

Viewpoints on the History of Digital Synthesis*

JULIUS O. SMITH III

*Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University, Stanford, California 94305 USA*

December 28, 2005

1 Introduction

In a highly stimulating *Science* article, Max Mathews painted an exciting vision of “the digital computer as a musical instrument” [7]. “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.” In *The Technology of Computer Music* [8], Mathews 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 more than 40% 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 personal computer.

Problem 2 remains unsolved, and cannot, in principle, ever be completely solved. Since it takes millions of samples to make a sound, nobody has the time to type in every sample of sound for a musical piece. Therefore, sound samples must be synthesized algorithmically, or derived from recordings of natural phenomena. In any 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.

The fundamental difficulty of digital synthesis is finding the smallest collection of synthesis techniques that span the gamut of musically desirable sounds with minimum redundancy. It is helpful when a technique is intuitively predictable. Predictability is good, for example, when analogies exist with well-known musical instruments, familiar sounds from daily experience, or established forms of communication (speech sounds).

The rest of this essay sketches one view of the development of digital synthesis techniques from the days of Max Mathews’s experiments to the present. In the first half we present experiences with synthesizer hardware and assess the current state of the art. The second half begins with a taxonomy of synthesis techniques, from which we extrapolate future trends. We anticipate that synthesis in the future will be dominated by spectral and physical models. Spectral models are based

*Keynote Paper, Proceedings of the International Computer Music Conference, Montreal, pp. 1-10, Oct. 1991. Revised with Curtis Roads for publication in *Cahiers de l’IRCAM*, 1992.

largely on the perception of sound, while physical models are based on mathematical descriptions of acoustic musical instruments.

Sound synthesis and signal processing are inherently technical subjects. Although mathematical details will not be presented, this essay assumes familiarity with the engineering concepts behind computer music, particularly a conceptual understanding of a variety of sound synthesis techniques. For a general introduction to this field, see [16, 14, 15], and back issues of the Computer Music Journal.

2 Historical View of Synthesizer Development

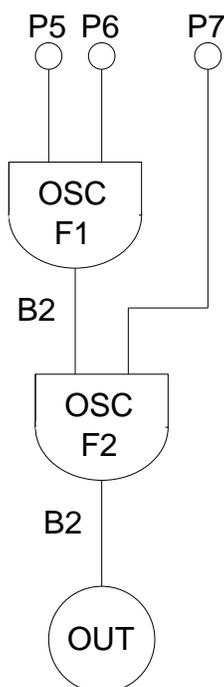


Figure 1: Example Music V unit generator diagram. The left input of each oscillator is the amplitude control, and the right input is the frequency control. In this example, control parameter P5 controls overall amplitude, P6 sets the note duration, and P6 controls the note frequencies. The upper oscillator serves as an amplitude envelope generator for the lower oscillator. Function F1 is trapezoidal in shape (the amplitude envelope), and F2 is the desired oscillator waveform and sets the timbre of the note.

With the Music III program, Max Mathews and Joan Miller introduced the concept of the unit generator for sound synthesis, a technique extended in the languages Music IV and Music V [8]. A unit generator is a fundamental building block, or module, used to build a variety of sound-generating algorithms. Each unit generator accepts numeric parameters and/or audio signal(s) as input, and produces an output signal.

The unit generators of Music V included an oscillator, filter, adder, multiplier, random number generator, and envelope generator. Basic signal processing and synthesis modules could be interconnected to create interesting synthetic sounds. The techniques of additive, subtractive, and

nonlinear synthesis (such as frequency modulation, or FM) could be implemented naturally with these modules. They were similar in function to the modules used in analog synthesizers at the time, such as voltage-controlled oscillators (VCOs), amplifiers (VCAs), and filters (VCFs). Analog synthesis, in turn, utilized modules from earlier audio electronics.

Composers wrote instrument definitions in *Music V* as networks of interconnected unit-generators. An instrument invocation was essentially a subroutine call with arguments, called pfields (parameter fields) supplied to the the instrument. A *Music V* score was essentially a time-ordered sequence of instrument calls. Since the instrument and score definitions in *Music V* completely specified the music computation in a procedural text format, *Music V* gave us a kind of 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 (Musical Instrument Digital Interface).

In the years following the availability of *Music V*, a number of research centers with access to large, mainframe computers and sound converters extended the music compiler in various ways. At Stanford University's Center for Research in Music and Acoustics (CCRMA), for example, *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 numerous universities and research centers around the world. 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 describes developments from CCRMA's point of view. The CCRMA story is applicable to other computer music laboratories that have invested significantly in digital synthesis hardware.

2.1 Experiences with the Samson Box

While the *Music V* software synthesis approach was somewhat 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 common for composers to spend hundreds of seconds of computer time for each second of sound produced. Student composers were forced to work between 3 AM and 6 AM 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 [5], affectionately known as the “Samson Box,” named after its designer Peter Samson. The Samson Box resembled a green refrigerator in the machine room at the Stanford Artificial Intelligence Laboratory, and it cost on the order of \$100,000. In its hardware architecture, it provided 256 generators (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 filter, random-number generator, or amplitude-modulator, among other functions). Up to 64 Kwords of delay memory with 32 access ports could be used to construct large wavetables and delay lines. A modifier could be combined with a delay port to construct a high-order comb filter or Schroeder allpass filter—fundamental building blocks of digital reverberators. Finally, four digital-to-analog converters came with the Box to supply four-channel sound output. These analog lines were fed to a 16-by-32 audio switch that routed sound to various listening stations around the lab.

The Samson Box was an 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 well supported. Much 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 sizable costs in moving from a general software synthesis environment to a constrained, special-purpose hardware synthesizer. Tens of man-years of effort went into software support. A large instrument library was written to manage the patching of hardware unit generators into instruments. Instead of directly controlling the synthesizer, instrument procedures written in the SAIL programming language were executed to produce synthesizer commands that were saved in a “command stream” file. Debugging tools were developed for disassembling, editing, and reassembling the synthesizer command-stream data. Reading and manipulating the synthesizer command stream was difficult but 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 hardware with nonsaturating overflow and lack of rounding control. General wavetables were not supported in the oscillators. Overall, it simply took a lot of systems programming work to make everything work right.

Another type of cost was incurred in moving from the general-purpose computer 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*, was incredibly difficult to implement on the Box, and an enormous 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 synthesis technology. It was more of a music instrument than a research tool.

2.2 Portability and the Limits of MIDI

Another problem with supporting special-purpose computer-music hardware is that it can be obsolete by the time its controlling software is considered usable. Hardware is changing so quickly and software environments are becoming 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 the Unix operating system—“the ultimate computer virus”—is that it ports readily to new processors. We don’t have time to recreate our software universe for new, special-purpose machines. A compromise that works well today is not far from the original *Music V* design: 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 synthesis and processing techniques on top of a relatively small number of unit generators comprising roughly 90% of the computational load. Most existing, widely available, computer-music software systems are built upon the unit generator concept, e.g., *Csound* (MIT), *cmusic* (UCSD), *MAX* (IRCAM), and the *Music Kit* (NeXT).

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. 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 do not always hear about new synthesis advances in the *Computer Music Journal*. Instead, we are likely to first hear opaque marketing terms such as “LA Synthesis” with no paper in the literature that explains the technique.

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 a MIDI note list 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 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!”

As Max Mathews said, “Direct digital synthesis makes it possible to compose directly with sound, rather than by having to assemble notes” [10]. Thus, part of the promise of computer music was to free composers of the note concept. In *Music V*, the note concept became a more abstract event that could denote any infusion of information into the sound-computing machinery. The sound generated by an event could be blended seamlessly with sound generated by surrounding events, obscuring any traditional concept of discrete notes. Commercial 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 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 in some way, such as in the original *Music V*.

The recently introduced Kurzweil K2000 synthesizer has a built-in graphical patching language. One sees “preset” patches on the LCD, but each position in the patch can be occupied by one of 20 different unit generators. This can be seen as a step in the direction of MIDI-based instrument definition. By supporting patch definition in terms of unit generators in the catch-all system exclusive message of MIDI, it becomes possible to create instruments on the fly in a musical performance on the synthesizer. A reasonable next step would be to move this function out of the system exclusive message and into the standard MIDI protocol.

A straightforward way to add instrument definitions to MIDI would be to define a set of standard unit generators and a syntax for patching them together and binding message-parameters and controllers to the unit-generator parameters. That way, in addition to the loosely described, standard timbre names, such as “honky-tonk piano,” in the General MIDI specification recently introduced by synthesizer manufacturers, there could also be a handful of simple yet powerful unit

generators capable of the General MIDI timbres and much more. Adding instrument definitions to MIDI would not significantly increase the size of the typical MIDI file, and the reproducibility of a MIDI performance—the whole point of General MIDI—would actually be correct. Of course, inexpensive 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 at least in terms of a generally understood vocabulary. In the more distant future, it would be ideal to have a means of supplying the “source code” for new unit generators in the MIDI stream. At that point, MIDI instrument definitions would finally be as general as those of the Mus10 program.

2.3 The Rise of Digital Signal Processing Chips

Happily, software synthesis is making bit of a comeback, thanks to more powerful processors. The Motorola DSP56001 digital signal processing (DSP) chip, for example, running at 25 MHz, can synthesize a simple model of a guitar at 44.1 KHz in real time. At present, the DSP56001 costs about as much as a good meal, but its price is dropping precipitously and its speed is going up on the order of a factor of two in the next revision.

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 the general-purpose processor chips found in most personal computers. However, the current software development tools for DSP chips are inferior. In current computer systems, 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. Programmers rarely use the C language compiler for the 56001, for example, because the generated assembly code is too voluminous and slow for most DSP applications, where the whole point is to get fast execution in a small amount of memory. As a result, signal processing code is generally written manually in an arcane assembly language. Due to the language barrier, separate development tools, and general difficulty of programming DSP chips, there has been no significant resurgence of software synthesis research using DSP chips as an implementation vehicle. The Silicon Graphics Indigo workstation has an imbedded DSP56001, but users are not provided a way to program it. On the NeXT Computer, where programming the DSP56001 is supported, the DSP is serving primarily 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 learn a whole new development environment just to write a few unit generators.

2.4 DSP versus RISC Processors

General-purpose processors as found in personal computers are not far behind DSP chips in their suitability as software synthesis engines. Sufficiently powerful chips exist today, but at high cost. The IRCAM/Ariel musical workstation uses two Intel i860 RISC (reduced instruction set computer) processors to perform multivoiced synthesis in real time. A single i860 can out-perform a DSP56001,

for example, at music synthesis. (In fairness, one can define the problem so that the DSP chip is faster for that problem. DSP chips are far more efficient in handling real-time interrupts and low-latency input/output.) While DSP chips are less expensive by an order of magnitude in terms of dollars per computation per second, and while DSP chips possess valuable features due to being specifically designed for real-time signal processing, use of a true general-purpose processor benefits from far superior compilers and development tools (at present). Furthermore, RISC processors are adding hardware features needed for efficient graphics and sound processing. Thus 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 cheaply, delivering tens and soon hundreds of millions of operations per second. However, it remains to be seen whether (a) first-class development environments will be created for DSP chips or (b) general-purpose RISC processors will be given the desirable features of DSP chips, or (c) a DSP chip will be created as a variant of the design for a RISC processor in such a way that it can still benefit from the development tools for the more general processor. There are many possible paths to the desired end result: highly efficient hardware control from high-level software.

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. But 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, and few have taken it on. One well-known music system programmer has stated that hand-coding inner loops for the i860 is one of the hardest things he's ever done. Another factor against RISC processors is the following: due to their deeply pipelined architecture, their performance in interrupt-prone real-time interactive applications is only a fraction of their advertised speed.

Nevertheless, social and economic factors indicate that general-purpose processors will enjoy many more man-years of software-support effort than any special-purpose processor is likely to see. Given that general-purpose RISC chips now offer fast, (single-cycle, pipelined), floating-point, multiply-add units, it would be relatively easy to incorporate remaining features of today's signal processing chips—possibly easier than providing a first-class software development environment for a new DSP chip.

3 Taxonomy of Digital Synthesis Techniques

The previous historical sketch focused more on synthesis engines than on techniques used to synthesize sound. In this section, we organize today's best-known synthesis techniques into the categories displayed in Fig. 2.

Some of these techniques will now be briefly discussed. Space limitations prevent detailed discussions and references for all techniques. The reader is referred to [16, 14, 15] and recent issues of the *Computer Music Journal* for further reading and references.

Sampling synthesis can be considered a descendant of the tape-based *musique concrète*. Jean-Claude Risset noted: “*Musique concrète* did open an infinite world of sounds for music, but the

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

Figure 2: A taxonomy of digital synthesis techniques.

control and manipulation one could exert upon them was rudimentary with respect to the richness of the sounds, which favored an esthetics of collage” [13]. 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. A major problem with sampling synthesizers, which try so hard to imitate existing instruments, is their lack of what we might call “prosodic rules” for musical phrasing. Individual notes may sound like realistic reproductions of traditional instrument tones. but when these tones are played in sequence, all of the note-to-note transitions—so important in instruments such as saxophones and the human voice—are missing. Speech scientists recognized long ago that juxtaposing phonemes made for brittle speech. Today’s samplers make brittle, frozen music.

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 waveshape stored in memory is repeatedly read 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 oscillator 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 slightly detuned fundamental frequencies in order to obtain a chorus-like effect. Cross-fading between wavetables was a convenient way to obtain 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) added tremolo, vibrato, and the Leslie (multipath delay and Doppler-shifting via spinning loudspeakers) 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 creates the period for the table. This approach, with interpolation among timbres, was used by Michael McNabb in the creation of *Dreamsong* [11]. 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. That is, given a sufficient number of wavetables as well as 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, and it is difficult in general to orthogonalize the dimensions of control. In any case, a great deal of memory is needed to make a multidimensional timbre space using tables. Perhaps a physical model is worth a thousand wavetables.

Principal-components synthesis was apparently first tried in the time domain by Stapleton and Bass at Purdue University [19]. The “principal components” corresponding to a set of sounds to be synthesized are “most important waveshapes,” which can be mixed to provide an approximation to all sounds in the desired set. Principal components are the eigenvectors corresponding to the largest eigenvalues of the covariance matrix formed as a sum of outer products of the waveforms in the desired set. Stapleton and Bass computed an optimal set of single periods for approximating a larger set of periodic musical tones via linear combinations. This would be a valuable complement to vector synthesis since it can provide smooth vectors between a variety of natural sounds. The frequency-domain form was laid out in [12] in the context of steady-state tone discrimination based on changes in harmonic amplitudes. In this domain, the principal components are fundamental spectral shapes that are mixed together to produce various spectra. This provides analysis support for spectral interpolation synthesis techniques [4].

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 [13]. 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, developed by Flanagan at Bell Laboratories and adapted to music applications by Andy Moorer and others, has provided analysis support for additive synthesis for many years. The PARSHL program, written in the summer of 1984 at CCRMA, extended the phase vocoder to inharmonic partials, motivated initially by the piano. Xavier Serra, for his CCRMA Ph.D. thesis research, added filtered noise to the inharmonic sinusoidal model [17]. Macaulay and Quatieri at MIT Lincoln Labs independently developed a variant of the tracking phase vocoder for speech coding applications, and since have extended additive synthesis to a wider variety of audio signal processing applications. 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. The frame-splicing problem is avoided when using a bank of sinusoidal oscillators because, for steady-state tones, the phase can be allowed to run free with only the amplitude and frequency controls changing over time; similarly, the frame-splicing problem is avoided when using a noise generator passing through a time-varying filter, as in the Serra sines+noise model.

Linear Predictive Coding (LPC) has been used successfully for music synthesis by Andy Moorer, Ken Steiglitz, Paul Lansky, and earlier (at lower sampling rates) by speech researchers at Bell Telephone Laboratories. 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 instrument bodies, 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 [2] is listed a spectral modeling technique because it is a variation on formant synthesis. The Klatt speech synthesizer is another example. VOSIM is similar in concept, but trades 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 of 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 (as in the 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 to be a special case of physical models for strings developed by McIntyre and Woodhouse, which led to its

extensions for musical use.

What the Karplus-Strong algorithm showed, to everyone’s surprise, was that a physical model for a real vibrating string 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 reminiscent of the Karplus-Strong algorithm. 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 [3, 18, 1].

4 The Control Problem

Issues in synthesis performance practice are too numerous to try to summarize here. The reader is referred to the survey chapter by Gareth Loy [6]. Suffice it to say that the musical control of digital musical instruments is still in its infancy despite some 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 that “knows” a given musical context. The latter is an artificial intelligence problem.

5 Projections for the Future

Abstract-algorithm synthesis seems destined to diminish in importance due to the lack of analysis support. As many algorithmic synthesis attempts showed us long ago, it is difficult to find a wide variety of musically pleasing sounds by exploring the parameters of a mathematical expression. Apart from a musical context that might impart some meaning, most sounds are simply uninteresting. The most straightforward way to obtain interesting sounds is to draw on past instrument technology or natural sounds. Both spectral-modeling and physical-modeling synthesis techniques can model such sounds. In both cases the model is determined by an analysis procedure that computes optimal model parameters to approximate a particular input sound. The musician manipulates the parameters to create musical 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.

Some characteristics of each case are listed in the following table:

Spectral Modeling	Physical Modeling
Fully general	Specialized case by case
Any basilar membrane skyline	Any instrument at some cost
Time and frequency domains	Time and space domains
Numerous time-freq envelopes	Several physical variables
Memory requirements large	More compact description
Large operation-count/sample	Small to large complexity
Stochastic part initially easy	Stochastic part usually tricky
Attacks difficult	Attacks natural
Articulations difficult	Articulations natural
Expressivity limited	Expressivity unlimited
Nonlinearities difficult	Nonlinearities natural
Delay/reverb hard	Delay/reverb natural
Can calibrate to nature	Can calibrate to nature
Can calibrate to any sound	May calibrate to own sound
Physics not too helpful	Physics very helpful
Cartoons from pictures	Working models from all clues
Evolution restricted	Evolution unbounded
Represents sound receiver	Represents sound source

Since spectral modeling constructs directly the spectrum received along the basilar membrane of the ear, its scope is inherently broader than that of physical modeling. 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 much easier to implement as a recursion than as a super-thin formant in a short-time spectrum. Of course, as Orion Larson says¹: “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 that could conceivably help solve the transient modeling problem. Most activity with wavelet transforms to date has been confined to basic constant-Q spectrum analysis, where the analysis filters are aligned to a logarithmic frequency grid and have a constant ratio of bandwidth to center frequency or Q. Spectral models are also not yet 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, should be developed in a more general-purpose and sharable way.

¹Private communication, 1976

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 that would determine the optimum grain waveform, and provide the counterpart of a short-time power spectral density that 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 explosions, rain, breaking glass, and the crushing of rocks, to name a few.

6 Acknowledgment

My thanks to Curtis Roads for instigating and/or contributing several improvements to this version of the text.

References

- [1] J.-M. Adrien and J. Morrison, “Mosaic: A modular program for synthesis by modal superposition,” in *Proceedings of the Colloquium on Physical Modeling*, (Grenoble, France), ACROE, 1990.
- [2] G. Bennett and X. Rodet, “Synthesis of the singing voice,” in *Current Directions in Computer Music Research* (M. V. Mathews and J. R. Pierce, eds.), pp. 19–44, Cambridge, MA: MIT Press, 1989.
- [3] C. Cadoz, A. Luciani, and J. L. Florens, eds., *Proceedings of the Colloquium on Physical Modeling*, Grenoble, France: ACROE, 1990.
- [4] R. Dannenberg, M.-H. Serra, and D. Rubine, “Spectral interpolation synthesis,” *Journal of the Audio Engineering Society*, 1988.
- [5] D. G. Loy, “Notes on the implementation of MUSBOX: A compiler for the Systems Concepts digital synthesizer,” in *The Music Machine* (C. Roads, ed.), pp. 333–349, Cambridge, MA: MIT Press, 1981.
- [6] G. Loy, “Composing with computers—a survey of some compositional formalisms and music programming languages,” in *Current Directions in Computer Music Research* (M. V. Mathews and J. R. Pierce, eds.), pp. 291–396, Cambridge, MA: MIT Press, 1989.
- [7] M. V. Mathews, “The digital computer as a musical instrument,” *Science*, vol. 142, no. 11, pp. 553–557, 1963.
- [8] M. V. Mathews, *The Technology of Computer Music*, Cambridge, MA: MIT Press, 1969.
- [9] M. V. Mathews and J. R. Pierce, eds., *Current Directions in Computer Music Research*, Cambridge, MA: MIT Press, 1989.
- [10] M. V. Mathews, F. R. Moore, and J.-C. Risset, “Computers and future music,” *Science*, vol. 183, no. 1, pp. 263–268, 1974.

- [11] M. M. McNabb, “Dreamsong: the composition,” *Computer Music Journal*, vol. 5, no. 4, pp. 36–53, 1981, Reprinted in [14].
- [12] R. Plomp, *Aspects of Tone Sensation*, New York: Academic Press, 1976.
- [13] J.-C. Risset, “Computer music experiments, 1964—...,” *Computer Music Journal*, vol. 9, pp. 11–18, Spring 1985, Reprinted in [14].
- [14] C. Roads, ed., *The Music Machine*, Cambridge, MA: MIT Press, 1989.
- [15] C. Roads, *The Computer Music Tutorial*, Cambridge, MA: MIT Press, 1996.
- [16] C. Roads and J. Strawn, eds., *Foundations of Computer Music*, Cambridge, MA: MIT Press, 1985.
- [17] X. Serra and J. O. Smith, “Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition,” *Computer Music Journal*, vol. 14, pp. 12–24, Winter 1990, Software and recent papers URL: [//www.iaa.upf.es/~xserra/](http://www.iaa.upf.es/~xserra/).
- [18] J. O. Smith, “Waveguide simulation of non-cylindrical acoustic tubes,” in *Proceedings of the 1991 International Computer Music Conference, Montreal*, pp. 304–307, Computer Music Association, 1991.
- [19] J. C. Stapleton and S. C. Bass, “Synthesis of musical tones based on the Karhunen-Loeve transform,” *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. ASSP-36, pp. 305–319, March 1988.