21 Understanding Csound's Spectral Data Types

Barry Vercoe

Most of the signals in Csound are discrete-time sequences of floating-point values, varying at a fixed audio rate or control rate, or at some nonperiodic rate determined by score events, MIDI events and control sensing. They can often replace each other as inputs to compound generators, so that an audio oscillator can take an amplitude that is variously an i-time constant, a control signal, or an audio signal. Also, a Csound instrument normally progresses from event constants to control signals to audio signals and we spend most of our sound-design time making this progression real.

Spectral data types are different. Although they represent audio and control signals and likewise vary at some fixed rate over time, they cannot be plugged into normal signal slots, nor can normal signals substitute for them. They create a separate network of data communication, often with several simultaneously different refresh rates, and generally maintain an orderly world of their own. Moreover, the entrance and exit to this world is the reverse of the above: we begin with audio signals and end with control signals. The goal of this chapter then is to make this progression seem just as real.

The reason for this reverse is signal analysis, sensing and detection. Csound utilities such as **lpanal** and **pvanal** do analyze signals, producing output that can lead to effective reconstruction under time or frequency modifications, but there is little real *sensing* or *detection* in these processes and certainly none that squarely represents what you and I would sense in a sound either acoustically or musically. While **lpanal** and **pvanal** exploit the elegance of all-pole filters and Fast Fourier Transforms (with linearly spaced filter bins of equal bandwidth), the human cochlea has evolved hair-cell collectors that are exponentially spaced with proportional bandwidths. Both systems have had their reasons. The problem with mathematically elegant analysis, however, is that it appeals mostly to computers (and some people). What the rest of

us really need for computer-assisted music performance are sound analysis and sensing mechanisms that work just like our own.

The Csound spectral data types are based on perceptually relevant methods of analysis and feature detection. From the initial massaging of audio input data to the gradual mounting of evidence favoring a certain pitch, pulse or tempo, the methods and opcodes I have devised are inspired by what we currently know about how humans "get" a musical signal. As a result, the opcodes enable one to build models of human auditory perception and detection within a running Csound instrument. This gives the resulting pitch and rhythm detectors both relevance and strength. In describing the nature and use of these opcodes below I will occasionally allude to their physiological and perceptual roots. For a more detailed account, however, see my chapter "Computational auditory pathways to music understanding" in (Vercoe 1997).

Opcodes

A feature-detecting sequence that uses spectral data types is formed from a small set of opcodes that can be grouped as shown in figure 21.1.

The connecting data object *wsig* contains not only spectral magnitudes, but also a battery of other information that makes it self-defining. In a chain of processing opcodes, each will modify its input spectral data, but the output object will retain the self-defining parts to pass on to the next opcode. This opcode will in turn "know" things, such as which spectrum opcode is periodically refreshing the first link in the chain, the time of last refresh, how often refreshes occur, how many spectral points there are per octave, whether they are magnitude or dB, the frequency range, etc. All of this means that an ending operator like **specdisp** or **specptrk** can tell from its

```
ENTERING:                  wsig        spectrum   xsig,     ...

PROCESSING AND VIEWING:    wsig        specaddm   wsig      ...
                           wsig        specdiff   wsigin
                           wsig        specscal   wsigin,   ...
                           wsig        spechist   wsigin
                           wsig        specfilt   wsigin,   ...
                                       specdisp   wsig,     ...

LEAVING:                   koct, kamp  specptrk   wsigin,   ...
                           ksum        specsum    wsig,     ...
```

Figure 21.1 Csound's spectral data type opcodes.

input how often it must do work and how detailed this must be. It can also opt to ignore some changes and work at a slower pace.

The originating spectral analysis of audio is done by:

```
wsig  spectrum  xsig, iprd, iocts, ifrqa, iq[, ihann, idbout,
                idisprd, idsinrs]
```

The analysis is done every *iprd* by a set of exponentially spaced Fourier matchings, wherein a windowed segment of the audio signal is multiplied by sinusoids with specific frequencies. The process is first performed for the top octave, for *ifrqs* different frequencies exponentially spaced. The window size is determined by *iq,* the ratio of Fourier center frequency to bandwidth. For efficiency, the data are not actually windowed at all, but the sinusoids are and these can be viewed by making *idsines* nonzero. Next, the audio data are downsampled and the process repeated for the next octave and so on, for as many octaves as requested. To fill the window of the lowest bin in the lowest octave, the downsampled data must be kept around for some time. The stored down-samples (dynamically changing) can be periodically displayed by giving *idisprd* a nonzero value.

Keeping downsampled audio to fill a slow-moving low-frequency window brings us to a problem we will encounter later. Both the Hanning and Hamming-shaped windows are symmetric (bell-shaped sinusoidal) and designed to focus analytic attention on their temporal center. To make the centers of all frequency-analysis windows coincide at the exact same time-point, the higher frequency windows are delayed until the low frequency window is complete. This introduces an input-output time delay across the **spectrum** opcode. The amount of delay depends on both the window size (indirectly *iq*) and the number of octaves (*iocts*). While this delaying strategy might at first seem unnecessarily fussy, the coincident windows turn out to make a big difference for some spectral operations like **specptrk**, as we will see shortly.

Once we have reliable spectral data, the other opcodes can then proceed with their work. The unit **specaddm** does a weighted add of two incoming *wsigs,* while **specdiff** calculates the difference between consecutive frames of a single varying spectrum. This latter can be seen as a delta analyzer, operating independently on each "channel" of the spectrum to produce a differential spectrum as output. In fact, it reports only the positive differences to produce a positive difference spectrum and is thus useful as an energy onset detector. The units **spechist** and **specfilt** are similar to each other, the first accumulating the values in each frequency channel to provide a running histogram of spectral distribution, while the second injects each new value into a first-order lowpass filter attached to each channel. We will see this used in one of the examples below.

The units **specptrk** and **specsum** have only control signal output and provide a way back into standard Csound instrument processing. The first is a pitch detector, which reports the frequency and amplitude as control signals.

```
koct, kamp  specptrk  wsig, kvar, ilo, ihi, istrt, idbthresh,
                      inptls, irolloff [, iodd, iconfs,
                      interp, ifprd, iwtflg]
```

The detection method involves matching the spectral data of *wsig* with a template of harmonic partials (optionally odd, with some roll-off per octave). Matching is done by cross-correlation to produce an internal spectrum of candidate pitches over a limited pitch range (*ilo* to *ihi*). The internal spectrum is then scanned for the strongest candidate, which, if confirmed over *iconfs* consecutive *wsigs,* is declared the winner. The output is then modified accordingly.

The combination of suitably scaled **spectrum** and **specptrk** units creates a robust pitch detector, capable of extracting the most prominent component of a mixed source signal (e.g., a sitar against a background drone). We can observe some of this at work: we can display the original spectrum via a **specdisp** and we can display the cross-correlation spectrum of the present unit by giving *ifprd* a nonzero value. When an incoming signal has almost no energy at the fundamental (e.g., a high bassoon-like nasal sound), this tracker will still report the human-perceived fundamental pitch. And whereas traditional pitch detectors have difficulty with fast-moving tones like octave scoops, this tracker will stay with the signal, largely because we have time-aligned all the windows of octave down-samples (as described above). Lastly, the pitch resolution of any tone is not restricted to the frequency bins-per-octave of the originating spectrum, but employs parabolic interpolation to obtain much higher resolution.

With an understanding of the above we are now in a position to consider some applications.

A Beat Tracker and Tempo Follower

Energy assessment in the human auditory system is a complex affair. It is not measured immediately but is integrated over time, and we cannot gauge the full intensity of a single impact for about 300 milliseconds. If another impact should occur within that period, the integration of the first is incomplete and the second impact becomes the beneficiary of the remainder (Povel and Okkerman 1981). Consequently, when a stream of impacts arrives grouped in pairs, the first of each pair will seem *softer* than the second, even when both have the same physical intensity. This leads us to the

perception of a "lilting" rhythm, and the same phenomenon is at the base of all human rhythmic perception.

A machine will not see it that way. An instrumentation-quality intensity detector will report something much closer to the truth. And if it is digital, even its own integration time (in the low nanoseconds) will be thwarted by the sample-and-hold process.

So how do we get a computer to hear rhythms the way we do? We could program a set of rules that would reinterpret the intensity patterns along human perceptual lines; for a complex score, this could be time consuming. Or we could model the above energy integration in the data gathering itself. This latter is the strategy implemented in Csound's spectral data processing, and an instrument that would track audible beats and follow a changing tempo in human-like fashion would look as shown in figure 21.2.

Every .01 seconds we form a new spectrum, 8 octaves, 12 frequencies per octave, with a bandwidth Q of 8. We use a *Hamming* window, request magnitudes in dB and skip the display of downsampled data. We next apply *Fletcher-Munson* scaling, using stored function tables *f 3* and *f 4,* to simulate the frequency-favoring effect of the auditory canal.

For the inner ear, calculation of a positive difference spectrum is relevant for the following reason: when the human cochlea receives a sudden increase in energy at a hair cell, the neural firing rates on its attached auditory nerve fibers register a sudden increase, then a rapid adaptation to more normal behavior. By contrast, when it receives a sudden decrease in energy, the hair cell almost ignores the event. Clearly our hearing has evolved to be highly sensitive to new onsets (life-threatening?) and almost oblivious to endings, and our music reflects this with event-oriented structures flavored with percussive sounds. We give our machine a similar predilection on each frequency channel with **specdiff**.

The energy integration phenomenon, however, is not visible on the auditory nerve fibers. Apparently this must be happening at a later stage of processing and we can measure it only by psychoacoustic experiment (Povel and Okkerman 1981). It is not yet clear how this actually works. We simply presume in the above model to inject the positive difference data directly into integrating filters (**specfilt**), whose time constants are frequency dependent and are conveyed via stored f-table *f 5.* Finally, we sum the energy sensation across all frequency bins to produce a running composite, in *ksum4.* This is a simple sum, purposely disregarding the effects of simultaneous masking on loudness perception (Zwicker and Scharf 1965), since our real goal is to compare the energies across time.

To the extent that *ksum4* adequately represents the fluctuation in our own sensations, we can now perform pulse and tempo estimation on a single channel of k-rate

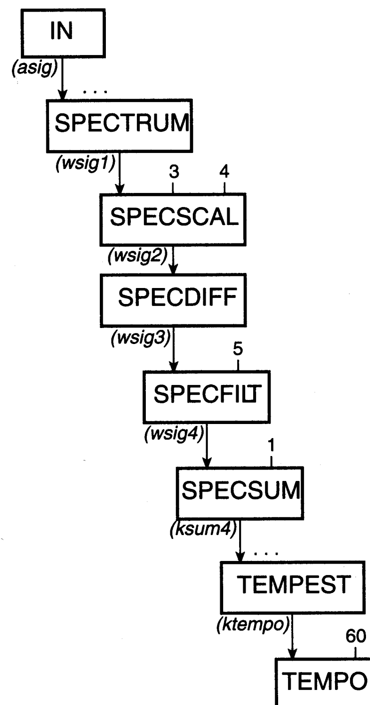

Figure 21.2 Block diagram of *instr 2101,* a beat tracker and tempo follower.

```
       instr     2101      ; BEAT TRACKER, TEMPO FOLLOWER
asig   in                             ; GET MICROPHONE INPUT
                                      ; FORM A SPECTRAL DATA TYPE
wsig1  spectrum  asig, .01, 8, 12, 8, 0, 1, 0
wsig2  specscal  wsig1, 3, 4          ; FLETCHER-MUNSON SCALING
wsig3  specdiff  wsig2                ; POSITIVE DIFFERENCE SPECTRUM
wsig4  specfilt  wsig3, 5             ; INJECT INTO INTEGRATING FILTERS
ksum4  specsum   wsig4, 1
                                      ; GET TEMPO...
ktempo tempest   ksum4, .01, .1, 3, 1, 30, .005, 90, 2, .04, 1
       tempo     ktempo, 60           ; ... AND CONTROL THE PERFORMANCE
       endin
```

Figure 21.3 Orchestra code for *instr 2101,* an instrument for taking microphone input and controlling the tempo of the performance based on beat tracking.

data. The **tempest** unit does not traffic in spectral data types, so it will not be described here beyond what is already covered in the Csound manual and further in (Vercoe 1997). It does however afford some good graphic display of the short-term (echoic) memory modeling from which the beat structure and tempo are derived, along with its development of rhythmic expectations that are an essential part of human beat and tempo tracking, and the reader is advised to try running the unit with the input values given above so as to observe them.

The final **tempo** opcode takes us beyond analysis and observation. Although it does nothing for the beat-tracking instrument itself, the **tempo** opcode takes the machine-estimated running *ktempo* and passes it to the Csound scheduler, which controls the timing of every new event. Therefor if the above instrument is inserted into another working orchestra and score, and the command-line flag **-t 60** is invoked, you can control that orchestra's performance by simply tapping into the microphone. A live demonstration of this was intially given at the 1990 ICMC (Vercoe and Ellis 1990), when Dan Ellis controlled the tempo of a Bach performance by tapping arbitrary drum rhythms on a table near a microphone.

A Pitch Tracker and Harmonizer

Since the same human ear that detects rhythms is also responsible for sensations of pitch, we can build a model of this new phenomenon using many of the same initial principles. The two paths of course eventually diverge, and we will be forced to consider some of the special needs of pitch acuity as we get deeper into the search. Given a good sense of pitch, it is not hard to build automatic harmonizers and pitch-to-MIDI converters. We will look briefly at both of these before forming some conclusions.

The two examples we will use, however, employ opcodes that are not part of the normal Csound distribution. These are from my Extended Csound, a version I have developed that can run complex instruments in real-time using the Analog Devices 21060 floating-point DSP accelerator (Vercoe 1996). In that system, some number of DSPs (1–6) on a plug-in audio card can dynamically share the computational load of a large Csound orchestra, which often contains new opcodes that extend both its repertoire and its real-time performance capacity. Although my personal exploration into these real-time complexities is currently dependent on such accelerators, I fully expect the experience gained will eventually migrate to more generally accessible platforms. The interested reader will also find additional presaging examples of Extended Csound on the CD-ROM that accompanies this volume.

A Csound instrument that can pitch-track an incoming audio signal and turn that into a five-part harmony would look as shown in figures 21.4 and 21.5.

First, we take one channel from our stereo microphone and give it some simple equalization (EQ) to heighten the voice partials. Our spectral analysis is similar to the above, with the following new considerations: we will form a new spectrum only every .02 seconds, since percussive rhythm is not the likely input. We request 6 octaves of downsampling, 24 frequencies per octave, with a bandwidth Q of 12. We also request a *Hanning* window and root magnitude spectral data.

The choice of 24 frequency bins of Q 12 merits some discussion. Both are weighted toward pitch-tracking rather than intensity measurement as was the case above, yet they still fall short of an ideal model of the ear. The human cochlea has about 400 hair-cell detectors per octave in this frequency region. On the other hand those detectors are broad-band, with a Q of 4 (1/3 octave). Broad-band implies fast energy collection, where things like binaural sensing of direction depend on accurate measurement of interaural time differences. This is not our goal here, and we opt for slower, more narrowly focused filters, one quarter-tone apart. The parabolic interpolation in **specptrk** will do the rest.

We are now sending **specptrk** some favorable data. The range restriction of 6.5 to 8.9 (in decimal octaves) is sufficient to cover my voice range even on a good day and we give it an initial hint of 7.5. So that it will not try to pitch-track just microphone noise, we set a minimum threshold of 10 dB, below which it will output zeroes for both pitch and amplitude. Since I may decide to sing some strange vocal sounds (e.g., with missing fundamentals), we ask for an internal template of 7 harmonic partials, with a rolloff of .7 per octave. We request just 3 confirmations of any octave leap (proportionally less for smaller intervals) and ask that the pitch and amplitude outputs be k-rate interpolated between consecutive analyses. Finally, we ask it to display the running cross-correlation spectrum so that we can observe the various pitch candidates in dynamic competition.

There was a price to pay for all this, it may be recalled. So that the tracker would stay locked onto fast-moving voice "scoops," we carefully delayed all channels of analysis until the low-frequency window was full and the other windows could be centrally aligned. The amount of delay incurred by **spectrum** is reported on the user console at i-time. For a sampling rate of 16K, a Q of 12 and 6 octaves of downsampling, that value is 66 milliseconds. Having now emerged from the spectral data type world with a running pitch value, we now delay the audio signal by this amount so that the audio and its pitch estimate are synchronized.

We are now ready for the harmonizer. The **harmon4** unit is not part of regular Csound, but an addition that exists in Extended Csound (Vercoe 1996). It is similar to Csound's **harmon** unit, but depends on other processing modules (such as

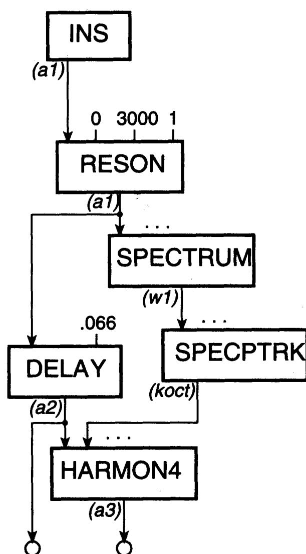

Figure 21.4 Block diagram of *instr 2102,* a pitch tracker and harmonizer.

```
            instr     2102      ; PITCH TRACKING HARMONIZER
a1, a0      ins                               ; GET MICROPHONE INPUT
a1          reson     a1, 0, 3000, 1          ; AND APPLY SOME EQ
w1          spectrum  a1, .02, 6, 24, 12, 1, 3  ; FORM A SPECTRAL DATA TYPE
                                              ; FIND THE PITCH
koct, kamp  specptrk  w1, 1, 6.5, 8.9, 7.5, 10, 7, .7, 0, 3, 1, .1
a2          delay     a1, .066                ; TIME ALIGN PITCH AND AUDIO
                                              ; ADD 4 NEW PARTS
a3          harmon4   a2, koct, 1.25, .75, 1.5, 1.875, 0, 6.5
            outs      a2, a3                  ; AND SEND ALL 5 TO OUTPUT
            endin
```

Figure 21.5 Orchestra code for *instr 2102,* a pitch tracking harmonizer instrument shown in figure 21.4.

specptrk) to provide a reliable pitch estimate. Like **harmon**, **harmon4** will pitch-shift the original audio stream while preserving the vocal formants and vowel quality. Also, like **harmon**, the pitch-shifts can be specified either as frequency ratios (with the source) or as specific cps values. Its main advantage is better sound quality and the ability to generate up to four vocal transpositions at once.

If you have an Extended Csound accelerator card you can run the **harmon4** instrument as shown above. The four transpositions are given as ratios from the source: .75, 1.25, 1.5 and 1.875, outlining a major triad in 6–4 position with an added major seventh at the top. This is basically the instrument that I demonstrated live at the 1996 International Computer Music Conference (Vercoe 1996), and the voice transposing quality is quite good. If you have only the standard Csound distribution, you should replace the **harmon4** line with the following:

```
a3        harmon    a2, cpsoct(koct), .2, 1.25, .75, 0, 110, .1
```

The transposed voice quality will not be as good, and there are only two added voices instead of four, but the example will serve to demonstrate the effect.

One can imagine many variants of the above. A simple one is to replace the fixed-ratio harmonies with independently derived pitches, as from an external MIDI keyboard, or from some algorithm cognizant, say, of the "changes" in a jazz standard. Another is to replace **harmon4** with a different generator, either a Csound looping sample oscillator reading a different sound (a voice-controlled trombone is fun), or a pitch-to-MIDI converter that would let you take your voice control outside the system to another device:

```
          midiout      kamp, koct, iampsens, ibendrng, ichan
```

The possibilities for experimentation and development here are quite unbounded and the reader is encouraged to develop his or her own instruments or opcodes that would take advantage of the feature detection that spectral data types provide.

Conclusion

Csound's spectral data types, based on perceptual methods of gathering and storing auditory relevant data, provide a fresh look at how to enable computer instruments to extract musically important information from audio signals. They offer a new future of computer-assisted ensemble performance connected by sound, not merely by electrical signals. While we do not yet have a full understanding of how humans do the feature extraction that informs both their own performance and their listening, we have shown that imbedding what we do know within a computer instrument can

give it an ensemble relevance that normally only live performers achieve. This is why listening to a live performance is still so exciting and this is where computer music eventually must go.

References

Povel, D., and H. Okkerman. 1981. "Accents in equitonal sequences." *Perception and Psychophysics* 30: 565–572.

Vercoe, B. 1996. "Extended Csound." *Proceedings of the International Computer Music Conference*, pp. 141–142.

Vercoe, B. 1997. "Computational Auditory Pathways to Music Understanding." In Deliege, I. and J. Sloboda (eds). *Perception and Cognition of Music (*pp. 307–326). East Sussex: Psychology Press.

Vercoe, B., and D. Ellis. 1990. "Real time Csound: software synthesis with sensing and control." *Proceedings of the International Computer Music Conference,* pp. 209–211.

Zwicker, E., and B, Scharf. 1965. "A model of loudness summation." *Psychological Review 72:* 3–26.