ICMC

INTERNATIONAL COMPUTER MUSIC CONFERENCE

MONTREAL 1991

PROCEEDINGS

Editors: Bo Alphonce, Bruce Pennycook

Published by ICMC-1991
Faculty of Music
McGill University

Viewpoints on the History of Digital Synthesis

Keynote Paper, ICMC-91

Julius Orion Smith III

Assoc. Prof. (Research), CCRMA, Music Dept., Stanford University, Stanford, CA 94305
Signal Processing Engineer, NeXT Inc., 900 Chesapeake Dr., Redwood City, CA 94063
email: jos@next.com or jos@ccrma.stanford.edu

Abstract - This essay sketches one view of the development of digital synthesis techniques from the days of Music V to the present. A taxonomy of digital synthesis techniques is proposed, and future projections are ventured. It is anticipated that synthesis in the future will be dominated by spectral and physical models.

Introduction

In a highly stimulating *Science* article, Max Mathews painted an exciting vision of "the digital computer as a musical instrument" (Mathews 1963). "Sound from Numbers," it pointed out, was a completely general way to synthesize sound because the bandwidth and dynamic range of hearing are bounded: "any perceivable sound can be so produced." The promise of computer music was that *the computer is capable of generating any sound that could ever come from a loudspeaker.* In *The Technology of Computer Music* (Mathews 1969), Max wrote

> "The two fundamental problems in sound synthesis are (1) the vast amount of data needed to specify a pressure function—hence the necessity of a very fast program—and (2) the need for a simple, powerful language in which to describe a complex sequence of sounds."

Problem (1) has been solved to a large extent by the march of technology. Digital processor performance has increased at the rate of 40-50% per year for the past fifteen years, and the trend shows no sign of weakening. At present, multiple voices of many synthesis techniques can be sustained in real time on a *single-chip*, general-purpose computer.

Problem (2) remains unsolved, and cannot, in principle, ever be completely solved. Nobody has the time to type in every sample of sound for a musical piece, (unless the piece is very short), nor does anyone know *how* to directly type the samples of natural-sounding waveforms. Therefore, sound samples must be *synthesized* from a much smaller set of numbers, or derived from *recordings* of natural phenomena, or both. In either case, a large number of samples must be specified or manipulated according a much smaller set of numbers. This implies a great sacrifice of generality. Fortunately, the vast majority of waveforms are either musically undesirable or musically equivalent to other waveforms, so we should in fact be able to give up most waveforms without giving up anything of musical value. The fundamental difficulty of digital synthesis becomes finding the smallest collection of synthesis techniques which span the space of musically useful sounds without redundancy. It is helpful when a technique is intuitively predictable. Predictability is good when analogies exist with well-known musical instruments.

Historical View of Digital Synthesis Development

In the Music V program (Mathews 1969), the concept of the *unit generator* was introduced. A unit generator is a fundamental building block for sound synthesis. It takes numeric parameters and/or audio signal(s) as input, and produces an output signal. The parameters are constrained to be constant over the duration of a note, or "event." The unit generators of Music V included an oscillator, filter, adder, multiplier, random number generator, and envelope generator. Thus, Music V unit-generators were basic signal processing/synthesis modules which could be combined to create interesting synthetic sounds. The techniques of *additive, subtractive,* and *nonlinear* synthesis (such as FM) could be implemented quite naturally with these elements. They were similar in function to the sound-generating/processing modules used in analog synthesizers at the time, such as voltage-controlled oscillators (VCO), amplifiers (VCA), and filters (VCF). Analog synthesis, in turn, utilized modules from earlier audio electronics.

Instrument definitions in Music V were written as unit-generator patches. An instrument invocation was essentially a subroutine call with arguments, called "P fields," which plugged into the unit-generator patch. A Music V "score" was essentially a time-stamped sequence of instrument calls. Since the instrument and score definitions in Music V completely specified the music computation in a procedural ASCII format, Music V gave us the musical counterpart of PostScript for 2D graphics (the standard marking language used first for laser printers and more recently for computer displays). Apparently, Music V was born at least three decades too soon to be accepted as the PostScript of the music world. Instead, we got MIDI.

In the years following the availability of Music V, a number of research centers with access to large, mainframe computers and D/A converters extended the music compiler in various ways. At CCRMA, for example, various Music V descendants, such as Mus10, introduced named variables, an Algol-style language for instrument definition, more built-in unit generators, piecewise-linear-functions for use as envelope parameters, and an instrument compiler. Descendants of Music V appeared at Princeton (Paul Lansky), MIT (Barry Vercoe), and UCSD (Dick Moore) as well a few other places. Computer music blossomed in the seventies, with many software and hardware systems appearing. It would not be feasible to adequately survey parallel developments throughout the world in this short essay, so the remainder of this historical sketch will describe developments from CCRMA's point of view. The CCRMA story is quite applicable to other computer music labs that have invested significantly in digital synthesis hardware.

While the Music V software synthesis approach was very general and powerful—a unit generator could do anything permitted by the underlying programming language—computational costs on a general-purpose computer were dauntingly high. It was not uncommon to be spending hundreds of seconds of computer time for each second of sound produced. Student composers were forced to work between 3AM and 6AM to finish their pieces. Pressure mounted to move the primitive sound-generating algorithms into special-purpose hardware.

In October 1977, CCRMA took delivery of the Systems Concepts Digital Synthesizer (Roads 1989, pp. 333-349; Loy 1981), affectionately known around CCRMA as the "Samson Box," named after its designer Pete Samson. The Samson Box resembled a large, green refrigerator in the machine room at the Stanford AI lab, and it cost on the order of $100,000. In its hardware architecture, it provided 256 "generators" which were waveform oscillators with several modes and controls, complete with amplitude and frequency envelope support, and 128 "modifiers," each of which could be a second-order digital filter, random-number generator, amplitude-modulator, signum function, allpass controller, and the like. Up to 64K words of delay memory with 32 ports could be used to construct reverberators, other delay effects, and large wavetables. Finally, four D/A converters came with "the Box" to supply four-channel sound output. These analog lines were fed to a 16-by-32 "audio switch" which routed sound to the various listening stations around the lab.

The Samson Box was a very elegant implementation of nearly all known, desirable, unit-generators in hardware form, and sound synthesis was sped up by three orders of magnitude in many cases. Additive, subtractive, and nonlinear FM synthesis and waveshaping were supported very nicely. A lot of music was produced by many composers on the Samson Box over more than a decade. It was a clear success.

The Samson Box, however, was not a panacea. There were very sizeable costs in moving from a general software synthesis environment to a constrained, special-purpose hardware synthesizer. Tens of man-years of effort were poured into software support: A large instrument library was written to manage the patching of hardware unit generators into instruments. Such patching had to be done indirectly via the synthesizer "command stream," that is, instrument procedures in SAIL executed to produce synthesizer commands which were saved in a file. Debugging tools were developed for disassembling, editing, and reassembling the synthesizer command stream. The command stream was difficult to work with, but it was unavoidable in serious debugging work. Software for managing the unique envelope hardware on the synthesizer was developed, requiring a lot of work. Filter support was complicated by the use of 20-bit fixed-point with non-saturating overflow and lack of rounding control. General wavetables were not supported in the oscillators. In general, it simply took a lot of work to make everything work right.

Another type of cost was incurred in moving over to the Samson Box. Research into new synthesis techniques slowed to a trickle. While editing an Algol-like description of a Mus10 instrument was easy, reconfiguring a complicated patch of Samson Box modules was much more difficult, and a lot of expertise was required to design, develop, and debug new instruments on the Box. Many new techniques such as waveguide synthesis and the Chant vocal synthesis method did not map easily onto the Samson Box architecture. Bowed strings based on a physical model could not be given a physically correct vibrato mechanism due to the way delay memory usage was constrained. Simple "Feedback FM" did not work because phase rather than frequency feedback is required. Most memorably, the simple interpolating delay-line, called Zdelay in Mus10, turned out to be incredibly difficult to implement on the Box, and an incredible amount of time was expended trying to do it. While the Samson Box was a paragon of design elegance and hardware excellence, it did not provide the proper foundation for future growth of digital synthesis technology. It was a computer-music production device more than a research tool.

Another problem with supporting special-purpose, computer-music hardware is that it can be obsolete by the time its controlling software is considered useable. Hardware is changing so quickly and software environments are getting so elaborate that we are almost forced to write software that will port easily from one hardware platform to the next. A major reason for the success of UNIX—"the ultimate computer virus"—is that it ports readily to new processors. We simply don't have time to recreate our software universe for new, special-purpose machines. A compromise that works well today is to write all software in a high-level language, but take the time to write hand-coded unit generators for each new processor that comes along. It is possible to implement all known techniques on top of a small number of unit generators which comprise more than 90% of the computational load. The NeXT Music Kit is built according to this model: it is an object-oriented system in which only the UnitGenerator and Orchestra classes must be rewritten for new hardware environments. As a result, the Music Kit can be ported to radically new unit-generator parameters. That way, in addition to the loosely described, standard timbres of General MIDI, such as "honky-tonk piano," there could also be a handful of simple yet powerful unit generators capable of the General MIDI timbres and much, much more. Adding instrument definitions to MIDI would not significantly increase the size of the typical MIDI file, and the *reproduceability* of a MIDI performance—the whole point of General MIDI—would actually be right. Of course, low-end synthesizers not capable of enough voices specified as unit-generator patches could continue to implement named timbres in their own way, as provided by General MIDI. It would also be helpful if General MIDI would define a few controller parameter names for each timbre, such as "brightness", "legato", and so on, so that greater expressiveness of performance is possible. In the more distant future, it would be ideal to have MIDI instrument definitions specifiable in a popular high-level language, as was done in Mus10.

Happily, software synthesis is making bit of a come-back, thanks to more powerful processors. The Motorola DSP56001 signal processing chip, for example, running at 25 MHz, can synthesize a simple guitar model at 44 kHz in real time. The Music Kit on the NeXT Computer uses the DSP56001 for synthesis. Because the DSP chip is a general-purpose processor with extensions for signal processing, it is much easier to program than most prior music-synthesis engines. While the controlling software for the Samson Box represents perhaps more than a hundred man-years of software effort, the DSP-based synthesis software on the NeXT Computer has absorbed only a few man-years so far, and to reach the same degree of completeness would require considerably less total effort.

For a given cost, DSP chips provide much more synthesis power than do general-purpose processor chips. However, current software development tools are significantly inferior for DSP chips. The DSP56001, for example, is still integrated as a "second computer" requiring its own assembly language, quirks and pitfalls, assembler, compiler, loader, debugger, and user-managed interprocessor communication. The DSP C compiler, for example, has simply not been useful, forcing *all* DSP code to be written in assembly language. Due presumably to the language barrier, separate development tools, and general difficulty of programming DSP chips, there does not appear to have been a significant resurgence of software synthesis research using DSP chips as an implementation vehicle. On the NeXT Computer, the "DSP" is serving mostly as a built-in synthesizer with a set of canned patches to choose from. Hardly anyone takes on programming the DSP; perhaps they feel life is too short to take up yet another computer.

General-purpose processors are not far behind digital signal processing chips in being suitable as software-synthesis engines. Sufficiently powerful chips exist today (at high cost). Already, the IRCAM/Ariel musical workstation uses two Intel i860 RISC processors to perform multivoiced synthesis in real time. A single i860 can out-perform a DSP56001, for example, at real-time music synthesis. (In fairness, one can define the problem so that the DSP chip is faster for that problem.) While DSP chips are less expensive by more than a factor of 10 in terms of dollars per computation per second, and while DSP chips possess valuable built-in conveniences due to being oriented specifically toward signal processing, use of a true general-purpose processor leverages off far superior compilers and development tools. Furthermore, RISC processors are adding hardware features needed for efficient graphics and sound processing. It is probably safe to say that software synthesis is on the threshold of returning forever to the form in which it started decades ago: written in high-level languages on fully general-purpose computers. It is now hard to justify the tens of man-years of software effort required to fully integrate and support special-purpose hardware for computer-music synthesis when one can buy mass-produced, general-purpose processors very cheaply, delivering tens and soon hundreds of megaflops per chip. Quality support of high-level languages and the ability to use immediately all previously existing software and development tools is an advantage not to be taken lightly.

processor families in a very short time. Similarly, Bill Schottstaedt at CCRMA wrote "Common Lisp Music" entirely in Common Lisp; a single Lisp macro, "Run," which encloses the sample loop of an instrument definition, tests for the presence of DSP chips (either the one on the NeXT CPU board or five on an Ariel QuintProcessor board) and compiles and loads Lisp code to any DSPs present for acceleration of execution. To port Bill's system to another processor family requires only a Common Lisp implementation on the new computer.

Over the past several years, MIDI-compatible digital synthesizers have been taking over the world as the synthesis engines used by composers and musicians everywhere, including CCRMA. MIDI-compatible digital synthesizers provide far more synthesis power per dollar than we ever saw before. Unfortunately, the development of new techniques is now primarily in the hands of industry. We are no longer likely to read about new synthesis advances in the *Computer Music Journal*. Instead, we continue to hear opaque marketing terms such as "LA Synthesis" with no paper in the literature that explains the technique. On the positive side, musical instrument manufacturers are more likely to hire our students to push forward the degree of sophistication in their synthesizers.

The ease of using MIDI synthesizers has sapped momentum from synthesis-algorithm research by composers. Many composers who once tried out their own ideas by writing their own unit generators and instruments are now settling for synthesis of a MIDI command stream instead. In times such as these, John Chowning would not likely have discovered FM synthesis: a novel timbral effect obtained when an oscillator's vibrato is increased to audio frequencies.

Unlike software synthesis or the Samson Box, MIDI synthesizers require very little effort to control. The MIDI specification simplifies the performance-instrument interface down to that of a piano-roll plus some continuous controllers. In other words, MIDI was designed to mechanize performance on a keyboard-controlled synthesizer. It was not designed to serve as an interchange format for computer-music. MIDI instrument control is limited to selecting a patch, triggering it with one of 128 key numbers, and optionally wiggling one or more controllers to which the patch may or may not respond in a useful way. Rarely is it possible to know precisely what the patch is actually doing, or what effect the controllers will have on the sound, if any. The advantage of MIDI is easy control of preset synthesis techniques. The disadvantage is greatly reduced generality of control, and greatly limited synthesis specification. As Andy Moorer is fond of saying, "no adjustment necessary—in fact, no adjustment possible!"

"Direct digital synthesis makes it possible to compose directly with sound, rather than by having to assemble notes" (Mathews et al. 1974). Thus, part of the promise of computer music was to free composers of the note concept. The note concept become a more abstract *event* which could denote any kind of infusion of information into the sound-computing machinery at a given time. The sound generated by an event could be blended seamlessly with sound generated by surrounding events, obscuring any traditional concept of discrete notes. Hardware synthesizers and MIDI have sent us back to the "Note Age" by placing a wall between the instrument and the note that is played on it. MIDI works against this promise of computer music, however understandably.

MIDI synthesizers offer only a tiny subset of the synthesis techniques possible in software. It seems unlikely that future MIDI extensions will recapture the generality of software synthesis until instrument definitions are provided for, as in the original Music V. A straightforward way to accomplish this would be to define a set of *standard unit generators* for MIDI, and a syntax for patching them together and binding message-parameters and controllers to

The new RISC chips are not a panacea for synthesis either. One of the promises of RISC is that compiler technology and the processor architecture are developed jointly to make sure high-level language programming can make maximally efficient use of the hardware . This promise has not yet been fulfilled. Hand-coding of unit generators in assembly language still increases the performance by a large integer factor on today's RISC processors relative to what the compilers provide. Unfortunately, optimal assembly-language programming is more work on a RISC processor than it is on an elegantly designed DSP chip. Nevertheless, socio-economic factors indicate that general-purpose processors will enjoy many, many more man-years of software-support effort than any special-purpose processor is likely to see. Given that general-purpose CPU chips now offer very fast, (single-cycle, pipelined), floating-point, multiply-add units, it would be relatively easy to incorporate remaining features of today's signal processing chips—apparently much easier than providing a first-class software development environment for a new, special-purpose piece of hardware.

Taxonomy of Digital Synthesis Techniques

The historical sketch above focused more on digital synthesis *engines* than on *techniques* used to synthesize sound. The traditional categories of synthesis were *additive, subtractive,* and *nonlinear.* In this section, an attempt will be made to organize today's best-known synthesis techniques into the categories displayed in the following table:

Processed Recording	Spectral Model	Physical Model	Abstract Algorithm
Concrète	Wavetable F	Ruiz Strings	VCO,VCA,VCF
Wavetable T	Additive	Karplus-Strong Ext.	Some Music V
Sampling	Phase Vocoder	Waveguide	Original FM
Vector	PARSHL	Modal	Feedback FM
Granular	Sines+Noise (Serra)	Cordis-Anima	Waveshaping
Prin. Comp. T	Prin. Comp. F	Mosaic	Phase Distortion
Wavelet T	Chant		Karplus-Strong
	VOSIM		
	Risset FM Brass		
	Chowning FM Voice		
	Subtractive		
	LPC		
	Inverse FFT		
	Xenakis Line Clusters		

Some of these techniques will now be briefly discussed. Space limitations prevent detailed discussions and references for all techniques. The reader is referred to (Roads and Strawn 1985, Roads 1989) and recent issues of the *Computer Music Journal* for further reading and references.

Sampling synthesis can be considered a descendant of *musique concrète.* Jean-Claude Risset noted: "*Musique concrète* did open an infinite world of sounds for music, but the control and manipulation one could exert upon them was rudimentary with respect to the richness of the sounds, which favored an esthetics of collage" (Roads 1989: Risset 1985). It is interesting that the same criticism can be applied to sampling synthesizers three decades later. A recorded sound can be transformed into any other sound by a linear transformation (some linear, time-varying filter). A loss of generality is therefore not inherent in the sampling approach. To date, however, highly general transformations of recorded material have not yet been introduced into the synthesis repertoire, except in a few disconnected research efforts. Derivative techniques such as *granular synthesis* are yielding significant new colors for the sonic palette. It can be argued also that spectral-modeling and wavelet-based synthesis are sampling methods with powerful transformation capabilities in the frequency domain.

"Wavetable T" denotes time-domain wavetable synthesis; this is the classic technique in which an arbitrary wave-shape is repeated to create a periodic sound. The original Music V oscillator supported this synthesis type, and to approximate a real (periodic) instrument tone, one could snip out a period of a recorded sound and load the table with it. The wavetable output is invariably multiplied by an amplitude-envelope. Of course, we quickly discovered that we also needed vibrato, and it often helped to add several wavetable units together with independent vibrato and/or slightly detuned fundamental frequencies in order to obtain a chorus-like effect. Panning between wavetables was a convenient way to get an evolving timbre. More than anything else, wavetable synthesis taught us that "periodic" sounds are generally poor sounds. Exact repetition is rarely musical. Electronic organs (the first digital one being the Allen organ) had to add tremolo, vibrato, and the Leslie (multipath delay and Doppler via spinning speakers and horns) as sonic post-processing in order to escape ear-fatiguing, periodic sounds.

"Wavetable F" denotes wavetable synthesis again, but as approached from the frequency domain. In this case, a desired harmonic spectrum is created—either a priori or from the results of a spectrum analysis—and an inverse Fourier series is used to create the period for the table. This approach, with interpolation among timbres, was used by Michael McNabb in the creation of Dreamsong (Roads 1989: McNabb 1981). It was used years earlier in psychoacoustics research by John Grey. An advantage of spectral-based wavetable synthesis is that *phase* is readily normalized, making *interpolation* between different wavetable timbres smoother and more predictable.

Vector synthesis is essentially multiple-wavetable synthesis with interpolation (and more recently, chaining of wavetables). This technique, with four-way interpolation, is used in the Korg Wavestation, for example. It points out a way that sampling synthesis can be made sensitive to an arbitrary number of performance control parameters. Given sufficiently many wavetables plus means for chaining, enveloping, and forming arbitrary linear combinations (interpolations) among them, it is possible to provide any number of expressive control parameters. Sampling synthesis need not be restricted to static table playback with looping and post-filtering. In principle, many wavetables may be necessary along each parameter dimension. Also, it is good to have a wavetable for every combination of parameters, implying that n parameters of control require 2^n wavetables, given two tables per parameter (i.e., no intermediate tables required). If the parameters are instead orthogonal, (e.g., formant bandwidths), n parameters can be implemented using interpolation among $2n$ wavetables. In any case, a lot of memory is likely to be used making a multidimensional timbre space using tables. Perhaps a physical model *is* worth a thousand wavetables.

Principle-components synthesis was apparently first tried in the time domain by Stapleton and Bass at Purdue University (Stapleton and Bass 1988). They computed an optimal set of *basis periods* for approximating a larger set of periodic musical tones via linear combinations. This would then be a valuable complement to vector synthesis since it can provide vectors which combine to span a wide variety of natural sounds. The frequency-domain form was laid out in (Plomp 1976) in the context of steady-state tone discrimination based on changes in harmonic amplitudes. In this domain, the principle components are fundamental *spectral shapes* which are mixed together to produce various spectra.

Additive synthesis historically models a spectrum as a set of discrete "lines" corresponding to sinusoids. The first analysis-driven additive synthesis for music appears to be Jean-Claude Risset's analysis and resynthesis of trumpet tones using Music V in 1964 (Roads 1989; Risset 1985). He also appears to have carried out the first piecewise-linear reduction of the harmonic amplitude envelopes, a technique that has become standard in additive synthesis based on oscillator banks. The phase vocoder has provided analysis support for additive synthesis for many years. The PARSHL program at CCRMA extended the phase vocoder to inharmonic partials, motivated initially by the piano, and Xavier Serra added filtered noise to the inharmonic sinusoidal model (Serra and Smith 1991). *Inverse-FFT* additive synthesis is implemented by writing any desired spectrum into an array and using the FFT algorithm to synthesize each frame of the time waveform (Chamberlin 1980); it undoubtedly has a big future in spectral-modeling synthesis since it is so general. The only tricky part is writing the spectrum for each frame in such a way that the frames splice together noiselessly in the time domain. Post-processing operations can be applied to the ideal desired spectrum to give optimal frame splicing in the time domain.

Linear Predictive Coding (LPC) has been used successfully for synthesis by Andy Moorer, Ken Steiglitz, and Paul Lansky, and earlier (at lower sampling rates) by speech researchers at Bell Labs. It is listed as a spectral modeling technique because there is evidence that the reason for the success of LPC in sound synthesis has more to do with the fact that the *upper spectral envelope* is estimated by the LPC algorithm than the fact that it has an interpretation as an estimator for the parameters of an all-pole model for the vocal tract. If this is so, direct spectral modeling should be able to do anything LPC can do, and more, and with greater flexibility. LPC has proven valuable for estimating loop-filter coefficients in waveguide models of strings and bores, so it could also be entered as a tool for sampling loop-filters in the "Physical Model" column. As a synthesis technique, it has the same transient-smearing problem that spectral modeling based on the short-time Fourier transform has. LPC can be viewed as one of many possible nonlinear smoothings of the short-time power spectrum, with good audio properties.

The *Chant* vocal synthesis technique (Mathews and Pierce 1989: Bennet and Rodet chapter) is listed a spectral modeling technique because it's a variation on *formant synthesis*. The Klatt speech synthesizer is another example. VOSIM is similar in concept, but trading sound quality for lower computational cost. Chant uses five exponentially decaying sinusoids tuned to the formant frequencies, prewindowed and repeated (overlapped) at the pitch frequency. Developing good Chant voices begins with a sung-vowel spectrum. The formants are measured, and Chant parameters are set to provide good approximations to these formants. Thus, the object of Chant is to *model the spectrum* as a regular sequence of harmonics multiplied by a formant envelope. LPC and *subtractive synthesis* also take this point view, except that the excitation can be white noise rather than a pulse train (i.e., any flat "excitation" spectrum will do). In more recent years, Chant has been extended to support *noise-modulated harmonics*, especially useful in the higher frequency regions. The problem is that real voices are not perfectly periodic, particularly when glottal closure is not complete, and higher-frequency harmonics look more like narrowband noise than spectral lines. Thus, a good spectral model should include provision for spectral lines that are somewhat "fuzzy." There are many ways to accomplish this "air brush" effect on the spectrum. Bill Schottstaedt, many years ago, added a little noise to the output of a modulating FM oscillator to achieve this effect on a formant group. Additive synthesis based on oscillators can accept a noise input in the same way, or any low-frequency amplitude- or phase-modulation can be used. Inverse-FFT synthesizers can simply write a broader "hill" into the spectrum instead of a discrete line (sampled window transform); the phase along the hill controls its shape and spread in the time domain. In the LPC world, it has been achieved, in effect, by *multipulse excitation*—that is, the "buzzy" impulse train is replaced by a small "tuft" of impulses, once per period. Multipulse LPC sounds more natural than single-pulse LPC.

The *Karplus-Strong algorithm* is listed as an abstract algorithm because it was conceived as a wavetable technique with a means of modifying the table each time through. It was later recognized as a special case of the physical model for strings being pursued by McIntyre, Woodhouse, and Schumacher, which led to its extensions for musical use. Cremer's "method of the rounded corner" appears to predate even McIntyre and Woodhouse. What the Karplus-Strong algorithm showed, to everyone's surprise, was that the "corner rounding function" could be simplified to a multiply-free, two-point average with musically useful results. Waveguide synthesis is a set of extensions in the direction of accurate physical modeling while maintaining the computational simplicity of the method of the rounded corner. It most efficiently models one-dimensional waveguides, such as strings and bores, yet it can be coupled in a rigorous way to the more general physical models in Cordis-Anima and Mosaic (ACROE 1990, Smith 1991).

The Control Problem

Issues in digital-synthesis performance practice are too numerous even to try to summarize here. The reader is referred to the survey chapter by Gareth Loy in (Mathews and Pierce 1989, pp. 291-396). Suffice it to say that the musical control of digital musical instruments is still in its infancy despite some very good work on the problems. Perhaps the problem can be better appreciated by considering that instruments made of metal and wood are played by human hands; therefore, to transfer past excellence in musical performance to new digital instruments requires either providing an interface for a human performer—a growing trend—or providing a software control layer which "knows" how to perform a given score in a given musical context. The latter is a real artificial intelligence problem.

Projections for the Future

Abstract-algorithm synthesis seems destined to diminish in "mind-share" due to the lack of analysis support. It is very difficult to find a wide variety of musically pleasing sounds by exploring the parameters of some mathematical expression. Most sounds are simply uninteresting. The space of sounds we hear in everyday life is but a tiny pin-point in the space of all possible sounds. The most straightforward way to obtain interesting sounds is to draw on nature in some way. Both spectral-modeling and physical-modeling synthesis techniques support incorporation and/or modeling of natural sounds. In both cases the model is determined by some analysis procedure which is capable of computing optimal model parameters for approximating a particular given sound. The parameters are then used to provide desirable variations.

Obtaining better control of *sampling* synthesis will require more general sound transformations. To proceed toward this goal, transformations must be understood in terms of what we hear. The best way we know to understand a sonic transformation is to study its effect on the *short-time spectrum*, where the spectrum-analysis parameters are tuned to match the characteristics of hearing as closely as possible. Thus, it appears inevitable that sampling synthesis will migrate toward spectral modeling.

If abstract methods disappear and sampling synthesis is absorbed into spectral modeling, this leaves only two categories: *physical-modeling* and *spectral-modeling*. This boils all synthesis techniques down to those which model either the *source* or the *receiver* of the sound. If it is agreed that nature-referenced techniques are required to obtain natural sounds, it is difficult to ask for greater generality than that covered by these two categories. Spectral modeling is the more general of the two, since it is capable of constructing an arbitrary stimulus along the basilar membrane of the ear. However, physical models provide more compact algorithms for generating familiar classes of sounds, such as strings and woodwinds. Also, they are generally more efficient at producing effects in the spectrum arising from attack articulations, long delays, pulsed noise, or nonlinearity in the physical instrument. It is also interesting to pause and consider how invariably performing musicians have interacted with resonators since the dawn of time in music. When a resonator has an impulse-response duration greater than that of a spectral frame (nominally the "integration time" of the ear), as happens with any string, then implementation of the resonator directly in the short-time spectrum becomes inconvenient. A resonator is a lot easier to implement as a recursion than as a super-thin formant in a short-time spectrum. Of course, as O. Larson says: "Anything is possible in software."

Spectral modeling has unsolved problems in the time domain: it is not yet known how to best modify a short-time Fourier analysis in the vicinity of an attack or other phase-sensitive transient. Phase is important during transients and not during steady-state intervals; a proper time-varying spectrum model should retain phase only where needed for accurate synthesis. The general question of *timbre perception* of non-stationary sounds becomes important. *Wavelet transforms* support more general signal building blocks which could conceivably help solve the transient modeling problem. Most activity with wavelet transforms to date has been confined to basic constant-Q spectrum analysis. Spectral models are also not yet terribly sophisticated; sinusoids and filtered noise with piecewise-linear envelopes are a good start, but surely there are other good primitives. Finally, tools for spectral modeling and transformation, such as spectral envelope and formant estimators, peak-finders, pitch-detectors, polyphonic peak associators, time compression/expansion transforms, and so on and on, should be developed in a more general-purpose and sharable way.

The use of granular synthesis to create swarms of "grains" of sound using wavelet kernels of some kind (Roads 1989: Roads 1978) appears promising as a basis for a future *statistical* time-domain modeling technique. It would be very interesting if a kind of wavelet transform could be developed which would determine the optimum grain waveform, and provide the counterpart of a short-time power spectral density which would indicate the statistical frequency of each grain scale at a given time. Such a tool could provide a compact, transformable description of sounds such as rain, breaking glass, and the crushing of rocks, to name a few.

References

Chamberlin 1980. *Musical Applications of Microprocessors.* New Jersey: Hayden Book Co., Inc.

ACROE 1990. Proceedings of the Colloquium on Physical Modeling, Grenoble, France.

Mathews, M. V. 1963. "The Digital Computer as a Musical Instrument." *Science* 142(11):553-557.

Mathews, M. V., et al. 1969. *The Technology of Computer Music.* Cambridge, Mass.: MIT Press.

Mathews M. V., F. R. Moore, and J.-C. Risset 1974. "Computers and Future Music." *Science* 183(1):263-268.

Mathews M. V., and J. R. Pierce, eds. 1989. *Current Directions in Computer Music Research.* Cambridge, Mass.: MIT Press.

Moore F. R. 1990. *Elements of Computer Music.* Englewood Cliffs, New Jersey: Prentice Hall.

Plomp, R. 1976. *Aspects of Tone Sensation.* New York: Academic Press.

Roads, C., and J. Strawn, eds. 1985. *Foundations of Computer Music.* Cambridge, Mass.: MIT Press.

Roads, C., ed.. 1989. *The Music Machine.* Cambridge, Mass.: MIT Press.

Serra X. J., and J. O. Smith 1991. "Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic plus Stochastic Decomposition." *Computer Music Journal* 14(4):12-24.

Smith, J.O. 1991. "Waveguide Simulation of Non-Cylindrical Acoustic Tubes." Elsewhere in this proceedings.

Stapleton, J. C., and S. C. Bass 1988. "Synthesis of Musical Tones Based on the Karhunen-Loève Transform." IEEE Tr. ASSP, 36(3):305-319