. [Available at http://cnmat.CNMAT.Berkeley.EDU/ ICMC97/papers-html/Latency.html]

[8] E. Brandt and R. Dannenberg. “Low-Latency music software using off-the-shelf operating systems.” *Proceedings of the ICMC,* 1998.

[9] R. Dannenberg and M. Goto. “Latency in Computer Audio Systems” *Proceedings of the ICMC,* 1997. [Available at http://raven.dartmouth.edu/~icma/array/spring97/articles.html]

[10] B. Bargen and P. Donnelly. *Inside DirectX.* Microsoft Press, *1998.*

[11] Microsoft. *DirectX Web Site*. URL: http://www.microsoft.com/directx/default.asp.

[12] P. Messik. *Maximum Midi.* Manning Publications, 1998.