

Spectral Signal Processing in Csound 5

Victor Lazzarini, Thomas Lysaght and Joseph Timoney
Music Technology Laboratory, National University of Ireland, Maynooth
Victor.Lazzarini@nuim.ie, Joseph.Timoney@cs.nuim.ie, Thomas.Lysaght@cs.nuim.ie

Abstract

This article explores the new unit generators that integrate the Csound 5 music programming language for frequency-domain signal (fsig) processing. This framework for spectral signals was introduced to Csound 4.13, by Richard Dobson and further extended by Victor Lazzarini in version 5. The latest release of Csound incorporates a variety of new opcodes for different types of spectral manipulations. This article introduces the fsig framework and the analysis and resynthesis unit generators. It describes in detail the different types of spectral processing made possible by these new opcodes.

1. Introduction

Csound 5 (ffitch 2005), released in February 2006, is a completely re-coded version of the popular MUSIC *N*-derived music programming language (Vercoe 1996). Among the many new features, it provides completely new host and module APIs, for embedding and extending the system, several new frontends and scripting language support, as well as numerous new opcodes, bringing the total number of unit generators to over 800. These include a set of opcodes designed to work with spectral signals, defined by the Csound orchestra language type *fsig*.

This signal type (Lazzarini 2005) was introduced by Richard Dobson to Csound 4.13, together with a few basic opcodes. It provides a framework for spectral processing, which was further extended by Victor Lazzarini (in Csound 5) to support partial track signals. Previously, spectral processing in Csound was limited to transformation and resynthesis of spectral datafiles. With the *fsig* framework, a Csound instrument can manipulate any input signal in the frequency-domain. The opcodes process streaming spectral signal, which is generated by an analysis (or a data file reader) opcode. The frequency-domain signal can be resynthesised using inverse-DFT overlap-add or additive synthesis.

2. Frequency-domain signals

Streaming frequency-domain signals are defined by the Csound *fsig* type. Such signals are processed at a rate that is dependent on the size of the DFT analysis frame and the number of overlapping frames (or the hopsize), effectively the rate of generation of new spectral frames. The ‘perform’-method of a spectral processing opcode is called every control period, but it only outputs a new frame if there is a new frame at its input. The *fsig* framework provides support for such checks. Consequently, the *fsig* rate is independent of the control rate and the processing is more efficient than the original datafile-based opcodes, which were tied to the orchestra control rate.

Fsig are self-describing. Unlike time-domain audio and control signals, they are furnished with the extra information about their features: DFT length, number of overlaps ($N/\text{hopsize}$), window size, window type, data format and frame count (current frame number, starting from 0). The actual format of the spectral data can vary, currently three types are being used: PVS_AMP_FREQ, amplitude and frequency pairs as produced by the phase vocoder and IFD; PVS_AMP_PHASE, amplitude and phase (polar DFT) data; and PVS_TRACKS, partial tracks of amplitude, frequency, phase and track ID (Lazzarini et al 2005). Of these, the first two will have a fixed size, namely the DFT size plus two extra values (holding the positive spectrum plus the Nyquist frequency, generated by the DFT of a real signal), or $N/2 + 1$ bins. These two formats will be henceforth referred to as ‘bin-frame’ data.

The PVS_TRACKS signal is actually of variable size, but ultimately having a maximum number of tracks equivalent to the number of analysis bins. The partial tracks will contain four items each; however not all partial track-processing opcodes will require all of them (the phase can be sometimes omitted). Crucial to its operation is the track ID information, as it is used to match tracks at consecutive frames. It is possible to introduce other types of bin-frame spectral signals, for instance, data in rectangular (real, imaginary) format.

Nevertheless, the musical generality of the PVS_AMP_FREQ has so far fulfilled the needs of most bin-frame processing applications.

3. Spectral Analysis

Streaming spectral data can be generated by three opcodes: the original `pvsanal` and `pvsfread` opcodes (written by Richard Dobson) plus the `pvsifd`, introduced in Csound 5. These unit generators produced data in bin-frame format, which can be further transformed into partial tracks, by the `partials` opcode.

3.1 Phase Vocoder

The `pvsanal` opcode, as well as by the `pvanal` Csound utility, which generates PVOCEX files for the `pvsfread`, perform phase vocoder analysis. They are loosely modelled on the original CARL phase vocoder (Dolson 1986). Effectively, they take a time-domain signal, window, rotate and feed it into a real-signal FFT operation, producing $N+2$ spectral samples, which are then polar-converted. The difference of phases on consecutive frames is taken and converted into frequency in Hz. The input signal frames are overlapped by a user-defined hopsize. The `pvsanal` opcode does this operation in a streaming fashion, generating a new frame every hopsize input samples.

3.2 Instantaneous Frequency Distribution

The `pvsifd` opcode implements the instantaneous frequency distribution analysis (Lazzarini et al 2005), which is based on frequency reassignment. At every hopsize input samples it takes two FFTs of the signal (windowed by a analysis window and its derivative) and generates two fsigs, one containing a PVS_AMP_FREQ signal, similar to the `pvsanal` output, and another containing a PVS_AMP_PHASE signal. This pair of fsigs can then be used for a full partial track analysis.

3.3 Partial track analysis

The `partials` opcode takes in two fsigs, with PVS_AMP_FREQ and PVS_AMP_PHASE and does a partial track analysis, generating a PVS_TRACKS signal containing a variable number of partial tracks. Each track will model one partial of the input signal, with amplitude, frequency, phase and partial ID data. It is possible to use only a single PVS_AMP_FREQ input, in which case the phase information will be omitted from the analysis output. In fact, the majority of the track processing opcodes do not require phase information. It is possible to feed partial track analysis with the output of a `pvsanal` or `pvsfread` opcode.

4. Spectral Processing

Csound 5 provides a comprehensive set of spectral processing opcodes that will take and produced fsigs. In this section, we will look at the different types of transformations, loosely classified in amplitude, frequency and combination effects, as discussed in (Verfaillie and Depalle 2004).

4.1 Amplitude transformations

The basic type of amplitude effects are filter-like processes, which will alter the amplitude functions, but leave the frequency (and phase) unaltered. Richard Dobson provided to Csound 4.13, a `pvsmask` opcode, which uses a function table of $N/2$ length as an amplitude response curve. The opcode multiplies each bin amplitude by a function table value, indexed by the bin number, effectively filtering the signal. In Csound 5, the `trfilter` opcode operates in a similar fashion, but process partial tracks instead of bin frames, thus the length of the function table is not required to be fixed to any particular size. Time-varying filter effects can be implemented using a table writing opcode, as show in the example below, which implements a comb filter-like effect:

```
aphs phasor 1 /* table writing index */
/* sinusoid signal to be fed into the
   table, kpkps is number of spec peaks */
afil oscili 2, kpkps/2, 1
/* table writing (table size=44100) */
tabw abs(afil), aphas, 3, 1
ffil trfilter ftrk, 1, 3 /* filtering */
```

In the example above, if the signal `kpkps` is modulated, then a flanger-type effect will result. Time-varying filtering can also be implemented by two other Csound 5 opcodes, `pvsfilter` and `pvsarp`. The latter provides a spectral ‘arpeggiation’ effect by zeroing some bin amplitudes and boosting others. The former takes two bin-frame fsigs and uses one of them as an amplitude response, multiplying the two amplitude functions together. This opcode can be also used in some cross-synthesis effects as well as filtering.

Also in the category of amplitude transformations we have the mask-based effects, such as noise cancellation that can be performed by the `pvsstencil` opcode. This takes an input signal and compares it, bin by bin with an amplitude response mask in a function table, performing amplitude scaling based on this comparison. For a denoiser type effect, it is possible to construct such a mask table from a PVOCEX file (using GEN43) and apply it to an input signal using this opcode.

4.2 Frequency transformations

The basic frequency effects implemented for streaming spectral signals are frequency scaling and frequency shifting. These are available for both bin-frame (`pvscale` and `pvshift`) and partial track signals (`trscale` and `trshift`). Frequency scaling of bin-frame signals is described by the following expression:

$$f_{out}[np] = f_{in}[n]p \quad (1)$$

where f_{in} and f_{out} are the input and output bin frequencies, respectively, n is the bin index and p is the scaling interval. A simple harmoniser example is shown below:

```
asig in
/* spectral processing, iscl is the
harmoniser interval */
fsig pvsanal asig, isiz, isiz/4, isiz, iw
ftps pvscale fsig, iscl, ikeepform, igain
atps pvsynth ftps
/* there is a N-sample delay between input
and output */
adp delayr .1
adel deltapn ifftsize
delayw asig
out atps+adel
```

In order to compensate for the N-sample delay between the input of the spectral process and its output, a short delay line is used, so the original and transposed signal can be time-aligned. The bin-frame frequency scaler and shifter opcodes have two special modes of operation that will attempt to preserve formants, for vocal applications. If the `ikeepform` variable is 1 or 2, one of these modes will be used. The method of formant preservation used here is perhaps not as accurate as the one described in (Rodet and Roebel 2005), but many times more efficient and yielding good results. Fig.1 compares the output and input of `pvscale` using formant preservation with pitch shifting by an interval of a fifth.

Frequency shifting adds a value to all frequencies in the input spectra, as defined by the following expression for bin-frame signals:

$$f_{out}\left[n + \frac{s}{bw}\right] = f_{in}[n] + s \quad (2)$$

where n is the bin index, s the frequency shift amount in Hz and bw is the bin bandwidth in Hz. This has the effect of destroying any harmonic relationships that might exist in an input sound.

With partial track processing, it is possible to split the tracks into one or more frequency regions and apply such transformation to those regions, altering the timbre of an instrument, but not completely destroying its pitch impression. This is demonstrated in the example below:

```
/* split tracks at 1500 Hz */
ftrkd, ftrku trsplit ftrk, 1500
/* shift upper frequencies by 150 Hz */
fshft trshift ftrku, 150
/* combine split tracks */
ftmix trmix ftrkd, fshft
```

It is important to note that frequency scaling, as well as shifting, is slightly simpler with partial tracks, if compared to bin-frame signals. In fact, it only requires the scaling or shifting of the partial frequencies, with not need for the bin-reallocation implied in eqs. 1 and 2.

4.3 Cross-synthesis and other effects

A number of cross-synthesis effects are possible, from morphing by interpolation of bin values (`pvscross` by Richard Dobson), to channel vocoder-like amplitude substitution (`pvsvoc`) and partial track cross-synthesis (`trcross`). A special signal combination effect is also implemented by the `pvsdemix` opcode, which is loosely based on the reverse-panning ADress algorithm. This opcode takes two signals, the left and right channels of a stereo mix and separates the instruments in the mix according to their panning position.

For partial track signals, there are a number of specialised opcodes that will manipulate and transform track data. As shown in a previous example, it is possible to split and mix tracks (`trsplit` and `trmix`), as well as isolate the highest and lowest-frequency tracks (`trhighest` and `trlowest`) and obtain their current frequency and amplitude values. It is also possible to realise some special effects, such as residual extraction, by combining original and track-resynthesis signals, in similar process to the one described in (Serra 1997). Another effect that involves the transformation of both frequency and amplitude is that of spectral blurring (Wishart 1995). This is based on averaging the amplitude and frequency functions of time that make up the analysed spectra. Effectively this is the application of a low-pass FIR, with a smoothing effect on the amplitude and frequency signals. The effect is implemented by the `pvsblur` opcode, which takes in a 'blur time' parameter, defining the averaging period (and the filter length). The operation has a side effect of delaying the signal by the amount of time required for blurring.

5. Resynthesis

Resynthesis of spectral data can be performed in a variety of ways, belonging to two basic categories, overlap-add inverse-DFT and additive synthesis.

5.1 Overlap-add

This is generally the most efficient way of resynthesising bin-frame amplitude-frequency data. It is performed by `pvsynth` opcode. This takes the amplitude and frequency pairs, integrates the frequencies to obtain the current phases, converts to rectangular data and applies an inverse DFT. The resulting time-domain signal block is then overlap-added to the correct time-aligned position at the output.

Partial tracks cannot be fed directly to the overlap-add resynthesis, but can be converted into bin-frame data. This conversion is performed by the `binit` opcode, which generates a frame of amplitude and frequency bins based on the track data input. This conversion is lossy, as only one track per bin is allowed into the output, but masking effects are taken into account, so that the resulting signal is perceptually similar to the track input.

5.2 Additive synthesis

Additive synthesis can be applied to bin-frame data or partial tracks, but, generally speaking, it is more suited to the latter. Richard Dobson contributed an additive resynthesis opcode, `pvsadd`, to the original set of `fsig` opcodes, which is reasonably fast, but producing a medium to low quality (due to the lack of interpolation) resynthesis of bin-frame data.

Partial track additive synthesis can, however, be more efficient and offers better quality. There are three additive opcodes in Csound 5 for track data: using linear (`tradsyn`) and cubic phase interpolation (`sinsyn` and `resyn`). Of the three, `tradsyn` is the most efficient and flexible, as it depends only on amplitude and frequency track data. The cubic-phase opcodes will have better fidelity in signal reconstruction, but will be slower, and in the case of `sinsyn`, will not allow any type of frequency (or timescale) transformations of the original analysis data.

6. Future Prospects

The `fsig` framework in Csound5 and its spectral opcodes provide a comprehensive, flexible and intuitive way to build frequency-domain effects computer instruments (fig.3). In addition to the existing unit generators, it is expected that new ones will be added to the set. For instance, a track morphing opcode, based on graph matching techniques, is under development by the

authors. Also, it is important to mention an interesting work on a sliding DFT analysis/resynthesis method (ffitch et al, 2005), which will eventually be incorporated into the system.

7. Acknowledgements

The authors would like to acknowledge the work of Richard Dobson in the development of the `fsig` framework and opcodes for Csound 4.13. Without this work none of the Csound 5 spectral unit generators described in this article would have been possible. It is also important to mention the contribution of John ffitch and Istvan Varga, among others, to the development of the Csound 5 infra-structure, which supported the addition of these opcodes.

References

- John ffitch. 2005. On the Design of Csound5, *Proceedings of the 3rd Linux Audio Conference*, ZKM, Karlsruhe, Germany, 37-42
- Barry Vercoe. 1986. Csound: A Manual of the Audio Processing System. MIT.
- Victor Lazzarini. 2005. Extensions to the Csound Language: from User-Defined to Plugin Opcodes and Beyond, *Proceedings of the 3rd Linux Audio Conference*, ZKM, Karlsruhe, Germany, 13-20.
- Victor Lazzarini, Joe Timoney and Tom Lysaght. 2005. Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking. *Proc. of ICMC2005*, Barcelona, Spain, pp 197-200
- Mark Dolson. 1986. The phase vocoder tutorial, *Computer Music Journal*, 10(4), pp. 14-27, 1986.
- Victor Lazzarini, Joe Timoney and Tom Lysaght. 2005. Alternative analysis-synthesis approaches for timescale, Frequency and other transformations of Musical Signals. *Proc. of DAFx05*, pp. 18-23.
- Vincent Verfaillie and Philip Depalle. 2004. Adaptive Effects Based on STFT, Using a Source-Filter Model. *Proc. of DAFx04*, pp. 296-301
- Axel Roebel and Xavier Rodet. 2005. Real Time Signal Transposition with Envelope Preservation in the Phase Vocoder. *Proc. of ICMC2005*, Barcelona, Spain, pp. 672 – 675.
- Xavier Serra. 1997. Musical Sound Modelling with Sinusoids plus Noise. G.D. Poli an others (eds.), *Musical Signal Processing*, Swets & Zeitlinger Publishers, 1997.
- Trevor Wishart. 1995. *Audible Design*, Orpheus the Pantomime, York, 1995.
- John ffitch, Richard Bradford and Richard Dobson. 2005. Sliding is smoother than jumping. *Proc. of ICMC2005*, Barcelona, Spain, pp.287-290.