DATA SONIFICATION AND SOUND VISUALIZATION

Sound can help us explore and analyze complex data sets in scientific computing. The authors describe a digital instrument for additive sound synthesis (Diass) and a program to visualize sounds in a virtual-reality environment (M4Cave).

Although most computational scientists routinely use visual imaging techniques to explore and analyze large data sets, they tend to be much less familiar with the use of sound. Yet sound signals carry significant amounts of information and can be used advantageously to increase the human/computer interface's bandwidth. The project described in this article focuses on scientific sonification—the faithful rendering of scientific data into sounds—and the visualization of sounds in a virtual-reality environment.

The project grew out of an attempt to apply the latest supercomputing technology to the process of music composition (see the "Computer-assisted music composition" sidebar). We developed its main tools, *Diass* (*d*igital *i*nstrument for *a*dditive *s*ound *s*ynthesis) and *M4Cave*, a program for the visualization of sound objects in a multimedia environment. We discuss Diass first; then we focus on M4Cave. Both are part of a comprehensive *music composition environment* that includes additional software for computer-assisted composition and automatic music notation. Figure 1 gives a schematic overview of the environment's various elements.

1521-9615/99/$10.00 © 1999 IEEE

HANS G. KAPER AND ELIZABETH WIEBEL
Argonne National Laboratory
SEVER TIPEI
University of Illinois

Past experiments

Scientific sonification is not a new concept. Sonar and Geiger counters are familiar examples of technologies that use sound to convey information about scientific data. But the sounds are simple, and the amount of information they carry is minimal. An early experiment demonstrated more than 20 years ago that sound can be successful in pattern recognition in chemistry.[1] The study related changes in chemical variables to changes in sound variables, which the test subjects could recognize after some practice. Recently, Sergei V. Pereverzev and his colleagues reported a successful application of scientific sonification in physics, where sounds revealed oscillations in superfluid helium that escaped detection by visual means.[2] These two studies involved sound only, however.

Several other experiments reported in the literature refer to situations where sounds were combined with visual images to further data analysis.[3–7] Most, if not all, of these attempts used MIDI-controlled synthesizer sounds, which have drastic limitations in the number and range of their control parameters.

Synthesis

Digital sound synthesis is a way to generate a stream of numbers representing an audio waveform's sampled values. To realize the sounds, we send these samples through a digital-to-analog converter (DAC), which converts the numbers to a continuously varying voltage that can be amplified and sent to a loudspeaker.

In software synthesis, the computer executes the audio wave's sample value calculations. The user implements the mathematical formula according to which sample values are to be calculated in a specific computer program, and the computer does all the calculations and sends the sample values to the DAC. Software synthesis contrasts in this way with hardware synthesis, where the process is hardwired in special circuitry. Software synthesis is the technique of choice if we wish to develop an instrument for data sonification. Hardware synthesis, although it has the advantage of high-speed operation, lacks the flexibility of software synthesis.

With software synthesis, we can realize any imaginable sound—provided we have the time to wait for the results. With a sampling rate of 44,100 samples per second, the time available per sample is only 20 microseconds—too short for real-time synthesis of reasonably complex sounds. For this reason, most of today's synthesis programs generate a sound file that is then played through a DAC. But data sonification in real time might become feasible on tomorrow's high-performance computing architectures.

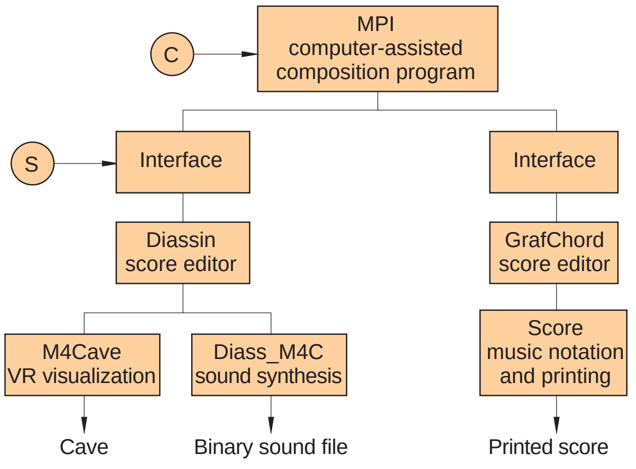

Figure 1. The music composition environment. C and S mark the data entry points for composition and sonification.

Computer-assisted music composition

The idea of using computers for music composition goes back to the 1950s, when Lejaren Hiller performed his experiments at the University of Illinois.[1] The premiere of his Quartet No. 4 for strings—*Illiac Suite*[2]—in May 1957 is generally regarded as the birth of computer music. Since then, computers have helped many composers to algorithmically synthesize new sounds and produce new pieces for acoustic as well as digital instruments.

Why would a composer need computer assistance when composing? A quick answer is that, as in many other areas, routine operations can be relegated to the machine. A more sophisticated reason is that the composer might rely on expert systems to write Bach-like chorales or imitate the mannerisms of Chopin or Rachmaninov. There are, however, more compelling reasons when composing is viewed as a speculative and experimental endeavor, rather than as an ability to manufacture pleasing sounds.[3]

Music is basically a dynamic event evolving in a multidimensional space; as such, it can be formalized.[4] The composer controls the evolution by supplying a set of rules and accepts the output as long as it is consistent with the program and input-data logic. If the set of rules allows for a certain degree of randomness, the output will differ every time a new "seed" is introduced. The same code and input data might thus produce an unlimited number of compositions, all belonging to the same equivalence class or *manifold composition*.[5] The members of a manifold composition are variants of the same piece; they share the same structure and are the result of the same process, but differ in the way specific events are arranged in time.

A nontraditional way of composing, the manifolds show how high-performance computing provides the composer with new means to try out compositional strategies or materials and hear the results in a reasonable amount of time.

References

1. L. Hiller and L. Isaacson, *Experimental Music*, McGraw-Hill, New York, 1959; reprinted by Greenwood Press, Westport, Conn., 1983.
2. L. Hiller, *Computer Music Retrospective*, Compact Disc WER 60128-50, WERGO Schallplatten GmbH, Mainz, Germany, 1989.
3. S. Tipei, "The Computer: A Composer's Collaborator," *Leonardo*, Vol. 22, No. 2, 1989, pp. 189–195.
4. I. Xenakis, *Formalized Music: Thought and Mathematics in Musical Composition*, revised ed., Pendragon Press, Stuyvesant, N.Y., 1992.
5. S. Tipei, "Manifold Compositions: A (Super)Computer-Assisted Composition Experiment in Progress," *Proc. Int'l Computer Music Conf.*, Int'l Computer Music Assoc., San Francisco, 1989, pp. 324–327.

Table 1. Static (S) and dynamic (D) control parameters in Diass, a digital instrument for additive sound synthesis.

Level	Description	Control parameter
Partial	Carrier (sine) wave	S: Starting time, duration, phase; D: Amplitude, frequency
	AM (tremolo) wave	S: Wave type, phase; D: Amplitude, frequency
	FM (vibrato) wave	S: Wave type, phase; D: Amplitude, frequency
	Amplitude transients	S: Max size; D: Shape
	Amplitude transient rate	S: Max rate; D: Rate shape
	Frequency transients	S: Max size; D: Shape
	Frequency transient rate	S: Max rate; D: Rate shape
Sound	Timbre	D: Partial-to-sound relation
	Loudness	S: Max size; D: Shape
	Glissando	S: Max size; D: Shape
	Crescendo/Decrescendo	S: Max size; D: Shape
	Localization	D: Panning
	Reverberation	S: Duration, decay rate, mix
	Hall	S: Hall size, reflection coefficient

Diass

Scientific sonification requires a flexible and powerful instrument to optimally convey information through sound. Diass, developed jointly by the authors and their students at the University of Illinois and Argonne National Laboratory, is such an instrument. It functions within the framework of M4C, the synthesis language James Beauchamp and his associates developed, and it uses additive synthesis to compose sounds of any complexity and with any desired acoustic effect.[8] Synthesis languages such as M4C are designed around the notion that the user creates an instrument together with a score that references the instrument. The synthesis program reads the instrument definition, feeds it the data from the score file, and computes the final audio signal, which is then written to a sound file for later playback.

The M4C synthesis language is embedded in the C language. As part of our project, we redesigned Diass and relevant parts of M4C for a distributed-memory environment. The parallel implementation uses the standard MPI message-passing library.

Like all additive-synthesis instruments, Diass creates sounds through a summation of simple sine waves. The basic formula is

$$S(t) = \sum_i P_i(t) = \sum_i a_i(t)\sin(2\pi f_i(t)t + \phi_i(t)).$$

The individual sine waves that make up a sound are commonly designated as the sound's *partials*, hence the symbol *P*. The sum extends over all partials active at time t; a_i is the amplitude, f_i the frequency, and ϕ_i the phase of the ith partial. These variables can be modulated periodically or otherwise; the modulations evolve on a slow timescale, typically on the order of a sound's duration. Phase modulation is barely distinguishable from frequency modulation, particularly in the case of time-varying frequency spectra, and it is not implemented in Diass.

Audible frequencies range roughly from 20 to 20,000 Hz, although in practice, the upper limit is one-half of the sampling frequency (this is called the *Nyquist criterion*).

The partials in a sound need not be in any harmonic relationship (that is, f_i need not be a multiple of some fundamental frequency f_0); they also don't need to share any other property. A sound's definition is purely operational. What distinguishes one sound from another is that certain operations are defined at the level of a sound and affect all the partials that constitute that sound.

A partial's evolution is subject to many other controls besides amplitude and frequency modulation. Moreover, these controls can affect a single partial or all the partials in a sound. For example, reverberation, which represents the combined effects of a hall's size and acoustic characteristics, affects all the partials in a sound simultaneously, although not necessarily in the same way. Furthermore, if a random element is present, it must be applied at the level of a sound; otherwise, a complex wave is perceived as a collection of independent sine waves instead of a single sound. Thus, all partials in a sound must access the same random number sequence, and the controls of any partial that changes its allegiance and moves from one sound to another must be adjusted accordingly.

Table 1 lists the control parameters that can be applied in Diass. Some, such as starting time and duration, do not change for a sound's duration; they are static and a single value determines them. Others are dynamic; an envelope—a normalized function consisting of linear and exponential segments—and a maximum size control their evolution. Not all control parameters are totally independent; some occur only in certain combinations, and some are designed to reinforce others.

The control parameters give Diass its flexibility and make it a suitable instrument for data sonification. However, because control parameters act at the partial and sound levels (or even at the level of a collection of sounds), the computational complexity is significant.

The score

Input for Diass consists of a raw score file detailing the controls. The raw score file is transformed into a score file for the instrument—a collection of *instrument cards* (I-cards), one for each partial, which M4C feeds to the instrument. The transformation is accomplished in several steps.

Among the controls are certain global operations (*macros*) defined at the level of a sound. In a first pass, these macros expand into controls for the individual partials. The next step consists of applying the loudness routines. These routines operate at the sound level and ensure that the sounds have the desired loudness. The final step consists of applying the *anticlip* routines. For various historical and technical reasons, sound samples are stored as 16-bit integers. The anticlip routines guarantee that none of the sample values the instrument produces from the score file exceed 16 bits. Because loudness and anticlip play a significant role in sonification, we discuss them in more detail.

Loudness. Loudness perception is subjective. Although a sound's perceived loudness relates to the amplitudes of its constituent partials, the relation is nonlinear and depends on the frequencies of the partials. At the most elementary level, pure sinusoidal waves of low or high frequencies require a higher energy flow and therefore a larger amplitude to achieve the same loudness level as similar waves at mid-range frequencies. When waves of different frequencies are superimposed to form a sound, the situation becomes still more complicated. The sum of two tones of the same frequency produced by two identical instruments played simultaneously is not perceived as twice as loud as the tone a single instrument produces.

An algorithm for data sonification must reflect these subjective experiences. For example, when we sonify two degrees of freedom, mapping one (say, x_1) to amplitude and the other (say, x_2) to frequency, then we should perceive equal loudness levels when x_1 has the same value, irrespective of the values of x_2. Also, when the variable x_1 increases or decreases, we should perceive a proportional increase or decrease in the loudness level (see the "Loudness" sidebar).

Loudness

Sound is transmitted through sound waves—periodic pressure variations that cause the eardrums to vibrate. But loudness perception has as much to do with the amount of energy that the sound wave carries as with the processing of this energy that takes place in the ear and the brain once the sound wave has hit the eardrums. The latter is a much more subjective part of the experience. The algorithms underlying the loudness routines of Diass therefore incorporate both formal definitions and relevant results of psychoacoustic research. We summarize the algorithm's most relevant elements.[1,2]

The definition of (perceived) loudness begins with the consideration of the energy carried by the sound wave. The intensity I of a pure tone (sinusoidal sound) is expressed in terms of its average pressure variation Δp (measured in newton/m^2),

$$I = 20 \times \log_{10} (\Delta p/\Delta p_0).$$

Δp_0 is a reference value, usually identified with a traveling wave of 1,000 Hz at the threshold of hearing; $\Delta p_0 = 2 \times 10^{-5}$ newton/m^2. The unit of I is the decibel (dB).

Because of the way the inner ear processes acoustical vibrations, the sensation of loudness is strongly frequency dependent. For instance, although an intensity of 50 dB at 1,000 Hz is considered *soft piano*, the same intensity is barely audible at 60 Hz. In other words, to produce a given loudness sensation at low frequencies, a much higher intensity (energy flow) is needed than at 1,000 Hz. The intensity I is therefore not a good measure of loudness if different frequencies are involved.

In the 1930s, Harvey Fletcher and W.A. Munson[3] performed a series of loudness-matching experiments, from which they derived a set of *curves of equal loudness*. These are curves in the frequency (f) versus intensity (I) plane; points on the same curve represent single continuously sounding pure tones that are perceived as being equally loud. They are similar to those

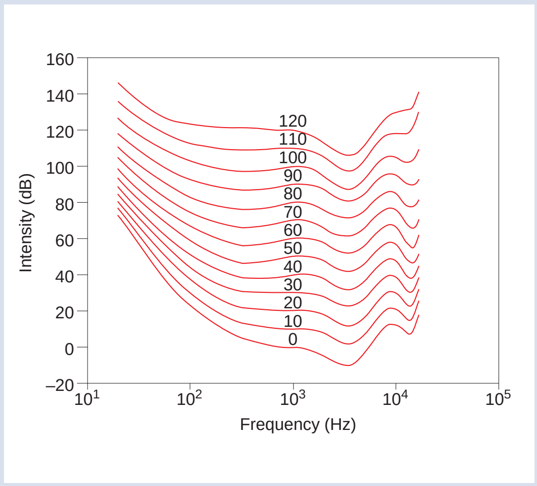

Figure A. Curves of equal loudness (marked in phons) in the frequency versus intensity plane.

recommended by the International Organization for Standardization[4] and are presented in Figure A. The curves show clearly that, to be perceived as equally loud, very low and very high frequencies require much higher intensities (energy) than frequencies in the middle range of the spectrum of audible sounds.

The (physical) loudness level L_p of a Fletcher-Munson curve is identified with the value of I at the reference frequency of 1,000 Hz. The unit of L_p is the phon. The Fletcher-Munson curves range from a loudness level of 0 to 120 phons over a frequency that ranges from 25 to 16,000 Hz.

L_p still does not measure loudness in an absolute manner: a tone whose L_p is twice as large does not sound twice as loud. Following Thomas D. Rossing,[5] we define the (subjective) loudness level L_s in terms of L_p by the formula $L_s = 2^{(L_p - 40)/10}$. The unit of L_s is a sone. To be effective, loudness scaling must be done on the basis of sones.

The loudness of a sound that is composed of several partials depends on how well the frequencies of the partials are separated. Each frequency f is associated with a *critical band*, whose width Δf is approximately given by the expression[6]

$$\Delta f \approx 25 + 75(1 + 1.4(f/1000)^2)^{0.69}.$$

Intensities within a critical band are added, and the loudness of a critical band can again be read from the Fletcher-Munson tables. If the frequencies of a sound's constituent partials are spread over several critical bands, its loudness is computed in accordance with a formula revised by Rossing,[5]

$$_s = L_{s,m} + 0.3\sum L_{s,i}$$

Here, $L_{s,m}$ is the loudness of the loudest critical band, and the sum extends over the remaining bands.

The loudness routines in Diass use critical band information and a table derived from the Fletcher-Munson curves to create complex sounds of specified loudness.

The waveform of Figure B, which was produced with Diass, illustrates the concept of equal loudness across the frequency spectrum and for different timbres. The waveform represents five sound clusters, each lasting 5.5 seconds (except the fourth, which lasts 5.7 seconds). The clusters, although of widely different structure,

The loudness routines in Diass incorporate the relevant results of psychoacoustic research[11] and give the user full control over a sound's perceived loudness. They also scale each partial so that each sample value fits in a 16-bit register.

Anticlip. When several sounds coexist and their waveforms are added, sample values might exceed 16 bits (overflow), even when the individual waveforms stay within the 16-bit limit. Overflow gives rise to *clipping*—a popping noise—when the sound file is played. The anticlip routines in Diass check the score for potential overflow and rescale the sounds as necessary, while preserving the ratio of perceived loudness levels. Thus it is possible to produce an entire sound file in a single run from the score file, even when the sounds cover a wide dynamic range.

To appreciate the difficulty inherent in scaling, consider a sound cluster consisting of numerous complex sounds, all very loud and resulting in clipping, followed by a barely audible sound with only two or three partials. If the cluster's amplitude is decreased to fit the register capacity, and that of the soft tiny sound following it is scaled proportionally, the latter disappears under system noise. However, if only the loud cluster is scaled, the relationship between the two sound events is completely distorted. Many times in the past, individual sounds or groups of sounds were generated separately and then merged with analog equipment or an additional digital mixer. The loudness and anticlip routines in Diass deal with this problem by adjusting both loud and soft sounds so that their perceived loudness matches the desired relationship and so no clipping occurs.

The editor

Features such as the loudness routines make Diass a fine-tuned, flexible, and precise instrument for data sonification. Of course, they require the specification of significant amounts of input data.

can be perceived at the same loudness level (2^5 sones).

The first sound cluster has 24 sounds. The fundamental frequencies of the sounds range from 40 to 5,000 Hz. Each sound is harmonically tuned; that is, it is made up of a fundamental and all its harmonics (partials whose frequencies are integer multiples of the fundamental frequency). The frequencies are limited to one-half of the sampling rate (the Nyquist criterion); hence, the number of partials in this cluster is 754 (at a sampling rate of 22,050 Hz). The second sound cluster has five sounds, harmonically tuned, with fundamental frequencies ranging from 40 to 4,000 Hz; the number of partials is 113. The third, fourth, and fifth cluster have 15, 1, and 10 sounds, with 453, 60, and 250 partials. All partials are assigned the same amplitude, which presents the worst-case scenario when we try to obtain the same perceived loudness for all clusters.

References

1. J.G. Roederer, *The Physics and Psychophysics of Music*, 3rd ed., Springer-Verlag, Berlin, 1995.
2. T.D. Rossing, *The Science of Sound*, Addison-Wesley, Reading, Mass., 1990.
3. H. Fletcher and W.A. Munson, "Loudness: Its Definition, Measurement, and Calculation," *J. Acoustical Soc. Am.*, Vol. 5, 1933, p. 82.
4. "Acoustics–Normal Equal-Loudness Level Contours," Publication 226, Int'l Organization for Standardization, Geneva, 1987.
5. T.D. Rossing, *The Science of Sound*, Addison-Wesley, Reading, Mass., 1990.
6. E. Zwicker and E. Terhardt, "Analytical Expressions for Critical-Band Rate and Critical Bandwidth as a Function of Frequency," *J. Acoustical Soc. Am.*, Vol. 68, 1980, p. 5.

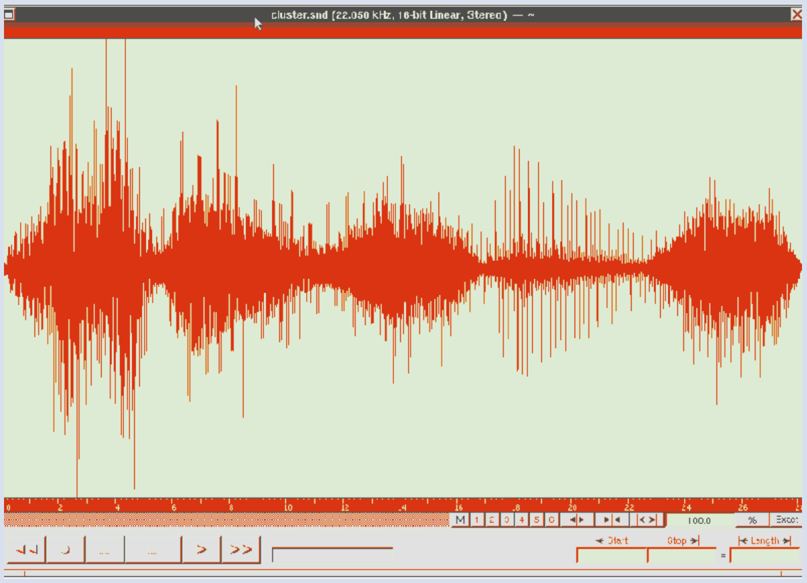

Figure B. A waveform of five sound clusters of equal perceived loudness.

The editor in Diass facilitates this process. It comes in slow and fast versions.

In the slow version, data are entered one at a time, either in response to questions from a menu or through a graphical user interface. The process lets the user build sounds step by step, experiment, and fine-tune the instrument. The slow version is suitable for sound composition and for designing prototype experiments in sonification. The fast version uses the same code but reads the responses to the menu questions from a script. Researchers use this version for sonification experiments.

Computing requirements

The sound-synthesis software is computationally intensive (see the "Computational complexity" sidebar). We parallelized Diass for the the workstation environment and on IBM's Scalable PowerParallel (SP) system. Parallelism is implemented at the sound level to minimize communication among the processors and enable all partials of a sound to access the same random number sequence. Parallel mode employs at least four processors—one to distribute the tasks and supervise the entire run (the *master* processor), a second to mix the results (the *mixer*), and at least two *slave* nodes to compute the samples one sound at a time. Sounds are computed in their starting-time order, irrespective of their duration or complexity. (A smart load-balancing algorithm would take into account the duration of the various sounds and the number of their partials.)

Performance depends greatly on the complexity of the sounds—that is, on the number of partials per sound and the number of active controls for each partial. Typically, the time to generate a two-channel sound file for a 2-min., 26-sec. musical composition with 236 sounds and 4,939 partials ranges from almost two hours on four processors to about 10 minutes on 34 SP processors. Figure 2 gives some indication of the

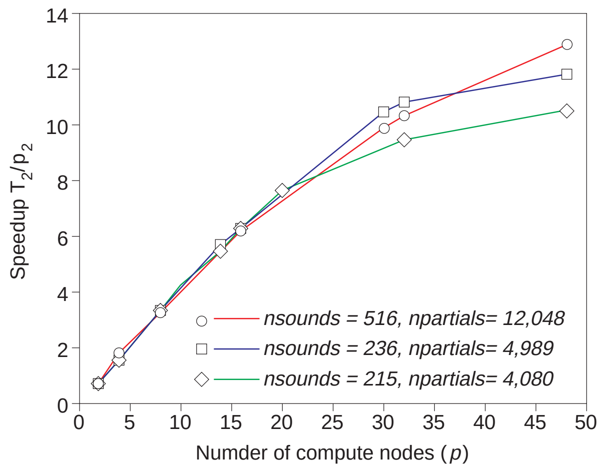

Figure 2. Timing results for the digital instrument for additive sound synthesis on an IBM Scalable PowerParallel system.

speedups we observe in a multiprocessing environment. The three graphs correspond to three variants of the same 2-min., 26-sec. piece with different complexities. The time T_p refers to a computation on $p + 2$ processors (p slaves); all times are approximate, because they were extracted from data given by LoadLeveler, a not-too-sophisticated SP timing instrument. Speedup is relative to the performance on four processors (two compute nodes). We observe the typical linear speedup until saturation sets in. The more complex the piece (the more partials), the later saturation sets in.

With a sampling rate of 44,100 samples per second and a two-channel output, a sound file occupies 176 Kbytes per second of sound, so the sound file for the 2-min., 26-sec. musical com-

Computational complexity

To get some idea of the computational complexity, consider the following simple scenario, where we wish to sonify time-varying data representing the values of two primary and several secondary observables measured over the course of an experiment. A natural choice is to map the primary observables onto loudness and frequency and to use amplitude and frequency modulation to monitor the secondary observables. We calculate the sample values of the sound wave S from this expression:

$$S(t) = a(t)\sin(2\pi f(t)t + \phi). \qquad (1)$$

The frequency f represents three degrees of freedom: the carrier frequency f^C, and the amplitude a^{FM} and frequency f^{FM} of the modulating wave,

$$f(t) = f^C(t) + a^{FM}(t)\sin(2\pi f^{FM}t + \phi^{FM}). \qquad (2)$$

The carrier frequency is identified with a primary observable; each of the remaining two degrees of freedom can be identified with a secondary observable.

The amplitude a in Equation 1 is similarly modulated,

$$a(t) = a^C(t) + a^{AM}(t)\sin(2\pi f^{AM}t + \phi^{AM}). \qquad (3)$$

We compute the carrier amplitude a^C from the observed loudness, which is identified with one of the (primary) observables, so its value is given. The amplitude a^{AM} and frequency f^{AM} of the modulation represent two more degrees of freedom, which can be identified with two other secondary observables. In total, we have two primary and four secondary variables (not counting the phases, which we assume to be static).

The amplitude $a^C(t)$ must be computed such that $S(t)$ has the perceived loudness level $L_s(t)$,

$$L_s(S(t)) = L_s(t). \qquad (4)$$

The loudness function L_s is a nonlinear function of the amplitude and frequency of the partial (sound). Its computation is done in the loudness routines of Diass and involves a significant number of operations, including table lookups (see the "Loudness" sidebar).

On the basis of these formulas, we can roughly estimate the number of operations (additions, multiplications, function evaluations—sine, exponential, or logarithm, and table lookups) required for the computation of a single sample value. The contribution that is most difficult to estimate is the computation of the carrier amplitude from the loudness; the data in Table A represent the minimum number of operations. Ignoring phases and so forth, we find at least 24 operations. Hence, at the standard rate of 44,100 samples per second, we need to perform more than 1.1 million operations per second.

The simultaneous sonification of more observables is obviously much more complicated; the complications grow exponentially. A careful estimate of the computational complexity requires an analysis of the anticlip routines, which is beyond this article's scope.

Table A. The number of operations per partial per sample value.

Equation	Adds	Mults	Function evaluation	Table lookups
(1)	1	3	1	—
(2)	2	3	1	—
(3)	2	3	1	—
(4)	1	3	2	1
Total	6	12	5	1

position takes close to 25.8 Mbytes of memory.

Our experiments to date

We have focused much of our work so far on the development of Diass.[12,13] In addition, we have used Diass for two preliminary experiments in scientific sonification: one in chemistry and the other in materials science.

The first experiment used data from Jeff Tilson, a computational chemist at Argonne National Laboratory, who studied the binding of a carbon atom to a protonated thiophene molecule. The data represented the difference in the energy levels before and after the binding reaction at 128^3 mesh points of a regular computational grid in space. Because the data were static, we arbitrarily identified time with one of the spatial coordinates and sonified data in planes parallel to this axis. We kept the time to traverse a plane over its full length at 30 seconds. In a typical experiment, we assigned a sound to every other point in the vertical direction, distributing the frequencies regularly over a specified frequency range. We then used the data in the horizontal direction to generate amplitude envelopes for each of the sounds. Thus, a sound would become louder or softer as the data increased or decreased, and the evolution of the loudness distribution within the ensemble of 64 sounds indicated the distribution of the energy difference before and after the reaction in space. The sound parameters chosen for the data representation varied from one experiment to another.

The second experiment involved data from a numerical simulation in materials science. The scientists were interested in patterns of motion of magnetic flux vortices through a superconducting medium. The medium was represented by 384×256 mesh points in a rectangular domain. As an external force drove the vortices from left to right across the domain, the vortices repelled each other but were attracted by regularly or randomly distributed defects in the material. In this experiment, frequency and frequency modulation (vibrato) represented movement in the plane, and changes in loudness were connected to changes in a vortex's speed. A traveling window of constant width captured the motion of a number of vortices simultaneously.

These investigations are ongoing, and we have not subjected the results to rigorous statistical evaluation. They merely serve to demonstrate the capabilities of Diass and explore various mappings from the degrees of freedom in the data to the parameters controlling the sound synthesis. You can hear samples at *http://mcs.anl.gov/appliedmath/Sonification/index.html*.

What we learned from Diass

Our general conclusions are that the sounds produced in each experiment conveyed information about the data's qualitative nature, and Diass is a flexible and sophisticated tool capable of rendering subtle variations in the data.

Changes in some control variables, such as time, frequency, and amplitude, are immediately recognizable. Changes in the combination of partials in a sound, identifiable through its timbre, can be recognized with some practice. Modifiers such as reverberation, amplitude modulation (*tremolo*), and frequency modulation (*vibrato*) enhance some effects. In some instances, a modifier might lump two, three, or more degrees of freedom together, such as hall size, duration, and acoustic properties in the case of reverberation. Through the proper manipulation of reverberation, loudness, and spectrum, we can create the illusion of sounds being produced at arbitrary locations in a room, even with only two speakers.

Most auditory processes are based on the recognition of time patterns.

Like the eye, the ear has a very high power of discrimination. Even a coarse grid, such as the temperate tuning used in Western music, includes about 100 identifiable discrete steps over the frequency range encompassed by a piano keyboard. Contemporary music, as well as some non-Western traditional music, successfully uses smaller increments of a quarter tone or less for a total of some 200 or more identifiable steps in the audible range. Equally discriminating power is available in the realm of timbre.

Sound is an obvious means to identify regularities in the time domain, both at the microlevel and on a larger scale, and to bring out transitions between random states and periodic happenings. Most auditory processes are based on the recognition of time patterns (periodic repetitions giving birth to pitch, amplitude, or frequency modulation; spectral consistency creating stable timbres in a complex sound; and so on), and the ear is highly attuned to detect such regularities.

Most conceptual problems in scientific sonification are related to finding suitable mappings between the space of data and the space of sounds. Common sense dictates letting the two domains share the coordinates of physical space and time (if these are relevant) and translating other degrees of freedom in the data into separate sound parameters. However, experimenting with alternative mappings might be advantageous. Sonification software must be flexible enough for a user to pair different sets of parameters into the two domains.

Any mapping between data and sound parameters must allow for redundancies to enable the exploration of data at different levels of complexity. Similar to visualization software, sonification software must have utilities for zooming, modifying the audio palette, switching between visual and aural representation of parameters, defining time loops, slowing down or speeding up, and so forth.

One limitation concerns the sheer volume of data in scientific sonification.

Our experiments also showed that Diass, at least in its present form, has its limitations. One limitation concerns the sheer volume of data in scientific sonification. The composition of a musical piece (the original intent behind Diass) typically entails the handling of a few thousand sounds, each with a dozen or so partials. The number of data points in the computational chemistry experiment ran into the millions, a difference of several orders of magnitude. By the same token, a typical amplitude envelope for a partial or sound in a musical composition involves 10 or even fewer segments, but both experiments required envelopes with well over 100 such segments. Another difficulty was that both experiments required sounds to be accurately located in space. Although panning is very effective in pinpointing the source on a horizontal line, suggesting the height of a sound is a major challenge. We hope that additions to the software as well as a contemplated eight-speaker system will help us get closer to a realistic 3D representation of sounds. Finally, to become an effective tool for sonification, Diass must operate in real time. We are addressing all three concerns in a C++ version of Diass under development.

Sound visualization in a VR environment

The notion of sound visualization might at first sight seem incongruous in the context of data sonification. However, as several researchers have recognized, a sound's structure is difficult to detect without proper training, and any means of aiding detection will enhance data sonification's value. Visualizing sounds is one of these means. In this project, in addition to developing Diass, we also focused on the visualization of sounds in the Cave (Cave Automatic Virtual Environment), a room-size virtual-reality environment (*www.evl.uic.edu/pape/CAVE/prog/CAVEGuide.html*), and on the ImmersaDesk, a 2D version.

M4Cave—a visualization tool

The software collectively known as M4Cave takes a score file from Diass and renders the sounds represented by the score as visual images in a Cave or ImmersaDesk. The images are computed on the fly and are made to correspond exactly to the sounds you would hear through a one-to-one mapping between the control parameters and visual attributes. The code, which is written in C++, uses OpenGL for visualizing objects.

Graphical representations

Currently, M4Cave can represent sounds as a collection of spheres (or cubes or polyhedra), as a cloud of confetti-like particles, or as a collection of planes.

The sphere representation is the most developed and incorporates more of a sound's parameters into the visualization. Sounds are visualized as stacks of spheres, each sphere corresponding to a partial in the sound. A sphere's position along the vertical axis is determined by the partial's frequency, and its size is proportional to the amplitude. A sound's position in the stereo field determines sphere placement in the room. The visual objects rotate or pulse when tremolo or vibrato is applied, and their color varies when reverberation is present. An optional grid in the background shows the octaves divided into 12 equal increments. Figure 3—taken from our Web site (*http://mcs.anl.gov/applied_math/Sonification/index.html*)—shows a visualization of nine sounds with different numbers of partials.

The plane and cloud representations were designed more on the basis of artistic considerations. (Remember that the visualization's purpose is to aid sound perception.) The cloud representation's strength is in showing tremolo and vibrato in the sound. The plane representation is unique in that it limits the visualization to only one partial (usually the fundamental) of each sound. The various representations can be combined, and a menu can vary the mappings chosen for each representation.

Preliminary findings

We used M4Cave to explore various mappings from the sound domain to the visual domain.

Besides the obvious short score files to test the implementation of these mappings, we used Diass-generated score files of various musical compositions, notably Sever Tipei's *A.N.L.-folds*.[14] A.N.L.-folds is an example of the *manifold composition* described in the "Computer-assisted music composition" sidebar. Each member of A.N.L.-folds lasts exactly 2 min., 26 sec. and comprises between 200 and 500 sounds of medium to great complexity. Figure 3 was taken from a run of one of these A.N.L.-folds.

The combination of visual images and sounds provides an extremely powerful tool for uncovering complicated structures. Sometimes, the sounds reveal features that are hidden to the eye; at other times, the visual images illuminate features that are not easily detectable in the sound. The two modes of perception reinforce each other, and both improve with practice.

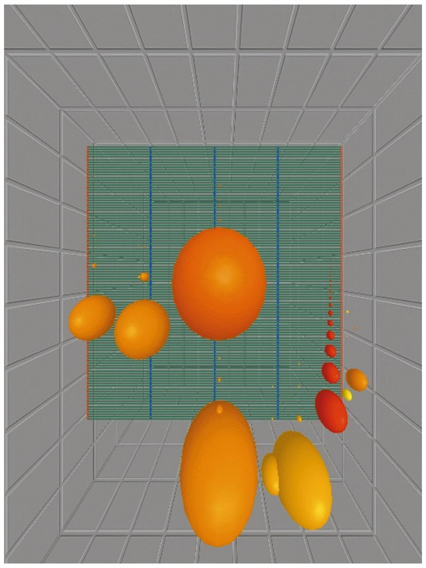

Figure 3. A visualization of nine sounds. (Picture taken from a Cave simulator.)

This project is unusual in several respects. It is somewhat speculative, in the sense that we don't have much experience with the use of sound in scientific computing. This is mostly why the involvement of someone expert in the intricacies of the sound world is critical for its success. In our case, the expertise comes from the realm of music composition.

When do we declare success? Can we reasonably expect that sonification will evolve to the same level of usefulness as visualization for computational science? The answers to these questions depend on your expectations. Our culture is visually oriented, and as a society, we watch rather than listen. Contemporary musical culture is often reduced to entertainment genres that use a simple vocabulary—no small impediment to discovering the world of sound's potential benefits. But given unusual and unexpected sonorities, we might yet discover that we have not lost the ability to listen.

When we engage in this type of research, it is easy to get swept up by unreasonable expectations, looking for the "killer application." But the killer application is a phantom not worth pursuing. What we can offer is a systematic investigation of a new tool's potential. If it helps us understand some computational data sets a little better, or if it enables us to explore these data sets more easily and in more detail, we have good reason to claim success. If the project adds to our understanding of aural semiotics, we have even more reason to claim success. And if none of these successes materializes, we can still claim that the people involved, both scientists and musicians, gained by becoming more familiar with each other's work and ways of thinking. Such a rapprochement has, in fact, already occurred and led to a new course entitled *Music, Science, and Technology* at the University of Illinois, Urbana-Champaign, where some of the issues presented here are being discussed in a formal educational context.

Acknowledgments

This work was partially supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, US Department of Energy, under Contract W-31-109-Eng-38. This project is a joint collaboration between Argonne National Laboratory and the University of Illinois at Urbana-Champaign.

References

1. E. Yeung, "Pattern Recognition by Audio Representation of Multivariate Analytical Data," *Analytical Chemistry*, Vol. 52, 1980, pp. 1120–1123.
2. S.V. Pereverzev et al., "Quantum Oscillations between Two Weakly Coupled Reservoirs of Superfluid ^{3}He," *Nature,* Vol. 388, 31 July 1997, pp. 449–451.
3. S. Bly, *Sound and Computer Information Presentation*, PhD thesis, Univ. of California, Davis, Calif., 1982.
4. J. Mezrich, S. Frysinger, and R. Slivjanovski, "Dynamic Representation of Multivariate Time Series Data," *J. Am. Statistical Assoc.,* Vol. 79, 1984, pp. 34–40.
5. S. Smith and M. Williams, *The Use of Sound in an Exploratory Visualization Experiment*, Tech. Report R-89-002, Computer Science Dept., Univ. of Massachusetts, Lowell, Mass., 1989.
6. E. Wenzel et al., "A System for Three-Dimensional Acoustic 'Visualization' in a Virtual Environment Workstation," *Proc. Visualization 90: First IEEE Conf. Visualization*, IEEE Computer Society Press, Los Alamitos, Calif., 1990, pp. 329–337.
7. G. Kramer, ed., *Auditory Display: Sonification, Audification, and Auditory Interfaces*, Addison-Wesley, Reading, Mass., 1994.
8. J. Beauchamp, *Music 4C Introduction*, Univ. of Illinois, Urbana-Champaign, Ill., Computer Music Project, School of Music, 1993; http://cmp-rs.music.uiuc.edu/cmp/software/m4c.html.
9. C. Roads, *The Computer Music Tutorial*, MIT Press, Cambridge, Mass. 1996.
10. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1994; www.mcs.anl.gov/mpi/index.html.
11. H.G. Kaper, D. Ralley, and S. Tipei, "Perceived Equal Loudness of Complex Tones: A Software Implementation for Computer Music Composition," *Proc. Int'l Conf. Music Perception and Cognition*, Int'l Computer Music Assoc., San Francisco, Calif., 1996, pp. 127–132.
12. C. Kriese and S. Tipei, "A Compositional Approach to Additive Synthesis on Supercomputers," *Proc. Int'l Computer Music Conf.*, Int'l Computer Music Assoc., 1992, pp. 394–395.
13. H. Kaper et al., "Additive Synthesis with DIASS_M4C on Argonne National Laboratory's IBM POWERparallel System (SP)," *Proc. Int'l Computer Music Conf.*, Int'l Computer Music Assoc., 1995, pp. 351–352.
14. S. Tipei, *A.N.L.-folds. mani 1943-0000; mani 1985r-2101; mani 1943r-0101; mani 1996m-1001; mani 1996t-2001*, Tech. Report ANL/MCS-P679-0897, Mathematics and Computer Science Division, Argonne Nat'l Laboratory, Argonne, Ill., 1996.

Hans G. Kaper is a senior mathematician at Argonne National Laboratory. His professional interests include applied mathematics, particularly mathematics of physical systems, and scientific computing. He is a corresponding member of the Royal Netherlands Academy of Sciences. His main interest outside mathematics is classical music. He is chairman of "Arts at Argonne," a concert impresario, and an accomplished pianist. He received his PhD in mathematics from the University of Groningen, The Netherlands. Contact him at Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4844; kaper@mcs.anl.gov; www.mcs.anl.gov/~kaper.

Sever Tipei is a professor of composition and music theory at the University of Illinois at Urbana-Champaign, where he also manages the Computer Music Project of the UIUC Experimental Music Studios. He regards the composition of music both as an experimental and as a speculative endeavor. He has a diploma in piano from the Bucharest Conservatory in Romania and a DMA in composition from the University of Michigan. Contact him at the Univ. of Illinois Experimental Music Studios, 2136 Music Bldg., 1114 W. Nevada St., Urbana, IL 61801; s-tipei@uiuc.edu; http://cmp-rs.music.uiuc.edu/people/tipei/index.html.

Elizabeth Wiebel participated in the Student Research Participation Program, sponsored by the Division of Educational Programs at Argonne National Laboratory. She received her BS in mathematics and computer science from St. Norbert College, Wisconsin. She will begin graduate work in computer science at the College of William & Mary, Williamsburg, Virginia this fall. Additionally, she studied and taught piano through the St. Norbert College Music Department. Contact her at 840 Messmer St., Fort Atkinson, WI 53538; ewiebel@mcs.anl.gov.