CDP PSOW

Operations on Pitch-Synchronous Grains

(with Command Line Usage)

The PSOW program group manipulates FOF-like sound grains in the time domain. This implies not only vocal sounds as source material, but more particularly vowel sounds, spoken or sung. **Input sounds must be MONO**.

These programs make use of a pitch trace extracted in binary form from an analysis file by **REPITCH GETPITCH Mode 1** and then converted to a *time frequency* text breakpoint file by **PTOBRK**. Note the three steps:

1. Analyse the input source file with PVOC
2. Get its binary pitch trace with REPITCH GETPITCH Mode **1** *without 'retaining pitch zeros'* (no **-z** flag)
3. Convert binary pitch trace file to a breakpoint file with PTOBRK. Either the **.brk** or the **.txt** extension may be used. (Both are used in this manual.)

You need to provide a *pitch-brkpnt-data* file for each sound used as an input to a PSOW process (when required by the Usage). Note that this also needs to happen for PSOW outputs which are then used as inputs to other processes.

See **'Introducing PSOW'** for a general introduction to this program set,
APPENDIX 1 for an additional discussion of `FOF',
APPENDIX 2 for batch files to automate the 3-step procedure listed above, and
APPENDIX 3 for a summary of the 6-Step procedure to follow when running the PSOW program set from within *Sound Loom*. Note that its PROCESS button for PSOW is named 'FOFS'.

Please note that the sound examples provided serve to illustrate the operation of the various functions within PSOW. Much more exploration of the program is needed to discover the full sound-shaping potential of this software.

Functions to manipulate FOF-like grains

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

Introducing PSOW
Explaining the nature and purpose of the PSOW program set.

CHOP
Chop sound into sections between specified grain (chunks) OR: Chop away sections of soundfile that you DON'T want to manipulate with PSOW functions.

CUTATGRAIN
Cut at exact grain time

DELETE
Time shrink sound by deleting a proportion of the pitch-synchronised grains

DUPL
Timestretch/transpose a sound by duplicating the pitch-synchronised grains

FEATURES
Impose new features on vocal-type sound, preserving or modifying FOF-grains

[FOFEX EXTRACT]
Extract FOFs to a file or to separate soundfiles

[FOFEX CONSTRUCT]
Superimpose FOFs to make output FOF

GRAB
Grab a pitch-synchronised grain from a file, and use it to create a new sound

IMPOSE
Impose vocal FOFs in 1st sound onto the 2nd sound

INTERLEAVE
Interleave FOFs from two different files

INTERP
Interpolate between 2 pitch-synchronised grains, to produce a new sound

LOCATE
Locate exact start time of nearest FOF-grain

[PTOBRK WITHZEROS]
Convert pitch trace from binary .frq to text breakpoint file (.txt or .brk) for PSOW

REINFORCE
Reinforce harmonics in a vocal-type FOF-grain file

REPLACE
Combine FOFs of 1st sound with the pitch of the 2nd sound

SPACE
Distribute the alternate FOFs in the sound over a stereo space

SPLIT
Split vocal FOFs into subharmonic and upwardly transposed pitch

STRETCH
Timestretch/transpose a sound by repositioning the pitch-synchronised grains. The grains themselves are not time-stretched

STRTRANS
Timestretch/transpose a sound by repositioning the pitch-synchronised grains

SUSTAIN
Sustain a pitch-synchronised FOF within a sound

SUSTAIN2
Sustain an explicitly specific FOF within a sound

SYNTH
Impose vocal FOFs on a stream of synthesised sound

[TWEET]
Replace FOFs in vocal sound by synthetic tweets or noise

Appendix 1 - About 'FOF'
Background and further information about the term 'FOF'.
Appendix 2 - Batch files
Batch files to automate the 3-step preparation of the *pitch-brkpnt-data* file
Appendix 3 - PSOW/FOF in *Sound Loom*
Step-by-step procedure to run PSOW from within *Sound Loom*.
Appendix 4 - Producing "Song" in *Sound Loom*
Using 'Sustain a specific FOF within a sound' to produce "Song"
.

Introducing the PSOW Program Set

Vocal sounds are amazingly complex. The cavities of the mouth and head form resonance chambers that impose fixed tonal qualities on the sound, whatever the pitch. These are called formants. Depending on the speaker, vocal sounds can have a wide pitch range and a musical quality. They are also made up of a series of impulses, known as FOFs.

The PSOW set works with the pitch and impulse qualities of the sound. Thus the first step is to extract the pitch trace of the sound, following the procedure described below. The impulses can be thought of as **small packets of waves**. The shape of the particular packet defines the colour (e.g. the vowel) of the vocal sounds, while their rate of their occurrence (how many per second) determines the pitch of the vocal sound. FOFs are smaller than syllables, and in fact smaller than audible 'grains' (for instance the tongue-flaps making up a rolled-'rr'), but larger than the wavecycles (they may contain several wave-cycles, defining particularly the vowel formants at that moment).

The general purpose of the PSOW programs is to attempt to find the FOFs in a sound, and work with them in various ways. They assume sounds are made up of FOFs; they may produce interesting but unpredictable results on sounds that are not intrinsically voice-like. Also, they will work best with spoken or sung vowels ... sibilants will usually produce some artefacts. **They apply only to mono sources**.

The basic procedure for preparing to use PSOW is this:

1. First use PVOC ANAL to analyse a MONO vocal soundfile. This produces an analysis file (**.ana**).

2. Extract the pitch trace from the analysis file with REPITCH GETPITCH, saving this as a **binary pitch data file** (Mode **1**, ensuring that you do NOT retain 'pitch-zeros'. This is the program default. For PSOW we need to retain **signal** zeros but not **pitch** zeros. To retain pitch-zeros, you have to use the **-z** flag – 'retain unpitched windows', so we avoid this. (Using pitch-extraction *directly* to a text file will not work for PSOW applications as it doesn't handle signal-zeros and pitch-zeros in quite the right way.)
   After extracting the pitch trace and saving to a binary pitch data file using the program default (i.e., optional **-z** flag is NOT used), we now have a binary pitch data file containing the pitch trace of the sound. It contains, by default, 'zeros', meaning indications of where there is no significant **signal** in the source. It does NOT contain 'pitch-zeros', meaning indications of where there is no significant **pitch** in the source.

3. Convert this binary pitch data file to **text data** using PTOBRK. **Your new text file is then used as the first parameter to the PSOW programs**, referred to in the Usage as *pitch-brkpnt-data*. Please note that the textfile (i.e., the input soundfile) must contain some significant pitch information for the PSOW program to work (sounds with multiple pitches, inharmonic or noisy sounds will probably not work very well with PSOW).

NB: *Soundshaper* runs this sequence automatically for the input soundfile(s), so you don't have to worry about it. It also accepts stereo and multi-channel input, which is split into separate MONO channel files before PSOW is run and re-assembled after processing.

Example processing sequence for the above: (The extensions are shown for clarity, but are not used in the GUIs.)

```
pvoc anal 1 six.wav six.ana
repitch getpitch 1 six.ana sixdummy.ana six.frq
ptobrk withzeros six.frq six.txt 20
```

PSOW processes also often contain a parameter ***segcnt***, indicating the number of FOFs in each block that is manipulated by the program. If this parameter is greater than 1 (say *N*), time-stretching, for example, will work on grains which contain *N* FOFs (rather than single FOFs), and you will hear a series of grains in the sound, similar to using brassage with low density and large grainsize.

PSOW CHOP – Chop sound into sections between specified FOF-grain (chunks) OR: Chop away sections of soundfile that you DON'T want to manipulate with PSOW functions

Usage

psow chop *infile outfile-rootname pitch-brkpnt-textfile time-graincount-pairs*

Example command line to chop grains into groups:

```
psow chop omahumc omahumcchop omahumc.brk omtimes.txt
```

omtimes.brk

```
Time    Graincount
1.0     5
3.5     5
7.0     5
```

The result of this process is the following:
omahumcchop.wav
skips over FOF 5-seg chunk at 1.0 - get with GRAB
omahumcchop_001.wav
skips over FOF 5-seg chunk at 3.5 - get with GRAB
omahumcchop_002.wav
skips over FOF 5-seg chunk at 7.0 - get with GRAB
omahumcchop_003.wav

Parameters

insndfile – input mono soundfile
outsndfile-rootname – base name for a series of soundfile outputs; the first has no number attached, and subsequent outfiles have '_001', '_002' etc. appended.
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information. See the Introduction to revise the steps needed to create this file.
time-graincount-pairs – This is a text file of cut times expressed as *time grain-count* value pairs.

> *Time* is the time of the grain where the file is cut. (The file is automatically cut at a start-of-grain boundary. The next segment begins AFTER the specified grain-chunk.)
> *Graincount* is the number of grains in the chunk, at a specified time, before the next cut-section starts. Recommended values are very low, e.g., 1 or 2. (Values greater than 1 will result in granulated outfiles, often the desired result.)

Understanding the PSOW CHOP Process

This process extracts lengths of pitch-focused sound from the input soundfile, saving each length as a separate soundfile. It does not remove anything from the original soundfile, which remains intact. These extracted lengths will be the sections that you DO NOT want to manipulate with one of the PSOW functions. The point is to **cut around** the material you want to FOF-process, process the material, and then splice back the extracted lengths (using 0 length splices).

This process cuts up and stores a sound as a sequence of segments, extracting the material **around** the points where you want to apply FOF processing. The output is therefore at least 2 sound segments. You can then use PSOW GRAB (using the same edit points) to capture specific FOFs in a sound, extend them, and then reinsert them into the original sound at the edit points, by splicing the segments cut using PSOW CHOP and the new sounds made with PSOW GRAB in the correct order.

Alternatively, you can use PSOW GRAB first, possibly making use of its processing features to develop the sound extracted right away. You would then use PSOW CHOP with the same *segcnt* (= *graincnt*) to save the material around these grabbed-and-transformed segments so that you could splice it all back together again. When the sound transformations are carried out while running PSOW GRAB, a special *pitch-brkpnt-data* text file for the grabbed portion is not needed, as the one for the sound from which the portion is grabbed is being used. It would be needed if you were going to use the grabbed portion with a different function, except for PSOW INTERP, which does not require a *pitch-brkpnt-data* file. See PSOW GRAB for an example of a sound transformed while grabbed.

The *time grain* pairs (File of *cuttime : grains-in-chunk* pairs) is a textfile containing pairs of values, where the first value is a time for a cut, and the second value determines how many FOFs are to be 'stepped over' after this cut point. Note that a 'FOF' is very short – only a few milliseconds. The cut times therefore indicate where the FOF material to use is located: PSOW CHOP cuts around them, and PSOW GRAB gets the FOF material at these points for processing, and SFEDIT JOIN splices the resulting files back into a single soundfile.

Additional Observations
The real point of PSOW CHOP is to cut away what you do not want to FOF-manipulate (e.g., consonants) and save this unwanted material as separate soundfiles. This means studying your sound in a sound editor (or by ear using FROM-TO playback) to determine where the good material you want to use as FOF source is. Note that the actual FOF material will be tiny: we are thinking 'grains', so the unused portions will contain the rest of the vowel material as well as consonants. It is not just a matter of finding start and end points for consonants and editing them out, a persistent misconception I have had about PSOW, and another way entirely of working with vocal material.

In the 'time-grain' value pairs text file, you specify the start times for your FOF grains and their length as a number of 'grain-chunks'. PSOW CHOP then cuts out all of the material before and after these FOF sections (i.e., the material 'around' them) saving it as a series of separate soundfiles. After manipulating the FOF material with the other PSOW functions, you are then able to put back the original material by splicing these separate soundfiles (SFEDIT JOIN) with your FOF-manipulations, thus recreating aspects of the original vocal material, but now with FOF-manipulated enhancements inbetween.

It is very important to remember that the CHOP process cuts 'around' the desired FOF source material. You do not have it yet! You need to use PSOW GRAB to extract the FOF grains that you will manipulate. But note that you can use the same times as used in PSOW CHOP: instead of cutting around them, you now cut out and keep the FOF grains. (PSOW CUTATGRAIN can also be used to cut out portions of *infile*.)

However, you cannot use these newly acquired FOF grains until you create a *pitch-brkpnt-data* file for them, except for PSOW INTERP (for which the duration parameter must be 0.0). **The lengths of the FOF-source file and the *pitch-brkpnt-data* file must match. This means repeating for each FOF grain soundfile the 3-step procedure you had to do before using PSOW CHOP:**

ANALYSE: pvoc anal 1 ingrabfile.wav outgrabfile.ana
GET PITCH TRACE: repitch getpitch 1 outgrabfile.ana outdummyfile outgrabfile.frq
CONVERT TO BREAKPOINT: ptobrk withzeros outgrabfile.frq outtgrabfile.txt 20

In the Appendix 2, I provide generic batch files to do this with any input.

NB: *Soundshaper* runs these processes behind the scenes for the input file.

Musical Applications

PSOW CHOP therefore enables you to carry out FOF processing with pinpoint precision, keeping the rest of the soundfile (now saved as a series of soundfiles) untouched. You then put it altogether again with SFEDIT JOIN. Note that you can repeat the same soundfile in the list to join, as well as reassemble the component soundfiles in any order.

This program is a prime candidate for putting together a batch processing sequence, either with a command line interpreter, *Sound Loom's* **Instruments** or *Soundshaper's* **Patch** system.

End of PSOW CHOP

PSOW CUTATGRAIN – Cut at exact grain time

Usage

psow cutatgrain mode *insndfile outsndfile pitch-brkpnt-data time*

Example command line to perform the cut:

```
psow cutatgrain 1 om omcutatgrain om.txt 0.4
```

Modes

1 Retain file BEFORE (exact) specified grain time.
2 Retain file AT and AFTER (exact) specified grain.

Parameters

insndfile – input mono soudfile
outsndfile – output cut soundfile
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
time – in seconds at which to cut the file

Understanding the PSOW CUTATGRAIN Process

This process allows you to cut a sound at a FOF boundary (the one nearest to the time you specify). This may be useful, as the files are cut at zero-crossings in the signal, and can hence be joined together, without splices, in various ways.

In *Sound Loom*, the time you set in CUTATGRAIN is automatically transferred to the edit-time box in GRAB (and *vice-versa*.

Musical Applications

Here we are specifying a time at which presumably we intend to perform a PSOW FOF-manipulation. With PSOW CUTATGRAIN we can cut and retain the portion of sound *before* this time (Mode **1**) or *at and after* this time (Mode **2**). The cut is automatically made at a zero-crossing, creating ideal conditions for a trouble-free splice later.

End of PSOW CUTATGRAIN

PSOW DELETE – Time-shrink sound by deleting a proportion of the pitch-synchronised grains

Usage

psow delete insndfile outsndfile pitch-brkpnt-data propkeep segcnt

Example command line to create a proportional data reduction (in this case, 1 in 3 are retained):

```
psow delete omahumc omahumcdel omahumc.brk 3 5
```

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile(s)
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
propkeep – proportion of chunks to keep. '2' keeps 1 in 2; '7' keeps 1 in 7, etc.
segcnt – number of grains in a chunk

Understanding the PSOW DELETE Process

This process will timeshrink a sound, without changing pitch or vowels. FOFS-per-grain (**segcnt**) greater than 1 give more realistic results.

This is a straightforward thinning process, like straining the sound through different sizes of mesh. Note that the 'mesh' is specified in two ways: the proportion to keep and the length of the grain chunk.

Musical Applications

Let's consider several types of 'mesh':

- keeping a high proportion of big chunks
- keeping a high proportion of small chunks
- keeping a low proportion of big chunks
- keeping a low proportion of small chunks
- keeping a midrange proportion of medium-sized chunks

End of PSOW DELETE

PSOW DUPL – Timestretch/transpose a sound by duplicating the pitch-synchronised grains

Usage

psow dupl *insndfile outsndfile pitch-brkpnt-data repeat-cnt segcnt*

Example command line to duplicate pitch-synchronised grains:

```
psow dupl om omdupl om.txt 8 5
```

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile with duplications
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
repeat-cnt – number of repetitions in each chunk.
segcnt – number of grains in a chunk.

Understanding the PSOW DUPL Process

This process time-stretches the sound, but preserves the pitch (like MODIFY SPEED) and preserves the vowel formants. FOFS-per-grain (**segcnt**) is best set at 1 but other small values do not alter the output a great deal. Also see **PSOW STRETCH**.

I have found that inputs made with PSOW GRAB can have very few grains in them, and you can get an error message "too few grains". For example, if the GRAB file has 4 grains in it and you give a *segcnt* of 5, it will fail for this reason. Note that it will also fail if *segcnt* is 4, because the count starts at 0: 0-1-2-3 is 4 grains. Thus in this case, *segcnt* = 3 will succeed.

Also, please remember that you cannot use a GRAB file until you have created the corresponding *pitch-brkpnt-data* file for it (the 3-step procedure).

Musical Applications

This is another way of lengthening a sound. The number of repetitions affects how much longer it will be, while the number of grains in a chunk affects granulation. If *segcnt* = 1, the result will be as smooth as it is possible to be with this kind of input and process. There may be pitch steps related to the pitch trace of the file. If *segcnt* is considerably longer, e.g., 10 or 15, the result will be more granulated, i.e., have a coarser grain.

End of PSOW DUPL

PSOW FEATURES – Impose new features on vocal-type sound, preserving or modifying FOF-grains

Usage

psow features mode *insndfile outsndfile pitch-brkpnt-data segcnt trans vibfrq vibdepth spectrans hoarseness attenuation subharmno subharmamp FOF-stretching* [**-a**]

Example command line to impose new features:

```
psow features 1 om omfeatures1 5 3 3 3 7 0.5 0.25 2 0.5 100
```

Modes

1 Transposition accompanied by 'timewarp': pitch is higher and the sound is shorter, i.e., standard Time Domain transposition.
2 The transposed pitch is accompanied by an additional lower pitch.

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
segcnt – number of grains in a chunk to be retained as-is.
transposition – pitch transposition in semitones. The two Modes of the program relate to this parameter.
vibfrq – frequency of any added vibrato.
vibdepth – depth in semitones of any added vibrato.
spectrans – amount of transposition of the spectrum in semitones (will not change the fundamental pitch).
hoarseness – degree of vocal hoarseness (roughness) to introduce. Range: 0 to 1.
attenuation – attentuation. Range 0 to 1. May be necessary when FOF-stretching, due to overlaps.
subharmno – amount by which the fundamental pitch is divided. Both 0 and 1 give NO subharmonics.
subharmamp – amplitude level of any subharmonic introduced. Range: 0 to 1.
FOF-stretching – time extension of the FOF components. This does NOT stretch the overall soundfile length. Range: 1 to 512.
-a – option to use an alternative algorithm for the FOF stretch.

Understanding the PSOW FEATURES Process

- Mode 1 transposing involving timewarp does its pitch transposition like MODIFY SPEED, shortening the sound if the pitch goes up.

- Mode 2 transposing involving double pitches does not change the timeframe of the sound but may introduce octavation or other double-pitch features when it transposes by large amounts.

PSOW FEATURES combines various elements of some of the other PSOW processes, and the parameters have the same functions, except that the transposition parameters are here entered in semitones rather than ratios.

Some additional information about the parameters may be useful:

- *attenuation* – This parameter attenuates the output (applies amplitude reduction), and may be necessary when using a 'FOF-stretching' value of more than 1.
- *subharmno* ('Subharmonic number') – This determines the pitch of the subharmonic generated. You can use 0 or 1 if you do not want a subharmonic. For example, 2 divides the fundamental pitch by 2, producing a pitch an octave lower, 3 divides it by 3, producing a pitch an octave and a 5th lower and so on.
- *subharmamp* ('subharmonic level') – This determines the loudness of any subharmonic introduced.
- *FOF-stretching* sustains individual FOFs (it does NOT time-stretch the sound) producing an effect akin to reverberation.

Musical Applications

This program is one of the most versatile in the set.

End of PSOW FEATURES

FOFEX – Operations using pitch-synchronous grains (FOFs) to (re)construct soundfiles
A sub-Group of two: FOFEX EXTRACT FOFEX CONSTRUCT

Usage

fofex NAME (mode) *infile outfile parameters*:
where NAME can be any one of
extract construct

Type **fofex extract** for more info on fofex extract..etc.

FOFEX EXTRACT – Extract FOFs to a file or to separate soundfiles

Usage

fofex extract 1 *infile outfiles 0 pitch-brkpnt-data minlevel fofcnt* [**-w**]
OR
fofex extract 1 *infile outfiles excludefile pitch-brkpnt-data minlevel fofcnt* [**-w**]

fofex extract 2 *infile outfile pitch-brkpnt-data time*
fofex extract 3 *infile generic_outfile_name pitch-brkpnt-data minlevel fofcnt* [**-w**]

Modes

1 All FOF(group)s extracted to special FOF-file, to use with **FOFEX CONSTRUCT**
2 A single FOF(group) is extracted
3 All FOF(group)s are extracted to separate soundfiles

Parameters

infile – input soundfile
outfile(s) – output binary FOF file or to soundfiles

OUTPUT OF MODE 1 is:
1) a fofbank soundfile containing extracted FOFs in ascending pitch order
2) a fofinfo textfile of position of (semitone) pitchsteps in soundfile

pitch-brkpnt-data – a breakpoint file of *time hertz* pairs, created by **PTOBRK** .

The file may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.

excludefile (Mode 1) – text file of pairs of times as sample-counts, defining areas in the source from which FOFs will NOT be extracted (i.e. sample-count at start of passage and sample-count at end of passage).
minlevel – level in dBs below the level of loudest FOF found, below which FOFs are rejected. NB Zero means no FOFS are rejected.
fofcnt – size of FOF group to extract
time (Mode 2) – time in file at which to extract FOF(group)
-w – FOFs are windowed (cosine smooth of edges)

Understanding the FOFEX EXTRACT Process

Note that this process is paired with **FOFEX CONSTRUCT** (see below). Suitable applications have yet to be determined.

To determine time as sample-count (for Mode 1's *excludefile*), see the utility **SNDINFO TIMESMP**

Musical Applications

End of FOFEX EXTRACT

FOFEX CONSTRUCT – Superimpose FOFs to make output FOF

Usage

fofex construct 1 *fofbank outfile fofinf pbrk env gain f1*
fofex construct 2-5 *fofbank outfile fofinf pbrk env gain* [**-n**]
fofex construct 6 *fofbank outfile fofinf pbrk env gain f1 f2 minf maxf* [**-n**]
fofex construct 7 *fofbank outfile fofinf pbrk env gain f1 f2 f3 minf maxk minl maxl* [**-n**]

Modes

1 Use 1 FOF only
2 All FOFs superimposed to make output FOF
3 Low FOFs superimposed to make output FOF
4 Midrange FOFs superimposed to make output FOF
5 High FOFs superimposed to make output FOF
6 Use 2 FOFs,varying weighting with pitch of output
7 Use 3 FOFs,vary weighting with pitch & level of output

Parameters

fofbank – soundfile of extracted FOFS created by **FOFEX EXTRACT**
outfile – output soundfile [or FOF-file?]
fofinf – a textfile with position of pitchsteps in soundfile, also created by the **FOFEX EXTRACT**
pbrk – breakpoint file of *time hertz* information, created by **PTOBRK**.
(May contain zeros (no-signal) but NOT pitch-zeros (no-pitch). Must have SOME freq. info.)
env – breakpoint-Envelope file, as text
gain – output gain
f1(f2 f3) – in mode 1, FOF to use, (lowest=1).n

In modes **6,7** use specifically numbered FOFs

minf maxf – minimum and maximum **frequencies** for variation of the internal FOF balance
minl maxl – minimum and maximum **levels** (0-1) for variation of the internal FOF balance
-n – set all FOFs to same level before synthesis

Understanding the FOFEX CONSTRUCT Process

This process is paired with **FOFEX EXTRACT** (see above).
Suitable applications have yet to be determined.

Musical Applications

...

End of FOFEX CONSTRUCT

PSOW GRAB – Grab a pitch-synchronised grain from a file, and use it to create a new sound

Usage

psow grab *insndfile outsndfile pitch-brkpnt-data time outfile_duration segcnt spectrans density randomisation gain*

Example command line to grab a pitch-synchronised grain. In the section for PSOW CHOP above, 3 times were specified, around which to CHOP. Now we will GRAB two FOF-chunks at those times, specifying 0 for the outfile duration so that we can use them with PSOW INTERP. Please note, these FOF-chunks are very short, so may not be audible upon playback.

```
psow grab omahumc omgrab1 omahumc.brk 1.0 0 5 0.001 1 0 1
psow grab omahumc omgrab2 omahumc.brk 1.0 0 5 0.001 1 0 1
```

Example command line for a sound that is transformed using PSOW GRAB facilities while it being grabbed:

```
psow grab omahumc omahumcgrab omahumc.brk 1.0 5 10 16 2 0.6 1
```

Also, please remember that you cannot use a GRAB file until you have created the corresponding *pitch-brkpnt-data* file for it (the 3-step procedure).

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile made from the grabbed grain-chunk
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
time – time in seconds at which to grab the grain(s).
duration – Duration of the output soundfile. If *duration* is set to 0, a single grain (chunk) is grabbed.
segcnt – The number of grains in a chunk.
density – The rate at which the grain-chunks in the output soundfile succeed one another. **1**: the grains follow one after the other; **2**: grain-chunks overlap by 2 grain-chunks; **3**: grain-chunks overlap by 3 grain-chunks, etc. **0.5**: grains are separated by an equivalent amount of silence, etc. Range: 0.125 to 8.0
spectrans – amount of transposition of the spectrum in semitones (will not change the fundamental pitch). Range: 0.001 to 8.0. NB: 0 for no transposition is not accepted.
randomisation – Randomisation of the position of the grain-chunks in the output soundfile. This randomisation introduces noisiness into the output sound. Range 0 to 1.
gain – Amplitude adjustment applied to the output. 1 = full amplitude.

Understanding the PSOW GRAB Process

This process grabs an individual (group of) FOF(s) and uses it to produce a new sound. You must specify the time in the source sound where the FOF-to-be-grabbed is located, and the duration of the output sound to produce. Duration ZERO grabs a single (group of) FOF(s). These outputs are specifically for use with **PSOW INTERP**. While they may actually run in other functions, they will be too short to do anything useful. **Therefore, when using GRAB to create FOF-grains to use with other functions, give a duration suitable for the purpose.**

Note that if there is no pitch at the grab-time you choose, you will not get any result. The other parameters provided with PSOW GRAB enable you do carry out a significant amount of processing at the same time.

- **Density** (pitch transposition) specifies transposition of the fundamental as a frequency ratio. So 2.0 is an octave up, and 0.25 is 2 octaves down)
- **Spectrans** (spectral transposition) specifies the transposition of the **spectrum** in the output sound. This is a bit like changing the vowel without changing the pitch.
- **Rand** (randomisation) randomizes the position of the FOFs in the output, introducing a hoarse noisiness.
- **Gain** (amplitude adjustment downwards) may be needed if density is greater than one, because the overlaying of grains will sum amplitudes.

In *Sound Loom*, when you grab a FOF, the time you set is automatically transferred to the edit-time box in CUTATGRAIN (and *vice-versa*.

Musical Applications

The most direct use for outputs of PSOW GRAB is as inputs to PSOW INTERP – interpolating between two pitch-synchronised grains. GRAB files can be used for other functions, but bear in mind the advice given above: first to create the matching *pitch-brkpnt-data* file, and to grab a sufficient number of grains appropriate for the function that you are then going to use.

End of PSOW GRAB

PSOW IMPOSE – Attempts to impose vocal FOFs in 1st sound onto a 2nd sound

Usage

psow impose *insndfile1 insndfile2 outsndfile pitch-brkpnt-data depth wsize gate*

Example command line to impose FOFs:

```
psow impose om sixi omimpsixi om.txt 1 20 -40
```

Parameters

insndfile1 – input mono soundfile from which to take data
insndfile2 – second mono soundfile onto which to impose data
outsndfile – output soundfile
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
depth – Depth of application of FOFs to the second sound. Range: 0 to 1.
wsize – Windowsize in milliseconds to track the envelope of the second sound (for normalisation purposes). Recommended value 20(?)
gate – Level in decibels in the second sound at which it is assumed to be zero (full amplitude). Sound above the gate level is retained. A recommended level would be -40 or less. The letters 'dB' are not used when specifying the *gate* level.

Understanding the PSOW IMPOSE Process

This is an experimental program that attempts to impose the FOF characteristics of one sound onto another.

Some of the parameters could use further explanation:

- **Depth** of application relates to the degree to which the 2nd soundfile is affected. If *depth* = 0, the 2nd sound is not altered. If *depth* = 1, the 2nd sound is completely altered by the 1st. Values inbetween give intermediate results: closer to 0 results in a limited degree of alteration, whereas values closer to one create more alteration.

- **Wsize** sets the length of the segments in the second soundfile to use to determine amplitude. Longer lengths mean less resolution, i.e., less responsiveness to changes of amplitude. The results affect how the second sound is normalised. In a fairly steady-state resultant sound, longer windows may be fine, but if there are rapid changes of amplitude, it will be better to use a finer resolution: a smaller *wsize*.

- **Gate** enables you to set the maximum level in the output soundfile. Note that the parameter is in decibels. A gain reduction of 0.7 represents -3dB. See our **Chart** of Gain-Decibel relationships.

Musical Applications

The pitch synchronous grains of the first soundfile, which emphasise vowel material, colour those of the second. The *weight* parameter in Mode **2** seems particularly useful. Note that the focus is on the psow-grains, and not formants (vowel resonances), so the aural results will be different from the 'cross-synthesis' achieved with **FORMANTS VOCODE**.

End of PSOW IMPOSE

PSOW INTERLEAVE – Interleave FOFs from two different files

Usage

psow interleave *insndfile1 insndfile2 outsndfile pitch-brkpnt-data1 pitch-brkpnt-data2 grouplenth bias balance weight*

Example command line to create an interleaved effect biased towards the 2nd soundfile:

```
psow interleave om sixi ominterlsixi om.txt sixi.txt 5 -1 0.8 0.8
```

Parameters

insndfile1 – input mono soundfile1
insndfile2 – second input mono soundfile
outsndfile – output interleaved soundfile
pitch-brkpnt-data1 – text file with a pitch trace in *time frequency* breakpoint form for *insndfile1*. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
pitch-brkpnt-data2 – similar file for *insndfile2*.
grouplength – the number of FOFs in each manipulated segment
bias – Is the output pitch biased to one or the other of the infiles? **0**: no bias; **1**: biased to the first soundfile; **-1**: biased to the second soundfile.
balance – the level balance of the components of the two input soundfiles to be placed in the output soundfile: **1**: equally loud; **> 1**: the first soundfile is louder; **< 1**: the second soundfile is louder.
weight – relative number of components from the two input soundfiles to place in the output soundfile: **1.0**: equal amounts; **> 1**: more of the first soundfile; **< 1**: more of the second soundfile.

Understanding the PSOW INTERLEAVE Process

This process alternates (groups of) FOFs from two different sounds and hence requires 2 pitch-data files, one corresponding to each of the input sounds.

You may notice in the example above that om.wav is 0.9 sec long, sixi.wav is 3.0 sec long, and the output ominterlsixi.wav is 0.929 seconds long, very close to the shortest soundfile. Sometimes the output is actually shorter than the shortest input. This is thought to be caused by FOFs not being found in some regions of the file (T. Wishart).

Using the parameters:

- ***Bias*** determines to what extent the output pitch is biased towards the pitch of one file or the other (values: 0, no bias, 1 biased to 1st, -1 biased to 2nd) with intermediate values giving intermediate degrees of *bias*.
- ***Balance*** determines the relative loudness of the two components. **1** gives an equal balance, while **values greater than 1** make the 1st louder **and values less than 1** make the 2nd louder.

- ***Weight*** (Mode **2**) determines the relative number of FOF components from the 2 sources in the input, and the parameter works like the *balance* parameter.

Musical Applications

This function performs the same sort of operation as **COMBINE INTERLEAVE**, but with the focus on FOF-grain type sonic material. One hears a pulsed interleaving of the two sounds. with *grplen* determining the resolution of the interleave (fine with low values, coarse – longer units – with higher values).

The three parameters to balance the relative strengths of the two inputs are very important and useful in determining the tonal quality of the output. Fractional values are allowed, and they can be adjusted with great precision.

End of PSOW INTERLEAVE

PSOW INTERP – Interpolate between 2 pitch-synchronised grains, to produce a new sound. (Grains acquired by PSOW GRAB, with duration 0.0)

Usage

psow interp *insndfile1 insndfile2 startdur interpdur enddur vibfrq vibdepth tremfrq tremdepth*

Example command line to create an interpolated effect:

```
psow interp omgrab1 omgrab2 ominterp1-2 0.25 5 0.25 1.7 0.6 1.5 2
```

Parameters

insndfile1 – first input mono soundfile made by PSOW GRAB from a single grain-chunk, with *duration* set to 0.0
insndfile2 – second input mono soundfile made by PSOW GRAB from a single grain-chunk (in the same or a different soundfile), with *duration* set to 0.0
outsndfile – output soundfile interpolating between the grain-chunks
startdur – length of time in seconds to sustain the initial grain
interpdur – duration of the interpolation, affecting the length of the outfile.
enddur – length of time in seconds to sustain the final grain
vibfrq – frequency of any added vibrato (can be 0)
vibdepth – depth in semitones of any added vibrato. Range: 0 to 3.
tremfrq – frequency of any added tremolo (can be 0)
tremdepth – depth of any added tremolo

> **N.B.:** This interpolation process assumes that your input soundfiles each contain a single pitchsync (FOF) grain obtained using PSOW GRAB with output duration 0.0. However, you can GRAB with a longer *segcnt* (keeping the output duration 0.0. This produces a more granulated result from PSOW INTERP. Also note that the two *vib* and two *trem* parameters will accept fractional values. The pitch trace files are not needed.

Understanding the PSOW INTERP Process

This process creates a new sound by interpolating between two SINGLE (groups of) FOFs extracted with zero duration from the same source sound (or different sources) by **PSOW GRAB**. It therefore does not need the usual *pitch-brkpnt-data* textfile. If you use input files not created with PSOW GRAB, you will get the error message: "File 1 is not a valid pitch-sync grain file."

Note that the process was intended to work with single FOFs as input, but will also work with groups of FOFs, and between groups of FOFs of different sizes. To get larger 'groups of FOFs', make *segcnt* longer when using PSOW GRAB. As noted above, a longer *segcnt* in the sources increases the granulation in the output of PSOW INTERP.

The output sound can be shaped using the various parameters. *Startdur* and *enddur* shape the beginning and the end by allowing you to sustain the first and last grains. The length of time *interpdur* over which to realise the interpolation between the two grains, can also be set. Note that 'vibrato' refers to the rate and amount of frequency change, and 'tremolo' does the same with amplitude.

Musical Applications

A 'plain vanilla' output can be made by setting the two *vib* and two *trem* parameters to 0. But when applied, they can thicken and otherwise shape the output sound in very precise ways. The aural difference between the two input sounds (vowels, usually) and the length of time over which the interpolation takes place, shape the nature of the transition.

You might want to compare the nature of the output (somewhat granulated) with transitions produced by MORPH BRIDE, MORPH GLIDE or COMBINE CROSS.

End of PSOW INTERP

PSOW LOCATE – Locate the exact start time of the nearest FOF-grain

Usage

psow locate *insndfile pitch-brkpnt-data time*

Example command line to find the exact location of a grain:

> **psow locate om om.txt 0.4**
> Resulting on-screen display:
> "INFO: TIME 0.400000 IS NEAREST GRAIN START 0.404354"

Parameters

> *insndfile* – input mono soundfile
> *outfile* – none - screen display only and therefore cannot use in a batchfile
> *pitch-brkpnt-data* – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
> *time* – time at which to find the start of a grain

Understanding the PSOW LOCATE Function

> This process returns the *exact start time* of the FOF nearest to the time in the source that you specify.

Musical Applications

> This utility can help to determine times for CUTATGRAIN or GRAB with more precision. It may be useful to know where exactly the file is being cut, e.g., when going on to some other process in which material has to synchronise exactly to the sample.

End of PSOW LOCATE

PTOBRK WITHZEROS – convert pitch trace from binary .frq to text breakpoint file (.txt or .brk) for PSOW

Usage

ptobrk withzeros *binary-pitchfile outtextfile min-pitch-dur*

Example command line to convert pitch trace from **.frq** to **.brk**:

```
ptobrk withzeros om.frq om.brk 20
```

Parameters

binary-pitchfile – binary pitch trace file created by REPITCH GETPITCH Mode **1 without retaining pitch zeros**. On the command line, do NOT use REPITCH's **-z** flag (which retains pitch zeros). The point is to eliminate the non-pitch data. The extension for this file should be **.frq**.
outtextfile – text breakpoint file format for the pitch trace data. This format is required by PSOW.
min-pitch-dur – gives minimum time (in milliseconds) that any stretch data must persist to be regarded as valid data. Range: 1 to 1000 ms. Recommended value: 20ms.

Understanding the PTOBRK WITHZEROS Function

This function should be used instead of REPITCH GETPITCH Mode **2**, which creates a breakpoint textfile of pitch data. REPITCH GETPITCH Mode **1** is used *without the* ***-z*** *flag* to create a **.frq** file with no-pitch and no-sound markers. PTOBRK WITHZEROS then converts this kind of **.frq** file to a *time herz* breakpoint file.

Musical Applications

This is a simple utility for creating the correct format for the pitch trace used in PSOW to make the FOF-grains pitch-synchronous.

The creation of this 'pitch-brkpnt-data' file for PSOW is in fact a 3-step process involving PVOC analysis, REPITCH GETPITCH and PTOBRK. This may sometimes be taken care of behind the scenes in a GUI, and may sometimes need to be done by hand. The PSOW Reference Manual **Appendix 2** provides batch files to speed up this process.

End of PTOBRK

PSOW REINFORCE – Reinforce harmonics in a vocal-type FOF-grain file

Usage

psow reinforce 1 *insndfile outsndfile reinforcement-data pitch-brkpnt-data* [**-d***delay*] [**-s**]
psow reinforce 2 *insndfile outsndfile reinforcement-data pitch-brkpnt-data* [**-w***weight*]

Example command line to create a soundfile with reinforced harmonics:

```
psow reinforce 1 om omreinfm1 reindat1.txt om.txt -d10
psow reinforce 2 om omreinfm2w10 reindat2.txt om.txt -w10
```

reindat1.txt

```
harm_no level
2       1
3       2
4       3
```

reindat2.txt

```
harm_no level
1.1     1
3.5     2
4.8     3
```

Modes

1 Reinforce the **harmonic** content of the sound.
2 Reinforce the sound with **inharmonic** partials.

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile
reinforcement-data – text file with pairs of values: for *harmonic_number level*. The first value represents a harmonic number from 2 to 256, and the second value is level relative to the source, with a range from > 0.0 to 16.0. In Mode **2** the 'harmonics' may be fractional, with a range from > 1 to 256. (For example, in Mode **2**, 1.1 is acceptable.)
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
-d*delay* – optional time in milliseconds by which to delay the onset of the added harmonics.
-s – option to omit FOFs generated for higher harmonics which coincide with FOFs of lower harmonics.
-w*weight* – sustain inharmonic components. A higher weight gives a longer sustain. Please note that a very high weight may cause buffers to overflow. Default: 4.0. Range: 1 to 256.

Understanding the PSOW REINFORCE Process

This process attempts to reinforce harmonics in the sound by overlaying FOFs in particular ways. The special data file required contains paired values for *harmonic_number* and *amplitude*. In Mode **2**, the harmonic numbers can be fractional and produces inharmonic spectra from pitched sounds. In addition, in Mode **2**, these constituents can be sustained for longer, using the *weight* parameter.

Musical Applications

Mode **1** enables us to re-weight the harmonic content and therefore tonal character of the pitched sound. In Mode **2**, the inharmonic character of the output is apparent in the example above.

A review of the harmonic series can be useful for this program, and those that involve subharmonics: namely that the 2nd harmonic is the octave, the 3rd a perfect fifth above that, the 4th two octaves, and the 5th harmonic is two octaves plus a major third, etc. See **PSOW SPACE** and **PSOW SPLIT**, which employ subharmonics.

End of PSOW REINFORCE

PSOW REPLACE – Combine FOFs of 1st sound with the pitch of the 2nd sound

Usage

psow replace insndfile1 insndfile2 outfile pitch-brkpnt-data1 pitch-brkpnt-data2 grpcnt

Example command line to create a FOF-pitch combination:

```
psow replace om sixi omreplacesixi om.txt sixi.txt 5
```

Parameters

insndfile1 – input mono soundfile1
insndfile2 – input mono soundfile2
outsndfile – output soundfile with FOF-pitch combination
pitch-brkpnt-data1 – text file with a pitch trace in *time frequency* breakpoint form for *insndfile1*. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
pitch-brkpnt-data2 – similar file for *insndfile2*.
groupcount – number of FOFs in a grain-chunk

Understanding the PSOW REPLACE Process

PSOW REPLACE combines the FOF structure of the first sound with the pitch of 2nd. It requires two pitch data files, one for each source sound.

Musical Applications

This function comes into its own when the pitch structure of the 2nd file has a distinctive character.

End of PSOW REPLACE

PSOW SPACE – Distribute the alternate FOFs in the sound over a stereo space

Usage

psow space *infile outfile pitch-brkpnt-data subharmno separation balance hisuppress*

Example command line to create spatial distribution:

```
psow space omahumc omahumcspace omahumc.brk 5 -1 1 1
```

Parameters

insndfile – input mono soundfile
outsndfile – output stereo soundfile
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
subharmno – subharmonic number, which divides the frequency of the source. Range: 2 to 5.
separation – spatial separation of alternate FOFs. Range: -1 to 1.

- **0:** no separation, all output is stereo-centred.
- **1:** alternate FOFs go to the widest spread, starting with the far right position.
- **-1:** alternate FOFs go to the widest spread, starting with the far left position.

balance – of left and right components. Range: 0 to 8.

- **1.0:** the leftward and rightward levels are equal
- **> 1:** the leftward signal is divided by *balance*: bias to the right
- **< 1:** the rightward signal is multiplied by *balance*: bias to the left.

hisuppress – suppress high-frequency components. Range: 0 to 1.

Understanding the PSOW SPACE Process

Distributes the alternate FOFs in the sound over the stereo space. Note that placing alternate FOFs to the left and then to the right, causes the sound to drop an octave in pitch, hence the subharmonic number parameter begins with value 2 (producing the octave downwards shift). Higher subharmonic values shift the heard pitch down further (via the subharmonic series).

The fact that the output soundfile sounds lower may be unexpected and deserves further explanation. Trevor Wishart writes: "Because FOFs are being alternated in space, it's like having 2 separate sound-streams, each at ½ the original pitch (the pitch drops by an octave). ***Subharmonic_number*** allows you to drop the pitch further (divide the frequency by 3, 4, 5 – i.e., down an 8^{va} + a 5^{th}, two 8^{vas}, and two 8^{vas} + a major 3^{rd} respectively.) It is like the harmonic series upside-down."

The ***separation*** parameter determines where the FOFs are placed in the space. With a value of 0, they all appear at the centre (and there is no pitch-shift, whatever the subharmonic number parameter), while 1 places the alternate FOFs at the extreme right and left (in that order), and -1 at the extreme left and right (in that order). Intermediate values give intermediate values of spatial separation, which means that fractional values are allowed.

The ***Balance***, with a range from 0 to 8, determines the relative loudness of the signals on left and right speakers. While **1.0** sets the balance equal in both speakers, note that when the value is greater than 1, the level on the left is lowered by **dividing** the signal by the number entered for *balance*, and when the value is less than 1, the level on the right is increased by **multiplying** by the number entered for *balance*. This use of multiplication and division means that the values entered can create major changes in level.

A muffling of the sound will result by making use of the ***hisuppress*** parameter. Together with the ability to lower the pitch of the sound, low, throaty sounds can be produced.

Musical Applications

This is the only PSOW function that produces a stereo output. Given the alternation between speakers, there is some similarity to effects that can be achieved with TEXTURE.

Low, gravelly intonations akin to Tibetan chanting are easily produced with PSOW SPACE. Also note that, unlike other **time domain** downward transpositions, the output sound is lower but not longer.

End of PSOW SPACE

PSOW SPLIT – Split vocal FOFs into subharmonic and upwardly transposed pitch

Usage

psow split *infile outfile pitch_brkpnt-data subharmno uptrans balance*

Example command line to create a split effect:

```
psow split om omsplit om.txt 3 12 4
```

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
subharmno – subharmonic number, which divides the frequency of the source. Range: 3 to 8.
uptrans – upward transposition in semitones. Range: 0 to 48.
balance – amplitude level of the up-transposed components relative to the subharmonics. Range: 0 to 8.

Understanding the PSOW SPLIT Process

Allows you to transpose the pitch up (without changing the timeframe or the vowels) and to add subharmonic frequencies, independently and simultaneously.

Musical Applications

This process makes the sound thicker, a voice sound hoarse and 'gravelly'. Note the wide range of acceptable transposition values and the ability to bring out the higher frequencies by using a higher value for *balance*.

End of PSOW SPLIT

PSOW STRETCH – Timestretch/transpose a sound by repositioning the pitch-synchronised grains. The grains themselves are not time-stretched

Usage

psow stretch *insndfile outsndfile pitch-brkpnt-data timestretch segcnt*

Example command line to create a timestretch effect:

```
psow stretch om omstr8 om.txt 8 3
psow stretch omahumc omahumcstr09 omahumc.brk 0.9 5
```

Parameters

insndfile – input mono soundfile (NB: not meant for use with files produced by PSOW GRAB)
outsndfile – output soundfile
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
timestretch – proportion by which the sound is to be stretched (> 1) or compressed (< 1). (Larger values create longer gaps.) Range: 0.1 to 10.0.
segcnt – number of grains in a chunk retained without change, while gaps between segments are stretched.

Understanding the PSOW STRETCH Process

This process time-stretches the sound, and changes the pitch (like MODIFY SPEED) but will preserve the vowel formants. FOFS-per-grain (**segcnt**) is best set at 1 for realistic results. The result when *segcnt* was one was continuous, but granular in character: a sequence of adjacent grains/pulses.

Musical Applications

Again, the nature of the FOF process as essentially granular in character affects the results. As seen, even 'realistic results' are pulsed. Note how gaps can be produced with higher values for *timestretch* as in our first example. This distinguishes the process from PSOW DUPL, while its granular character is essentially different from normal time-stretch. Higher values for *segcnt* make the FOF-chunks longer, giving more of the source and reducing the gaps (a little).

On the other hand, *timestretch* values less than 1, as in our second example, make the stretched sound more continuous. However, there does seem to be a tendency for amplitude overload with values less than 0.9, probably because the grains start to overlap. This tendency increases when *segcnt* is increased.

There is therefore an important interplay between the amount of *timestretch* and the values for *segcnt*.

End of PSOW STRETCH

PSOW STRTRANS – Timestretch/transpose a sound by repositioning the pitch-synchronised grains, and overlapping them

Usage

psow strtrans *infile outfile pitch-brkpnt-data timestretch segcnt trans*

Example command line to create a timestretch with overlap effect:

```
psow strtrans om omstrt om.txt 6 5 7
```

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
timestretch – proportion by which the sound is to be stretched (> 1) or compressed (< 1). Range: 1 to 10.0.
segcnt – number of grains in a chunk retained without change, while gaps between segments are stretched.
trans – transposition in semitones, corresponds to overlap between successive segments. **NB:** this parameter interacts with *timestretch* in unpredictable ways.

Understanding the PSOW STRTRANS Process

This process will timestretch a sound, without changing vowels. Each FOF is individually repeated and hence, with many repeats, the process produces artefacts similar to waveset duplication. The pitch can also be (independently) shifted (without changing the vowels). These 2 parameters tend to interact to produce subharmonic artefacts.

Musical Applications

With PSOW STRTRANS the output can be more continuous, whereas PSOW STRETCH produces discrete pulsations, whether adjacent or with a gap. Also, it appears that higher values for *trans*, along with a larger *timestretch* factor tends to create a stepped effect, similar to Sample-Hold or FOCUS FREEZE and FOCUS HOLD.

End of PSOW STRTRANS

PSOW SUSTAIN – Freeze and sustain a sound on a specified pitch-synchronised grain

Usage

psow sustain *insndfile outsndfile pitch-brkpnt-data time dur segcnt vibfrq vibdepth transpose gain* [**-s**]

Example command line to create a freeze and sustain effect:

```
psow sustain om omsust om.txt 0.4 4 5 3 3 -7 5 -s
```

NB: There is a **special Appendix** with a detailed discussion about using PSOW SUSTAIN to create 'Song'.

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
time – time at which to freeze the grain(s).
duration – duration of output soundfile. This must be greater than the duration of the input soundfile.
segcnt – number of grains in a chunk.
vibfrq – frequency of any added vibrato.
vibdepth – depth in semitones of any added vibrato. Range: 0.0 to 3.0.
transpose – transposition of grain in semitones (Range: -48 to +24 semitones). This parameter may be time-varying, but time = 0 will be the start of an expanded grain – not (necessarily) the start of the sound.
gain – loudness contour of the entire output (Range: 0 to 10)
-s – option to smooth the grabbed fofs

Understanding the PSOW SUSTAIN Process

This process plays the sound up until a specified time. It then sustains the (group of) FOF(s) found at that time for *dur* length of time. Having done so, it plays the remainder of the sound. You may add vibrato to the sustained FOF.

Musical Applications

This is a way of lengthening a particular FOF sound and is particularly useful to stretch out the central portion of a sound, leaving onset and conclusion the same. The sound is reasonably clean and smooth if the *vib* parameters are set to 0, but becomes more pulsated as their values increase, sometimes with unpredictable artefacts because of the granular nature of the process. The smoothest results are achieved by using the **-s** flag.

End of PSOW SUSTAIN

PSOW SUSTAIN2 – Freeze and sustain a sound on an explicitly specified grain

Usage

psow sustain2 *insndfile outsndfile start end dur vibfrq vibdepth nudge*

Example command line to create a freeze and sustain effect, with a tight vibrato:

```
psow sustain om omsust2 om.txt 0.4 0.41 4 1 1 0
```

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile
start – time in seconds at which to cut the grain.
end – time in seconds of the end of the grain.
duration – duration of output soundfile. This must be greater than the input soundfile's duration.
vibfrq – frequency of any added vibrato.
vibdepth – depth in semitones of any added vibrato. Range: 0.0 to 3.0.
nudge – move selected grain position by *nudge* zero crossings.

Understanding the PSOW SUSTAIN2 Process

Note that the usual *pitch-brkpnt-data* file is not required.

This process behaves very similarly to PSOW SUSTAIN, but asks you to specify the precise start and end times at which the FOF is located. Larger durations between the *start* and *end* times produce a coarser result.

The *nudge* parameter is particularly important because it allows you to shift the position of the sustained FOF-grain just a little bit. You can produce phasing effects by creating several outputs, each with a slightly different *nudge* amount, and then mixing them together, all starting in the mix at the same time.

Musical Applications

We can produce cleanly sustained sounds, sounds that wobble slowly, sounds that are thicker and more unpredictable, and sounds with phasing effects (after mixing several different outputs).

End of PSOW SUSTAIN2

PSOW SYNTH – Impose vocal FOFs on a stream of synthesised sound (Experimental program)

Usage

psow synth mode *insndfile outsndfile* [*oscdatafile*] *pitch-brkpnt-data depth*

Example command lines to create a impose FOFs on a synthesised sound:

```
psow synth 1 om omsynthm1 oscdatm1.txt om.txt 1
psow synth 2 om omsynthm2 oscdatm2.txt om.txt 1
psow synth 3 om omsynthm3 oscdatm3.txt om.txt 1
psow synth 4 om omsynthm4 oscdatm4.txt om.txt 1
psow synth 5 om omsynthm5 om.txt 1
```

oscdatm1.txt

```
FRQ     Amplitude
200     0.4
400     0.5
800     0.6
1000    0.7
2000    0.8
4000    0.9
```

oscdatm2.txt

```
MPV     Amplitude
48      1.0
60      0.9
67      0.8
76      0.7
```

oscdatm3.txt

```
TIME FRQ AMP ...

0.2 200 0.4  400 0.5  800 0.6  1000 0.7  2000 0.8  4000 0.9
0.5 420 0.4  440 0.5  480 0.6   500 0.7  1000 0.8  2000 0.9
```

oscdatm4.txt

```
TIME MPV AMP ...
0.2 48.0 0.5  60.00 0.6  67 0.7  72.00 0.8  76.0 0.9
0.5 66.5 0.9  66.75 0.8  67 0.7  67.25 0.6  67.5 0.5
```

Modes

OSCDATAFILE: note that all amplitudes are in the 0 to 1 range

1 fixed frequency bands: each line of *oscdatafile* has a pair of values for a *frequency* and an *amplitude*. Amplitude range is from 0 to 1.
2 fixed MIDI bands: each line of *oscdatafile* has a pair of values for a *midipitch* and an *amplitude*. Amplitude range is from 0 to 1.
3 variable frequency bands: each line of *oscdatafile frequency* and *amplitude* values in the format used for **FILTER VARIBANK**. Time Frq Amp Frq Amp ... Amplitude range is from 0 to 1.
4 variable MIDI bands: each line of *oscdatafile* has *midipitch* and *amplitude* values in the format used for FILTER VARIBANK: Time MPV Amp MPV Amp ... Amplitude range is from 0 to 1.
5 noise: no *oscdatafile*; the synthetic source is noise.

Parameters

insndfile – input mono soundfile
outsndfile – output soundfile
[*oscdatafile*] – optional file for amplitude values (Amplitude range is from 0 to 1.) Modes **1-4** provide various data formats, while Mode **5** implements the option NOT to have this file, in which case the synthetic source is noise.
pitch-brkpnt-data – text file with a pitch trace in *time frequency* breakpoint form. It may contain zeros (indicating moments of no-signal) but NOT pitch-zeros (indicating moments of no-pitch). It must contain SOME significant frequency information.
depth – depth of application of FOFs to the synthesised sound.

Understanding the PSOW SYNTH Process

The program runs, but the results, at least with this short sound, seem not to show the frequency spread expected. So far, attempts to produce interesting results have not achieved anything.

Musical Applications

I think the best thing to do at this stage is to try it out with various types of source files: shorter, longer, with distinct pitch traces, with significant vowel changes etc. One expects to find results similar to, but distinguishable from time-varying filtering – see **FILTER VARIBANK**.

End of PSOW SYNTH

TWEET – Replace FOFs in vocal sound by synthetic tweets or noise

Usage

tweet tweet 1 *infile outfile exclude pitchdata minlevel pkcnt chirp* [**-w**]
tweet tweet 2 *infile outfile exclude pitchdata minlevel frq chirp* [**-w**]
tweet tweet 3 *infile outfile exclude pitchdata minlevel* [**-w**]

Modes

1 Replace FOFs by synthetic tweets, based on number of peaks
2 Replace FOFs by synthetic tweets, based on frequency of peaks
3 Replace FOFs by noise

Parameters

infile –input soundfile.
outfile – output soundfile.
exclude – '0' or text file containing pairs of times from which FOFs will **not** be extracted.
Enter '0' if not wanted.
pitchdata – breakpoint pitch data file of time and frequency, as created by **REPITCH GETPITCH Mode 2**.
The pitch data may contain zeros (indicating moments of no-signal), but it must contain ***some*** significant frequency information.
minlevel – a level in dBs below the level of loudest FOF found; FOFs below this level are rejected. (Zero = **no** FOFS are rejected.)
pkcnt – Mode 1: number of peaks in the impulse: frequency changes with FOF size. (Range: 1 to 200)
frq – Mode 2: frequency of peaks in the impulse: frequency is fixed. (Range:1 to 200)
chirp – glissing of impulse. NB Care: output may clip. (Range: 0 to 30)
-w – FOFs are windowed (cosine smoothing of edges).

Understanding the TWEET Process

TWEET replaces FOFs found in the infile with synthetic sounds. It may be compared with PSOW SYNTH, which impose vocal FOFs on a stream of synthesized sound.

Musical Applications

...

End of TWEET

Appendix 1 – ABOUT 'FOF'

`FOF' = *Fonction d'Onde Formantique*. It is a spectral synthesis method developed at IRCAM in 1984 by Xavier Rodet as part of the **Chant** project. The idea was to find an effective way to model the sound of the human voice. Trevor Wishart writes: "There is a difference between this 'FOF-Synthesis' and PSOW. FOF-Synthesis synthesises small wave-packets that are like the wave-packets generated by the human voice. PSOW tries to **find** these wave-packets in a real signal. This is tricky, so don't expect to get perfect results in all cases!"

In *Computer Sound Synthesis for the Electronic Musician* (Focal Press, 1998), Eduardo Reck Miranda adds that "this mechanism resembles the granular synthesis technique, with the difference that the envelope of the 'FOF grain' was especially designed to facilitate the production of formants" (p.144).

Richard Dobson describes the technique in this way:
"FOF (Formant Wave Function Synthesis) was developed as part of IRCAM's 'CHANT' project, which was devoted to the development of techniques for the realistic synthesis of the singing voice. It models the human voice as the sound from an impulse generator (the equivalent to the vocal chords) passing through a set of band-pass filters (representing the characteristics of the vocal tract), each filter corresponding to a vocal formant. Since the output from the impulse generator can be considered as a sequence of 'grains', the technique can be seen to be closely related to granular synthesis, the difference being that in FOF the grains are regular and synchronous, generating a coherent periodic waveform." (Richard Dobson. *A Dictionary of Electronic and Computer Music Technology*. Oxford University Press. 1992)

Note the key words: vocal, formant, band-pass filter, grains, regular and synchronous. This program set by Trevor Wishart is called PSOW: 'Pitch Synchronous (Operations on Waveforms?)'. It is not a spectral synthesis method, but manipulates grains of sound in the time domain. His description of the grains as 'pitch synchronous' is what connects it to the FOF technique: the implication is sounds that are vocal and have a regular, i.e., pitched waveform: in effect, vowels.

This means that the PSOW functions are designed to work with vowel sounds produced by the human voice. Trying to use them with consonants or other unpitched or complex sounds is not likely to produce usable results. Sounds with **no sibilants** and **no silences** are recommended for best results. Vowels can be extracted from speech (see SFEDIT CUT and PSOW CHOP, CUTATGRAIN and GRAB) to be used as source material, and the singing voice also produces ideal source material.

The purpose of the PSOW functions therefore becomes clear: to enhance the vowel material within speech (possibly restoring the consonants afterwards), or to produce pitched complexes with sung vowels. It is another form of CDP's approach to musique concrète, this time focused on vocal sound sources.

APPENDIX 2 – Batch files for the 3-step preparation of the pitch-brkpnt-data (text) file.

Appendix 2A - FRQTOTXT.BAT - PC Batch generic file

```
rem frqtotxt.bat - PC batch file to carry out the 3 steps to make the
rem                      make the breakpoint pitch file needed by PSOW to
rem                use with a FOF-source file (lengths must match)

rem Edit this file to replace 'yourfile' with the actual name of your file
rem at both the beginning and the end of the sequence.

echo on
copysfx yourfile.wav infile.wav
pvoc anal 1 infile.wav infile.ana
repitch getpitch 1 infile.ana infilepchdummy.wav infile.frq
ptobrk withzeros infile.frq infile.txt
ren infile.txt yourfile.txt

rem Delete (temporary) files no longer needed:
del infile.wav
del infile.ana
del infilepchdummy.wav
del infile.frq
echo off
```

Appendix 2B - FRQTOTXT.SH - MAC generic Bash shell script

```
echo frqtotxt.sh - MAC Bash shell script to prepare pitch-brkpnt-data file for
echo                 use with a FOF-source file (lengths must match)
echo Edit this file to replace 'yourfile' with the actual name of your file
echo at both the beginning and the end of the sequence.

echo
echo copysfx yourfile.aiff infile.aiff
copysfx yourfile.aiff infile.aiff
echo pvoc anal 1 infile.aiff infile.ana
pvoc anal 1 infile.aiff infile.ana
echo repitch getpitch 1 infile.ana infilepchdummy.aiff infile.frq
repitch getpitch 1 infile.ana infilepchdummy.aiff infile.frq
echo ptobrk withzeros infile.frq infile.txt
ptobrk withzeros infile.frq infile.txt
echo move infile.txt yourfile.txt
move infile.txt yourfile.txt

echo Delete (temporary) files no longer needed:
echo rm infile.aiff
rm infile.aiff
echo rm infile.ana
rm infile.ana
echo rm infilepchdummy.aiff
rm infilepchdummy.aiff
echo rm infile.frq
rm infile.frq
echo
```

Appendix 2C - FRQTOTXT.TCL - TCL script to automate the name changes (put *infile* and *outfile* on the command line) – invoke with tclsh

```
#frqtotxt.tcl - Starting with an input soundfile, this routine
#                   carries out the 3-step process ending with
#                   PTOBRK.EXE to convert the REPITCH GETPITCH binary
#                   pitch data file to a text file for use with PSOW.
#                   NB. Use REPITCH GETPITCH Mode 1, without -z flag
#                   i.e., without retaining pitch zeros.
# NB: apart from checking the command line arguments there is no
# error checking in this version of the program.

#OUTPUT MESSAGES TO THE USER
#Invoke with 'tclsh' or messages will not appear:
#tclsh frqtotxt.tcl yourinfile.wav theoutfile.txt

proc Usage {} {
        Inf "Usage: frqtotxt.tcl yourinfile.wav theoutfile.txt\n"
        Inf "NB: Infile must be mono\n"
        return
}

proc Inf {errmessage} {
        puts stdout $errmessage
}

#COMMAND LINE ARGUMENT CHECK

if {$argc < 2} {
        Usage
        exit;
}

if {![file exists [lindex $argv 0]]} {
        Inf "Input soundfile [lindex $argv 0] does not exist.\n"
        exit;
}

if {[file exists [lindex $argv 1]]} {
        Inf "Output soundfile [lindex $argv 1] already exists.\n"
        exit;
}

#BODY OF THE 3-STEP FRQ-TO-TXT PROCESS

exec copysfx [lindex $argv 0] infile.wav
exec pvoc anal 1 infile.wav infile.ana
exec repitch getpitch 1 infile.ana infilepchdummy.wav infile.frq
exec ptobrk withzeros infile.frq infile.txt 20
file rename infile.txt [lindex $argv 1]

# DELETE (temporary) files no longer needed

file delete infile.wav
file delete infile.ana
file delete infilepchdummy.wav
file delete infile.frq
```

APPENDIX 3 – Step-by-step procedure when running the PSOW/FOF program set from within *Sound Loom*.

1. Place your source sound file in CHOSEN FILES and analyse it with PVOC ANAL. SAVE. Click on Recycle Output. The analysis file now becomes the new input.

2. Run the process REPITCH GETPITCH *extract pitch from analysis data: to binary file*. Do NOT tick parameter 8 (*keep pitch zeros*). [TW: If we mark (keep) the pitch zeros (areas of no discernible pitch), this tells PSOW that they are unusable for finding FOFS. However, they could be, for example, whispered speech, so we still want PSOW to make use of these areas. Thus we do NOT 'keep pitch zeros'.] All the default parameter settings that come up are therefore OK and you just need to click on RUN to produce the binary pitch data file.

   The process produces 2 outputs, an analysis file ('cdptest0') and a pitch-data file ('cdptest1'). When you save **cdptest1** it will automatically get a **.frq** extension. When you SAVE, you will be asked whether to Save As a Generic Name. If you say YES, your analysis file and binary pitch data file will have the same name, but with **.ana** and **.frq** extensions respectively. If you say NO, you will be able to enter different names. (I usually use NO and put the word 'dummy' into the analysis file name so I know it's the one to delete.)

3. Re-set the CHOSEN FILES: Select ONLY your new **.frq** file, so that REPITCH GETPITCH can use it in the next step – therefore you also need to take away the **.ana** file used in Step 2 above so that only the **.frq** file is in CHOSEN FILES.

4. Run the Process REPITCH GETPITCH, and under 'CONVERT PITCHDATA FORMAT' click on the lower option: *ditto, BUT retain no-signal info* – to correlate with command line/Terminal use, this is when PTOBRK is run. The **no-signal** information is different from the **no-pitch** information, and is needed by the PSOW routines because where there is no signal there can be no FOFs to discover. NB: do NOT select the upper line, which reads *convert binary pitch data to text*, which is a different process and does not produce the data needed by PSOW.

   Now SAVE. The output file will have a **.txt** extension.

   When running the FOF functions, most of them will want to use the pitch data text file (**.txt**) you just made as the first parameter. However, it doesn't go onto the CHOSEN FILES list. Rather, you leave it on the Workspace, where it is automatically placed when you SAVE it, and from which it is fetched with the 'Get File' button on the Process Parameter Page for the function when you run it.

5. Now put ONLY the original **.aiff** soundfile back into CHOSEN FILES (which means that you take the **.frq** file away from CHOSEN FILES). Now the FOF functions can use it.

6. Set the other parameters of the given FOF function as needed, and RUN, etc. Again, the **.txt** file you just made will usually be the first parameter, and you 'get' it by using the 'Get File' button on the Parameter Page.

Remember that you have alternatives for the 3-step preparatory procedure to make the binary pitch text file. One of these is to run the batch file provided via the Terminal (see Appendix 2 of the Manual. The files themselves have been placed in the **html** directory, but you will probably want to copy them to your working directory).

Another way to facilitate the procedure would be to set up these 3 processes as a *Sound Loom* **Instrument**. Remember to keep the 2 parameters defining the pitch range as active parameters in the Instrument, by ticking the little checkboxes when you build the Instrument.

APPENDIX 4 – Using 'Sustain a specific FOF within a sound' to produce "Song"

by Trevor Wishart

This process uses advanced facilities on the *Sound Loom* GUI. It is a good example of the detailed sound manipulation techniques that Trevor uses to create his remarkable compositions. [Ed.]

This process, on the **FOFS** menu of *Sound Loom*, can be used to extend pitched vowels in a source, and make them sing. Success depends on the choice of material, and the parameters applied, and output may have to be modified by small edits and/or (low pass, or band pass) filtering to produce a satisfactory result.

- The pitch of the original sound must first be extracted and saved in textfile format. To do this, place the analysis file version of the file you want to use in CHOSEN FILES and then use **extract pitch from analysis file: to textfile**. (*A successful pitch-track is essential*). The resulting textfile containing the pitch trace data forms the first parameter for the Song from sustained FOFs function. You place the soundfile version of the sound you want to use and go to **FOFS: sustain a specific FOF within a sound**. The text pitch file you just made is used for the first parameter: PITCH BREAKPOINT FILE. You load it in using **Get File**. (The file will be on the **Workspace** and should be listed automatically.) Click on it and then on **Use**. It will now be shown as the (breakpoint) file to be used.

- The second parameter, FREEZE, is the time at which the sound is to be frozen and extended. You can use the **Sound View** window to see the sound, and mark an appropriate time. Choose a moment within a clearly pitched vowel sound. If the output you get is at an unexpected pitch, or has an unsatisfactory quality, try adjusting the freeze time by very small amounts.

- The OUPUT DURATION OF WHOLE SOUND parameter will determine by how much the FOF is extended. For example, if the original sound is 2 seconds long, and the specified output duration is 8 seconds, the FOF will be extended by 6 seconds duration. **NB:** If you do **not** set the output duration, it defaults to the duration of the source, and the process will refuse to run. Similarly, if you set the output duration to a value *less than the source duration*, the process will refuse to run.

- Normally you will want to grab *just one* grain. Grabbing more than 1 grain usually tranposes the pitch downwards (2 grains down an octave, 3 grains down 1 octave and a 5th, and so on down the subharmonic series). Large number of grains produce audible loop-repetition effects.

- If you now run the process (with *Vibrato Depth* and *Transposition* both set to zero), you should produce an output with a static frozen FOF. From this you can ascertain if the pitch is what you expected.

- If the output has high frequency emphasis (sounds artificially 'nasal'), try turning on SMOOTH FOFS and running again. (Using this sometimes produces a discontinuity or 'bump' at the start of the frozen portion. If this happens you can edit it out later).

To produce a more realistically 'sung' result:

- Set the *Vibrato Frequency* to somewhere between **6** and **15** cycles.
- Set the *Vibrato Depth* between **.25** and **.65**

Test the output by running the process again.

To confine this vibrato to the extended FOF *only* (and to possibly add some randomised *Frequency* and *Depth* variation), use the **VibLocal** button.

- The *Depth* and *Frequency* must be preset, in the parameter boxes, to *non-zero* values, before **VibLocal** is used.
- The degree of randomisation of either parameter (if required) can be set in the small window that pops up. Change the values using the **Arrow** keys.
- Breakpoint files will be generated and placed in the *Vibrato Frequency* and *Depth* parameter boxes.

You can now rerun the process and hear the output sound.

- You can cause the pitch of the FOF to change by supplying a semitone transposition breakpoint file for the GRAIN PITCH TRANSPOSITION parameter.

If you have ***Tabula Vigilans*** there will be a small piano-keyboard type key to the right of this parameter. You can use this to create such a breakpoint file.

- First acknowledge that MIDI capture will proceed (click on the window which appears).
- Next hit the *reference pitch* from which the transpositions will take place. Normally this will be the pitch of the extended FOF itself.
- Now play a pitch line. It is best to use staccato articulation, as the events can be rejoined as required, afterwards.
- Hit the bottom note on your MIDI keyboard, to stop the MIDI capture.
- You will be asked if you wish to pre-sustain the entered pitches. The captured data, if staccato, will consist of a series of silence-separated events with quite steep on/off envelopes. If you intend to smooth the onsets of these events, you need to pre-sustain them by the likely smoothing (see below).
- You will next be asked if you wish to extend the last entered event to meet the end-portion of the sound. If you selected to freeze the sound at a moment not at the very end of the source, there will still be a segment of unmodified source retained **after** the frozen extension.

If you opt to do the extend, your last MIDI event (provided it occurs **before** the end of the frozen segment) will be sustained to join onto this end-segment of the source.

- You will also be asked if you wish to apply the loudness envelope from the MIDI entry. Preserving the loudness not only retains the loudness envelope, but is also used to separate the extended FOF into distinct events separated by silence (only if you separated your MIDI input events by silence). If you opt to do this, a breakpoint file will be automatically generated and placed in the LOUDNESS CONTOUR parameter box.

If you now run the process again, you will hear a sound where the sustained FOF is transposed by your input data. Typically, the steep envelopes will make it sound rather artificial.

- Finally, if you generated silence-separated events, you can articulate these, from the **Artic** button. (The *Artic* button will **not** function until you have generated such silence-separated events). The **Artic** interface offers several options.

- You may '**Stitch**' the end of the sustained FOF to the remnant of the end of the source. Often there will be a loudness discontinuity here, manifesting itself as a bump or click in the sound. The **Stitch** facility allows you to specify an envelope to apply to the output, at this point, to remove this discontinuity.
- In the panel below, each silence separated event in your FOF-extension is assigend an event number and a set of options to apply to each event.
- Before you can articualte any event you must turn the articulation **On** by ticking the relevant **Articulation On** box, or the **All Events: Artic On** box.
- Once the articulation for an event is **On** you can enter values for *Decay to minimum of* and *Minimum at time fraction* in the boxes below the event number. These set the minimum level to which the event will decay, and the time-proportion between events at which the minimum is reached. (Also see the alternative, *Rise Time* and *Decay Time* below).
- Events may also be *Sustain*-ed, i.e., they are extended to meet the next event, but not slurred into them.
- Events may also be *Slur*-red, i.e. they are extended to merge into the next event, with no articulation between them.
- The *Gain* of each event can be adjusted.
- You can enter explicit values for the *Rise Time* and *Decay Time* of that event.
- You can also set global rise and decay times in the boxes above (GLOBALS) using the **Arrow** keys. You can apply these to all the **On** events by clicking on *Set Rise* or *Set Decay*. You can also remember, and recall these global values (from one session to another) using the **Save** and **Recall** buttons.

Note that if you set a rise-time of (say) **0.3** seconds, but, when you created the source from MIDI input, you set the pre-sustain to *less than 0.3* seconds, the previous pitch will extend into the rise-attack and the previous pitch will be heard to glissando towards the new pitch.

When you run the Articulation, it writes a new LOUDNESS CONTOUR envelope to the parameter box.

You must *Run the Process again* to hear the result.

Last Updated 27 Apr 2023
Documentation: Archer Endrich, revised R. Fraser