TUTORIAL ARTICLE

The phase vocoder: theory and practice

RAJMIL FISCHMAN

Music Department, Keele University, Keele, Staffordshire ST5 5BG, UK
E-mail: r.a.fischman@cc.keele.ac.uk

This article discusses the theoretical principles governing the short time Fourier transform (STFT) and its musical applications, using the CARL phase vocoder and the Composers' Desktop Project (CDP) spectral transformations. The former was originally developed by Dolson (1986, 1991) and ported to personal computer systems by Atkins, Bentley, Henderson, Orton and Wishart (Atkins, Bentley, Endrich, Fischman, Malham, Orton and Wishart 1987, Wishart 1994a). The latter was developed by Wishart (1994b). When possible, theoretical matters are presented in a qualitative way, keeping the number and complexity of mathematical expressions within the essential requirements for those interested in musical applications. The latter are accompanied by sound examples, including excerpts from pieces by the author.

1. THEORETICAL BACKGROUND

The mathematical principles for the analysis of signals developed by Fourier (1768–1830) have been seminal in the generation of a number of important techniques used in electroacoustic music; from linear processes such as additive and subtractive synthesis to nonlinear amplitude and frequency modulation. In its simplest form, Fourier analysis states that any periodic signal (i.e. a signal which consists of the repetition of a finite waveform) can be represented as a sum of sinewaves called *harmonics*, where each harmonic has a particular amplitude, frequency and phase. Furthermore, if the frequency of the signal is *f*, the frequencies of the harmonics will be multiples of *f* (e.g. *f*, 2*f*, 3*f*, 4*f*, 5*f*, etc.). The set of harmonics which represents a particular signal is referred to as its *spectrum*. Following this reasoning, it is possible to reproduce a known periodic sound using the appropriate combination of harmonics obtained from its analysis. It is also possible to imagine new sonorities and synthesise these by inventing new spectral combinations.

However, most natural sounds are not periodic. In fact, it is aperiodicity which imbues them with characteristic liveliness. Therefore, analysis of natural signals requires an improvement of the original Fourier approach. One way of dealing with this problem consists of imagining that the signal changes its period every instant; thus, at any given moment, a different set of harmonics may be used to represent it. Alternatively, it is possible to consider that the signal is composed of a particular set of sinusoids generically called *components* or *partials* – with amplitudes and frequencies which change in time. Regardless of which approach is adopted, the end result is a spectrum which changes as time goes by. In this case, the signal is said to have a *dynamic spectrum*.

The time-varying Fourier model may be used in order to devise techniques which change the spectral characteristics of a sound. For this purpose, a signal may be analysed by decomposition into a set of sinusoids. These may then be processed by mechanisms which alter their amplitudes and frequencies. Finally, the processed components may be used in order to resynthesise a new signal. This procedure is illustrated in figure 1.

1.1. Analysis stage

The first stage of a Fourier processing cycle is analysis. Its implementation presents two issues which much be solved. In the first place, the analysis mechanism must be able to detect each individual component at any given moment. In the second place, it must account for the fact that the spectrum changes as time goes by.

A step towards solving the first problem may consist of a device which is capable of detecting and isolating a single component. This may be achieved by means of a *band-pass* filter. If a component of frequency f_i falls within bw, the bandwidth of the filter, and all the other components of the signal are outside bw, only f_i will go through, as shown in figure 2. If f_i is present, its amplitude will be obtained from the output of the filter.

The same process may be applied to the remaining components using a bank of filters with centre frequencies which spread throughout the spectrum.



Figure 1. Typical Fourier processing cycle.

Organised Sound **2**(2): 127–45 © 1997 Cambridge University Press. Printed in the United Kingdom.

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

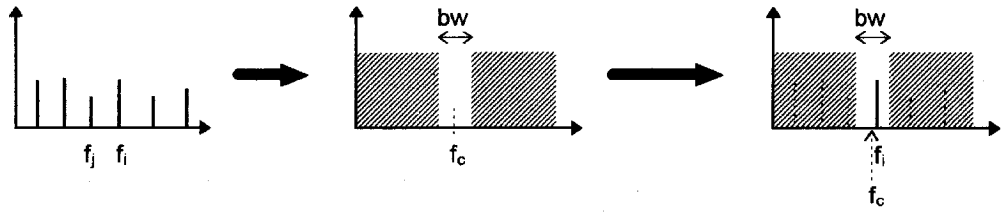


Figure 2. Use of a band-pass filter in order to detect a component with frequency f_i.

When the signal is passed through the bank, the output of each filter lets through a different component.

Ideally, the second problem may be solved by evaluating the spectrum at every instant, using the filter-bank. In practice, the spectrum may be sampled at regular intervals, as often as necessary in order to detect significant changes.

1.1.1. Detection of individual components

Determining the bandwidth of the filters requires some care. If it is too wide in comparison with the spacing of the partials, more than one partial will pass, as shown in figure 3. On the other hand, narrower bandwidths require larger numbers of filters in order to cover the whole spectrum, increasing the number of calculations performed.

The sound processing systems described in this paper do not deal with signals which are continuous functions of time, as assumed in the description of Fourier synthesis above. In practice, the digital audio systems used in computer music deal with *discrete time samples*, so that the signals can be converted into a digital format and manipulated within computer programs forming the computer music system. The *sampling theorem* states that no spectral information will be lost in this sampling process, provided that the sampling rate sr is not less than that which would result in two samples being taken per cycle of the highest frequency component present in the sampled signal. It follows that the highest frequency component which can be accurately represented in the sampling process is sr/2, also known as the *Nyquist* frequency.

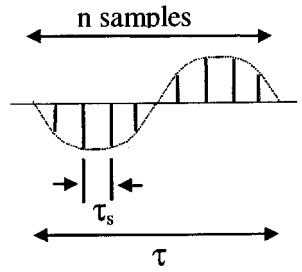


Figure 4. A sampled signal.

If the sampled signal appearing at the output of a filter is as shown in figure 4, then the period τ $(1/f)$ of the sampled waveform is given by

$$\tau = n\tau_s,$$

where τ_s (1/sr) is the time interval between samples. Hence the output frequency f is given by

(1) $$f = \frac{\mathrm{sr}}{n},$$

and

(2) $$n = \frac{\mathrm{sr}}{f}.$$

Therefore, we must ensure that a sufficient number of samples are processed in order to be able to detect the lowest partial of a signal. For example, if the lowest partial is 40 Hz and we are sampling at 4 kHz, then we need at least 4,000/40 = 100 samples. However, if the sampling rate is 40 kHz, we will need at least 40,000/40 = 1,000 samples.

A filter bank may be implemented using a *fast Fourier transform* (FFT) algorithm. This converts a

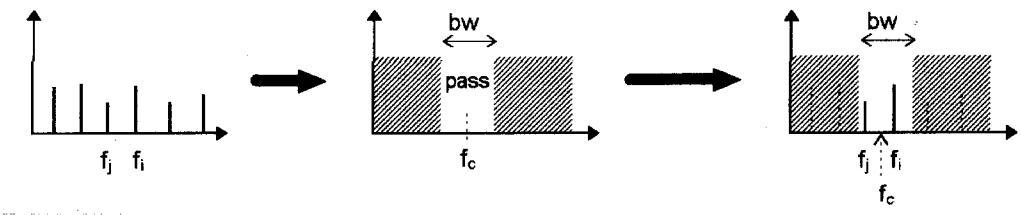


Figure 3. Filter with bandwidth which is too wide to isolate the component f_i.

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

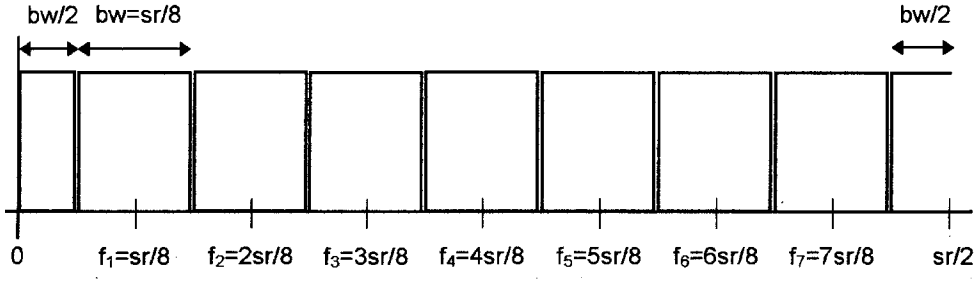


Figure 5. Bank of filters representing an ideal 16-sample fast Fourier transform.

block of 2^N samples into a spectral representation which is conceptually equivalent to the output of a linearly spaced bank of band-pass filters, as represented in figure 5. Typically, all the filters have the same bandwidth bw,[1] except for the first one, which is a low-pass with cut-off frequency equal to bw/2, and the last one, which is a high-pass with cut-off sr/2 − bw/2. The filters are normally referred to as *channels*. It is also important to note that the radix 2 FFT constrains the number of samples analysed to powers of two (e.g. 2, 4, 8, 16, 32, etc.).

In order to make effective use of the filter bank, the frequency of the lowest detectable partial may coincide with the centre frequency of the first band-pass. From equation (1), if the number of samples used is 2^N, the centre frequency of this filter is

$$f_1 = \frac{\text{sr}}{2^N} = \frac{\text{Nyquist}}{2^{N-1}}. \tag{3}$$

We can now calculate the bandwidth of the filters if we assume that the centre frequency is exactly in the middle of the pass band. Examining figure 5 and remembering that the low-pass filter has a cut-off equal to bw/2, we have

$$f_1 = \frac{\text{bw}}{2} + \frac{\text{bw}}{2} = \text{bw}. \tag{4}$$

Finally, the number of band-pass filters in the bank is

$$n_{\text{bp}} = \frac{\text{Nyquist} - \text{width of low-pass} - \text{width of high-pass}}{\text{bandwidth}},$$

and from equations (3) and (4) we have

$$n_{\text{bp}} = \frac{\text{sr}/2 - \text{bw}/2 - \text{bw}/2}{\text{bw}} = \frac{\text{sr}/2 - \text{bw}}{\text{bw}} = \frac{\text{sr}/2 - \text{sr}/2^N}{\text{sr}/2^N} = 2^{N-1} - 1. \tag{5}$$

[1] There are other techniques, such as wavelet algorithms, which produce banks of filters with bandwidths which become narrower as the frequency decreases (constant *Q*). See, for example, Kronland-Martinet (1988) and Kronland-Martinet and Grossman (1991).

If we count the low-pass and high-pass as one filter which is split, we may arrive at the following conclusion:

> *The number of samples used in order to analyse a signal determines the number and width of the band-pass filters. Using a* 2^N *sample FFT divides the spectrum into* 2^{N-1} *channels of width equal to* Nyquist/2^{N-1}.

1.1.2. Detection of spectral changes

As stated above, in order to follow changes in the spectrum of a signal as it evolves in time, it is necessary to sample this spectrum at appropriate intervals. The simplest way of carrying out this procedure consists of the application of a FFT to consecutive groups of samples, as shown in figure 6. If these 'spectral snapshots' are large, and hence spaced at regular intervals, a great deal of time-varying spectral detail may be lost. On the other hand, if snapshots are taken too often, the number of samples in each snapshot may be too small to produce reasonable resolution of the spectrum into its components (see the argument above). The spectral snapshots are referred to as as *frames* and the rate at which the spectrum is sampled (i.e. the number of frames per second) is the *analysis sampling rate*.

A further problem arises with the isolation of groups of samples from the original signal. Within that signal, the first and last samples of the sample group were preceded and followed by other samples.

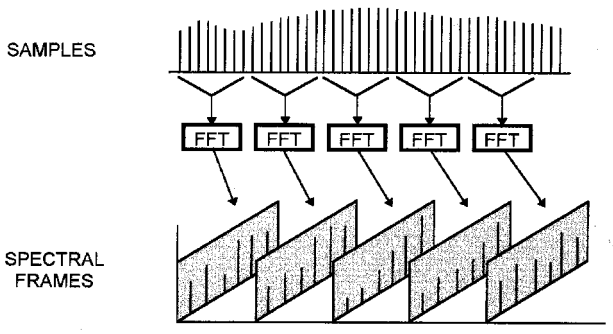


Figure 6. Fourier transform taken every eight samples.

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

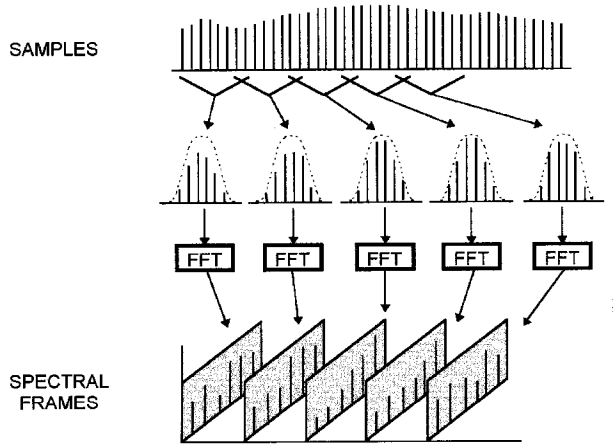


Figure 7. Short time Fourier transform (STFT) with envelope and two-sample overlap.

However, when the sample groups are analysed in isolation, these end samples are preceded and followed by nothing at all; the sample groups appear to begin and end abruptly, producing apparent 'transients' in the signal (we might hear these as clicks). Therefore the FFT derived spectrum will be distorted by the edge transients of each group. This problem may be alleviated if a few samples preceding and following the group are respectively faded in and out, smoothing the edges. In other words, groups are overlapped and given an envelope, as shown in figure 7. Overlap may also increase the resolution of the FFT, since it is possible to analyse more samples within the same temporal interval.

The process of isolating a consecutive group of samples is called *windowing*, because it resembles the action of hiding the signal behind a 'wall', using a sliding widow which restricts the view to a limited number of samples at a time. The shape of the envelope applied to the samples determines the *window type* and the number of samples in a window is the *window length*. The procedure consisting of FFT windowed spectral sampling at regular intervals is known as the *short time Fourier transform* (STFT).

The most common types of window used in STFT processes correspond to one of the following types: Bartlett, Blackmann, Hanning, Hamming, and Kaiser. The first of these is just a triangular envelope (figure 8(b)). Hanning is a raised cosine bell (figure 8(c)). Blackmann and Hamming are variants of the cosine bell and Kaiser is a more complex window based on a special function (prolate spheroidal), which allows tighter control of the shapes of the FFT band-pass filters. If no envelope is applied to the samples, the window is said to be rectangular (figure 8(a)).

Overlap is measured using a *decimation factor*, which indicates the number of overlaps throughout the duration of a single window. Figures 9(a) and (b) show respectively decimation factors of 2 and 4 (overlap of 2 and overlap of 4 windows).

It is now possible to find a relationship between window length and the original sampling rate of the sound signal. If the windows do not overlap (decimation factor $D = 1$), the analysis sampling rate asr is simply

$$\text{asr} = \frac{\text{sr}}{\text{window length}} = \frac{\text{sr}}{2^N}. \tag{6}$$

If $D > 1$ (the windows overlap) then the analysis sampling rate is effectively increased. For example, if $D = 2$, the window start times are separated by half the window width. Hence the analysis sampling rate doubles compared with the non-overlapping case. We can write this as

$$\text{asr} = \frac{\text{sr}}{2^N} \times D. \tag{7}$$

The discussion above may be summarised as follows:

The window length and decimation factor determine the time resolution of the STFT used to measure the spectral evolution of a signal.

1.1.3. Real filters – leakage

The filters considered so far are an idealisation of a desired frequency response. In real filters, the transition between stop and pass is not immediate – it has a slope. In addition, the stop region has ripples which let through a small percentage of the unwanted part of a signal. This is known as *spectral leakage*. The amount of leakage depends upon the type of window in use.

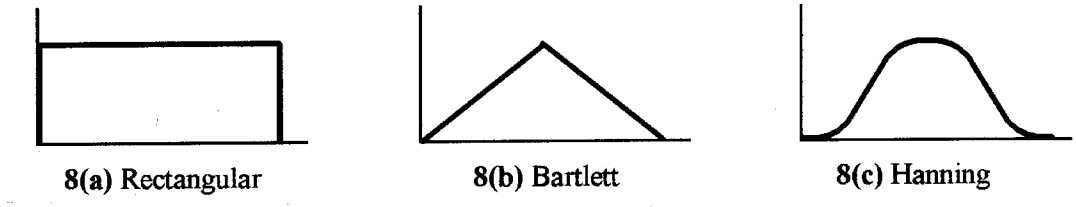


Figure 8. STFT window types.

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press



9(a) Decimation factor = 2

9(b) Decimation factor = 4

Figure 9. Examples of multiple window overlap.

1.2. Processing stage

The set of spectral frames produced by the STFT provides a description of the way the spectrum of a signal changes in time. This type of information allows processing procedures which may not be possible by other means. In order to illustrate this, we may imagine a periodic sound which must be transposed an octave upwards. This could be achieved by sampling the sound at a certain sampling rate and playing it at twice that sampling rate. Therefore, if the original period repeats f cycles in one second, doubling the sampling rate will repeat $2f$ cycles in one second, doubling the frequency of the fundamental and raising the pitch by an octave. However, the new sampling rate uses twice as many samples every second; therefore, the duration will be halved.

On the other hand, the duration of a STFT representation depends on the number of frames per second used in the analysis. Therefore, if this is not altered, the duration will remain the same. All that is required is to transpose each frame by an octave, which is done by doubling the frequency of each component, i.e. multiplying it by 2. Generalising this process, it is possible to transpose a signal by any ratio without altering its duration. Also, the pitch may vary in time by varying the transposition ratio accordingly.

Furthermore, the number of frames may be manipulated in order to *time-stretch* a sound, altering its duration without changing its pitch. For example, duration may be doubled by repeating each frame, or halved, by omitting every other. Time-stretch by repetition may result in a mechanical effect similar to echo, which may be pronounced when multiplying the duration by a large factor. More accurate doubling of the duration may be achieved by inserting a new frame between two 'old' ones. The component amplitudes and frequencies of each new frame may be found by averaging the respective amplitudes and frequencies of the preceding and following old frames. More sophisticated interpolation procedures may be used in order to time-stretch by any factor or make this factor a time-varying quantity.

1.3. Resynthesis stage

The FFT has a counterpart – the *inverse fast Fourier transform* (IFFT) – which converts spectral information into samples. Therefore, it is possible to devise a procedure which reverses the analysis stage using the IFFT in order to convert the spectral frames into a time-sampled signal, by converting spectral frames into overlapped windows with samples.

1.4. Choice of parameters for musical applications

The success of a specific process depends on the appropriate choice of STFT parameters. We have already seen that it is important to ensure that the bandwidth of the FFT filters matches the desired spectral resolution, since band-pass filters which are too wide may not be effective in separating frequency components. Given a desired resolution Δf, the number of filters required may be determined from equation (5) and the conclusions of section 1.1.1, as follows:

$$n \geqslant \frac{\text{Nyquist}}{\Delta f}. \tag{8}$$

For example, in order to obtain a channel bandwidth of 10 Hz at a sampling rate of 44.1 kHz (Nyquist = 22.05 kHz), the number of filters must be equal or larger than 22,050/12 = 1,837.5; the lowest power of two which fulfils this condition is 2,048.

Another aspect which determines spectral resolution is the fact that human hearing works on a logarithmic scale while the STFT is linear. This results in poorer resolution at lower frequencies. Following the example above, a resolution of 10 Hz may be appropriate for signals with components above 1,000 Hz, in which case the resulting error of 5 Hz is 0.5 per cent, corresponding to less than a twelfth of a semitone. However, for a component of 100 Hz, the error is 5 per cent, close to a semitone. At 50 Hz the error is two semitones, and so on.

Finally, since the number of channels is determined by the length of the analsyis window (see conclusions in sections 1.1.1 and 1.1.2), a large number of filters will require larger window lengths, producing less frames per second. This will affect the accuracy of detection of spectral changes. In the example above, the number of samples required in order to implement 2,048 ($=2^{11}$) filters is 4,096 ($=2^{12}$). Assuming a decimation factor of 8, using equation (7) and rounding the result to the nearest integer, the number of frames analysed in one second is 86.

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

Doubling the window length will halve the number of frames to 43. Using the latter window length and halving the decimation factor will halve the last result to 22 frames, which may be too low for fast evolving portions of a sound, such as attacks and transients.

To summarise, the chosen number of filters (or the equivalent window length) is a compromise between frequency and time resolution. The former is determined by the bandwidth of the channels and the latter by the number of frames per time unit. Choice is also influenced by the particular process applied and the nature of the sound. Normally, slow changing sounds may be better analysed using higher spectral resolution, particularly if they are quasi-periodic and do not contain a large amount of high-frequency partials. Lower frequencies require even higher spectral resolution. On the other hand, fast-changing spectra may require higher temporal resolution. Time-stretching processes may favour temporal resolution while purely spectral operations, such as transposition, may require more accurate frequency discrimination. In practice, musical applications using sampling rates near 44.1 kHz require between 256 and 4,096 filters.

2. THE CARL PHASE VOCODER

The CARL phase vocoder (Dolson 1986, 1991, Wishart 1994a)[2] consists of software which implements basic STFT operations – such as filtering, time-stretch and transposition (pitch-shift) – on soundfiles. It also enables the implementation of more complex procedures by producing analysis files which may be processed by other software. As long as the output of further processing conforms to the format for analysis files, the processed signal may be resynthesised using the phase vocoder again.

Channels are numbered in ascending frequency order, starting from 0, corresponding to the low-pass filter, followed by 1 (first band-pass), 2 (second band-pass), etc. If there are 2^{N-1} filters, the number corresponding to the high-pass filter will be 2^{N-1} (see figure 5).

2.2. Analysis file format

Spectral data is stored as a succession of spectral frames, each with a set of amplitude/frequency floating point numbers representing the channels. Both values vary from frame to frame: the value of the amplitude depends on the relative strength of each partial and the calculation of the frequency takes into account changes in phase. The term *bin* describes a single channel which changes its amplitude and frequency throughout a succession of spectral frames.

In addition to spectral data, analysis files store information about the type of FFT process carried out, such as sampling rate, window length and type, analysis sampling rate, decimation, etc.

2.3. Phase vocoder commands

The CARL phase vocoder, ported to the PC by the Composers' Desktop Project (Atkins *et al*. 1987, Wishart 1994a), currently runs in an MS-DOS environment or equivalent. It is invoked by typing a command with the following general syntax:

pvoc [*flags*] *input output*

This command may be used in three basic modes:

(1) Implementation of a full Fourier processing cycle involving analysis and resynthesis. In this case, the input and the output are both files containing sampled sounds.
(2) Analysis only. In this case, the input is a file containing sampled sound and the output is a STFT analysis file.
(3) Synthesis only. In this case, the input is a STFT analysis file and the output is a file containing samples.

The flags are used for the following purposes:

(1) Defining the mode. For example, -A is used for analysis only and -S for synthesis only.
(2) Setting STFT parameters. For example, -N sets the number of band-pass filters and -D the decimation factor.
(3) Carrying out a specific Fourier processing cycle. For example, -P is used to transpose a sound and -T to time-stretch.

Typing *pvoc -h* provides a user message with a list of flags. Further information may be obtained from the Composers' Desktop Project (CDP) manual (Wishart 1994a).

2.4. Processing of stereo soundfiles

In its current form, PVOC can only be applied to mono soundfiles. In order to process stereo sounds, it is possible to use two commands from the CDP 'Groucho' sound processing programs – originally produced by Bentley (Dobson and Endrich 1994) – in order to implement the following process:

(1) Channel separation, using CHANNEL. This produces two soundfiles, one containing the left stereo channel and the other containing the right channel.
(2) Phase vocoder processing of left and right. This produces two new soundfiles, one for the left channel of the output and one for the right.

[2] See Moorer (1978) and Gordon and Strawn (1985) for other implementations of the phase vocoder.

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

(3) Combination of output left and right into one stereo soundfile using INTERLEAVE.

This process may be automated using a batchfile, such as PVTSST.BAT, which is listed below:

```
REM PVTSST.BAT

REM Usage: pvtsst in-file-name out-file-name stretch-factor

REM channel separation

channel -ot%1.wav 1 2

REM Time-stretch

pvoc -N512 -T%3 t_c1.wav x_c1.wav

pvoc -N512 -T%3 t_c2.wav x_c2.wav

REM interleave

interl %2.wav x_c1.wav x_c2.wav

REM Clean leftovers

del t_c1.wav

del t_c2.wav

del x_c1.wav

del x_c2.wav
```

In order to time-stretch the soundfile f1 by a factor of 3.2, creating the soundfile f1x, we need to type

```
pvtsst f1 f1x 3.2
```

3. COMPOSERS' DESKTOP PROJECT (CDP) SPECTRAL TRANSFORMATIONS[3]

This consists of a suite of programs implemented by Wishart (1994b) which makes use of analysis files produced by the phase vocoder in order to expand the available spectral processing palette. A typical STFT processing cycle using the spectral transformations consists of the following steps:

(1) Analysis of the input using PVOC.
(2) Processing using a CDP spectral transformation.
(3) Resynthesis using PVOC.

Detailed information about each program may be found in the manual (Wishart 1994b).

3.1. Use of batchfiles

The process above requires at least three commands. Typing each of these every time a particular operation is applied to a soundfile may become tedious and introduce errors. In addition, it may be necessary to delete intermediate files created by the whole procedure in order to save hard disk space, since analysis files are usually large. Therefore, it is useful to automate a generic process using batchfiles. As shown above, this may also be extended to the treatment of stereo files.

[3] © Trevor Wishart and the Composers' Desktop Project (CDP), c/o 12 Goodwood Way, Cepen Park South, Chippenham, Wiltshire SN14 0SY, UK. Tel: +44 1249 461361.
E-mail: tendrich@cix.compulink.uk

4. MUSICAL EXAMPLES

The following set of examples demonstrates various processes which may be achieved using the phase vocoder on its own and in combination with the CDP spectral transformations. The original sounds were sampled at 44.1 kHz. Audio examples appear on the annual *Organised Sound* CD. The reader is encouraged to process the original sounds with the batchfiles and commands provided in this article, which should produce the same results as those on the CD.

4.1. High-resolution filtering

The phase vocoder offers two optional flags: *-i*X causes resynthesises from the *X*th channel, omitting any channels below *X*; *-j*Y causes resynthesises up to the *Y*th channel, omitting any channels above it. Equations (3) and (4) may be used in order to calculate f_{ch}, the centre frequency of channel ch. Assuming 2^{N-1} filters,

$$f_{ch} = ch \times \frac{Nyquist}{2^{N-1}} = ch \times \frac{sr}{2^N}. \tag{9}$$

Alternatively, it is also possible to find a channel which corresponds to a given frequency f_{ch}:

$$ch = \frac{f_{ch} \times 2^{N-1}}{Nyquist} = \frac{f_{ch} \times 2^N}{sr}. \tag{10}$$

Audio example 1 demonstrates the use of the phase vocoder as a sharp band-pass filter. A middle A (*c.* 440 Hz) played by an oboe is followed by its fundamental, second harmonic, third harmonic, a combination of fourth to eighth harmonics and a combination of harmonics nine and above.

Isolation of the fundamental was achieved using PVOC as a low-pass filter with cut-off slightly over 440 Hz. Replacing $f = 440$ Hz, $2^{N-1} = 2{,}048$ filters and Nyquist = 22,050 Hz in equation (10) and rounding the result to the nearest integer, the channel corresponding to 440 Hz is 41. Therefore, the PVOC command uses *-j*42. The second harmonic was obtained using a band-pass filter between channels 80 ($=2 \times 40$) and 84 ($=2 \times 42$). The last sound was produced using the phase vocoder as a high-pass filter. The phase vocoder was applied to the soundfile PVX1.WAV in order to obtain the various filtered versions using the following commands:

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

```
pvoc -N4096 -j42 PVX1 PVX11

pvoc -N4096 -i80 -j84 PVX1 PVX12

pvoc -N4096 -i120 -j128 PVX1 PVX13

pvoc -N4096 -i160 -j332 PVX1 PVX148

pvoc -N4096 -i336 pvx1 PVX19+
```

It should be mentioned that a spectral split into bands may be achieved using the phase vocoder in combination with SPECSPLI (Wishart 1994b), a CDP spectral transformation which acts as a filter and also allows transposition and amplitude scaling of the resulting frequency band.

4.2. Time-stretch

A special flag -T may be used in conjunction with a factor in order to scale the duration of a sound. If the factor is larger than one the duration will be extended; a factor smaller than one will shorten the sound. A factor of 1 will produce no change. PVOC implements this type of time-stretch by adding or omitting frames (see section 1.3).

Audio example 2 contains a sound followed by a time-stretch of 4.1. Since the original is a stereo soundfile (PVX2.WAV), the process was carried out using the batchfile PVTSST.BAT – listed in section 2.4 – and the following command:

```
pvtsst PVX2 PVX2X 4.1
```

It should be noticed that in this case the number of filters used was 256 (option -N512), which favoured temporal accuracy of the STFT at the expense of spectral resolution, as explained in section 1.4.

When carrying out a time-stretch, some care must be taken with fast-changing spectra. For instance, extending the duration of a sound by a factor may smear its attack. This was not an issue in audio example 2, since the sound fades in. On the other hand, audio example 3 presents a case in which smearing is critical. It contains a group of sonorities resembling cracks (soundfile PVX4.WAV) which are followed by a time-stretch, again by a factor of 4.1, using the following command:

```
pvtsst PVX3 PVX3X 4.1
```

In order to avoid attack smearing, PVOC may be used in conjunction with SPECSTR (Wishart 1994b), a program which implements variable time-stretch, given as a set of time vs stretch-factor breakpoints in a function file. Audio example 5 contains another group of cracks (PVX4.WAV) which are followed by a time-varying transposition which preserves the attacks: the function file ensures that these are time-stretched by 1 (no change), while the portions between attacks are stretched by variable factors up to 150, affecting the decay and the interval between the cracks. An additional advantage of SPECSTR is that time-stretch is carried out using interpolation instead of simple frame repetition or omission; therefore, the resulting output is less likely to have the 'echo effect' described in section 1.2.

The process was implemented using the batchfile PVTSS.BAT, listed in Appendix A1. The command for this batchfile requires the name of the input soundfile and the name of the function file. The output of PVTSS.BAT has the same name as the function file with the extension '.WAV'. The command and function file used in audio example 4 are listed below:

```
command:        pvtss PVX4 PVX4X

function file:  0       1       <-\
                0.034   1       <-/  preserves attack
                0.07    125
                0.12    1       <-\
                0.653   1       <-/  preserves attack
                0.734   100
                1       1       <-\
                1.125   1       <-/  preserves attack
                1.16    150
                1.06    1
```

A further example of variable time-stretch may be found in Fischman (1994).

4.3. Transposition

PVOC allows direct transposition of a soundfile using the flag -P immediately followed by a transposition factor, which is used to scale the frequency of each bin. Audio example 5 compares transposition by scaling the frequency of every FFT frame with transposition by changing the sampling rate (resampling). The former preserves the duration of the original while the latter results in a shorter duration. The sounds are presented in the following order:

(1) Original (soundfile PVX5.WAV).
(2) Phase vocoder transposition by a factor of 1.66 (8.77 semitones above the original pitch), producing the soundfile PVX5P.WAV.

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

(3) Transposition by the same factor using the *Groucho* program FTRANS (Dobson and Endrich 1994), which transposes by resampling.

Since the original sound is stereo, the batchfile PVTRS.BAT – listed in Appendix A2 – was used with the command

```
pvtrs pvx5 pvx5p 1.66
```

4.4. Spectral shift and stretch

The principle of multiplying spectral components by a factor in order to transpose a spectral frame may be extended to a more general case. In the first place, a process which only multiplies some of the frequencies may be implemented. For instance, only frequencies above or below a certain channel may be multiplied by a factor. If the sound is harmonic and the factor is not an integer, the output will have an inharmonic spectrum composed of two harmonic blocks. This type of process is known as *spectral shift*. For example, a signal with components at 100, 200, 300, 400 and 500 Hz may have the last three shifted by a factor of 1.3. The resulting spectrum will contain the following partials:

100 Hz (unchanged)

200 Hz (unchanged)

$1.3 \times 300 = 390$ Hz

$1.3 \times 400 = 520$ Hz

$1.3 \times 500 = 650$ Hz

Note that the ratio between the shifted components (3:4:5) is preserved. However, the multiplication process may not preserve the ratio between components. Multiplying these by a factor which increases with frequency will result in stretching of the spectrum of the sound; if it decreases with frequency, the spectrum will be compressed. Multiplication by a factor which is a function of frequency is referred to as *spectral stretch*. For instance, the three upper frequencies of the previous example may be multiplied by a factor which changes by 0.3 every 100 Hz, resulting in the following spectrum:

100 Hz (unchanged)

200 Hz (unchanged)

$1.3 \times 300 = 390$ Hz

$1.6 \times 400 = 640$ Hz

$1.9 \times 500 = 950$ Hz

Note that the ratio of the three upper components is now 39:64:95 and not 3:4:5.

Finally, it is possible to make the shift or stretch factors functions of time, creating glissandi of groups of partials. Time-varying shift and stretch processes are respectively implemented in the programs SPECSH and SPECE (Wishart 1994b).

Audio example 6 consists of a texture formed by short percussive sounds. These are followed by the output of time-varying shift and stretch processes with similar parameters. The number of filters is 2,048 (2^{N-1}) and the components processed have frequencies above channel 120, which from equation (9) is 1,292 Hz. Both, shift and stretch, begin after 50 ms, preserving the attack of the first sound. The spectral shift increases from 0 to 2.4345 in 4 seconds. Its effect is immediate since all frequencies above channel 120 are multiplied by the same factor. The result will be heard as a glissando which immediately affects the sharpness of the sound. On the other hand, the stretching process does not multiply all frequencies by the same value: the factor is an exponential function of frequency. The highest partial varies from 0 to 2.4345 in four seconds; however, lower components will reach lower maxima. For instance, components immediately above 1,292 Hz will experience little change. Therefore, the character of the sound will be preserved to a greater extent than in the case of the shift.

The processes above were carried out using batchfiles PVSHU.BAT and PVSTU.BAT – listed in Appendices A3 and A4 – with the commands

```
pvshu PVX6 PVX6SH 120 2.4345 0.05 4 0 0          (shift)

pvstu PVX6 PVX6SH 120 2.4345 0.05 4 0 0 1.6      (stretch)
```

Further examples of shifting and stretching may be found in Wishart (1988).

4.5. Vocoding

PVOC may also be used in conjunction with the program SPECVOCO (Wishart 1994b) in order to impose the spectral envelope of one sound onto the spectrum of another. This is particularly effective in the case of speech, which is strongly characterised by the way the spectrum changes radically through time with rapid evolution of the spectral envelope. Typical vocoding processes makes use of the evolving spectra of such sounds to shape the spectra of other (target) sounds, such as musical instruments, noise, etc.

When carrying out a vocoding process it is important to bear in mind that the data in the frequency channels of the vocoding spectrum which will amplify or attenuate the data in corresponding frequency channels of the target sound must work on sufficiently prominent components in that sound. Otherwise the target may not be significantly shaped by the vocoding process.

Audio example 7 contains an iterative sound (PVX71.WAV) with strongly defined spatial localisation and movement which 'vocodes' a second sound

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

(PVX72.WAV) which is relatively static. The output has the same dynamic behaviour as the first sound with components belonging to the second. This example also includes an excerpt from the beginning of '*Dance Suite*' for string quartet and tape (Fischman 1996b) – in which a similar vocoding process is used – and a passage from *Alma Latina* (Fischman 1996a), which contains various transpositions of the same sound.

The process was implemented with PVVOS.BAT – listed in Appendix A5 – and the command

```
pvvos PVX71 PVX72 PVX7V 5
```

PVVOS.BAT uses the program SPECVOCO in order to impose the spectral envelope of PVX71.WAV on the spectrum of PVX72.WAV, producing the output soundfile PVX7V.WAV. The value 5 indicates that a point in the spectral envelope should be generated for every five channels.

4.6. Spectral tracing

This technique consists of keeping on a moment by moment basis only the *N* loudest components of a sound. The result usually 'softens' the original and may also help in getting rid of background noise. Audio example 8 contains a stereo texture before and after tracing its thirteen loudest partials. The process produces a smoother bubbling sound. It was realised with batchfile PVSPTRS.BAT – listed in Appendix A6 – and the command

```
pvsptrs PVX8 PVX8T 13
```

PVSPTRS.BAT invokes the spectral transformation SPECTRAC, which in this case is used to keep the thirteen loudest partials of the input soundfile PVX8.WAV.

4.7. Frame interleave

The program SPECLIFN (Wishart 1994b) interleaves groups of frames from analyses of two or more sounds. Each group is considered to be a *leaf*; its size refers to the number of frames it contains. For example, if spectra of two sounds are interleaved with a leaf size of 4, the output spectrum will consist of four frames of the first sound, followed by four of the second, followed by four of the third, and so on. This may create various degrees of granularity, depending on the size of the leaf and on the number and spectral characteristics of the input sounds. For instance, larger leaf sizes may create relatively slow pulsating effects, whereas smaller sizes may create relatively compact granular textures.

Audio example 9 contains the recording of a deep breath (PVX9.WAV). This is followed by transpositions of this sound by factors of 0.34 (18.68 semitones below the original pitch) and 0.66 (7.19 semitones below the original pitch). The output resulting from interleaving the previous three sounds with a leaf size of 4 frames follows next. Finally, an excerpt from *Alma Latina* (Fischman 1996a) which includes similar processing of another vocal sound is played: the original and variations of the interleaved sound were used as a sonic *moto* throughout this piece.

The transpositions of PVX9.WAV were carried out using the batchfile PVTRS.BAT – listed in Appendix A2 – with the following commands

```
pvtrs PVX9 PVX9P1 0.34

pvtrs PVX9 PVX9P2 0.66
```

The interleaving process was carried out using batchfile PVLF3.BAT – listed in Appendix A7 – with the following command:

```
pvlf3 PVX9 PVX9P1 PVX9P2 PVX9LF 4
```

4.8. Spectral sweep

It is possible to sweep the spectrum of a sound with a time-varying band-pass filter, which acts as a window, only letting through a narrow frequency band at any point in time. If the centre frequency of the filter increases, an ascending arpeggio will be heard. If it decreases, the arpeggio will descend. The program SPECARPE (Wishart 1994b) implements this type of behaviour.

Audio example 10 illustrates an arpeggiation process which ascends and descends between 200 Hz and 4.5 kHz, with a filter bandwidth of 30 Hz. The sweep is controlled by a sinewave oscillating at 0.25 Hz (a whole sweep lasts 4 s) starting from 0 and moving upwards (phase = 0). Partials which do not correspond to the pass region are stopped (the sweep type is On/Off). We first hear the original sound. This is followed by the result of the arpeggiation. Finally, an excerpt including ascending and descending parts of the sweep is played. These belong to the tape part of *The Day After* . . . – for string quartet and tape (Fischman 1995).

The process above was realised with the batchfile PVARPS.BAT – listed in Appendix A8 – and the following command:

```
pvarps PVX10 PVX10A 1 0 0.25 1 200 4500 30
```

4.9. Spectral interpolation

Under certain conditions, it is possible to achieve a convincing transformation of a sound into another by cross-fading their analysis files. This process is also known as *spectral interpolation* and is discussed in

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

detail in Wishart (1988) and Fischman (1994), who also present more elaborate examples.

Audio example 11 illustrates a transformation from an oboe middle A (PVX111.WAV) into a sound with vocal characteristics and the same pitch, which may be described as a cry (PVX112.WAV). The oboe is heard first, then the cry and finally the transformation. This process was implemented using the batchfile PVVI.BAT – listed in Appendix A9 – and the command:

```
pvvi PVX111 PVX112 PVX11VI 0.7 3.5 0.7 3.6 2 2
```

4.10. Further possibilities

There are now over fifty spectral manipulations working on PVOC data within the CDP environment. Some of the more interesting are:

- pitch extraction and manipulation, including transfer of pitch from one sound to another,
- tuning the spectrum to specified harmonic fields,
- various types of accurate filtering,
- focusing the spectrum on its most prominent constituents (averaged over any specified time window),
- freezing the spectrum and interpolating, or stepping between isolated spectral windows,
- chorusing and noise blurring of the spectrum, and
- inserting shepherd tone glissandi within the evolving spectral envelope of the source.

5. CONCLUSION

The combination of phase vocoder and the CDP sound transformations offers a powerful and effective set of tools for the manipulation of sounds in the frequency domain. Effective use of these tools is dependent on the understanding of both the theoretical principles of the short time Fourier transform (including its limitations) and their application to composition. The former are necessary in order to understand the processes carried out by the various programs and the significance and effects of their parameters. The latter is part of the set of skills acquired via regular practice of a discipline and from experience. It is hoped that this article will contribute to an understanding of the theoretical issues in a way which is accessible to readers with either musical or scientific backgrounds, and serve as a means for communicating experience acquired by the author and others in the process of using the available tools for musical composition.

APPENDICES – LISTING OF BATCHFILES

A1. Stereo time-stretch batchfile PVTSS.BAT

```
REM PVTSS.BAT
REM Usage: PVTSS input_file timestretch_file
REM ==== CHANNEL SEPARATION ====
channel -ot %1.wav 1 2
REM ==== LEFT CHANNEL ====
REM ANALYSIS
pvoc -N512 -A t_c1.wav a_c1.wav
REM Clean leftovers
del t_c1.wav
REM TIME-STRETCH
specstr a_c1.wav %2 ax_c1.wav
REM Clean leftovers
del a_c1.wav
REM SYNTHESIS
```

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

```
pvoc -S ax_c1.wav x_c1.wav
REM Clean leftovers
del ax_c1.wav
REM ==== RIGHT CHANNEL ====
REM ANALYSIS
pvoc -N512 -A t_c2.wav a_c2.wav
REM Clean leftovers
del t_c2.wav
REM TIME-STRETCH
specstr a_c2.wav %2 ax_c2.wav
REM Clean leftovers
del a_c2.wav
REM SYNTHESIS
pvoc -S ax_c2.wav x_c2.wav
REM Clean leftovers
del ax_c2.wav
REM ==== INTERLEAVE ====
interl %2 x_c1.wav x_c2.wav
REM Clean leftovers
del x_c1.wav
del x_c2.wav
```

A2. PVTRS.BAT: stereo transposition

```
REM PVTRS.BAT
REM Usage: pvtrs in-file-name out-file-name factor.
REM CHANNEL SEPARATION
channel -ot %1.wav 1 2
REM TRANSPOSITION
pvoc -N2048 -P%3 t_c1.wav tt_c1.wav
pvoc -N2048 -P%3 t_c2.wav tt_c2.wav
REM INTERLEAVE
interl %2.wav tt_c1.wav tt_c2.wav
REM remove separated channels
```

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

```
del t_c1.wav
del t_c2.wav
del tt_c1.wav
del tt_c2.wav
```

A3. PVSHU.BAT: spectral shift upwards

```
REM PVSHU.BAT
REM Usage: pvshu input_file output_file fdcno shift att div conv end
REM ANALYSIS
pvoc -N4096 -A %1.wav a_c1.wav
REM Clean leftovers
del t_c1.wav
REM SHIFT
specsh a_c1.wav ax_c1.wav 1 %3 %4 %5 %6 %7 %8
REM Clean leftovers
del a_c1.wav
REM SYNTHESIS
pvoc -S ax_c1.wav %2.wav
REM Clean leftovers
del ax_c1.wav
```

A4. PVSTU.BAT: spectral stretch upwards

```
REM PVSTU.BAT
REM Usage: PVSTU input_file output_file fdcno stretch att div conv end exp
REM  ANALYSIS
pvoc -N4096 -A %1.wav a.wav
REM STRETCH
spece a.wav ax.wav 0 %3 %4 %5 %6 %7 %8 %9
REM SYNTHESIS
pvoc -S ax.wav %2.wav
REM Clean leftovers
del a.wav
```

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

```
del ax.wav
```

A5. PVVOS.BAT: vocoding

```
REM PVVOS.BAT
REM Usage: pvvos input1 input2 output chanPerPoint
REM ==== CHANNEL SEPARATION ====
channel -ot1 %1.wav 1 2
channel -ot2 %2.wav 1 2
REM ==== LEFT CHANNEL ====
REM ANALYSIS
pvoc -A -N2048 t1_c1.wav t1_c1a.wav
pvoc -A -N2048 t2_c1.wav t2_c1a.wav
REM INTERPOLATION
specvoco t1_c1a.wav t2_c1a.wav ti_c1a.wav -f%4
REM SYNTHESIS
pvoc -S ti_c1a.wav ti_c1.wav
REM delete leftovers
del t1_c1.wav
del t2_c1.wav
del t1_c1a.wav
del t2_c1a.wav
del ti_c1a.wav
REM RIGHT CHANNEL
REM ANALYSIS
pvoc -A -N2048 t1_c2.wav t1_c2a.wav
pvoc -A -N2048 t2_c2.wav t2_c2a.wav
REM INTERPOLATION
specvoco t1_c2a.wav t2_c2a.wav ti_c2a.wav -f%4
REM SYNTHESIS
pvoc -S ti_c2a.wav ti_c2.wav
REM ==== INTERLEAVE ====
interl %3.wav ti_c1.wav ti_c2.wav
```

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

```
REM delete leftovers
del t1_c2.wav
del t2_c2.wav
del t1_c2a.wav
del t2_c2a.wav
del ti_c2a.wav
del ti_c1.wav
del ti_c2.wav
```

A6. PVSPTRS.BAT: spectral tracing

```
REM PVSPTRS.BAT
REM Usage: PVSPTRS input_file output_file partials
REM ==== CHANNEL SEPARATION ====
channel -ot %1.wav 1 2
REM ==== LEFT CHANNEL ====
REM ANALYSIS
pvoc -N512 -A t_c1.wav a_c1.wav
REM Clean leftovers
del t_c1.wav
REM SPECTRAC
spectrac a_c1.wav ax_c1.wav %3
REM Clean leftovers
del a_c1.wav
REM SYNTHESIS
pvoc -S ax_c1.wav x_c1.wav
REM Clean leftovers
del ax_c1.wav
REM ==== RIGHT CHANNEL ====
REM ANALYSIS
pvoc -N512 -A t_c2.wav a_c2.wav
REM Clean leftovers
del t_c2.wav
```

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

```
REM SPECTRAC
spectrac a_c2.wav ax_c2.wav %3
REM Clean leftovers
del a_c2.wav
REM SYNTHESIS
pvoc -S ax_c2.wav x_c2.wav
REM Clean leftovers
del ax_c2.wav
REM ==== INTERLEAVE ====
interl %2.wav x_c1.wav x_c2.wav
REM Clean leftovers
del x_c1.wav
del x_c2.wav
```

A7. PVLF3.BAT: spectral interleave of 3 sounds

```
REM PVLF3.BAT
REM Usage: pvlf3 input1 input2 input3 output leaf-size
REM CHANNEL SEPARATION
channel -ot1 %1.wav 1 2
channel -ot2 %2.wav 1 2
channel -ot3 %3.wav 1 2
REM ANALYSIS
pvoc -A -N2048 t1_c1.wav t1_c1a.wav
pvoc -A -N2048 t1_c2.wav t1_c2a.wav
pvoc -A -N2048 t2_c1.wav t2_c1a.wav
pvoc -A -N2048 t2_c2.wav t2_c2a.wav
pvoc -A -N2048 t3_c1.wav t3_c1a.wav
pvoc -A -N2048 t3_c2.wav t3_c2a.wav
REM SPECTRAL INTERLEAVE
speclifn t1_c1a.wav t2_c1a.wav t3_c1a.wav tlf_c1a.wav %5
speclifn t1_c2a.wav t2_c2a.wav t3_c2a.wav tlf_c2a.wav %5
REM SYNTHESIS
```

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

```
pvoc -S tlf_c1a.wav tlf_c1.wav
pvoc -S tlf_c2a.wav tlf_c2.wav
REM INTERLEAVE
interl %4.wav tlf_c1.wav tlf_c2.wav
REM delete leftovers
del t1_c1.wav
del t1_c2.wav
del t2_c1.wav
del t2_c2.wav
del t3_c1.wav
del t3_c2.wav
del t1_c1a.wav
del t1_c2a.wav
del t2_c1a.wav
del t2_c2a.wav
del t3_c1a.wav
del t3_c2a.wav
del tlf_c1a.wav
del tlf_c2a.wav
del tlf_c1.wav
del tlf_c2.wav
```

A8. PVARPS.BAT: spectral sweep

```
REM PVARPS.BAT
REM Usage: PVARPS input_file out_file T U V W X Y Z
REM CHANNEL SEPARATION
channel -ot %1.wav 1 2
REM ANALYSIS
pvoc -N512 -A t_c1.wav a_c1.wav
pvoc -N512 -A t_c2.wav a_c2.wav
REM Clean leftovers
del t_c1.wav
del t_c2.wav
```

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

```
REM ARPEGIATE
specarpe a_c1.wav ar_c1.wav -w%3 -p%4 -f%5 -t%6 -l%7 -u%8 -b%9
specarpe a_c2.wav ar_c2.wav -w%3 -p%4 -f%5 -t%6 -l%7 -u%8 -b%9
REM Clean leftovers
del a_c1.wav
del a_c2.wav
REM
REM SYNTHESIS
pvoc -S ar_c1.wav r_c1.wav
pvoc -S ar_c2.wav r_c2.wav
REM Clean leftovers
del ar_c1.wav
del ar_c2.wav
REM INTERLEAVE
interl %2 r_c1.wav r_c2.wav
REM Clean leftovers
del r_c1.wav
del r_c2.wav
```

A9. PVVI.BAT: spectral interpolation

```
REM PVVI.BAT
REM Usage: pvvi input1 input2 output amp.s amp.e phas.s phas.e exp.amp exp.phas
REM ANALYSIS
pvoc -A -N2048 %1.wav t1.wav
pvoc -A -N2048 %2.wav t2.wav
REM INTERPOLATION
vocinte t1.wav t2.wav ti.wav %4 %5 %6 %7 %8 %9
REM SYNTHESIS
pvoc -S ti.wav %3.wav
REM delete leftovers
del t1.wav
del t2.wav
del ti.wav
```

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press

REFERENCES

Atkins, M., Bentley, A., Endrich, T., Fischman, R., Malham, D., Orton, R., and Wishart, T. 1987. The Composers' Desktop Project. In S. Tipei and J. Beauchamp (eds.) *Proc. Int. Computer Music Conf.*, pp. 146–50. San Francisco: Computer Music Association.

Dobson, R., and Endrich, T. J. 1994. CDP 'Groucho' sound processing programs. In T. J. Endrich (ed.) *Composers' Desktop Project* (*CDP*). *Professional Computer Music System User Guide, Release 2*. York: CDP Ltd.

Dodge, C., and Jerse, T. A. 1985. *Computer Music. Synthesis, Composition and Performance*. New York: Schirmer Books.

Dolson, M. 1986. The phase vocoder: a tutorial. *Computer Music Journal* **10**(4): 14–27.

Dolson, M. 1991. Fourier-transform-based timbral manipulations. In V. Matthews and J. R. Pierce (eds.) *Current Directions in Computer Music Research*, pp. 105–12. Cambridge, MA: MIT Press.

Fischman, R. 1991. *Musical Applications of Digital Synthesis and Processing Techniques. Realisation using Csound and the Phase Vocoder*. Unpublished addendum to DPhil composition folio. York University, UK.

Fischman, R. 1994. Sound processing in Los Dados Eternos. In N. Osborne, P. Nelson and S. Emmerson (eds.) *Contemporary Music Review. Timbre Composition in Electroacoustic Music* **10**(2): 181–9. Switzerland: Harwood.

Fischman, R. 1995. *The Day After*. Piece for string quartet and tape.

Fischman, R. 1996a. *Alma Latina*. Piece for tape.

Fischman, R. 1996b. '*Dance Suite*'. Piece for string quartet and tape.

Gordon, J., and Strawn, J. 1985. An introduction to the phase vocoder. In J. Strawn (ed.) *Digital Audio Signal Processing: An Anthology*, pp. 221–70. Los Altos, CA: William Kaufmann Inc.

Kronland-Martinet, R. 1988. The use of the wavelet transform for the analysis, synthesis and processing of speech and music sounds. *Computer Music Journal* **12**(4): 11–20 (sound examples on soundsheet with **13**(1), 1989).

Kronland-Martinet, R., and Grossman, A. 1991. Application of time–frequency and time-scale methods (wavelet transform) to the analysis, synthesis, and transformation of natural sounds. In G. De Poli, A. Picciali and C. Roads (eds.) *Representations of Musical Signals*, pp. 45–85. Cambridge, MA: MIT Press.

Moorer, J. A. 1978. The use of the phase vocoder in computer music applications. *Journal of the Audio Engineering Society* **24**(9): 717–27.

Oppenheim, A. V., and Schaffer, R. W. 1975. *Digital Signal Processing*. New Jersey: Prentice/Hall Inc.

Williams, C. S. 1986. *Designing Digital Filters*. New Jersey: Prentice/Hall Inc.

Wishart, T. 1988. The composition of Vox-5. *Computer Music Journal* **12**(4): 21–7 (sound examples on soundsheet with **13**(1), 1989).

Wishart, T. 1994a. The phase vocoder. Introduction and reference manual. In T. J. Endrich (ed.) *Composers' Desktop Project* (*CDP*). *Professional Computer Music System User Guide, Release 2*. York: CDP Ltd.

Wishart, T. 1994b. Spectral transformations. In T. J. Endrich (ed.) *Composers' Desktop Project* (*CDP*). *Professional Computer Music System User Guide, Release 2*. York: CDP Ltd.

https://doi.org/10.1017/S1355771897009060 Published online by Cambridge University Press