with near but distinct x_0. In short, such a measure would provide a means to gain control over the predictabilty of the system behavior, thus making possible the user's meaningful choices of n. Indeed, in many cases higher iterates will certainly not result into sounds of more interesting structural properties - though they will certainly result in a longer computation time. Sometimes a large n may even destroy in the signal any observable relation with the signal's generating orbit in the phase space.

Some conclusions

Relavant theoretical details and the empirical use of synthesis by functional iteration must be both investigated in more depth. This is necessary, although a most peculiar aspect of such approach to sound synthesis lies exactly in the possibility of an explorative, nonlinear style of sonic design. Indeed, it is dubious that further analytical knowledge would make the model of better use in a linear and completely deterministic approach. Functional iteration synthesis provide *indeterministic* models of sonic material: the composer must learn his/her strategy by interacting with a source of structured information activated at the level of the microstructure of music, within and through the sound.

We think that the concept of *iteration*, being a concrete source of unpredictable but self-organized, consistent behavior, can capture large-scale design features which are particularly useful when one works at the microstructural level of sound. Methods of chaos theory have been already proposed and used as models of syntactical articulation of music (Pressing, 1988) and as powerful control structures of granular synthesis techniques (Di Scipio, 1990; Truax, 1990b). Our study proposes the extension of this approach to the level of the sample itself, by operationalizing a model of sonological emergence which projects the compositional process down to the micro-time scale in the musical structure. In so doing, it also implies a blurring of the conventional distinction between *sound* and *structure*, since with this kind of model the composer can generate entire fabrics of sound events and extended sonic textures that can hardly be perceived and conceived as separate partial components of the musical structure.

References

Collet, P. & Eckmann, J.P (1980) *Iterated Maps on the Interval as Dynamical Systems.* Boston: Birkhauser

De Poli, G., Piccialli, A. and Roads, C. eds (1991) *Representations of Musical Signals.* Cambridge Ma.: MIT Press

De Poli, G. (1981) Tecniche Numeriche di Sintesi della Musica. *Quaderni del LIMB*, v.1, 12-45

Di Scipio, A. (1990) Composition by Exploration of Nonlinear Dynamical Systems. *Proceedings ICMC90*, SanFrancisco: ICMA, 324-327

Di Scipio, A. (1994) Micro-time Sonic Design and the Formation of Timbre. *Contemporary Music Review* 10(2). In press.

Di Scipio, A. (1993) Models of Material and of Musical Design Become Inseparable. A Study in Composition-theory. *Proceedings International Conference on Cognitive Musicology*, Jyäskylä: Jyäskylän Yliopisto, 300-316

Feigenbaum, M (1980) Universal Behavior in Nonlinear Systems. *Los Alamos Science*, 1, 4-27

May, R. (1977) Simple Mathematical Models with Very Complicated Dynamics. *Nature*, 261, 459-67

Pressing, J. (1988) Nonlinear Maps as Generators of Musical Design. *Computer Music Journal*, 12(2), 35-46

Truax, B. (1990a) Composing with Real-time Granular Sound. *Perspectives of New Music*, 28(2), 120-134

Truax, B. (1990b) Chaotic Nonlinear Systems and Digital Synthesis. An Exploratory Study. *Proceedings ICMC90*, SanFrancisco: ICMA, 100-103

Modular programming of instruments in cmusic

CARLOS CERANA
Laboratorio de Investigación y Producción Musical (LIPM)
Junín 1930- 1113 Buenos Aires - República Argentina
tel/fax: 54-1-804-0877
email: rqcerana@arcriba.edu.ar

Abstract

This paper explains a system developed at the Laboratorio de Investigación y Producción Musical (LIPM), for modular programming of instruments in cmusic. The system allows users to build up their own patches using ready-made pieces of code, in a self-explanatory process. Every musician trained in hardware synthesizers programming can easily develop a complex instrument following simple rules, what turns it useful both for composition and for teaching software synthesis. The system allows coherent connections between envelope generators, devices for pitch and amplitude modulation, and for additive, subtractive, FM and waveshaping synthesis. As it is made exclusively of cmusic operators, users can add their own modules to follow their particular needs.

What it is

AMI (Aid of Modular Instruments) is an attempt to develop a user-friendly interface for instrument programming in cmusic.

The system allows the user to design complex instruments by adding simple modules. These modules are easily understood by every musician trained in hardware synthesizers programming, and their names are self-explanatory.

What it does

In its present state, the system allows coherent connections between envelope generators, devices for pitch and amplitude modulation, for additive, subtractive, FM and waveshaping synthesis, and for spatialization of sound.

User interface

Users build up their own patches by assembling modules, that are ready-made pieces of code. The system guarantees the coherence of connections between modules. By following certain simple rules it is easy to develop a complex instrument, because the constitutive parts are already debugged.

The resultant instrument is later invoked from the score by means of a macro.

What it looks like

A patch designed with AMI is a macro, that is interpreted by the C preprocessor before compiling the cmusic score. The modules that compose the instrument are invoked by other macros, nested in the main one.

An instrument definition in AMI should have the following parts:

1) Header: here is the #define directive, the name of the instrument and its arguments (the parameters that are left open to be controlled from the score).

2) A module (WHEN), that sets the action time for all of them.

3) A module for defining the use of global control for the amplitude.

4) Units for controlling the different aspects of sound (frequency, intensity, spectrum), and their evolution upon time. Some of these modules are specific for a particular synthesis system, others are common to all of them.

5) A module for the audio output.

Let's see an example:

```
#define FMPAIR(beginning,duration,intensity,note)\               <-- header
   WHEN(beginning,duration)\                                 <-- action time
   NOGLOBALAMP\                                  <-- global control of amplitude
   NOTREMOLO\                                                <-- tremolo unit
   NOENVPITCH\                                              <-- pitch envelope
   NOVIBRATO\                                                <-- vibrato unit
   NOMODULATOR1\                                       <-- modulation unit (FM)
   ENVMODULATION3S(0,.5,1,.7,.2,.8)\                    <-- modulation envelope
   MODULATOR1(3,note,1.7,1)\                           <-- modulation unit (FM)
   ENVAMPLITUDE3S(.3,1,.8,1)\                            <-- amplitude envelope
   CARRIER1(intensity,note,1)\                                 <-- carrier (FM)
   POSITION(3,5)                                               <-- audio output
```

The definition of the instrument FMPAIR tells us that it is not affected by global control of the amplitude, nor uses tremolo, vibrato or pitch envelope units. Nevertheless, all these features must be declared and its place held by *ad hoc* modules (NO-modules).

Other modules show us that the instrument is an FM pair. The MODULATOR1 has a modulation index of 3, and a frequency relation of 1.7 with the carrier. The waveshape of both operators is a sine, set by the value 1 as their last parameters. ENVMODULATION3S and ENVAMPLITUDE3S are the envelope generators for modulator and carrier. They both have three segments of straight-line transitions (that is the meaning of the 3S). The first and last values for the modulation envelope are its relative levels at the beginning and at the end of sound; amplitude envelopes begins and ends in zero (for what it is not explicitly said, and they have fewer arguments).

POSITION is an output module that uses the cmusic SPACE unit to spatialize the sound as it were coming from a fixed source.

The values for *beginning*, *duration*, *intensity* and *note* remain undefined, and are later controlled note by note from the score:

```
FMPAIR(0,4,-16dB,C(1))
FMPAIR(4,2,-12dB,A(0))
```

All cmusic pre and post operators can be used with the system (Moore,1990).

It's important to notice the usage of the NOMODULATOR1 module. It says that there is no previous modulation unit affecting the MODULATOR1 (so the instrument is only an FM pair).

Another more complex example:

```
MORECOMPLEXFM(beginning,intensity,note,veltremolo)\              <-- header
  WHEN(beginning,4)\                                          <-- action time
  NOGLOBALAMP\                                  <-- global control of amplitude
  NOVIBRATO\                                                  <-- vibrato unit
  NOMODULATOR1\                                       <-- modulation unit (FM)
  ENVPITCH2C(-20Hz,4,.5,0Hz,-5,5Hz)\                        <-- pitch envelope
  TREMOLO(veltremolo,8)\                                      <-- tremolo unit
  ENVMODULATION2S(.9,.5,.2,.8)\                        <-- modulation envelope
  MODULATOR1(1.5,note,.73,1)\                         <-- modulation unit (FM)
  NOTREMOLO\                                                  <-- tremolo unit
  ENVMODULATION3S(.2,.4,1,.6,.3,.6)\                   <-- modulation envelope
  MODULATOR1(3,note,1.7,1)\                           <-- modulation unit (FM)
  ENVAMPLITUDE3S(.3,1,.4,.3)\                           <-- amplitude envelope
  CARRIER1(intensity,note,1)\                                 <-- carrier (FM)
  MONO                                                        <-- audio output
```

The instrument MORECOMPLEXFM uses cascade FM synthesis. The output of the module MODULATOR1 is taken to feed it again, as the module is invoked twice. It is important to notice that in spite of the fact that the module is the same, its parameters are different, and so are the modulation envelopes that affect it.

Other important feature of the system is shown by the usage of pitch and amplitude modifiers. ENVPITCH2C (pitch envelope generator of two curved segments) affects the two modulators and the carrier. TREMOLO (a very simple unit for amplitude modulation) affects only the first modulator, because its action is turned off by NOTREMOLO before the appearance of the second modulator and the carrier.

How it works

The central concept of AMI is the idea of small, incomplete instruments, whose outputs are blocks going nowhere. The values passed to those blocks are taken by the following partial instruments, which have open inputs coming from nowhere. After processing the data, the output is again "hanged in the air" and taken by the next operator, and so on untill the last module of the instrument.

Playing a note in an instrument designed with AMI means to play a chord with all the modules that compound it, but only the one with the audio output actually "sounds".

The whole system relies on the coherence of naming the input-output blocks. To prevent the unwanted appearance of remaining data in a block (that could be left by other instrument playing at the same time), I introduced the NO-modules. Their sole mission is to clear a particular block, to be sure that its value is neutral to the processing.

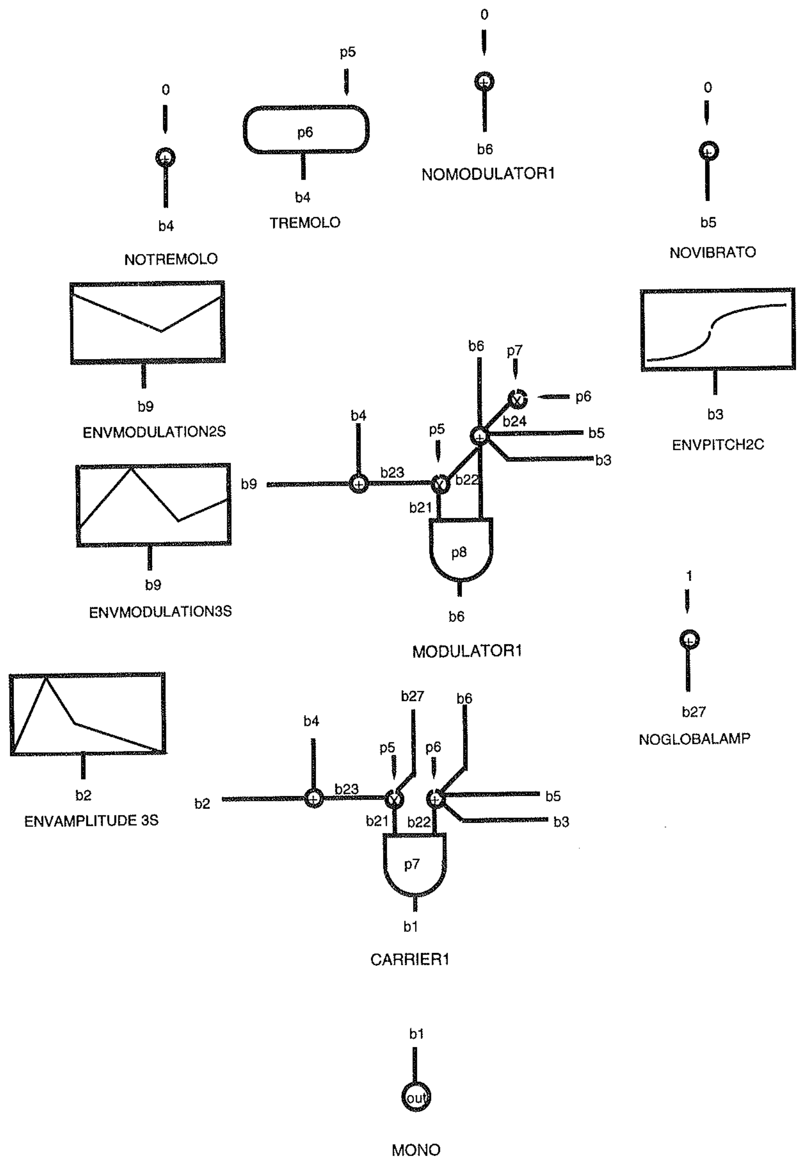
Elements of the system

The modules are macros that play a note in one of the incomplete instruments mentioned above, which themselves are defined using cmusic syntax.

The following two examples will enlighten the subject:

Example 1

```
ins 0 carrier1 ;
          adn b22 p6 b3 b5 b6 ;
          adn b23 b2 b4 ;
          mult b21 p5 b23 b27 ;
          osc b1 b21 b22 p7 d ;
          end ;
```



```
#define CARRIER1(intensity,note,waveshape)\
note p2 carrier1 p4 intensity note waveshape ;
```

The macro CARRIER1 invokes the instrument carrier1, and plays a note in it at the moment ***beginning*** (parameter 2), lasting for *duration* (parameter 4). These two parameters are supposed to be set by the macro WHEN (that's why it must appear as the first module in the definition of the patch).

Example 2

```
ins 0 nil4;
        adn b4 0 ;
        end ;

#define NOTREMOLO\
note p2 nil4 p4 ;
```

The nil4 instrument cleans the block 4, by feeding an additive operator with a zero. Then it is invoked by the macro NOTREMOLO to play a note.

Structure of modules used in the MORECOMPLEXFM instrument are displayed in the figure. The TREMOLO unit is not analyzed here because it itself is composed by modules, and its complete discussion lies beyond the scope of the present paper.

Further developments

The system could easily be expanded to follow everyone's particular needs. As it is made exclusively of cmusic operators, it is possible for users to add their own modules. The only need is to preserve the coherence in naming the input-output blocks, as it was done among the existing modules.

There is also a project for developing a graphic interface for the system, that could result in a significant improvement.

Conclusions

AMI for cmusic allows a comfortable approach to software synthesis, that makes it useful for educational purposes, as well as for composition. The system is easy to deal with for musicians experienced in hardware synthesizers, since its modules remind the operators of such devices.

The primary concern of the system is friendliness and clarity, rather than computational efficiency. I consider that for anybody at the first stages of contact with computer music languages, the waste of time in debugging is much more important that the one in compiling.

References

Cerana, Carlos (1994). *AMI: Ayuda mediante Módulos de Instrumentos para cmusic.* Buenos Aires: LIPM.
Moore, F. Richard (1990). *Elements of Computer Music.* Englewood Cliffs, NJ: Prentice-Hall.

Acknowledgments

The development of the system described in this paper was initiated as a result of a residence program between CRCA (University of California, San Diego), CCRMA (Stanford University) and LIPM, sponsored by the Rockefeller Foundation.

I wish to thank Tim Labor, who patiently tutored my first steps in cmusic, and F. Richard Moore, for their clever and opportune advice and encouragement.

On the Improvement of the Real-Time Performance of Two Fundamental Frequency Recognition Algorithms

ANDREW CHOI
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong

Abstract

Many existing fundamental frequency recognition (FFR) algorithms return reliable results when the analysis window is sufficiently wide. In some applications, however, the response time, i.e., the sum of the width of the analysis window and the computation time for the FFR algorithm, must be made as short as possible. This paper studies the effect of window width on the accuracy of two FFR algorithms and describes a new algorithm with improved accuracy for narrow analysis windows. The new algorithm uses dynamic programming to match harmonics to peaks in the constant-Q transform of the signal. A modification to another FFR algorithm that enhances its performance in real time is also considered.

Introduction

A pitched musical sound is composed chiefly of harmonic components whose frequencies are integral multiples of a fundamental frequency. The problem of fundamental frequency recognition (FFR) is encountered in the automatic analysis of these signals, such as in pitch-to-MIDI systems that enable acoustic instruments to be used as controllers of digital synthesizers.

FFR algorithms that operate in the frequency domain perform spectral analysis on the signal by segments and apply a pattern matching technique to the spectrum to determine each segment's fundamental frequency. Amuedo (1985), for example, identifies sinusoidal components in a signal by the peaks in the power spectrum and examines how the hypothesis for each component to be the fundamental frequency is reinforced by the other components. Pearson and Wilson (1990) consider a multiresolution approach for the spectral analysis step. Doval and Rodet (1991a, 1991b) apply a maximum likelihood analysis to determine the fundamental frequency also using peaks in the power spectrum. Brown (1992) computes the cross-correlation of the constant-Q transform of a segment of the signal with a fixed comb pattern. The calculation of the constant-Q transform and a fast algorithm for approximating it are considered in (Brown 1991) and (Brown and Puckette 1992), respectively.

An alternative approach for designing FFR algorithms is based on computing an autocorrelation between the waveform and a delayed version of itself and determining the fundamental frequency by maximizing the degree of their similarity. Ney (1982) uses time-warping to account for small variations in the signal waveform. The estimated period is the amount of shift that results in the best match of a segment of the signal with a future segment. Lane (1990) adapts the center frequency of a bandpass filter to match the fundamental frequency of the signal using a convergence algorithm. Cook et al. (1993) use a least mean square adaptive algorithm to determine the coefficients of a filter that predicts a segment of a signal from an earlier segment. The phase of the filter is computed from these coefficients, which is then used to estimate the period. Another technique, described in (Brown and Puckette 1993), first determines a coarse estimate of the fundamental frequency using a frequency-domain algorithm. The phase change of the component closest in frequency to the coarse estimate between two segments of the signal separated by one sample is then used to estimate the fundamental frequency accurately.

Accuracy is an important measure of performance of an FFR algorithm. In applications where synthesizers with continuous pitch are controlled, the resolution at which the FFR algorithm can distinguish