STREAMING FREQUENCY-DOMAIN DAFX IN CSOUND 5

Victor Lazzarini

Music Technology Laboratory
National University of Ireland, Maynooth
victor.lazzarini@nuim.ie

Joe Timoney, Tom Lysaght

National University of Ireland, Maynooth
Department of Computer Science
{Joseph.Timoney|Tom.Lysaght}@cs.nuim.ie

ABSTRACT

This article discusses the implementation of frequency domain digital audio effects using the Csound 5 music programming language, with its streaming frequency-domain signal (fsig) framework. Introduced to Csound 4.13, by Richard Dobson, it was 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 DAFx made possible by these new opcodes.

1. INTRODUCTION

Csound 5 [1], released in February 2006, is a completely re-coded version of the popular MUSIC N-derived music programming language [2]. 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 1000. These include a set of opcodes designed to work with spectral signals, defined by the Csound orchestra language type $fsig$.

This signal type [3] 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. THE FSIG FRAMEWORK

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. Fsigs are self-describing, so, unlike time-domain audio and control signals, are furnished with the extra information about their feature, including: DFT length, number of overlaps (N/hopsize), window size, window type and data format.

2.1. Data Formats

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 [4]. 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 actually holds frames 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 binframe 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 [5]. The `pvsanal` opcode does this operation in a streaming fashion, generating a new frame every hopsize input samples. The following example takes the channel 1 input signal to Csound and produces a fsig with a framesize of 1024 (513 bins), updated every 256 samples, using a 1024-sample hanning window.

```
asig  inch     1
fs1   pvsanal  asig, 1024, 256, 1024, 1
```

4. INSTANTANEOUS FREQUENCY DISTRIBUTION

The `pvsifd` opcode implements the instantaneous frequency distribution analysis [6]. It actually 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. The following example is equivalent to the previous one, except that the opcode generate two fsigs as output and uses an analysis window that is always the same size as the DFT.

```
asig      inch    1
fs1,fs2   pvsifd  asig, 1024, 256, 1
```

4.1. 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. The first example outputs a complete (amplitude, frequency and phase) partial track signal, with an analysis threshold of 0.003 (0.3%) , a minimum number of 1 time point for each track and a maximum gap of 3 time-points for track continuity. It limits the output to a maximum of 500 tracks:

```
asig      inch     1
fs1,fs2   pvsifd   asig, 1024, 256, 1
ftrk      partials fs1, fs2, 0.003, 1,3,500
```

However, as previously discussed, it is also possible to feed a amplitude/frequency-only input to partial track analysis, resulting in tracks with no phase information. The following example takes its input from a PVOCEX analysis file, using the file reader opcode:

```
ktim  line      0, idur, idur
fs1   pvsfread  ktim, "input.pvx"
ftrk  partials  fs1,fs1, 0.003,1 ,3,500
```

5. SPECTRAL PROCESSING

In this section, we will look at the different types of transformations, loosely classified in amplitude, frequency and combination effects, as discussed in [7].

5.1. 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, kpks is number of spec peaks */
afil oscili 2, kpks/2, 1
/* table writing (table size=44100) */
     tabw    abs(afil), aphs, 3, 1
ffil trfilter ftrk, 1, 3 /* filtering */
```

In the example above, if the signal kpks 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. For instance, a very narrow band-pass filter can be created by using the spectra of a sinusoid and the spectra of an arbitrary source as inputs:

```
asig inch     1  /* input */
asin oscili 1, kcf, 1 /* sinusoid */
/* input spectral signal */
fs1  pvsanal  asig, 1024, 256, 1024, 1
/* sinus spectral signal */
fs1  pvsanal  asin, 1024, 256, 1024, 1
/* filtering */
ffil pvsfilter asig,asin, 1
```

Also in the category of amplitude transformations we have the mask-based effects, such as noise cancellation that can be performed by the `pvstencil` 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:

```
asig   diskin2  "input.wav", 1,0,1
ifn    ftgen 1,0, isiz/2,-43, "noise.pvx"
fsig   pvsanal  asig, isiz,isiz/4,isiz,iw
fclean pvstencil fsig, kattn, klvl,ifn
```

In the example above, the amount of amplitude change is controlled by `kattn`, and the noise threshold can also be adjusted by the `klvl` variable.

5.2. 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 binframe signals is described by the following expression:

$$f_{\text{out}}[np] = f_{\text{in}}[n]p \tag{1}$$

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 [8], but many times more efficient and yielding good results. A comparison of the output and input of `pvscale` using formant preservation with pitch shifting demonstrates that original formants are fairly well preserved in the frequency-scaled spectrum (fig. 1).

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

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

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 eqns. (1) and (2).

5.3. 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. The example below demonstrates its use. In it, the `kpos` variable controls the pan position for signal extraction (-1 to 1) and kwidth controls the 'search width' around that position. The demix operation works on the basis of 100 discrete pan positions on each side of the stereo image.

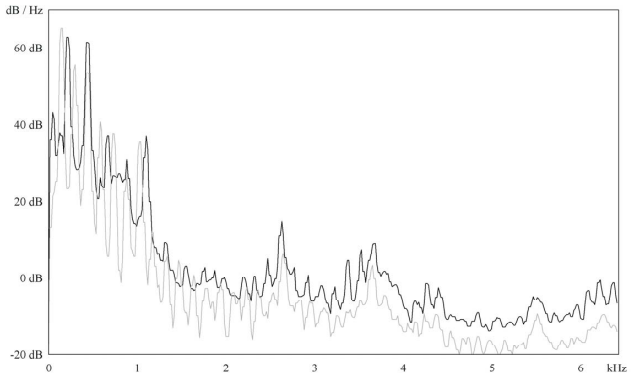

Figure 1: *Comparison between original (grey) and transposed (black) spectra. A good amount of formant preservation is clearly seen in the picture.*

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 [9].

Another effect that involves the transformation of both frequency and amplitude is that of spectral blurring [10], based on time-averaging the amplitude and frequency spectral functions. The effect is implemented by the `pvsblur` opcode, which takes in a 'blur time' parameter, defining the averaging period.

6. RESYNTHESIS

6.1. 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 ovelap-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.

6.2. 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.

7. CONCLUSION AND FUTURE PROSPECTS

The fsig framework in Csound5 and its spectral opcodes provide a comprehensive, flexible and intuitive way to build frequencydomain effects computer instruments. It is expected that new opcodes will be added to the exiting set. Also, it is important to mention the work on a sliding DFT analysis/resynthesis method by ffitch and Dobson [11], which will eventually be incorporated into the system.

8. 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. 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.

9. REFERENCES

[1] J. ffitch, "On the design of Csound 5," in *Proc. 3rd Linux Audio Conf.*, 2005, pp. 37–42.

[2] V. Lazzarini, "Extensions to the Csound language: from user-defined to plugin opcodes and beyond," in *Proc. of the 3rd Linux Audio Conf.*, 2005, pp. 13–20.

[3] V. Lazzarini, J. Timoney, and T. Lysaght, "Alternative analysis-synthesis approaches for timescale, frequency and other transformations of musical signals," in *Proc. Int. Conf. on Digital Audio Effects (DAFx-05),* Madrid, Spain, 2005, pp. 18–23.

[4] V. Lazzarini, J. Timoney, and T. Lysagh, "Time-stretching using the instantaneous frequency distribution and partial tracking," in *Proc. Int. Comp. Music Conf. (ICMC'05),* Barcelona, Spain, 2005, pp. 14–27.

[5] V. Verfaille and P. Depalle, "Adaptive effects based on STFT, using a source-filter model," in *Proc. Int. Conf. on Digital Audio Effects (DAFx-04),* Naples, Italy, 2004, pp. 296–301.

[6] X. Rodet and A. Röbel, "Real time signal transposition with envelope preservation in the phase vocoder," in *Proc. Int. Comp. Music Conf. (ICMC'05),* Barcelona, Spain, 2005, pp. 672–675.

[7] J. ffitch, R. Bradford, and R. Dobson, "Sliding is smoother than jumping," in *Proc. Int. Comp. Music Conf. (ICMC'05),* Barcelona, Spain, 2005, pp. 287–290.

[8] M. Dolson, "The phase vocoder: a tutorial," *Computer Music J.*, vol. 10, no. 4, pp. 14–27, 1986.

[9] B. Vercoe, *Csound: A Manual of the Audio Processing System*. MIT Media Lab, 1986.

[10] T. Wishart, *Audible Design*. Orpheus the Pantomime, York, 1996.

[11] X. Serra, *Musical Signal Processing*. G. D. Poli and A. Picialli and S. T. Pope and C. Roads Eds. Swets & Zeitlinger, 1996, ch. Musical sound modeling with sinusoids plus noise, pp. 91–122.