Extension 5: Sound *Text by R. Luke DuBois*
Excerpt from Processing: a programming handbook for visual designers
and artists Casey Reas and Ben Fry

The history of music is, in many ways, the history of technology. From developments in the writing and transcription of music (notation) to the design of spaces for the performance of music (acoustics) to the creation of musical instruments, composers and musicians have availed themselves of advances in human understanding to perfect and advance their professions. Unsurprisingly, therefore, we find that in the machine age these same people found themselves first in line to take advantage of the new techniques and possibilities offered by electricity, telecommunications, and, in the last century, digital computers to leverage all of these systems to create new and expressive forms of sonic art. Indeed, the development of phonography (the ability to reproduce sound mechanically) has, by itself, had such a transformative effect on aural culture that it seems inconceivable now to step back to an age where sound could emanate only from its original source. The ability to create, manipulate, and losslessly reproduce sound by digital means is having, at the time of this writing, an equally revolutionary effect on how we listen. As a result, the artist today working with sound has not only a huge array of tools to work with, but also a medium exceptionally well suited to technological experimentation.

Composers adopted digital computers slowly as a creative tool because of their initial lack of real-time responsiveness and intuitive interface. Although the first documented use of the computer to make music occurred in 1951 on the CSIRAC machine in Sydney, Australia, the genesis of most foundational technology in computer music as we know it today came when Max Mathews, a researcher at Bell Labs in the United States, developed a piece of software for the IBM 704 mainframe called MUSIC. In 1957, the MUSIC program rendered a 17-second composition by Newmann Guttmann called "In the Silver Scale". Originally tasked with the development of human-comprehensible synthesized speech, Mathews developed a system for encoding and decoding sound waves digitally, as well as a system for designing and implementing digital audio processes computationally. His assumptions about these representational schemes are still largely in use and will be described later in this text. The advent of faster machines, computer music programming languages, and digital systems capable of real-time interactivity brought about a rapid transition from analog to computer technology for the creation and manipulation of sound, a process that by the 1990s was largely comprehensive.

Sound programmers (composers, sound artists, etc.) use computers for a variety of tasks in the creative process. Many artists use the computer as a tool for the algorithmic and computer-assisted composition of music that is then realized off-line. For Lejaren Hiller's *Illiac Suite* for string quartet (1957), the composer ran an algorithm on the computer to generate notated instructions for live musicians to read and perform, much like any other piece of notated music. This computational approach to composition dovetails nicely with the aesthetic trends of twentieth-century musical modernism, including the controversial notion of the composer as "researcher," best articulated by serialists such as Milton Babbitt and Pierre Boulez, the founder of IRCAM. This use of the computer to manipulate the symbolic language of music has proven indispensable to many artists, some of whom have successfully adopted techniques from computational research in artificial intelligence to attempt the modeling of preexisting musical styles and forms; for example, David Cope's *5000 works. . .* and Brad Garton's *Rough Raga Riffs*

use stochastic techniques from information theory such as Markov chains to simulate the music of J. S. Bach and the styles of Indian Carnatic sitar music, respectively. If music can be thought of as a set of informatics to describe an organization of sound, the synthesis and manipulation of sound itself is the second category in which artists can exploit the power of computational systems. The use of the computer as a producer of synthesized sound liberates the artist from preconceived notions of instrumental capabilities and allows her/him to focus directly on the *timbre* of the sonic artifact, leading to the trope that computers allow us to make any sound we can imagine. Composers such as Jean-Claude Risset (*The Bell Labs Catalogue*), Iannis Xenakis (*GENDYN3*), and Barry Truax (*Riverrun*), have seen the computer as a crucial tool in investigating sound itself for compositional possibilities, be they imitative of real instruments (Risset), or formal studies in the stochastic arrangements of synthesized sound masses (Xenakis) using techniques culminating in the principles of granular synthesis (Truax). The computer also offers extensive possibilities for the assembly and manipulation of preexisting sound along the musique concre`te model, though with all the alternatives a digital computer can offer. The compositional process of digital sampling, whether used in pop recordings (Brian Eno and David Byrne's *My Life in the Bush of Ghosts*, Public Enemy's *Fear of a Black Planet*) or conceptual compositions (John Oswald's *Plunderphonics*, Chris Bailey's *Ow, My Head*), is aided tremendously by the digital form sound can now take. Computers also enable the transcoding of an audio signal into representations that allow for radical reinvestigation, as in the timestretching works of Leif Inge (*9 Beet Stretch*, a 24-hour "stretching" of Beethoven's Ninth Symphony) and the time-lapse phonography of this text's author (*Messiah*, a 5-minute "compression" of Handel's *Messiah*).

Artists working with sound will often combine the two approaches, allowing for the creation of generative works of sound art where the underlying structural system, as well as the sound generation and delivery, are computationally determined. Artists such as Michael Schumacher, Stephen Vitiello, Carl Stone, and Richard James (the Aphex Twin) all use this approach. Most excitingly, computers offer immense possibilities as actors and interactive agents in sonic *performance*, allowing performers to integrate algorithmic accompaniment (George Lewis), hyperinstrument design (Laetitia Sonami, *Interface*), and digital effects processing (Pauline Oliveros, Mari Kimura) into their performance repertoire. Now that we've talked a bit about the potential for sonic arts on the computer, we'll investigate some of the specific underlying technologies that enable us to work with sound in the digital domain.

Digital representation of sound and music

Sound typically enters the computer from the outside world (and vice versa) according to the time-domain representation explained earlier. Before it is digitized, the acoustic pressure wave of sound is first converted into an electromagnetic wave of sound that is a direct analog of the acoustic wave. This electrical signal is then fed to a piece of computer hardware called an analog-to-digital converter (ADC or A/D), which then digitizes the sound by sampling the amplitude of the pressure wave at a regular interval and quantifying the pressure readings numerically, passing them upstream in small packets, or vectors, to the main processor, where they can be stored or processed. Similarly, vectors of digital samples can be sent downstream from the computer to a hardware device called a digital-to-analog converter (DAC or D/A) which takes the numeric values and uses them to construct a smoothed-out electromagnetic pressure wave that can then be fed to a speaker or other device for playback.

Synthesis

Digital audio systems typically perform a variety of tasks by running processes in *signal processing networks*. Each node in the network typically performs a simple task that either generates or processes an audio signal. Most software for generating and manipulating sound on the computer follows this paradigm, originally outlined by Max Mathews as the *unit generator* model of computer music, where a map or function graph of a signal processing chain is executed for every sample (or vector of samples) passing through the system. A simple algorithm for synthesizing sound with a computer could be implemented using this paradigm with only three unit generators, described as follows.

First, let's assume we have a unit generator that generates a repeating sound waveform and has a controllable parameter for the frequency at which it repeats. We refer to this piece of code as an oscillator. Most typical digital oscillators work by playing back small tables or arrays of PCM audio data that outlines a specific waveform. These *wavetables* can contain incredibly simple patterns (e.g., a sine or square wave) or complex patterns from the outside world (e.g., a professionally recorded segment of a piano playing a single note).

If we play our oscillator directly (i.e., set its frequency to an audible value and route it directly to the D/A) we will hear a constant tone as the wavetable repeats over and over again. In order to attain a more nuanced and articulate sound, we may want to vary the volume of the oscillator over time so that it remains silent until we want a sound to occur. The oscillator will then increase in volume so that we can hear it. When we want the sound to silence again, we fade the oscillator down. Rather than rewriting the oscillator itself to accommodate instructions for volume control, we could design a second unit generator that takes a list of time and amplitude instructions and uses those to generate a so-called *envelope*, or ramp that changes over time. Our *envelope generator* generates an audio signal in the range of 0 to 1, though the sound from it is never experienced directly. Our third unit generator simply multiplies, sample per sample, the output of our oscillator with the output of our envelope generator. This *amplifier* code allows us to use our envelope ramp to dynamically change the volume of the oscillator, allowing the sound to fade in and out as we like.

In a commercial synthesizer, further algorithms could be inserted into the signal network, for example a filter that could shape the frequency content of the oscillator before it gets to the amplifier. Many synthesis algorithms depend on more than one oscillator, either in parallel (e.g., additive synthesis, in which you create a rich sound by adding many simple waveforms) or through *modulation* (e.g., frequency modulation, where one oscillator modulates the pitch of another).

Sampling

Rather than using a small waveform in computer memory as an oscillator, we could use a longer piece of recorded audio stored as an AIFF or WAV file on our computer's hard disk. This *sample* could then be played back at varying rates, affecting its pitch. For example, playing back a sound at twice the speed at which it was recorded will result in its rising in pitch by an octave. Similarly, playing a sound at half speed will cause it to drop in pitch by an octave.

Most samplers (i.e., musical instruments based on playing back audio recordings as sound sources) work by assuming that a recording has a *base frequency* that, though

often linked to the real pitch of an instrument in the recording, is ultimately arbitrary and simply signifies the frequency at which the sampler will play back the recording at normal speed. For example, if we record a cellist playing a sound at 220 hertz (the musical note A below middle C in the Western scale), we would want that recording to play back normally when we ask our sampler to play us a sound at 220 hertz. If we ask our sampler for a sound at a different frequency, our sampler will divide the requested frequency by the base frequency and use that ratio to determine the playback speed of the sampler. For example, if we want to hear a 440 hertz sound from our cello sample, we play it back at double speed. If we want to hear a sound at middle C (261.62558 hertz), we play back our sample at 1.189207136 times the original speed.

Many samplers use recordings that have meta-data associated with them to help give the sampler algorithm information that it needs to play back the sound correctly. The base frequency is often one of these pieces of information, as are *loop points* within the recording that the sampler can safely use to make the sound repeat for longer than the length of the original recording. For example, an orchestral string sample loaded into a commercial sampler may last for only a few seconds, but a record producer or keyboard player may need the sound to last much longer; in this case, the recording is designed so that in the middle of the recording there is a region that can be safely repeated, ad infinitum if need be, to create a sense of a much longer recording.

Effects processing

In addition to serving as a generator of sound, computers are used increasingly as machines for *processing* audio. The field of digital audio processing (DAP) is one of the most extensive areas for research in both the academic computer music communities and the commercial music industry. Faster computing speeds and the increased standardization of digital audio processing systems has allowed most techniques for sound processing to happen in real time, either using software algorithms or audio DSP coprocessors such as the Digidesign TDM and T|C Electronics Powercore cards.
As we saw with audio representation, audio effects processing is typically done using either time- or frequency-domain algorithms that process a stream of audio vectors. An echo effect, for example, can be easily implemented by creating a buffer of sample memory to delay a sound and play it back later, mixing it in with the original. Extremely short delays (of one or two samples) can be used to implement digital filters, which attenuate or boost different frequency ranges in the sound. Slightly longer delays create resonation points called *comb filters* that form an important building block in simulating the short echoes in room reverberation. A variable-delay comb filter creates the resonant swooshing effect called *flanging*. Longer delays are used to create a variety of echo, reverberation, and looping systems, and can also be used to create *pitch shifters* (by varying the playback speed of a slightly delayed sound).

Audio analysis

A final important area of research, especially in interactive sound environments, is the derivation of information from audio analysis. Speech recognition is perhaps the most obvious application of this, and a variety of paradigms for recognizing speech exist today, largely divided between "trained" systems (which accept a wide vocabulary from a single user) and "untrained" systems (which attempt to understand a small set of words spoken by anyone). Many of the tools implemented in speech recognition systems can be abstracted to derive a wealth of information from virtually any sound source. Interactive systems that "listen" to an audio input typically use a few simple techniques to abstract a complex sound source into a control source that can be mapped

as a parameter in interaction design. For example, a plot of average amplitude of an audio signal over time can be used to modulate a variable continuously through a technique called *envelope following*. Similarly, a threshold of amplitude can be set to trigger an event when the sound reaches a certain level; this technique of *attack detection* ("attack" is a common term for the onset of a sound) can be used, for example, to create a visual action synchronized with percussive sounds coming into the computer. The technique of pitch tracking, which uses a variety of analysis techniques to attempt to discern the fundamental frequency of an input sound that is reasonably harmonic, is often used in interactive computer music to track a musician in real time, comparing her/his notes against a "score" in the computer's memory. This technology of score-following can be used to sequence interactive events in a computer program without having to rely on absolute timing information, allowing musicians to deviate from a strict tempo, improvise, or otherwise inject a more fluid musicianship into a performance.

A wide variety of timbral analysis tools also exist to transform an audio signal into data that can be mapped to computer-mediated interactive events. Simple algorithms such as *zero-crossing counters*, which tabulate the number of times a time-domain audio signal crosses from positive to negative polarity, can be used to derive the amount of noise in an audio signal. Fourier analysis can also be used to find, for example, the five loudest frequency components in a sound, allowing the sound to be examined for harmonicity or timbral brightness. Filter banks and envelope followers can be combined to split a sound into overlapping frequency ranges that can then be used to drive another process. This technique is used in a common piece of effects hardware called the *vocoder*, in which a harmonic signal (such as a synthesizer) has different frequency ranges boosted or attenuated by a noisy signal (usually speech). The effect is that of one sound "talking" through another sound; it is among a family of techniques called cross-synthesis.

Tools for sound programming

A wide variety of tools are available to the digital artist working with sound. Sound recording, editing, mixing, and playback are typically accomplished through digital sound editors and so-called digital audio workstation (DAW) environments. Sound editors range from open source and free software (MixViews, Audacity) to professionallevel two-track mastering programs (BIAS Software's Peak application, Digidesign's Sound Designer). These programs typically allow you to import and record sounds, edit them with clipboard functionality (copy, paste, etc.), and perform a variety of simple digital sound processing (DSP) tasks nondestructively on the sound file itself, such as signal normalization, fading edits, and sample-rate conversion. Often these programs will act as hosts for software plug-ins originally designed for working inside of DAW software.

Digital audio workstation suites offer a full range of multitrack recording, playback, processing, and mixing tools, allowing for the production of large-scale, highly layered projects. DAW software is now considered standard in the music recording and production industry, gradually replacing reel-to-reel tape as the medium for producing commercial recordings. The Avid/Digidesign Pro Tools software, considered the industry standard, allows for the recording and mixing of many tracks of audio in real time along a timeline roughly similar to that in a video NLE (nonlinear editing) environment . Automation curves can be drawn to specify different parameters (volume, pan) of these tracks, which contain clips of audio ("regions" or "soundbites") that can be assembled

and edited nondestructively. The Pro Tools system uses hardware-accelerated DSP cards to facilitate mixing as well as to host plug-ins that allow for the high-quality processing of audio tracks in real time. Other DAW software applications, such as Apple's Logic Audio, Mark of the Unicorn's Digital Performer, Steinberg's Nuendo, and Cakewalk's Sonar, perform many of the same tasks using software-only platforms. All of these platforms also support third-party audio plug-ins written in a variety of formats, such as Apple's AudioUnits (AU), Steinberg's Virtual Studio Technology (VST), or Microsoft's DirectX format. Most DAW programs also include extensive support for MIDI, allowing the package to control and sequence external synthesizers, samplers, and drum machines; as well as software plug-in "instruments" that run inside the DAW itself as sound generators.

Classic computer music "languages," most of which are derived from Max Mathews' MUSIC program, are still in wide use today. Some of these, such as CSound (developed by Barry Vercoe at MIT) have wide followings and are taught in computer music studios as standard tools for electroacoustic composition. The majority of these MUSIC-N programs use text files for input, though they are increasingly available with graphical editors for many tasks. Typically, two text files are used; the first contains a description of the sound to be generated using a specification language that defines one or more "instruments" made by combining simple unit generators. A second file contains the "score," a list of instructions specifying which instrument in the first file plays what event, when, for how long, and with what variable parameters. Most of these programs go beyond simple task-based synthesis and audio processing to facilitate algorithmic composition, often by building on top of a standard programming language; F. Richard Moore's CLM package, for example, is built on top of Common LISP. Some of these languages have been retrofitted in recent years to work in real time (as opposed to rendering a sound file to disk); Real-Time Cmix, for example, contains a C-style parser as well as support for connectivity from clients over network sockets and MIDI.

A number of computer music environments were begun with the premise of real-time interaction as a foundational principle of the system. The Max development environment for real-time media, first developed at IRCAM in the 1980s and currently developed by Cycling'74, is a visual programming system based on a control graph of "objects" that execute and pass messages to one another in real time. The MSP extensions to Max allow for the design of customizable synthesis and signal-processing systems, all of which run in real time. A variety of sibling languages to Max exist, including Pure Data (developed by the original author of Max, Miller Puckette) and jMax (a Java-based version of Max still maintained at IRCAM). James McCartney's SuperCollider program and Ge Wang and Perry Cook's ChucK software are both textual languages designed to execute real-time interactive sound algorithms.
Finally, standard computer languages have a variety of APIs to choose from when working with sound. Phil Burke's JSyn (Java Synthesis) provides a unit generator-based API for doing real-time sound synthesis and processing in Java. The CCRMA Synthesis ToolKit (STK) is a C++ library of routines aimed at low-level synthesizer design and, centered on physical modeling synthesis technology.

Ess, a sound library for Processing that has many features in common with the above-mentioned languages, is used in the examples for this text. Because of the overhead of doing real-time signal processing in the Java language, it will typically be more efficient to work in one of the other environments listed above if your needs require substantial real-time audio performance.

Conclusion

A wide variety of tools and techniques are available for working computationally with sound, due to the close integration of digital technology and sound creation over the last half-century. Whether your goal is to implement a complex reactive synthesis environment or simply to mix some audio recordings, software exists to help you fill your needs. Furthermore, sound-friendly visual development environments (such as Max) allow for you to create custom software from scratch. A basic understanding of the principles behind digital audio recording, manipulation, and synthesis can be indispensable in order to better translate your creative ideas into the sonic medium. As the tools improve and the discourse of multimedia becomes more interdisciplinary, sound will become even better integrated into digital arts education and practice.