Return to Index

A Prototype Real-time Plugin Framework for the Phase Vocoder

Richard Dobson

Media Technology Research Centre, University of Bath, Bath, BA2 7AY

Within the last few years, consumer-level computers have reached levels of processing power that at last enable the real-time capability of the phase vocoder to be realized. It remains important to implement the phase vocoder as efficiently as possible, but this capability raises new issues relating to the design and implementation of streaming audio systems in general. Where a series of transformations is to be applied to a sound, it is clearly inefficient to chain complete phase vocoders in series, where the obvious solution is to apply the transformations themselves in series, inside a single analysis-resynthesis framework. The fact that this frequency-domain processing can now easily be done in real-time suggests that such a framework is no longer speculative, and consideration can be given to practical questions of design, implementation, and, in particular, integration with existing framework paradigms.

2. Implementation of a real-time Phase Vocoder

2.1 Core analysis parameters.

As noted above, the analysis and resynthesis performed by the phase vocoder is based on the use of the FFT, which in the forward transform converts a block of time-domain samples (typically a power-of-two size such as 1024, for optimum speed) into an analysis frame of complex (real, imaginary) data representing the detected frequency components of the signal (Dolson 1986, Jaffe 1987). The inverse transform converts this (possibly modified) frame back into the time domain.

a conventional soundfile, and there is no technical reason why such a framework cannot combine frequency-domain and time-domain tracks within a project.

These are core timings for the self-contained phase vocoder, which is written mostly as inline code, and which relies on short source sounds. As will be shown, the requirements of a plugin framework, and modifications for reliable continuous real-time performance, degrade these figures slightly, fortunately not so much as to give problems in practice.

2.3 A simple plugin framework.

The model developed here is of an initial analysis stage, which feeds analysis frames through one or more plugin transformations, selected and controlled by the composer, and resynthesized at the end. For simplicity, it is assumed that each plugin will over-write the incoming frame. A key principle of this model is that the analysis and resynthesis engines are independent, and may indeed be developed as such. Thus the format of the analysis frame needs to be specified formally. The approach taken here is to define a new streamable file format for analysis data, which can thus also define the format for analysis frames streamed through a framework. A C++ class wrapper has been developed from the original C code, facilitating not only the dynamic insertion of plugins, but also the creation of multi-channel processors and alternative resynthesis engines such as an oscillator bank. The analysis and resynthesis objects are created separately, and in accordance with the principles of object-oriented programming, do not share private data. They need not, in principle, be implemented identically. Where processing speed permits, the use of an oscillator bank for resynthesis may increasingly be preferred, especially where extensive pitch-related transformations are being used (Moore 1990: 247).

Three VST and DirectShow plugins have been implemented for this project, based on a simple C++ abstract base class, and drawing on algorithms by Trevor Wishart (Wishart 1994); they are available for free download, for both PC and Macintosh platforms (Dobson 2001). They demonstrate amplitude-only, frequency-only, and combination transforms, respectively:

pvexag:	exaggerates the spectral envelope (fourfold overlap)
pvtransp:	pitch shift +- one octave (sixfold overlap)
pvaccu:	spectral accumulation, with glissando (fourfold overlap)

The increased overlap for **pvtransp** reflects the need to ensure a reasonably clean pitch shift over half an octave.

2.3.1 The PVOC_EX file format.

Each phase vocoder implementation studied in the course of this project has implemented its own file format, in most cases without regard to portability, and in some cases with little or no format information. This is despite that fact that the data is virtually identical (the main difference being in the amplitude scaling employed). A Csound analysis file generated on an Intel platform cannot be used by Csound running on a big-endian platform such as the Macintosh. The CDP file formats are portable, being based on the WAVE and AIFF formats, but support only mono sounds, and their use is inappropriate now that the 32bit floating-point sample type is fully defined for both formats.

PVOC_EX has been developed as a robust, portable and streamable file format which can be supported by ati.76 -032b1147my formats, enabling use in a variety of contexts. Full details of the specification, together with downloadable command-line implementations for the Windows and Linux platforms, have been published on the Internet (Dobson 2000b). PVOC_EX is intended as a replacement for CDP's current file formats, and support for it is also being added to Csound. Prototype programs have been developed to convert the Csound and PVC formats to PVOC_EX. By being based on a streaming audio format, it can also be regarded as the specification for analysis data streamed through a plugin framework, as described below.

2.3.2 Real-time considerations.

A key element of the overlap-add resynthesis technique is the maintenance of a running phase, incrementing as each frame is processed:

```
for(i=0, i0=syn, i1=syn+1; i<= NO2; i++, i0+=2,  i1+=2){
  mag = *i0;
  oldOutPhase[i] += *i1 - ((float) i * F);
  phase = oldOutPhase[ i] * TwoPioverR;
  *i0 = (float)((double)mag * cos((double)phase));
  *i1 = (float)((double)mag * sin((double)phase));
}
```

[CARL: pvoc.c]

This technique assumes the ability of the standard C sin and cos functions to return a correct result for an input phase value outside the nominal range. For the short sounds typically used with disk-based processing, this assumption is reasonable. However, in a prototype implementation using a SHARC dsp chip, without the benefit of double-precision processing, degradation of the audio was easily apparent after as little as ten minutes. Clearly, for stability over long periods, the phase calculation must be refined to keep it within bounds. The obvious, if costly, solution is to apply normalization to the phase value each time:

```
oldOutPhase[ i] += *i1 - ((float) i * F);
oldOutPhase[ i]  =  fmod(oldOutPhase[ i], TWOPI)
```

As, even in the SHARC implementation, a small amount of range error in phase can be tolerated, the otherwise unacceptable extra processing burden incurred by this solution can be mitigated by applying the correction incrementally by analysis bin over successive analysis frames. This entails a change to ensure both that the smallest values are accumulated, and that both corrected and uncorrected values are valid:

```
float angledif, phase;
angledif = TwoPioverR * (*i1  - ((float) i * F));
phase = *(oldOutPhase + i) +angledif;
if(i== bin_index)
        phase = (float) fmod(phase,TWOPI);
*(oldOutPhase + i) = phase;
```

This leads to each bin receiving a correction at the rate:

(overlap * (N/2)+1) / SR   secs.

Thus, at SR = 44100, with N = 1024 and overlap = 128, each bin is corrected every 1.489 seconds, well within the safe range.

With this change, and a with few other minor adjustments to the CARL code, the prototype plugins have been run without audio degradation for over eight hours, with an overall penalty in processing time of a modest 3 percent. This is increased to around ten percent, (referenced to Table 1) firstly by the overhead of the conversion to C++, and secondly by the need to support time-domain sample blocks of arbitrary size, currently requiring a method call (albeit inline) every sample.

The primary host application used for testing the VST plugin implementations was the shareware program AudioMulch, by Ross Bencina (http:/www.audiomulch.com). This provides an explicit percentage report, as plugins are running. The availability of low-load synthesis sources, combined with a static display, enables plugins to be tested with a minimum of overhead. Table 2 lists some representative measurements for monophonic processing at the 44100 sample rate, using the same platform as given in Table 1. It can be seen that for a given FFT size, CPU load is almost exactly proportionate to the overlap size, indicating that the phase vocoder can be used in this way with a high level of predictability.

2.3.3 Future prospects for the phase vocoder.

The examples described above have been realized on what even today can be considered a low-powered workstation, and rely on a sizeable latency, through the use of the overlap-add technique. However, it is most important to note that this is not the only way to implement a phase vocoder, and that this latency is no more than a consequence of the need to reduce computational cost. In a far-reaching article, James Moorer (

otherwise all connected processing nodes need to receive the dimension of the signal as a parameter, either directly from the host, or, more naturally, from the source node. This is the principle underlying the DirectShow filter graph (where ‘filter’ is Microsoft’s generic name for any source, transform or sink node), in which format negotiation is handled bi-directionally by the pins which connect filters to each other. In most cases the primary determinant of format acceptance is the rendering hardware (it is especially annoying when a plugin refuses a format that the hardware can accept), while transform filters can be designed to accept (within reason) any set number of dimensions, at least so long as each dimension is of the same type. This requirement cannot be presumed to be met in all cases, and would need to be defined as part of the framework specification. The SDIF format (Wright, Chaudhary, Freed, Khoury and Wessel 1999), though not strictly speaking a

4. Conclusions.

Three examples have been given of multi-dimensional signals, demonstrating frequency domain, time domain, and control data, that it is argued the next generation of audio frameworks should seek to support. The computationally most demanding of these, the phase

Endrich, A. 1997. Composer’s Desktop Project - a musical imperative. **2**(1): 29-33.
Gerzon, M. A. 1972. Periphony: with-height sound reproduction.
21(1): 2-10.
Jaffe, D.A. 1987. Spectrum analysis tutorial, part 1: the discrete Fourier transform.
11(2): 9-24.
Malham, D. and Myatt, A. 1995. 3-d sound spatialization using ambisonic techniques.
19(4): 58-70.
McAulay, R.J. and Quatieri, T.F. 1986. Speech analysis/synthesis based on a sinusoidal representation.
34(4): 744-754.
Moore, F.R. 1990. . New Jersey: Prentice Hall.
Moorer, J. A. 2000. Audio in the new Millennium. **48**(5): 490-498.
Roads, C. 1996. Cambridge, MA: MIT Press.
Serra, X. and Smith, J.O. 1990. Spectral modelljng synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. **14**(4): 12-24.
Vercoe, B. 2000. Understanding Csound’s spectral data types. In R. Boulanger (ed.) . Cambridge, MA: MIT Press.
Wishart, T. 1988. The composition of Vox-5. **12**(4): 21-27.
Wishart, T. 1994. . York: Orpheus the Pantomime.
Wright, M., Chaudhary, A., Freed, A., Khoury,S., and Wessel, D. 1999. Audio applications of the Sound Description Interchange Format. , Audio Engineering Society.

Table 1.

Program	Implementation	Execution time (seconds)
		(average of ten runs)
plainpv	(Moore/PVC)	19.05

pvoc	(CARL/CDP)	12.0
pvocex	(pvoc/FFTW)	8.0