UC Irvine
Writings / Presentations

Title
STRATEGIES FOR CONTINUOUS PITCH AND AMPLITUDE TRACKING IN REALTIME INTERACTIVE IMPROVISATION SOFTWARE

Permalink
https://escholarship.org/uc/item/7ck721n4

Author
Dobrian, Christopher

Publication Date
2004

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

STRATEGIES FOR CONTINUOUS PITCH AND AMPLITUDE TRACKING IN REALTIME INTERACTIVE IMPROVISATION SOFTWARE

Christopher Dobrian
Department of Music
University of California, Irvine
Irvine CA 92697-2775 USA
dobrian@uci.edu

ABSTRACT

In a realtime interactive work for live performer and computer, the immanently human musical expression of the live performer is not easily equalled by algorithmically generated artificial expression in the computer sound. In cases when we expect the computer to display interactivity in the context of improvisation, pre-programmed emulations of expressivity in the computer are often no match for the charisma of an experienced improviser. This article proposes to achieve expressivity in computer sound by "stealing" expressivity from the live performer. By capturing, analyzing, and storing expressive characteristics found in the audio signal received from the acoustic instrument, the computer can use those same characteristic expressive sound gestures, either verbatim or with modifications. This can lead to a more balanced sense of interactivity in works for live performer and computer.

1. INTRODUCTION

In the genre of pieces for live instrumental performance in combination with computer, many composers are employing custom-written software that functions in real time during performance in a so-called "interactive" relationship with the live performer. The rubric of "interactivity" is in fact often applied to any work that involves realtime software, regardless of whether the software actually exemplifies true *inter*action between computer and human performer. A significant distinction may be drawn between software that is "reactive" and software that is truly *inter*active.

A shortcoming of many works for live performer of an acoustic instrument in combination with computer is that these two elements—human and computer—are inherently disparate, both in terms of sound quality and expression. Because the means of sound generation by computer is significantly different from that of traditional acoustic instruments, digitally synthesized and processed sound is often significantly different from live instrumental sound, and in many cases lacks the complexity of sound production and performance gesture found in acoustic instruments. This disparity of sound between instrument and computer can be used intentionally to create an obvious contrast, but one may also strive to introduce more complexity and interest in the computer sound. Although digital sounds can be made arbitrarily complex mathematically, this is not necessarily equivalent to the type of complexity we find engaging in human performance. This paper discusses an approach to using some of the complexity of human performance for direct control of computer audio, based on the premise of "stealing" expressivity from a live performer.

2. CHARACTERISTICS OF INTERACTIVITY

If a computer responds instantly to the sound or gestures of a live performer based on a programmed algorithm, this is not necessarily an example of "interactive" computer music. The program is *re*acting to its input in a pre-determined way. The computer can only be purported to be acting autonomously if it is programmed to make some decisions of its own that are not fully predicted by the algorithm. This implies inclusion of some elements of unpredictability: the use of pseudo-randomness at some structural level. Likewise, the relationship between computer and live performer cannot be described as interactive if the performer is playing from a fully fixed score, because the unpredictable behavior of the computer will have no influence on the live performer. The prefix inter- in the word "interactivity" implies mutual influence between agents that are also in some way autonomous decision makers. Neither agent may be fully predetermined in its behavior; each must be able to modify its behavior—to improvise—based on unpredictable behavior by the other. Therefore, improvisation by both human and computer is an essential component of any truly interactive work. In order for a computer to respond effectively to improvisation by a live performer, it must have some capability for cognition as well as independent action. Although computer cognition, intelligence, and expression are all artificial, they can still give the impression of interactivity.

3. STEALING EXPRESSIVITY

Just as the –ivity suffix in the word "interactivity" connotes "a quality of" interaction that can only be artificial in a machine, "expressivity" for a computer can only be a demonstration of an artificial or simulated quality of being expressive in the sense that we apply it to human music making: the conveyance of meaning or feeling. Much research has been focused on formulaic or algorithmic modeling of expressive phrasing (e.g., in jazz or classical music), and on developing new control interfaces that will permit more intimate and intuitive physical control of digital sound parameters. In this article I propose a direct method of achieving expressivity in computer music, by "stealing" it from a

live performer. Rather than trying to abstract a concept of musical expression and devise a formulaic generative method to simulate it, it can be more effective in some cases simply to capture and use expressive characteristics of an actual performance. This is particularly appropriate in an improvised interactive context, when the type of computer expression that might be a suitable response to the live performer may not be known in advance.

It is very difficult to describe formulaically the characteristics of computer sound that will make it seem expressive in a way that is comparable to human musical expression, but we can use human expressive gestures to shape the computer sound. The composer and programmer must determine what characteristics of the live performance are important to expression in a given work, and what characteristics of the performance can be captured and analyzed. When the live performer is using only an acoustic instrument, without MIDI, sensors, or other digital input, the primary source of information from human to computer is the sound of the instrument itself. The computer program can analyze the audio signal, derive characteristics that the composer has deemed important, and use that information directly to control the musical expressivity of its own sound.

4. PITCH AND AMPLITUDE TRACKING

In interactive pieces for acoustic instrument and computer, direct analysis of the audio signal received from the instrument is the main source of information for any cognitive function in the computer program. From the audio signal we can, with varying degrees of success, analyze the pitch, amplitude, and timbre (spectral content) of the signal, and from that information we can derive some information about specific events, notes, dynamic changes, and rhythm. Because of the difficulty of performing such analysis accurately and meaningfully, one must carefully consider what information is desired—which can vary based on the musical context of the moment—and must develop strategies for acquiring that information reliably.

In my recent pieces for flute and computer, I have focused on pitch and amplitude tracking in an effort to give the computer some sense of the musical expression of the live performer. In the following paragraphs I describe some of the strategies I have used. Notably, I will focus on the tracking strategies I developed that are specially suited to the Korean flute *daegeum*, an instrument that idiomatically introduces some stylistic and expressive traits that are often not the focus of Western classical music.

5. CONTINUOUS PITCH TRACKING

For computer pitch analysis of melodic instruments, the fiddle~ software [1, 6] has had widespread usage among programmers in Pd [5] and MSP [3, 7]. (Other software for pitch detection in MSP exists, such as pitch~ [4], yin~ [2], and others. A comparison of all of those methods is beyond the scope of this article.) The fiddle~ implementation in MSP is largely successful for detecting pitch in most Western classical music contexts where the music is conceptually organized as discrete notes each having a single stable fundamental pitch.

In the idiomatic style of the daegeum, however, the identity of a note with a single stable pitch is much more difficult to specify, and indeed is musically inappropriate in many cases. Often in daegeum music, the conceptual equivalent of a note—that is, a distinct period of relative pitch stability within a scale—entails a constant fluctuation of pitch ranging as wide as a semitone above or below the conceptual pitch center. Much of the musical interest and expressivity in this idiom lies in the curve of this pitch variation, in combination with simultaneous variations of amplitude and timbre, as much as or more than the sequential organization of discrete scale steps as would be the case in most Western music. For a computer to capture the expressive nature of pitch in daegeum music, ascribing a single pitch to a note is inadequate; it is the ongoing curve of pitch fluctuation that is more important.

The fiddle~ object does provide the MSP programmer with an ongoing report of estimated pitch without trying to determine a single precise pitch per note, and it is this ongoing report that is more useful for idiomatic daegeum music. However, because of the wide range of spectral variation possible within a single sustained daegeum note, fiddle~ is prone to make some "wrong" pitch assessments that are briefly displaced by an octave from the perceived overall pitch. Although octave melodic leaps are not uncommon in daegeum music, they most commonly appear between phrases, and only much more rarely occur as a smooth legato transition; therefore it is reasonable and beneficial to ignore instantaneous leaps equal to or greater than an octave, on the assumption that they are but spurious artifacts of fiddle~'s analysis. It is also possible to ignore any of fiddle~'s pitch estimates that lie beyond the range of the instrument—estimates that may be the result of turbulent embouchure noise, an inadvertent grunt or loud inhalation by the performer, etc. Pitch estimates that occur when the note has insufficient amplitude can also be filtered out. Once the "bad" guesses are removed, and the "good" guesses have been smoothed with low pass filtering, the result is a pitch curve that perceptually very closely resembles that of the performance (Figure 1).

The continuous pitch curve, now in the form of a signal, can be used to control the pitch of synthesized sounds, modulating oscillators, pitch shifting, etc. in real time, can be transposed or modified with any standard DSP technique such as compression, and/or can be recorded in a buffer~ for later use.

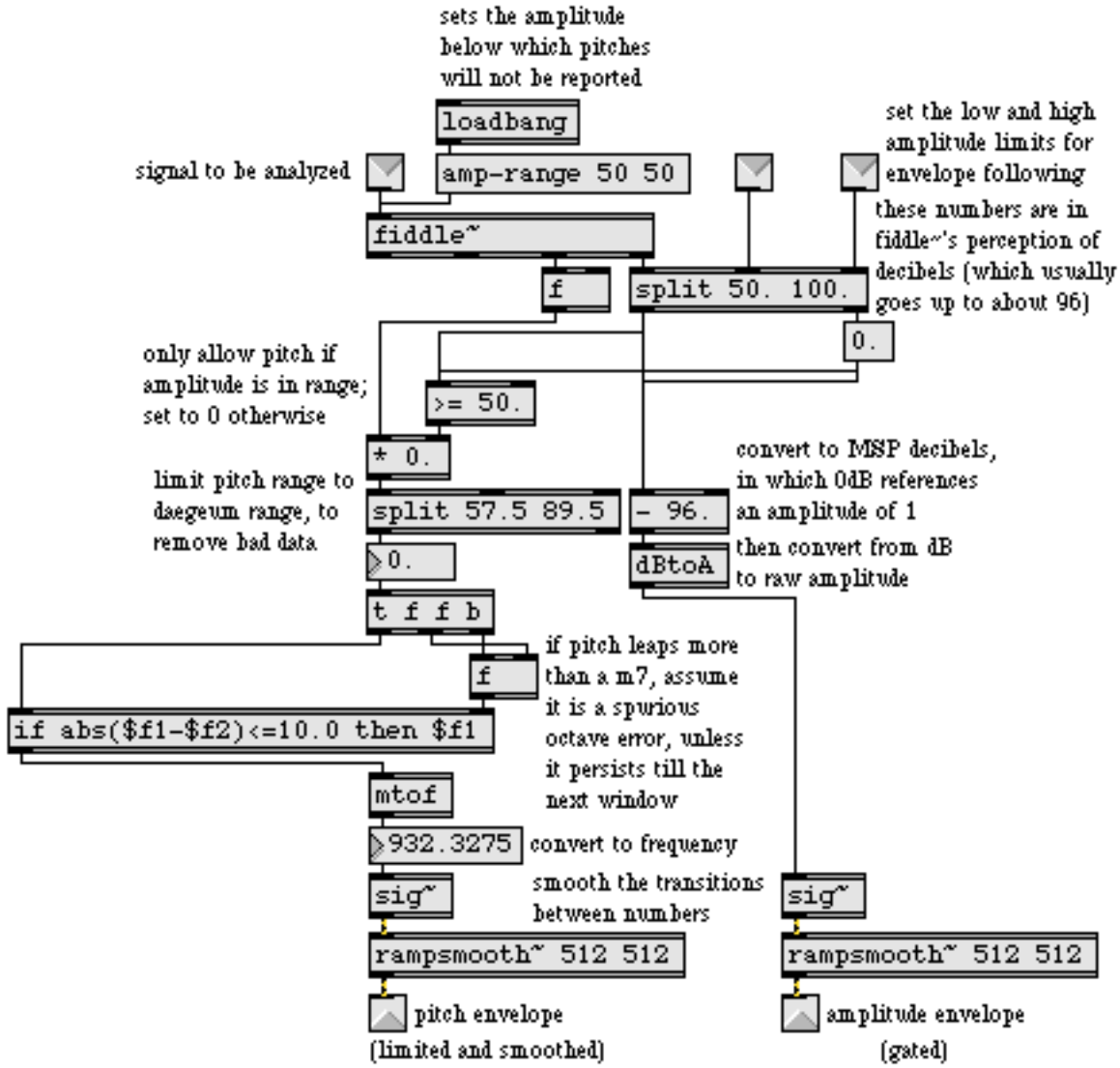

Figure 1. Continuous pitch follower, filtering out spurious values and smoothing the curve

6. AMPLITUDE TRACKING

In MSP it's a simple matter to make an envelope follower that tracks and mimics the over-all amplitude envelope of an audio signal. At a periodic control rate much lower than the audio sampling rate—usually well below the fundamental frequency of the note being analyzed—one can accumulate samples and find the maximum magnitude within the control period, using the peakamp~ object. These maxima describe the general amplitude envelope of the sound, and can be converted back to an MSP signal with interpolation, for use as a control signal for synthesized sounds (Figure 2). As with the continuous pitch curve, the amplitude envelope can be modified with DSP, repurposed to control other sonic parameters, and/or recorded into a buffer~ for later use. Once stored, these control signals derived from the live performance can be treated as musical motives that can recur verbatim or with modifications.

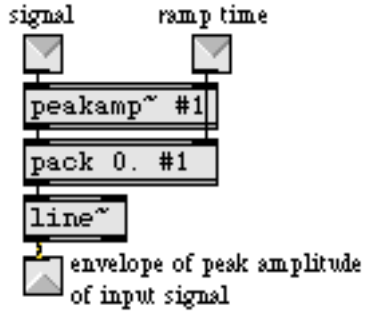

Figure 2. Simple amplitude envelope follower

As was done with the pitch follower described above, we may wish to ignore signals of low amplitude that we deem to be beneath the level of the desired signal, i.e. part of the ambient noise floor. When dealing with signal amplitude, eliminating low level signals can be accomplished by gating—"ducking" all the way to 0 any signal below a given threshold—or by downward dynamic expansion—increasingly reducing a signal the more it falls below a threshold. A ducking

feature can easily be added to the envelope follower just by comparing the incoming peak amplitude to a threshold and converting the value to 0 if it is below the threshold (Figure 3).

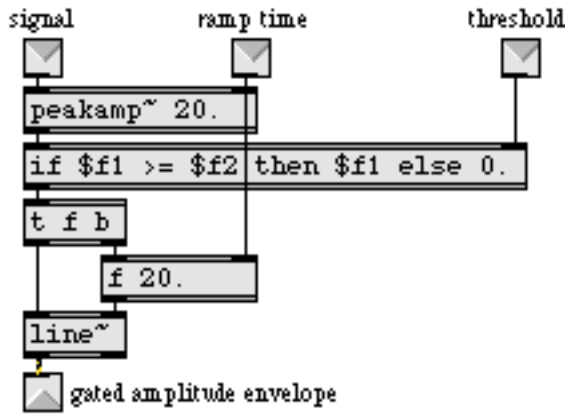

Figure 3. Simple amplitude envelope follower with low amplitudes converted to 0

In actual practice, some more refinements are usually necessary. Precaution should be taken so that peak amplitude levels that hover around the threshold, fluctuating rapidly just above and just below it, do not cause the gate to be opened and shut repeatedly, which would result in a distorted envelope. This can be accomplished by setting a longer attack and/or release time for the opening and closing of the gate. It has the added advantage of creating more graceful attack and release at the beginning and end of the amplitude envelope signal, adjustable to the needs of the situation. The following figure shows one solution, in which the amplitude envelope signal (which might be coming from the simple follower shown in Figure 2 above) is gated by a thresh~ object, with the effect of the thresh~ being smoothed by rampsmooth~. In this example, thresholds and ramp times can be specified independently for the attack and release of the thresh~ gate (Figure 4).

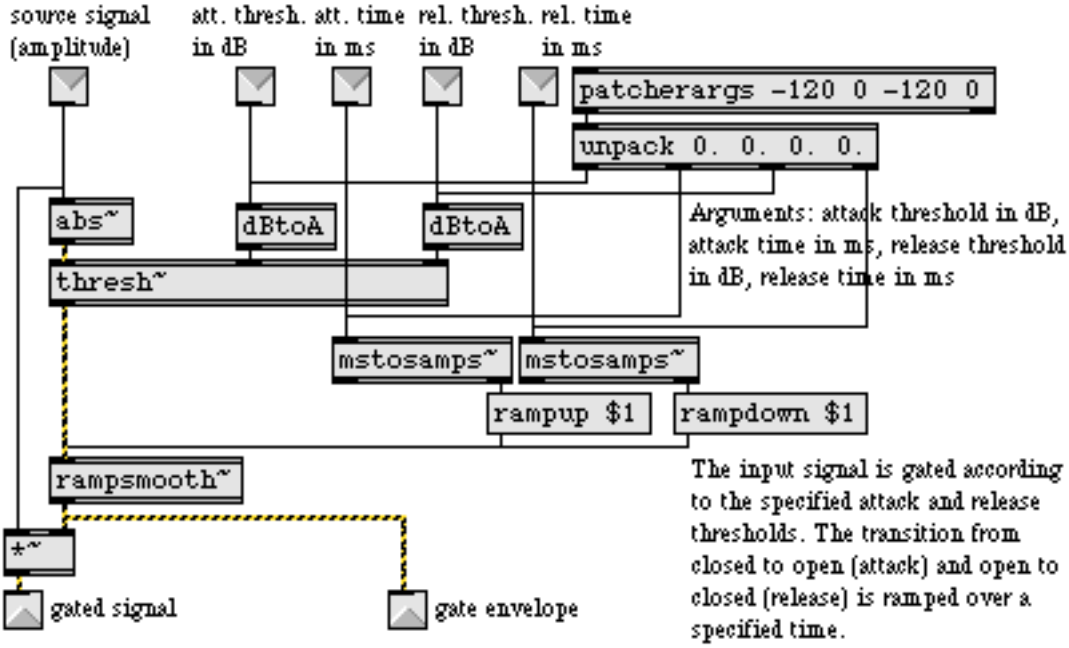

Figure 4. Attack and release thresholds and ramp times for gated envelope follower

7. NOTE BOUNDARY DETECTION

The periodic peak amplitude values gathered by peakamp~ can be used to recognize sound events and silences, which can be interpreted as the beginning and ending boundaries of individual notes or phrases. Once again, the problem is not quite so simple as just detecting when the peak amplitude goes above or below a particular threshold. An instrument such as the daegeum has a very wide dynamic range and idiomatically wide amplitude vibrato (a.k.a. tremolo) that might easily dip below the ambient noise floor many times within the course of a note. In order to avoid wrongly interpreting such an amplitude vibrato as repeated notes, one can specify a "wait time" before determining that a note has ended; if the peak amplitude goes back above the threshold before the wait time has passed, the note is presumed to be continuing. In the same spirit, one might specify a minimum note duration, less than which a note cannot be presumed to have ended.

The daegeum, like most flutes, often has a very gradual attack and release, and this must be taken into account when evaluating the attack amplitude of a note. If we only look at the amplitude at the moment it passes the detection threshold, we may get an inaccurate assessment of the note's attack velocity, because the true peak of the attack—the "downbeat", or "sync point" if you will—may actually occur a fraction of a second later than when it first passes the threshold. In order to allow for this possibility, we can track the increasing amplitude from the time it passes the threshold on the attack, and only report the true peak of the attack once we see that the amplitude is steady or is starting to

diminish.

The following figure demonstrates these ideas. The input amplitude is expected as a floating-point value expressed in decibels. This would come from an input signal, the peak amplitude of which is periodically evaluated by peakamp~ and then converted to decibels with atodb. Some default values for attack threshold, release threshold, minimum note duration, and "wait time" are shown here, but they can be adjusted by new values received in the inlets (Figure 5).

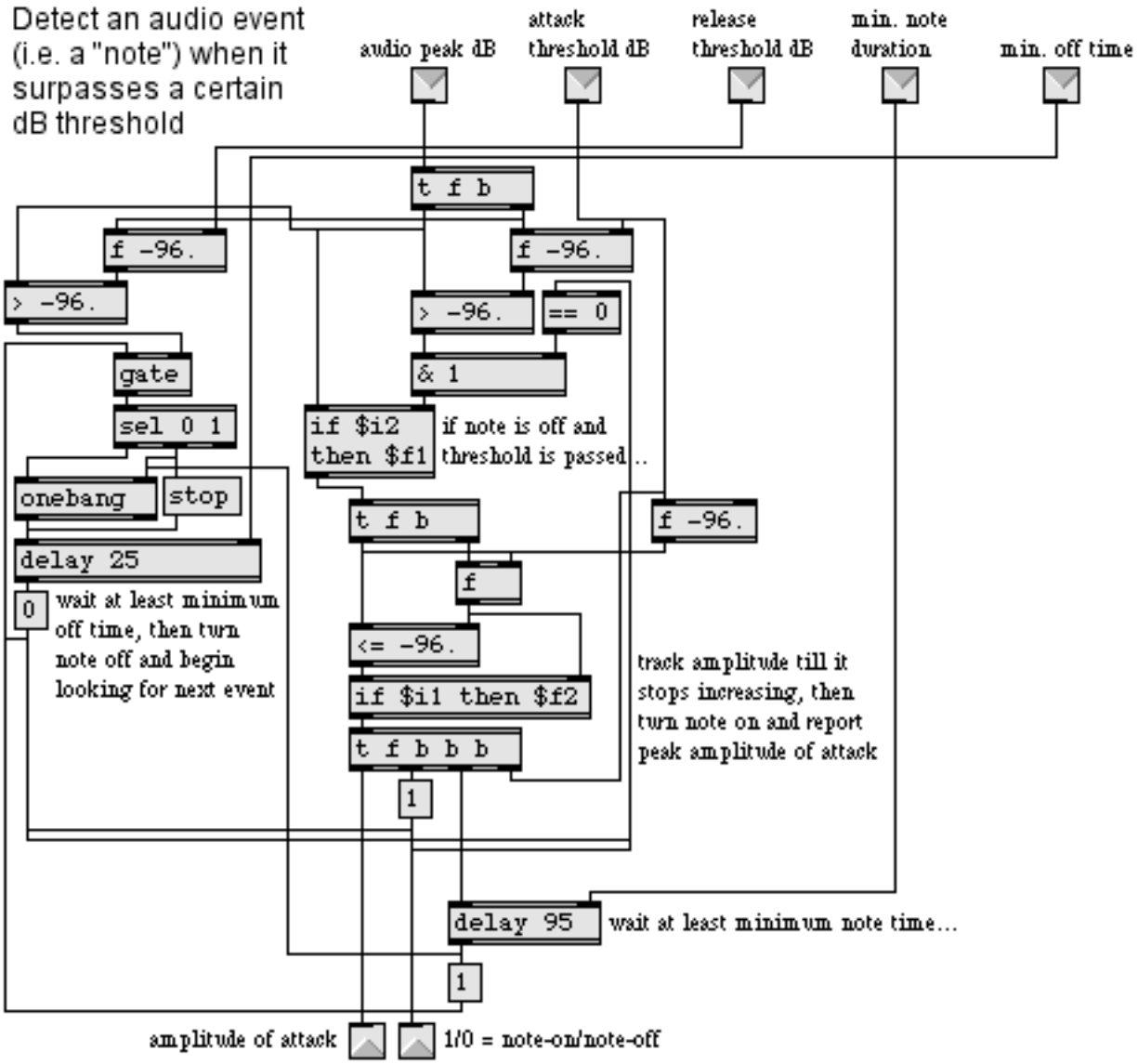

Figure 5. Detect a sound event and report its beginning, ending, and attack amplitude

When an amplitude value comes in, we compare it to the threshold. If it exceeds the threshold and no note is currently on, then we track future amplitudes until they are no longer increasing. At that time, the note is presumed to have reached its peak attack amplitude, so we send a 1 out the right outlet to say that an event has occurred, and we send the peak attack amplitude out the left outlet. Also, we update the note's on/off status (in the & object) and we start the note duration and wait time clocks (the delay objects). From that time on, incoming amplitudes are continually evaluated to see if they remain above the release threshold. If not, and once both the minimum duration and the wait time have passed, the note is reported as having ended.

This type of note boundary detection is very useful for triggering events or processes based on the attack detected in an incoming audio signal, and ending them when a note release is detected. For example a synthesized or recorded sound can be started and stopped in response to these note boundaries, and peak attack amplitude can be used to determine how loudly the computer will play its sound. A program might also make higher level evaluation of a series of events, such as estimating tempo, measuring and storing played rhythms, etc.

8. USE OF TRACKED PITCH AND AMPLITUDE DATA

The control signals derived from the amplitude and pitch of an input audio signal can be applied instantaneously to control parameters of the computer audio. Additionally, they can be modified in MSP before being used, and/or can be applied to parameters

other than amplitude and pitch, such as panning, filter cutoff frequency, etc. Figure 6 shows a straightforward example of how the patches shown in the previous figures can be used to provide expression to a synthesized sound. The patches called pitchfollower~, envelopefollower~, threshgate~, and detectevent correspond to Figures 1, 2, 4, and 5 above. The patcher soundsynthesis could be any synthesis algorithm, with the four inlets being used to provide amplitude curve, frequency curve, attack peak amplitude, and note on/off indicator. Note how easily the frequency curve can be transposed, and the amplitude curve can be repurposed to control panning from left to right as the amplitude increases.

This example (Figure 6) shows realtime use of expressive information signals from a live performer. Alternatively, once those control signals have been recorded into a buffer~, they become motives that can be recalled later in the performance, and can be modified by any DSP technique.

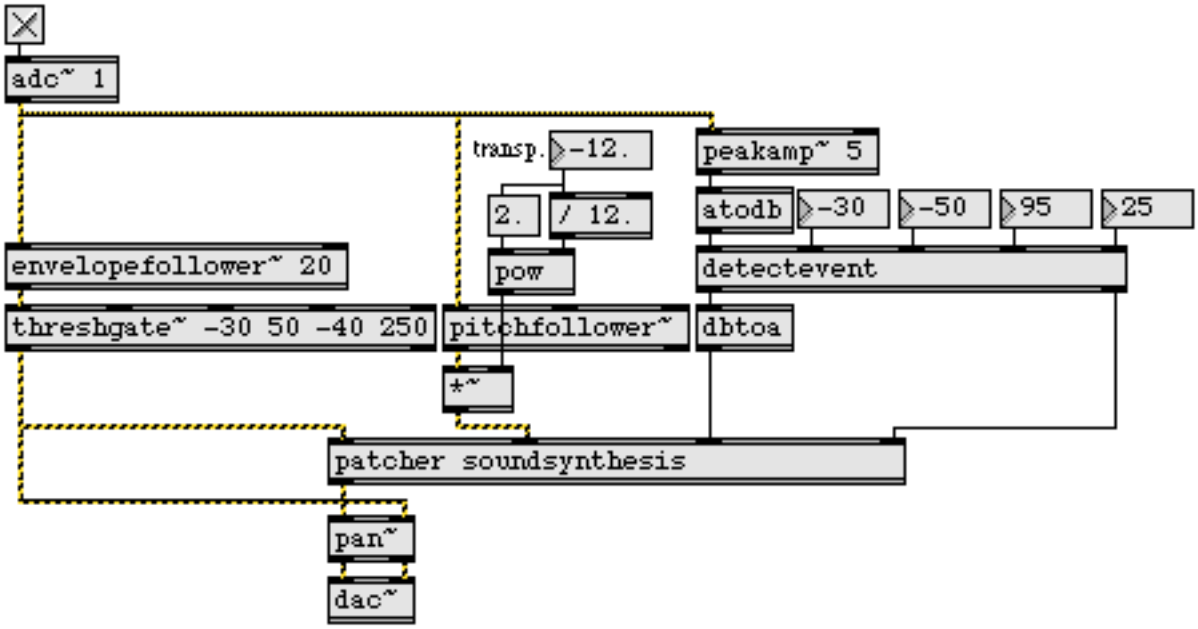

Figure 6. Amplitude and pitch curves used to control synthesis and panning

9. CONCLUSION

For expressivity in a realtime improvised interactive work, it is often effective to "steal" expressive information from a live performer and apply it to the computer music. In works for acoustic instrument and computer, the audio signal is the primary source of such information. Continuous curves derived from the pitch and amplitude of the sound source can be used to control parameters of the computer audio, lending a sense of human expressivity to the computer music. These expressive curves, when expressed as a signal in MSP, can be modified with DSP, can be reassigned to other sonic parameters, and can become motives for improvisational use by the computer later in the piece. The computer sounds more expressive because it is in fact basing its own sonic gestures on information derived directly from the live performance.

10. REFERENCES

[1] Brown, J. C. and Puckette, M. 1993. "A High-Resolution Fundamental Frequency Determination Based on Phase Changes of the Fourier Transform". Journal of the Acoustical Society of America, 94:2, pp. 662-667.

[2] Cheveigné, A. and Kawahara, H. 2002. "YIN, a fundamental frequency estimator for speech and music". Journal of the Acoustical Society of America, 111:4, pp. 1917-1930.

[3] Dobrian, C. 1998. *MSP: The Documentation*. San Francisco: Cycling '74.

[4] Jehan, T. and Schoner, B. 2001. "An Audio-Driven, Spectral Analysis-Based, Perceptual Synthesis Engine". Proceedings of the International Computer Music Conference. San Francisco: International Computer Music Association.

[5] Puckette, M. 1996. "Pure Data". Proceedings of the International Computer Music Conference. San Francisco: International Computer Music Association, pp. 269-272.

[6] Puckette, Miller and Apel, Theodore. 1998. "Real-time audio analysis tools for Pd and MSP". Proceedings of the International Computer Music Conference. San Francisco: International Computer Music Association, pp. 109-112.

[7] Zicarelli, D. 1998. "An Extensible Real-Time Signal Processing Environment for Max". Proceedings of the International Computer Music Conference. San Francisco: International Computer Music Association.