Realtime Software Synthesis for Psychoacoustic Experiments

David S. Sullivan Jr., Stephan Moore, and Ichiro Fujinaga

Computer Music Department
The Peabody Institute of the Johns Hopkins University
One East Mount Vernon Place
Baltimore MD 21202 U.S.A.

sullivan@peabody.jhu.edu
stephan@peabody.jhu.edu
ich@peabody.jhu.edu

Abstract

New realtime sound synthesis software will allow psychoacoustic researchers to efficiently design and implement sophisticated test instruments that involve realtime interactivity with test subjects. Such interaction in psychoacoustic experiments has historically been constrained by the same limitations affecting realtime sound synthesis. A new model for sound synthesis, made possible through recent advances in computer hardware, supports software that synthesizes CD-quality audio in realtime and can base this synthesis on realtime user interactivity. We demonstrate this new software-based model in experimental settings, and discuss its nature and abilities.

Because of the large amount of information required to describe CD-quality sound, and the time-sensitive nature of sound production, synthesis software for personal computers has not been primarily designed to perform realtime sound synthesis. Continuous data have not been easy to incorporate into current or subsequent stimuli. These limitations have made multiple pieces of equipment necessary in most setups. In the new realtime software synthesis model, all input, sound synthesis, and output are controlled by one device.

Because of recent advances in processor power and standard memory configurations, realtime synthesis has become practical, and engineers are now designing for it. Three new examples of realtime synthesis software are SuperCollider, MSP, and Pd (Pure Data). These applications have the advantage of combining great processor power with a highly configurable user interface, and can accomplish complex manipulation of sound in realtime. The combination of powerful, relatively low-priced computers with this new software makes possible a degree of control and flexibility not previously available to most researchers.

While experimenters are given new tools with these packages, the presence of these instruments as software on a computer facilitates integration of standard experimental techniques. With these new tools, researchers will have access to a new degree of flexibility and precision, enabling them to create more subtle and replicable test instruments that can interact with subjects in realtime.

Overview

New realtime sound synthesis software will allow psychoacoustic researchers to efficiently design and implement sophisticated test instruments that involve realtime interactivity with test subjects. Such interaction in psychoacoustic experiments has historically been constrained by the same limitations affecting realtime sound synthesis. A new model for sound synthesis, made possible through recent advances in computer hardware, supports software that synthesizes CD-quality audio in realtime and can base this synthesis on realtime user interactivity. We demonstrate this new software-based model in experimental settings, and discuss its nature and abilities.

Because of the large amount of information required to describe CD-quality sound and the time-sensitive nature of sound production, synthesis software for personal computers has

not been primarily designed to perform realtime sound synthesis. Continuous data have not been easy to incorporate into current or subsequent stimuli. These limitations have made multiple pieces of equipment necessary in most setups. In the new realtime software synthesis model, all input, sound synthesis, and output are controlled by one device.

Several tools have been developed in the past to help manage audio and sound synthesis in a computer environment. One tool developed in an effort to greatly reduce necessary data flow is the Musical Instrument Digital Interface (MIDI). MIDI has several limitations, however, and because no sound wave is described in the MIDI data, different synthesizers will produce very different sounds when interpreting the same MIDI message, making it difficult to replicate studies done with MIDI.

Some software tools, such as cmusic or Cmix, allow for elaborate descriptions of the sounds produced, but are not designed for realtime synthesis based on continuous interaction. Another piece of software, Csound, does have limited ability for realtime manipulation, but was not specifically designed for that purpose. Because of recent advances in processor power and standard memory configurations, realtime synthesis has become practical, and engineers are now designing for it. Three new examples of realtime synthesis software are SuperCollider (McCartney 1996), Max (Puckette 1988) with MSP, and Pd (Pure Data) (Puckette 1997) with the Graphics Environment for Multimedia (GEM) (Danks 1997). These applications have the advantage of combining great processor power with a highly configurable user interface, and can accomplish complex manipulation of sound in realtime. Each allows exacting control over almost all aspects of output, such as waveshape and frequency, based on user interaction. This provides for unprecedented flexibility in the description of the stimuli, the selection of the stimuli altogether, and the degree of precision in the calibration of responses. Researchers are also able to custom-design an instrument, and hear the instrument in realtime as they create it, greatly streamlining the development process.

Controlling sound synthesis in realtime is of great benefit to both the implementation and flexibility of the resultant experimental instrument. These software packages can have a great and immediate impact largely because of their intuitive interfaces, and relatively gentle learning curves. Any previous attempts at manipulation of sound based on realtime information would have required a great amount of programming on the part of the experimenter, and would have been limited by the technology available. The complexity of the synthesis algorithm determines the precision with which the sound can be controlled. More complex algorithms can greatly increase the number of computations required to produce one second of sound. The combination of powerful, relatively low-priced computers with this new software makes possible a degree of control and flexibility not previously available to most researchers.

While experimenters are given new tools with these packages, the presence of these instruments as software on a computer facilitates integration of standard experimental techniques. For example, experiments could be inexpensively stored on writeable CDs, which allow for random access to data, and the software could draw on a vast number of possible stimuli given the current state of interaction between the instrument and the subject. It would also be easy to record subjects' responses, and to access them randomly. Subjects

asked to describe sound could easily review their profile of a previous sound, to allow for a more accurate relative description of a current sound, for example. Experimenters familiar with MIDI could make use of a software implementation of MIDI, providing a virtual synthesizer within the computer.

Other Software

There are several useful tools for software synthesis currently available that generate sound in realtime. Even Csound, the venerable descendent of the first software synthesis programs (Mathews 1969), has over time developed some realtime functionality. NetSound provides a method of describing sound in a manner that requires low bandwidth, and uses client software, such as Csound, to synthesize the sound locally. Common Music runs on several platforms, including NeXT, Macintosh, SGI, and SUN. It allows a researcher or composer to design a project in the Common Music environment, and then send it to a "target" for realization. Current targets include: MIDI, Csound, Common Lisp Music, Music Kit, Cmix, cmusic, M4C, RT, Mix, and Common Music Notation.

Cmix uses the MINC (MINC Is Not C) programming language to build instruction sets that Cmix uses to drive its sound synthesis. It has an open architecture, which has fostered the design of RTcmix, which adds realtime synthesis functionality to Cmix. Kyma is a software synthesis language that uses sound objects, or streams of samples, as its building blocks (Scaletti 1989). Roger Dannenberg helped create a series of tools that began in 1984 with Arctic, but required special hardware, and progressed through Canon, Fugue, and now Nyquist, which runs on UNIX, Windows NT, Windows 95, and the Mac OS (Dannenberg 1997a). Cecilia 2.0 is a productivity-minded environment that uses Csound as its synthesis engine, while not requiring the user to know how to program in Csound. Cecilia couples with Cybil to generate scores, and runs on the Macintosh, SGI, and Linux. ObjektSynth (BeOS) synthesizes sound in realtime, but only runs under the BeOS. JSyn allows Java programmers to use methods written in the C programming language to generate sound in realtime. Reality is a PC-based, realtime synthesis package capable of multi-timbral signal generation using several simultaneous synthesis techniques (Smith 1998).

These software synthesis tools are part of a strong trend away from stand-alone synthesis and effects modules towards multipurpose software for personal computers. The IRCAM Musical Workstation is hybrid, and is described by Miller Puckette as a system using an i860 chip, with its own operating system (Puckette 1991b). Dannenberg views this as impressive, but still not enough to stop the move toward synthesis environments designed for the personal computer (Dannenberg 1997b). We chose MSP, SuperCollider, and Pd with GEM, because of their accessibility to non-programmers, (particularly true of MSP and Pd/GEM), and because their design allows for realtime generation of sound on personal computers, based on realtime user interactivity.

MAX/MSP, SuperCollider, and Pd/GEM

MSP, which runs only on the Macintosh, is a new set of externals (objects not included in the original Max environment) that are designed to run within the Max environment. Max uses a graphical programming interface with a very gentle learning curve making it possible

to create complex designs without knowing any programming language. The MSP externals allow a user to influence the synthesized sound in realtime.

Our example of an instrument in MSP tests the affect of vibrato on subjects' ability to quickly determine pitch (Yoo et al. 1998). In this experiment, the subject is first presented with either a straight pitch or a vibrato pitch, followed by a straight pitch. The subjects are asked to determine if the second pitch is higher or lower than the first pitch. In the second part of the experiment, the order is reversed, with subjects hearing either a straight or vibrato tone for the second pitch. Max allowed for an intuitive user interface, which has visual feedback for responses entered by selecting one of four choices from the keyboard (Figure 1). Also, all of the stimuli were recordings of an acoustic violin stored as soundfiles on the hard disk. These files are accessed in random order during the course of the experiment.

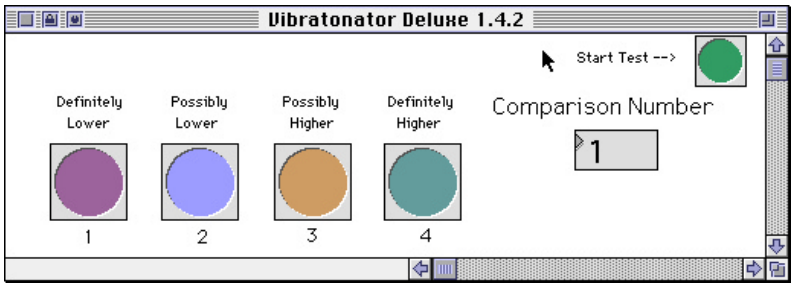

Figure 1. An example of a GUI in Max/MSP.

While Max and Pd both use a graphical programming environment, SuperCollider uses a more traditional, but very powerful text-based programming paradigm, and is designed to run only on the Macintosh. Its syntax is borrowed from the commonly-used programming languages SmallTalk and C, and may initially be more difficult to master for a researcher with little programming background, as compared to a programming environment that is completely graphical. However, SuperCollider implements an easily configurable graphical user interface that has intuitive controls such as buttons and sliders that can be assigned to any parameter of the synthesis. This makes it very simple to not only have the synthesis occur in realtime, but also to base that synthesis on a subject's interaction with the instrument. SuperCollider allows for the programming of synthesized instruments in a higher-level language than has been widely available previously (McCartney 1996).

In our trial FM matching experiment using SuperCollider, the subject is asked to manipulate two sliders to match the pitch and timbre of a frequency-modulated test tone (Figure 2). The sliders correspond to the carrier frequency and index of the FM tone produced. The subject may ask to hear the tone they are being asked to match, hear the tone that results from their slider settings, and change that tone in realtime as they listen and move the sliders. Additionally, subjects finalize their response, return to a previous test, advance to the next test, or end the testing session. The results of the test may then be recovered as text. The subject never has to interact with the program other than through the GUI.

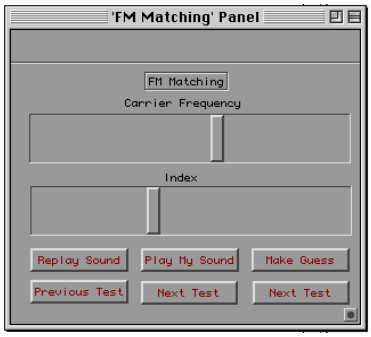

Figure 2. An example of a simple GUI in SuperCollider

Pd continues and updates Max's visual programming paradigm. Pd is available for the SGI IRIX and Windows/NT, and support for the integration of graphics with sound has been added in the form of the GEM. GEM also uses a visual programming language, and can operate within the Pd environment, processing video and images in realtime, and manipulating polygonal graphics. This allows for the easy integration of aural and visual stimuli into a test instrument.

Here we demonstrate a test instrument designed to determine if visual cues aid in memory of melodic patterns. Additionally, this instrument should give some indication of whether melodies based on the smallest interval that the subject can distinguish are more difficult to remember. In the first part of this experiment, the subject is asked if they are able to distinguish between two successive sine tones, based on pitch height (frequency). The pitches are played in pairs in various proximities in frequency to each other until the subject is unable to correctly distinguish between the pitches. The subject is then asked to follow a similar pattern in distinguishing between colors. Colors are shown in a simple box, with a gradient showing the range of colors displayed next to the box. In the second part of the experiment, subjects are asked to recall melodies with and without visual cues. Additionally, they are asked to recall melodies that are constructed of intervals that are relatively widely spaced, as well as melodies constructed using the subject's individual minimum threshold interval. The visual cues are also given in both wide and minimum threshold spacings.

In addition to describing the color of an object by determining its constituent red, green, and blue components, GEM uses a fourth variable, α, that describes the translucence of an object. Objects can range anywhere from transparent to opaque. It would be easy to design a similar experiment that would test subjects' tolerance of interfering noise in both the audio and visual components of this instrument.

Conclusion

Pd/GEM, SuperCollider, and MSP are some of the first examples of a new generation of software synthesis tools that take advantage of new processing power and flexibility. They

increase the user's ability to design instruments that can not only generate audio in realtime, but also react to realtime input, and in the case of Pd and GEM, as realtime video tools as well. The trend is expected to continue.

> Superscalar architectures are expected to compute 500 to 1,000 million instructions per second (MIPS) by the end of the decade. Software synthesis on superscalars will offer greater speed, flexibility, simplicity, and integration than today's systems based on digital signal processing (DSP) chips (Dannenberg 1997b, 83).

Because of these new tools, researchers will have a new degree of flexibility and precision, enabling them to create more subtle and replicable test instruments that can interact with subjects in realtime.

Bibliography

Danks, M. 1997. Real-time image and video processing in GEM. *Proceedings of the International Computer Music Conference.* 220–3.

Dannenberg, R. B. 1997a. Machine Tongues XIX: Nyquist, a language for composition and sound synthesis. *Computer Music Journal* 21 (3): 50–60.

Dannenberg, R. B., and N. Thompson. 1997b. Real-time software synthesis on superscalar architectures. *Computer Music Journal* 21 (3): 83–94.

Lansky, P. 1990. Cmix release notes and manuals. Department of Music, Princeton University. Princeton, New Jersey: Princeton University.

Lindemann, E., F. Dechelle, B. Smith, and M. Starkier. 1991. The architecture of the IRCAM Musical Workstation. *Computer Music Journal* 15 (3): 41–9.

Mathews, M. V. 1969. The technology of computer music. Cambridge, Massachusetts: MIT Press.

McCartney, J. 1996. SuperCollider: A realtime sound synthesis programming language. Austin, Texas.

Puckette, M. 1997. Pure Data. *Proceedings of the International Computer Music Conference.* 224–7.

Puckette, M. 1991a. Combining event and signal processing in the Max graphical programming environment. *Computer Music Journal* 15 (3): 68–77.

Puckette, M. 1991b. FTS: A real-time monitor for multiprocessor music synthesis. *Computer Music Journal* 15 (3): 58–67.

Puckette, M. 1988. The Patcher. *Proceedings of the International Computer Music Conference.*

Scaletti, C. 1989. The Kyma/Platypus computer music workstation. *Computer Music Journal* 13 (2): 23–38.

Smith, D. 1998. Real-time software synthesis. *Computer Music Journal* 22 (1): 5–6.

Vercoe, B., and D. Ellis. 1990. Real-time Csound: Software synthesis with sensing and control. *Proceedings of the International Computer Music Conference.* 209–11.

Yoo, L., D. S. Sullivan Jr., S. Moore, and I. Fujinaga. 1998. The effect of vibrato on response time in determining the pitch relationship of violin tones. *Proceedings of the International Conference of Music Perception and Cognition.*