CDP BLUR (Time) Functions

(with Command Line Usage)

Functions which BLUR (Time) analysis data

(Names in brackets mean that these are separate programs. The others are sub-modules of BLUR.)

AVRG
Average spectral energy over *N* adjacent channels
BLUR
Blur the spectral data over time
CHORUS
Add random variation to amplitude or frequency in analysis channels
DRUNK
Modify sound by a drunken walk along analysis windows
NOISE
Add noise to spectrum
SCATTER
Randomly thin out the spectrum
[SELFSIM]
Replace spectral windows with the most similar, louder window(s)
SHUFFLE
Shuffle analysis windows according to a specific scheme
SPREAD
Spread spectral peaks
SUPPRESS
Suppress the most prominent channel data
WEAVE
Weave amongst the analysis windows in a specified pattern

Technical Discussion
Analysis Windows

BLUR AVRG – Average spectral energy over *N* adjacent channels

Usage

blur avrg *infile outfile N*

infile – input analysis file made with PVOC
outfile – output analysis file
N – no. of adjacent channels – must be <= half the no. of channels in the *infile*.

N may vary over time

Understanding the BLUR AVERAGE Process

The averaging of energy information causes high energy data to cross channels. The result is the intrusion of timbral data into channels which previously didn't have that data. Test results led to a 'roughening' of the sound, though without going as far as distorting it.

Note that this process relates to channels, not to windows (as does the BLUR BLUR process). This means that the averaging directly affects the frequency components of the sound.

The energy (amplitude) in adjacent channels is averaged out over *Average-span* adjacent channels, without affecting the frequencies of those channels. The result is to broaden, or defocus, any energy peaks in the spectrum.

This reduces the frequency definition of the sound, which is why it is in the BLUR category and is, in effect, a type of filtering.

Musical Applications

BLUR AVRG appears to be useful to enrich the timbre of a sound and make its texture rougher.

End of BLUR AVRG

BLUR BLUR – Time-average the spectrum

Usage

blur blur *infile outfile blurring*

infile – input analysis file made with PVOC
outfile – output analysis file
blurring – the number of windows over which to average the spectrum

blurring may vary over time: provide the name of a *time blurring* breakpoint file for this parameter.

Understanding the BLUR BLUR Process

BLUR BLUR time-averages the spectrum. It 'blurs' detail in the time dimension by interpolating between the spectral envelope values of the start and end windows *blurring* windows. Note that it is not interpolating continuously over all the windows inbetween, just between the data in the start and end windows. The overall result is somewhat affected by just how different the data is in these two windows. The interpolation process produces a 'straight line' (linear) scale of values between the start and end points.

BLUR BLUR differs from **HILITE BLTR** in the absence of the 'trace' parameter. This means that the blurring occurs without any reduction in the number of partials.

Musical Applications

As with all of these programs, the result is greatly affected by the nature of the input sound. The more windows blurred, the more blurring or smoothing of the sound you might expect to happen. However, you might not notice much difference if the sound is already constant (or similar at the start and end points). You will probably need a sound with a great deal of internal change for the blurring to have a perceptible effect.

The musical results of this process begin with a softening of the attack transients, so is highly effective with, for example, plucked or percussive sounds. A time-varying transition from the original to a very blurred effect can be achieved by a simple breakpoint file which sets a low value for *blurring* at the beginning of the sound and a high value at the end. It is possible for the sound to disappear gradually into its own ambience. This powerful technique can therefore useful in the creation of 'ambient' music, as well as any degree of softening of the original sound.

Also see: **HILITE BLTR** and **HILITE TRACE**.

End of BLUR BLUR

BLUR CHORUS – Chorusing by randomising amplitudes and/or frequencies of partials

Usage

blur chorus 1 *infile outfile aspread*
blur chorus 2-4 *infile outfile fspread*
blur chorus 5-7 *infile outfile aspread fspread*

Modes

1 Randomise partial amplitudes
2 Randomise partial frequencies
3 Randomise partial frequencies upwards only
4 Randomise partial frequencies downwards only
5 Randomise partial amplitudes AND frequencies
6 Randomise partial amplitudes, and frequencies upwards only
7 Randomise partial amplitudes, and frequencies downwards only

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
aspread – maximum random scatter of partial-amps (Range 1-1028)
fspread – maximum random scatter of partial-frqs (Range 1-4)

aspread and *fspread* may vary over time

Understanding the BLUR CHORUS Process

Formerly SPECHORU, this process attempts to achieve a chorusing effect by randomising the amplitude and frequency values of the partials. It is based on an idea of Stephen McAdams. See his *Spectral Fusion and the Creation of Auditory Images* (1981).

If very large amplitude *aspread* values are used, the sound will turn to noise. The ranges shown for *aspread* and *fspread* appear to be rather extreme, but they are rescaled within the program. There is in fact no mathematical limit to the values which can be entered. In practice, the 'chorusing' effect itself is achieved by values just a little above 1!. Values of 2 or 3 begin to create a granular effect, and values of 10, 100 and 1000, for example, create more and more noise.

Musical Applications

'Chorusing' is one of the standard effects used to enliven and enrich a sound. It becomes available here in an open-ended context, where the effect can be driven beyond the usual bounds into granulation and noise effects, useful when a certain textural or gritty quality is needed.

In Mode 5, a burbly but comprehensible vocal transformation may occur with *aspread* = 30 and *fspread* = 4.

End of BLUR CHORUS

BLUR DRUNK – Modify sound by a drunken walk along analysis windows

Usage

blur drunk *infile outfile range starttime duration* [**-z**]

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
range –the maximum step (in windows) for the drunken walk: <= 64
starttime– the time (in seconds) in the file at which the walk should begin
duration– the required duration of the outfile after re-synthesis it may be longer than *infile*
-z eliminates zero steps (window-repeats) in the drunken walk

Understanding the BLUR DRUNK Process

Beginning at *starttime*, a user-defined point in the *infile*, BLUR DRUNK moves through the file reading windows, but jumping either forwards or backwards, from one window to another, in a random way. The size of these random jumps cannot be greater than *range*. This random process is known as a drunken walk. It proceeds until the specified duration of the output sound has been generated.

If the process tries to leap right out of the soundfile at its end or its start, it is automatically reflected back into the sound near to the leap point. The process never reads backwards through the file, though it may leap backwards before starting its next read.

Musical Applications

This process can be used to lurch about in a soundfile, producing a jumbled version of *infile* to varying degrees. The salient parameter is *range*. If *range* is small, the output will tend to linger around your start-point in the file, progressing very slowly away from it in an arbitrary direction. This results in a mix of time-stretching and slow wandering through the source. If *range* is large, the drunken walk tends to leap about wildly in the file, scrambling the source sound.

End of BLUR DRUNK

BLUR NOISE – Put noise in the spectrum

Usage

blur noise *infile outfile noise*

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
noise – Range 0 (no noise in spectrum) to 1 (spectrum saturated with noise)

noise may vary over time

Understanding the BLUR NOISE Process

This functions enables one to move a sound source towards pure noise, by making the data in every channel – most of which is actually low level noise – equally loud. Total saturation will reduce all sounds to a very similar noise signal. Partial or gradual saturation is possible, through the use of values less than 1 or by using a time-varying breakpoint file. These work in the usual way, with gradual change between different time points to which are assigned different *noise* values.

Musical Applications

This technique can be used to cause sound material to emerge from obscurity to clarity, or *v.vs.*. Also, a carefully chosen *noise* factor can be used to colour a sound or soften its edges.

End of BLUR NOISE

BLUR SCATTER – Randomly thin out the spectrum

Usage

blur scatter *infile outfile keep* [**-b***blocksize*] [**-r**] [**-n**]

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
keep – number of (randomly chosen) blocks to keep in each spectral window (Range: 1 to no. of chans.)
-b*blocksize* – frequency range of each block (default is width of 1 analysis channel. (Rounded internally to a multiple of channel width.)
-r number of blocks actually selected is randomised between 1 and *keep*
-n turn OFF normalisation of resulting sound

keep and *blocksize* may vary over time

Understanding the BLUR SCATTER Process

This function throws away a specified proportion of the analysis data in each window. Unlike **HILITE TRACE**, the channels chosen to be suppressed are selected entirely at random. The spectrum is thus thinned out in an unpredictable fashion.

Remember that the number of channels in an analysis file is determined by the value for the **-N** flag of the Phase Vocoder (now referred to as **-c***points*). For example, if it was **-c**1024, there will be 512 + 1 channels, each containing frequency and amplitude data.

Before scattering, the channels are gathered together into blocks of adjacent channels, and these blocks-of-channels are then either retained or discarded (at random).

Blocksize sets how many of these channels form one block. *Keep* sets how many of these blocks (i.e., groups of channels) will be randomly chosen and retained from the total of 513.

The values for *keep* and *blocksize* therefore need to be worked out in relation to the value given for **-c**. For example, retaining *keep* = 10 and *blocksize* = 6 will retain a total of 60 (randomly selected) channels from the analysis data.

Musical Applications

The musical result will be more variable than HILITE TRACE. The latter retains the *N* **loudest** channels, thus always keeping the most audibly prominent data of the original sound. BLUR SCATTER may or may not pick up audibly prominent data as it makes its random selections. The original sound material will therefore be more variably retained or cast aside, and this variability can be increased by using a *time keep* and/or a *time blocksize* breakpoint file.

End of BLUR SCATTER

SELFSIM – Replace spectral windows with the most similar, louder window(s) of the same analysis file

Usage

selfsim selfsim *inanalfile outanalfile self-similarity-index*

Example command line to expand the area of similar windows :

```
selfsim selfsim in.ana out.ana 2
```

Parameters

inanalfile – input (mono) analysis file
outanalfile – output (mono) analysis file
self-similarity-index – the number of similar windows to replace. Value = 1 uses the loudest window to replace the most similar window, then the next loudest window to replace the window most similar to it, and so on, with appropriate overall-loudness scaling. With value = 2, the loudest windows replaces the **two** most similar windows, and so on. Thus, if window A replaces window B, and window C is most similar to window B, then window A also replaces window C, etc.

Understanding the SELFSIM Process

This process systematically replaces spectral windows that are less prominent (i.e.,not as loud, no special peaks ...) with spectral windows in the same file that are more prominent in some way *and are the most similar to them in spectral envelope* . At the same time it scales amplitudes for any overall difference between the two windows.

Musical Applications

It is important to observe the effect of higher values for the *self-similarity-index*. You might describe this program as a feature repeater. On the one hand it is looking for prominent windows and extending their presence in the sound. On the other hand, these prominent windows are also examined for similarity of spectral envelope. Thus the features of this spectral envelope are repeated more often in the resulting sound – hence the 'self-similar' idea. The overall effect is going to depend on the nature of the prominent windows, and one can expect that a fairly steady-state sound will not be much affected by this process. Try it on exciting sounds!

End of SELFSIM

BLUR SHUFFLE – Shuffle order of analysis windows in file

Usage

blur shuffle *infile outfile domain-image grpsize*

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
domain-image – this consists of two strings of letters, separated by a hyphen ('-'); the first string is the *domain* and the second string is the *image*, e.g., 'abc-abbabcc'.

The *domain* letters represent a group of consecutive *infile* analysis windows, e.g., 'abcd'.
The *image* is any permutation of, or selection from, these *domain* letters these letters may be omitted or repeated in the *image* string, e.g., 'aaaaaaaadaaa'.

grpsize – the number of analysis windows corresponding to each letter of *domain*.

Each letter then represents a group of *grpsize* windows and the whole group is treated as one unit in the shuffling process.

Understanding the BLUR SHUFFLE Process

SHUFFLE shuffles windows in a PVOC analysis data file. This is a spectral version of 'brassage' (see **MODIFY BRASSAGE** and **MODIFY SAUSAGE**). As with most of these programs, the degree of audible difference is going to depend on how much the sound changes in the first place. Shuffling spectral windows in a fairly uniform sound may not have much effect. Experimenting with this program ought to lead to the unexpected.

To illustrate how this works, suppose there is a *domain* of three letters: A B C. This means that there will be three consecutive windows, numbered 1 2 3 in the order in which they occur in the analysis file.

If the *image* is ordered C B A, then these three windows will be rearranged in the output into the order 3 2 1.

grpsize has not been set in this example, so the default of 1 is operating. If we then set *grpsize* to, say 3, the A will stand for 3 windows, starting with 1 2 3, B for windows 4 5 6, and C for windows 7 8 9. When the shuffling takes place according to the *Image* **CBA**, the result will be 7 8 9, 4 5 6, 1 2 3. The process continues in this way throughout the analysis file. Larger values for *grpsize* (e.g., 20) can lead to the repetition of fragments of sound material. The following table summarises these inputs.

BLUR SHUFFLE EXAMPLE

Domain	*Image*	*Groupsize*
ABC	CBA	3
123 456 789	789 456 123	

The command line for the above would be:

```
blur shuffle infile outfile ABC CBA 3
```

Note that the size of the *domain* and the value for *grpsize* are limited by the amount of RAM available on your computer.

Musical Applications

The *domain-image* parameter is what makes SHUFFLE particularly useful. The *domain* identifies a series of consecutive windows, and the *image* defines the order of those windows (some may be omitted). Then this whole pattern repeats as a block, affecting each consecutive set of windows in the source file. The control over re-ordering makes it possible to jumble, retrograde, mix forwards and backwards etc. the successive windows. An increasing *grpsize* makes more of the source audible at any given time, which means that one can play with the degree to which the source can be identified.

Thus one is stepping through the source in chunks of varying size and re-ordering the contents of those chunks. The re-orderings play with familiarity, create repeatable sub-patterns, and will lead to additional timbral changes depending on the degree to which differing spectral data get mixed together.

End of BLUR SHUFFLE

BLUR SPREAD – Spread peaks of spectrum, introducing controlled noisiness

Usage

blur spread *infile outfile* **-f***N* | **-p***N* **-i** [**-s***spread*]

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
-f*N* extract formant envelope linear frequency-wise, using 1 point for every *N* equally-spaced frequency channels
-p*N* extract formant envelope linear pitch-wise, using *N* equally-spaced pitch bands per octave
-i quicksearch for formants (less accurate)
-s*spread* degree of spreading of spectrum (Range 0-1; Default is 1).

spread may vary over time

Understanding the BLUR SPREAD Process

This process introduces noise into the spectrum in a way which is coherent with (i.e., related to) the spectral envelope. The spectral envelope (formants) in each window is retained, and the level in every channel is made to approximate this average spectral contour to a greater or lesser extent, depending on *spread*. This process tends to exaggerate the less prominent (noise) constituents of the spectrum.

Musical Applications

This is a distortion technique which can be handled in a very sensitive manner because *spread* can be so finely adjusted, or altered in a time-varying manner. If applied to speech, the noise introduces a certain edginess, so if an effect such as unvoiced speech is sought, FORMANTS VOCODE is recommended, which allows some low or high frequency filtering.

End of BLUR SPREAD

BLUR SUPPRESS – Suppress the *N* loudest partials (on a window by window basis)

Usage

blur suppress *infile outfile N*

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
N – the number of spectral components to reject

N may vary over time

Understanding the BLUR SUPPRESS Process

This process is the opposite of **HILITE TRACE**, which retains the *N* loudest partials. Whereas with TRACE many components have to be removed before the sound starts to be seriously affected, here the change to *infile* will be more immediate when a relatively small value for *N* is used.

Musical Applications

The loudest partials will not necessarily relate to either the higher or lower parts of the frequency spectrum. They will relate simply to whatever is most audible prominent in the sound, such as possibly the fundamental of a pitched sound. This will have the effect of revealing the more strictly timbral aspects of the sound.

This process can also be useful in preparing a sound for another process which will benefit from suppressing its more audible features. As an example of this, ... (I did a sound in which I couldn't get rid of the fundamental, which was reinforced by the process, when I really wanted to bring out other things.)

Also see: **HILITE TRACE** and **HILITE BLTR**.

End of BLUR SUPPRESS

BLUR WEAVE – Modify sound by weaving amongst analysis windows

Usage

blur weave *infile outfile weavfile*

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
weavfile contains a list of integers which define successive **steps** (in windows) through the input file. The start window is always numbered '0'. Steps may be forward or backwards, adding the positive or negative integer of the weave formula to the number of the window at which one has arrived to show which will be the next window selected (see tables below). The step sequence is repeated until the end of the *infile* is reached.

The weave must obey the following rules:

RULE 1: NO step can exceed 127 forwards or -128 backwards
RULE 2: NO window reached in a weave can be BEFORE the start window of the weave.
RULE 3: FINAL window must be AFTER the weave start window.

Otherwise, weave may be forward or backward, or remain at same window.

Understanding the BLUR WEAVE Process

BLUR WEAVE is similar to SHUFFLE but is a smoother process which plots a regular course through the analysis windows, weaving backwards and forwards as instructed by the *weavfile*. Note that the weave process enables one to jump to points forwards or backwards in the soundfile, *at which point it takes a single window*. Thus one of the keys to understanding WEAVE is to realise that it only takes one window at a time after it has moved somewhere. It doesn't fill in with the windows inbetween. The *weavfile* therefore creates a sequence of single windows taken from various points relative to the start window. This is what makes it a form of brassage in the spectral dimension. The pattern in the *weavfile* then repeats, starting from the window it has now reached.

BLUR WEAVE can shorten or lengthen a sound:

Shorten: suppose *3 windows are selected in a way which spans a total of 7 windows.* For example, the weave pattern 3, -1, 5, starting from window 0 will take 3 steps, thus moving to window 3 (and selecting it for use in the *outanalysisfile*), move back one (3-1=2) and take window 2, and move forward 5 (2+5=7) and take window 7. Thus only 3 out of the 7 possible windows are written to the *outfile*, making the latter shorter by a ratio of 3:7.

new window count		0	1	2	3
source window	0				
	1				
	2			-1	
	3		3		
	4				
	5				
	6				
	7				5

weavfile produces reduction by 0.43

Lengthen: suppose *a few windows are used over and over* (the hovering effect mentioned above), such as with the step pattern 2 -1 1 -1 1 -1 1 -1 1 2. Here 3 windows are expanded to 10, a ratio of 10:3.

new window count		0	1	2	3	4	5	6	7	8	9	10
source window	0											
	1			-1		-1		-1		-1		
	2		2		1		1		1		1	
	3											
	4											2

weavfile produces expansion by 3.33

Musical Applications

WEAVE can be used to shorten or extend a sound and/or to blur its characteristics: by taking windows out of sequence, earlier (backwards) or later (forwards) in the source. Windows taken from relatively distant locations of course mix up the sound quite a bit. A very different effect is achieved by having the windows hover around a certain area. The fact that the pattern repeats leads to additional possibilities to create pulsations in the sound, though the timbral effect is likely to vary due to changing spectral content as the weave moves through the source.

End of BLUR WEAVE

Technical Discussion of Analysis Windows

These various functions to shuffle windows provide a tremendously varied set of tools with which to re-pattern the stream of analysis data. Before comparing them to one another, let's take a moment to revise just what a window is and just what of the original sound it encapsulates.

A window contains the spectral data in all the channels for a given duration of *infile*. If the number of samples used for the window is 1024 (indicated by the **-c***points* flag – formerly the **-N** flag), then the number of channels is 1024 divided by 2 (= 512) plus an extra channel (= 513).

This number divided by the sample rate gives the duration of one window. E.g., 512/44100 = 0.01163 sec. The hamming window usually used envelopes these samples in a certain way to maximise amplitude but avoid clicks by starting from and returning to 0. (The windows are made to overlap in order to avoid holes created by the enveloping.) Therefore, shuffling windows means moving about these little chunks of time, *with all the timbral data – the changing frequency/amplitude content contained therein*. The new juxtapositions of this timbral data are likely to lead to new colours in the output sound.

A brief overview of the SHUFFLE functions can help to clarify how they differ, and therefore what one may expect to gain by using each of them.

DRUNK
: jump about within specific spans of file, always reading forward, and change the location of these spans. This can provide a series of lurching dislocations, really pulp up the sound to create a new timbral entity.

SHUFFLE
: re-order *domain* windows according to *image* pattern, with *grpsize* windows for each domain element; the order of windows can therefore be made to go backwards as well as forwards, but the implementation of the pattern always moves constantly forward. Disjunctions, retrogrades, tiny repeating patterns etc. all become possible.

WEAVE
: contraction is likely to create a new timbral entity, while expansion will augment the original with repeating sub-patterns.

End of Technical Discussion

Last Updated 17 Sep. 2016
Documentation: Archer Endrich, revised R.Fraser