CDP MODIFY Functions

(with Command Line Usage)

Functions to MODIFY soundfiles

MODIFY **BRASSAGE**
Granular reconstitution of soundfile
MODIFY **CONVOLVE**
Convolve the first soundfile with the second
DSHIFT
Adds Doppler shift to panning
MODIFY **FINDPAN**
Find stereo-pan position of a sound in a stereo file
MODIFY **LOUDNESS**
Alter loudness or balance of sound
NEWDELAY
Delay with pitch-defined output sound
PHASE
Invert phase or enhance stereo separation of a sound
MODIFY **RADICAL**
Radically modify:

1. Reverse
2. Shred
3. Scrub
4. Resolution
5. Ring-Modulate
6. Cross-Modulate
7. Quantise (c.f. Resolution)

MODIFY **REVECHO**
Add reverberation or echo to the sound
MODIFY **SAUSAGE**
Brassage on several sources
MODIFY **SCALEDPAN**
Distribute sound in stereo space, scaling pan data to soundfile duration
MODIFY **SHUDDER**
Shudder a stereo soundfile
MODIFY **SPACE**
Spatialise, or alter the spatialisation of, a soundfile
MODIFY **SPACEFORM**
Create a sinusoidal spatial distribution data file
MODIFY **SPEED**
Change speed (& pitch) of sound
MODIFY **STACK**
Create a mix that stacks transposed versions of source on top of one another
VERGES
Play source, with specified brief moments glissing up or down

SEE ALSO:

[FASTCONV]
Multi-channel fast convolution

[WRAPPAGE]
Granular reconstitution of one or more soundfiles over multi-channel space

[GRAINMILL]
Stand-alone graphic program for BRASSAGE and SAUSAGE

MODIFY BRASSAGE – Granular reconstitution of soundfile

Usage

modify brassage 1 *infile outfile pitchshift*
modify brassage 2 *infile outfile velocity*
modify brassage 3 *infile outfile density pitch amp* [**-r***range*]
modify brassage 4 *infile outfile grainsize* [**-r***range*]
modify brassage 5 *infile outfile density*

modify brassage 6 *infile outfile velocity density grainsize pitchshift amp space bsplice esplice*
[**-r***range*] [**-j***jitter*] [**-l***outlength*] [**-c***channel*] [**-x**] [**-n**]

modify brassage 7 *infile outfile velocity density hvelocity hdensity*
grainsize pitchshift amp space bsplice esplice
hgrainsize hpitchshift hamp hspace hbsplice hesplice
[**-r***range*] [**-j***jitter*] [**-l***outlength*] [**-c***channel*] [**-x**] [**-n**]

Modes

1 PITCHSHIFT: shift the pitch of *infile* while retaining (more or less) the same duration
2 TIMESTRETCH: stretch or compress the *infile* in time, while retaining the same pitch
3 REVERB: use 3 parameters to create a kind of reverberant effect

pitch – transposition factor (Range: -0.33 to 0.33)

4 SCRAMBLE: random reordering of grains within a timeframe

NB: To get a scramble effect, you **do** need to provide a timeframe (in ms) with **-r***range* to overcome the default setting of 0.

5 GRANULATE: 'granulate' (put a grainy surface on) a source

A *density* of 1.0 will achieve this; < 1 will introduce gaps, and values > 1.1 (out of a range which ends at 2.0) begin to sound smooth again.

6 BRASSAGE: powerful segmentation/fragmentation procedures using constants or time-varying breakpoint files
7 FULL MONTY: finely tuned granular textures using parameter ranges with upper and lower limits, which can be set as constants or as time-varying breakpoint files, or mixtures of the two. You are recommended to use the graphic program *GrainMill* for this, if it is available.

Parameters

In *Sound Loom*, the upper limit parameters for Mode 7, which here begin with an 'h', are called 'limit' instead. A number of the parameters are also given different names. These are noted below, prefixed with 'SL:'.

infile – soundfile to be processed, normally, but not necessarily, mono (see *space*)
outfile – output soundfile written after processing
velocity (SL: *timeshrink*) – speed of advance in *infile*, relative to *outfile* (Range: >= 0)

This is the inverse of a *timestretch* (i.e., 1/n: higher values make the output shorter, lower values – less than 1 – make it longer). This permits an infinite timestretch. Remember this when using Mode **2**.

density – amount of grain overlap (Range: > 0)

Values < 1 leaves intergrain silence, i.e., gaps. Extremely small values will cease to perform predictably.

grainsize – size of the grains in milliseconds (Range: must be > 2 * the length of the splices; overall range: 2.0ms to 1997.12ms; Default: 50ms)
pitchshift – transposition of grains in + or - (fractions of) semitones
amp (SL: *grain loudness range*) – gain applied to the grains (Range: 0 to 1; Default: 1.0)

Use only if you want amplitude to vary over a range &/or in time.

bsplice (SL: *start splice*) – length of start-splices on grains in ms (Default: 5)
esplice (SL: *end splice*) – length of end-splices on grains in ms (Default: 5)
space (SL: *spatial position*) – set stereo position in *outfile* 0 = L, 1 = R (Range: 0 to *N*) **-r***range* (SL: *search range*) – of search for next grain, before *infile* 'now' (Default: 0 ms)
-j*jitter* (SL: *scatter*) – randomisation of grain position; Range 0 to 1 (Default: 0.5)
-l*outlength* – maximum *outfile* length (if end of data is not reached)

Set to zero (the Default) for this parameter to be ignored. But if *velocity* is 0 **anywhere**, *outlength* must be given.

-c*channel* (SL: *channel to extract*) – extract and work on just one channel of a stereo input (Range: 1 or 2) Set *channel* to **0** (the Default) for this parameter to be ignored.
-x – do exponential splices (Default: linear)
-n – no interpolation for pitch values (quick but dirty)

The presence of the upper limit parameters –
hvelocity (SL: *timeshrink limit*),
hdensity (SL: *density limit*),
hgrainsize (SL: *grainsize limit*),
hpitchshift (SL: *pitchshift limit*),
hamp (SL: *loudness range limit*),
hbsplice (SL: *startsplice limit*),
hesplice (SL: *endsplice limit*), and
hspace (SL: *spatial position limit*)
– make it possible to specify a range of values. When set, the program chooses random values for that parameter **for each grain** between the lower and upper limits. For example, if *pitchshift* is -3 and *hpitchshift* is 4, the pitch of each grain will be somewhere within the interval of a perfect 5^{th}.
NB: Furthermore, any of these parameters may vary in time (provide the name of a *time value* breakpoint file).

All parameters – except *outlength* and *channel* may vary over time.

Understanding the MODIFY BRASSAGE Process

MODIFY BRASSAGE is a re-working of the original CDP program called GRANULA. Those with a PC system can (and are advised to) use the graphic version, ***GrainMill***, with built-in graphic breakpoint editing. This command-line version offers greater flexibility with the shapes of the grains, handled with the *bsplice* and *esplice* parameters. The fact that *velocity* is an inverse function makes stretching time less intuitive, but allows scope for (even more) extreme values than the graphic version.

MODIFY BRASSAGE is mainly designed as a way of texturing pre-existing sample data. It is therefore not the same as a 'granular synthesis' program which synthesises sound using granula techniques. A search for 'granular synthesis' on the web will provide a great deal of informatiion on current work in this area.

This command line is exceptionally complex, especially because the low and high values for a number of (optional) parameters are not adjacent. (This was done in order to bring the command line into line with the requirements of a graphic interface.) It is recommended that you create a template batchfile which lines up the parameters as shown above, that is, with a newline which enables you to put the high values directly under the low values. But remember to delete the newline before saving (or it won't read the rest of the command line) and save to a new name so that you don't overwrite your template.

The flexibility with which the parameters can be handled is enormous. You can have:

- constant values
- random values selected (by the program) from a range specified by low and high limits
- time-varying values specified by breakpoint files
- a mixture of constant values and breakpoint files, such as a constant low limit and a time-varying high limit, or *v.vs*, or even breakpoint files as both lower and upper limits. (**NB:** the 'lines' created by joining up the points in the files for the upper and lower limits must not cross: at any given time location in the two files, a lower-file value must not be higher than a higher-file value, and *v.vs.*).

The different modes of the program now make it easier to realise a specific musical objective, in the manner of presets. Use Modes **6** and **7** for 'brassage' (serious fragmentation of soundfiles), for more control and for creating the more complex, granular types of texture.

The three key parameters are:

- ***grainsize*** – how coarse or fine the granulation will be; 'granular synthesis' typically uses gransizes of less than 50ms
- ***velocity*** – how the *infile* is read; it actually relates to *grainsize*:
  - a value of **1** means that the read-steps are the same as the *grainsize*, and the length of the *outfile* will therefore be the same as that of the *infile*
  - a value **less than 1** means that the read-steps are shorter than *grainsize*, material will overlap and the length of the *outfile* will be longer than that of the *infile*: **time-stretch**
  - a value **greater than 1** means that the read- steps are longer than *grainsize*, material will be skipped, and the length of the *outfile* will be shorter than that of the *infile*: **time-compression**

- ***density*** **–** how the *outfile* is constructed:
  - a value of **1** means that the *outfile* lays out *grainsize* grains end to end; the *infile* and the *outfile* will be the same length, but the *outfile* will have a 'granulated' surface texture due to the action of the splices on each grain.
  - a value **less than 1** means that the time between grains in the *outfile* will be longer than the grains themselves, introducing audible gaps ('pointillist' effects). E.g., a *grainsize* of 0.05 (50ms) divided by 0.5 (a value for *density* less than 1) = 0.1. Thus grains only 5/100^{ths} of a second long will occur every tenth of a second in the *outfile*, leaving significant gaps of 95/100^{ths} of a second.
  - a value **greater than 1** means that the time between grains in the *outfile* will be shorter than the grains themselves, causing the grains to overlap in time: the next grain will start before the previous has finished. Thus two (or more) grains will be mixed in the *outfile*, making the texture more dense. If there is no pitch transposition (of the grains), this will be heard as a more intense, thicker sound, with internal 'edges' determined by the nature of the grains and their splice slopes. If there is pitch transposition (of the grains), the grains will be scattered in a vertical space defined by the pitch range. E.g., a *grainsize* of 0.05 (50ms) divided by 2.0 (a value for *density* greater than 1) = 0.025. Thus grains which are 5/100^{ths} of a second long will occur every 2.5 100^{ths} of a second, overlapping therefore at each grain's half-way point. This is how the 'massive' textures characteristic of granular synthesis are produced.

The other parameters serve to create qualitative differences in the granular textures.

The most noticeable of these is ***pitchshift***, which adds or subtracts semitones or fractions of semitones to/from *each grain*.

- Using constants, you can 'tune' the texture, perhaps creating a number of differently tuned texture streams which will then be mixed in some way to create textures with a harmonic flavour.
- Using ranges, you can create thick, churning textures by placing a high density output within a narrow pitch band, or conversely, widely scatter grains (thickly or thinly depending on *density*) over a wide range.
- Using a constant and a breakpoint file makes it possible to create, for example, triangular shaped ranges (i.e., pitch fields), the base being the constant, and the triangular shape affected by the breakpoint file.
- Using double-breakpoint ranges make it possible to design pitch fields which move through time in a variety of patterns. But bear in mind that the pitch shape of the original material forms the starting point for these transpositions. Also remember that the two breakpoint shapes (lower and upper) are not supposed to cross (lower rising above the line of the upper, or upper dipping down below the line of the lower).

The ***amplitude*** parameter is also very important, for variation in amplitude add a natural suppleness to the texture, and can also be used to create aurally directional movement: crescendi and decrescendi. In case of overload, which the program does its best to avoid, this parameter can be used to scale down the output (use as a constant). The maximum value for *amplitude* is 1.0, so it is not possible to scale the *infile* amplitude upwards. There is usually some loss of signal level due to all the splices, so it is generally wise to ensure that the *infile* is at or close to a maximum level.

The ***space*** parameter is another way to enrich the output by using left and right limits to scatter the grains in horizontal space or have the granular stream move in a directional manner. Using constants enable you to locate the output at a specific point in the horizontal plane.

Range and ***jitter*** are another way to loosen things up. The former provides a search range in the *infile* from within which grains are selected, and the latter adds a further degree of randomisation to the time placement of grains in the *outfile*. With short *grainsize* and high *density*, *jitter* will have a more a more subtle effect, noticeable in many cases as a modulation or softening of the comb filter effect arising from overlapping grains. When *density* is 1 (in which grains are normally end-to-end), it will have the effect of overlapping some grains, while leaving (very small) spaces between others.

Musical Applications

One of the main pioneers in this area is Barry Truax. I recommend you try to get to listen to his famous compositions *Riverrun*, *The Wings of Nike*, and *Pacific*. Also see his articles:

- "Composing with real-time granular sound" (*Perspectives of New Music* 28(2) 120-134
- "Discovering Inner Complexity: Time Shifting and Transposition with Real-time Granulation Technique" (*CMJ* 18(2) 38-48

Also see *The Computer Music Tutorial* by Curtis Roads (et al.), MIT Press, 1996), pp. 168-184 and pp. 440-444.

It is also useful to study the role of texture in 20th century composition. The relationship of melody and textural accompaniment in the work of Bohuslav Martinu indicate how the 'background' often becomes aural 'foreground'. Pieces such as Stockhausen's *Momente* and *Gruppen* are classic examples of a full-blown collage/texture technique, as are the amazing collages in Berio's *Laborintus II*.

But 'brassage' and 'granular' textures really came into their own once the computer became a musical tool, when it became possible to create sound textures out of tiny fragments of sound, e.g., 0.035 sec long. MODIFY BRASSAGE provides an enormously powerful tool with which to explore the effects and side-effects of these processes.

If you have the CDP ***GrainMill*** program, you will be able to realise and study the HTML tutorial for it written by Philippos Theocharidis when he was at the University of Newcastle. The sounds can be made very easily by loading the settings files he provides.

For the moment, the best way into the program is first of all to run Modes **1-5** to explore the presets. Don't be afraid to try extreme values, for they will more easily reveal what this wonderful program can do. After that point, I recommend that you revise the section above about the 'three key parameters' and try out the different settings described for *grainsize*, *velocity* and *density*. (But do use template batch files for Modes **6** and **7**.)

As a basic point of reference, consider the following combinations:

velocity	***density***	**result in the *outfile***
< 1	< 1	time-stretched, with sonic material repeated, but with gaps in the *outfile*
< 1	> 1	time-stretched, with sonic material repeated and overlapping grains in the *outfile*
> 1	< 1	time-compression, with sonic material skipped, and gaps in the *outfile*
> 1	> 1	time-compression, with sonic material skipped, and overlapping grains in the *outfile*

The addition of a pitch range to the above may make the effects more observable. With time-varying parameters, you can move fluidly between these various basic configurations.

ALSO SEE: **MODIFY SAUSAGE**, which is able to apply the same functionality to several source files.

End of MODIFY BRASSAGE

MODIFY CONVOLVE – Convolve the first sound with the second

Usage

modify convolve 1 *insndfile1 insndfile2 outsndfile*
modify convolve 2 *insndfile1 insndfile2 outsndfile transposfile*

Parameters

insndfile1 – first input soundfile to be convolve with
insndfile2 – the second input soundfile, which must not be longer the *insndfile1*; both soundfiles must have the same channel count
outsndfile – output soundfile
transposfile – textfile of *time semitone-transposition* transposition pairs

Understanding the MODIFY CONVOLVE Process

Convolution is based on complex mathematical operations between two source soundfiles. There is a great deal of information about convolution available in Wikipedia, but your editor does not know just which ones Trevor Wishart uses in his program.

Musical Applications

The result of convolution is unpredictable but fascinating. The operation of this particular sub-module can take rather a long time, so you are recommended also to make use of **FASTCONV**.

End of MODIFY CONVOLVE

DSHIFT – Add Doppler shift to panning

Usage

dshift [**-d***N*] *inpanfile outtransposefile* [distance between speakers]

Example command line to create a Doppler effect:

```
dshift -d15 pan.brk dshift.brk 12.5
```

Parameters

-d*N* – a flag to set the time it takes for a sound to change direction. *N* is the time in milliseconds. Default: 10ms. This flag is useful in order to avoid clicks.
inpanfile – input PAN breakpoint file to use when adding Doppler effect
outtransposefile – output transposition file (in semitones) to use with MODIFY SPEED Mode 2 (on the soundfile panned with the PAN breakpoint file) to create the Doppler effect
distance_between_speakers – optional parameter to specify the space in meters between the speaker pair that will play back the sound

Understanding the DSHIFT Function

The Doppler effect is the phenomenon of pitch change that we hear when something emitting a loud noise, such as a horn or siren, goes quickly past us. This program was written by Rajmil Fischman. It has been implemented in *Soundshaper*, but not yet in *Sound Loom*, either on the PC or on the MAC. It can be run from the Command Line on the PC, but has not yet been ported to the MAC.

There are 3 main steps to creating this effect with DSHIFT:

1. Start with a pan breakpoint file and use it with **PAN** (MODIFY SPACE 1) to create a sound that moves between the speakers in a stereo field.
2. Then use DSHIFT to use the pan breakpoint file again as a guide to creating the Doppler pitch change effect. The output of DSHIFT is a pitch transposition file in semitones.
3. Run **MODIFY SPEED** with the panned soundfile and the pitch transposition file as inputs, to create the final, Doppler-shifted, soundfile.

To operate DSHIFT in *Soundshaper*:

1. Pan a mono sound as normal, using a breakpoint pan file. The resulting panned soundfile is the output of this process.
2. Select SPARE FILE and load the pan breakpoint file.
3. Run DSHIFT from the Data Menu: "Doppler-Shift Transposition".
4. Name the output breakpoint transposition file (in semitones); the default name is 'dshift.brk'.
5. Deselect SPARE FILE (the pan file) and run MODIFY SPEED Mode 2 (semitones), with the panned soundfile you made in Step 1 as the soundfile input and 'dshift.brk' (or your own name) as the time-varying transposition file. The output of SPEED is a panned soundfile that also changes in pitch.

Musical Applications

This is a 'real life' acoustic effect that can be used to enhance literal recreations of a natural environment. The case that most easily comes to mind is that of a vehicle passing by, especially if it is blasting a horn or siren as it moves. More abstract sounds can be used, with a 'real life' effect adding verisimilitude to a sound coming from a strange or unfamiliar source.

End of DSHIFT

MODIFY FINDPAN – Find stereo-pan position of a sound in a stereo file

Usage

modify findpan *infile time*

Parameters

infile – input stereo soundfile, which has been panned over time in the stereo field
time – the time-point in the sound at which you wish to query the pan position

Understanding the MODIFY FINDPAN Function

If you are carefully positioning sounds in stereo-space it might be useful to know exactly where on the stereo-stage a panning sound is at any particular time. This process assumes the input is an (originally mono) sound which you then panned so that it moves around the stereo space.

The process uses the file loudness data on the two channels to calculate where in the stereo space the sound appears to be. It assumes that the *infile* contains a sound that has previously been panned to a position in the stereo field. If this is not the case, the results will be misleading. For example, it could be a stereo file which is not in fact moving across the spatial field.

It is important that the input soundfile was originally mono.

- If it had always been a stereo file, you could not find out where anything is in the stereo stage by comparing the left and right channels – they could, for example, carry completely different sounds.
- If the file were originally mono and then panned, the left and right channels will have the same signal, but at different levels.
- Thus, by comparing those levels, you can tell where the sound is on the stereo stage.

Musical Applications

Suppose you have made a sound that moves about in space, such as across the horizontal field from left to right. Now you want to place another sound such that it is in exactly the same spatial location as the moving sound at a specific time point. The question is: where is that spatial location?

MODIFY FINDPAN asks you to supply a stereo input soundfile and a *time*. It then returns the pan location of the input soundfile at this time-point. You then use this information to specify the placement of the sound you want to add to the mix.

End of MODIFY FINDPAN

MODIFY LOUDNESS – Alter balance or loudness of sound

Usage

modify loudness 1 *infile outfile gain*
modify loudness 2 *infile outfile gain*
modify loudness 3 *infile outfile* **-l***level*
modify loudness 4 *infile outfile* **-l***level*
modify loudness 5 *infile infile2 outfile*
modify loudness 6 *infile outfile*
modify loudness 7 *infile infile2* [*infile3 ...*]
modify loudness 8 *infile infile2* [*infile3 ...*] *outfile*

Modes

1 GAIN: adjust level by factor *gain*
2 dBGAIN: adjust level by dB *gain*
3 NORMALISE: force level (if necessary) to the maximum possible, or to the *level* given (Range: 0 to 1)
4 FORCE LEVEL: force level to maximum possible, or to the *level* given (Range: 0 to 1)
5 BALANCE: force the maxmimum level of *infile* to the maximum level of *infile2*
6 INVERT PHASE: invert phase of the sound
7 FIND LOUDEST: find loudest of 2 or more files
8 EQUALISE: force all of 2 or more files to level of loudest file. The input files are rescaled in input order, with output names as *outfile, outfile1, outfile2, outfile3* etc.

Parameters

infile – soundfile to have its level altered
infile2 – soundfile to which *infile* is to be balanced
outfile – resulting soundfile, with readjusted level. When there are 2 or more input files, *outfile* is a generic name to which are added '1', '2' etc. to identify the matching output files.
gain – in Mode **1**, floating point multiplier: **< 1.0** reduces level, **> 1** increases level;
OR, in Mode **2**, level expressed in dB (Range: -96dB to 96dB). **NB:** every drop of -6dB halves the previous level.

> **NB:** a *gain* value of **-1** inverts the phase of the sound. This can sometimes be used to maintain high signal levels without clipping. See SUBMIX GETLEVEL.

-l*level* – amplitude level (Range: 0 to 1)

Understanding the MODIFY LOUDNESS Process

Information about the current level of a soundfile is given by SNDINFO MAXSAMP. Information about the units used to express loudness is summarised in the chart **Getting a Handle on dB values**. This chart shows both *gain* multipliers and dB values. (Click on 'Back' on your Browser to return from these locations.)

It is best to adjust level earlier rather than later in a set of processes. Also, it is important to start with a source sound which has a reasonable signal level. Artefacts will appear if a low signal is boosted too much.

Level overflows reported by *Csound* are usually due to a fault in the orchestra file.

Musical Applications

This will be one of the 'frequently used' functions, e.g., to:

- readjust a level which has either dropped a little too much or overflowed due to processing
- to boost a source signal which is a little too low to enter into a processing sequence
- to reduce the level of a soundfile after PVOC (the Phase Vocoder) has reported overflow. You can use the *gain* factor recommended by PVOC, or double-check levels with MAXSAMP before applying *gain*.
- When filters of time-varying Q are used, the output level tends to drop as the Q increases. This can be compensated for by using MODIFY LOUDNESS in Mode **5** (Balance sources), submitting the original file and the filtered file as the two sources. The resultant soundfile will have the sound of the filtered file with the amplitude contour of the original sound.
- And it can also be used to impose an envelope on a sound (just like ENVEL IMPOSE), if you use a time-varying *gain* value in Mode 1.

End of MODIFY LOUDNESS

NEWDELAY – Delay with pitch-defined output sound

Usage

newdelay newdelay *insndfile outsndfile midipitch mix feedback*

Example command line to create a pitch-specified delay :

```
newdelay newdelay in.wav out.wav 67 0.75 0.1
```

Parameters

insndfile – input soundfile
outsndfile – output soundfile
midipitch – pitch of the output, expressed as a MIDI Pitch Value. MIDI Range: -76 to +136. (See Notechart)
mix – the amount of delayed signal in the final mix. **0** gives a 'dry' result, **1** a 'wet' result
feedback – produces resonance related to delay time (with short times)

midipitch may vary over time.

Understanding the NEWDELAY Process

CDP already has a time-variable delay in **MODIFY REVECHO**, Mode **2**. This program specifies the delay time as a MIDI (or pseudo-MIDI) pitch value. In a delay line, with sufficient feedback, very short delays of typically less than 40ms (25Hz) create a pitched resonance. *Midipitch* allows you to set this delay time according to the required pitch. However, discrete echoes can also be expressed as a pseudo-MIDI value (e.g., -12 produces delays of about 1/4 sec.), thereby making an important connection between pitch and rhythm.

End of NEWDELAY NEWDELAY

PHASE – Invert phase or enhance stereo separation of a sound

Usage

phase phase 1 *insndfile outsndfile*
phase phase 2 *instereofile outstereofile* [**-t***transfer*]

Example command line to create phased signal:

```
phase phase 1 four.wav fourinv.wav
```

Modes

1 Invert phase of a sound.
2 Enhance stereo separation by phase shifting. Input file must be stereo.

Parameters

infile – input soundfile; must be stereo for Mode **2**
outfile – output soundfile, stereo in Mode **2**.
-t*transfer* – amount of signal to be used in phase-cancellation. Range: 0 to 1. 1 = all.

Understanding the PHASE PHASE Process

In Mode **1** playback of the output inverted soundfile shows that there is no audible difference from the input soundfile. However, if you mix the original sound with the phase-inverted sound, you will generate a silent output: positive and negative amplitude values cancel out. However, if you mix the input and output with an offset, you get signal and hear an overlap of the two soundfiles (the cancelling out does not fully happen). You can try this with **SUBMIX MERGE**.

Also see: **MODIFY LOUDNESS** Mode 6, which likewise inverts the phase.

Mode **2** is supposed to be a means of creating an enhanced stereo image by suppressing any aspects of the Left image which have bled to the Right, by adding (part of) the phase inverted Left signal on the Right, and *vice versa*. T Wishart writes: I learned of this idea from a mixing engineer I met in France, but I'm still not convinced that this achieves anything except in very special circumstances (with particular types of material). The output may sound the same as the input, except perhaps on the largest PA systems.

Musical Applications

Inverting the phase of signals might be appropriate in some mixing or texturing procedures. See the technical literature.

End of PHASE PHASE

MODIFY RADICAL – radical modifications: Reverse, Shred, Scrub, Lower Resolution, Ring Modulate, Cross Modulate, Quantise

Usage

modify radical 1 *infile outfile*
modify radical 2 *infile outfile repeats chunklen* [**-s***scatter*] [**-n**]
modify radical 3 *infile outfile dur* [**-l***down*] [**-h***up*] [**-s***start*] [**-e***end*]
modify radical 4 *infile outfile bit_resolution srate_division*
modify radical 5 *infile outfile modulating-frq*
modify radical 6 *infile1 infile2 outfile*
modify radical 7 *infile1 outfile bit_resolution*

Modes

1 REVERSE: Sound plays backwards
2 SHRED: Sound is shredded, within its existing duration
3 SCRUB BACK & FORTH: As if hand-winding tape spools over a tape head
4 LOSE RESOLUTION: Sound is converted to a lower sample rate or bit-resolution
5 RING MODULATE: Against input modulating frequency, creating sidebands
6 CROSS MODULATE: Two input soundfiles are multiplied, creating complex sidebands
7 QUANTISE: Sound is converted to specific bit-resolution (mid-rise).

Parameters

infile – input soundfile
infile2 – 2nd input soundfile, for cross modulation
outfile – output soundfile written after processing
repeats – number of repeats of shredding process
chunklen – average length of chunks to cut and permutate
-s*scatter* – randomisation of cuts (Range: 0 to K, where K = duration of *infile*/*chunklen* Default = 1)

- If *scatter* = 0, reorders without shredding
- **NB1:** *chunklen* * *scatter* must be less than the length of the program's sound buffer (Default is 1 Mb or the setting for CDP_MEMORY_BBSIZE)
- *Scatter* results in chunks of variable length.
- **NB2:** If the input sound is greater than the internal buffer length, each buffer of sound is shredded independently

-n – use this flag for a smoother output
dur – minimum length of *outfile* required
-l*down* – lowest downward tranposition in semitones
-h*up* – highest upward transposition in semitones
-s*start* – scrubs start before time *start* seconds
-e*end* – scrubs end after time *end*
bit_resolution – Range: 1 - 16 Default 16-bit.

srate_division – divide the sample-rate. Range: 1 to 256 Default 1 (normal)

- The value entered will be rounded to a power of 2
- Works **only on Mono files**

modulating-frq – number of cycles per second

Understanding the MODIFY RADICAL Process

Let's take this Mode by Mode:

- **Mode 1 REVERSE:** The soundfile is re-written back to front: starting at the end and ending at the beginning.

- **Mode 2 SHRED:** The soundfile is (randomly) segmented, and these segments reordered by means of a permutation process. As the number of *repeats* increases, it gets more and more jumbled, literally 'reducing it to shreds'. The **-n** adds splicing that results in a smoother output.

- **Mode 3 SCRUB:** This is an acceleration/deceleration process which models an editing procedure used in the 'classical' tape studio: the desired edit point was found by turning the tape spools by hand so that the tape moved (very slowly!) across the tape head. You could hear locate exactly where silence began or ended, where clicks came etc., although the sound was very low because of the slow speed.

  The SCRUB function could also be used creatively, to create extreme speed modifications of the source sound on the tape, e.g. an abrupt acceleration-deceleration as the tape went from not-moving, to fast-moving, to stopped as the hands jerked the tape across the heads.

- **Mode 4 LOSE RESOLUTION:** Reducing the sample rate reduces the level of the Nyquist frequency (sample_rate/2), thereby lowering the frequency level which can be safely handling during processing. Lowering the bit-resolution reduces the precision of the numerical expression of the data, making the digital 'quantisation' of the sonic material coarser. This means that time-varying information is lost.

- **Mode 5 RING MODULATION:** multiplies two (bipolar) signals. In this case, one signal is a soundfile and the other is a *modulating_frq*. This creates two 'sidebands' which are the sum and the difference of the two signals, while the carrier signal disappears. The result is a timbrally 'hollow' sound. (See Curtis Roads, *The Computer Music Tutorial*, pp. 215-220).

- **Mode 6 CROSS MODULATION:** multiplies two different soundfiles, producing a very strange mixture of the two. Each frequency of the second sound ring-modulates the first sound.

- **Mode 7 QUANTISE:** This lowers the bit-resolution, like the second parameter of LOSE RESOLUTION.

Musical Applications

Again, Mode by Mode:

- **Mode 1 REVERSE:** Reversing a sound can be used simply as a first stage in moving a familiar sound towards more abstract sonic material. Also, sounds can have a very characteristic 'signature' when played backwards (e.g., piano tones sound like crescendoing chords), so reversal can be used to achieve these effects.
- **Mode 2 SHRED:** Jumbling segments can be used for humorous effect, to texture a sound, or to create a series of randomly placed impulses (if the sound has sharp attacks in it).
- **Mode 3 SCRUB:** Scrub can be used for crazily improvised changes of speed within a sound.
- **Mode 4 LOSE RESOLUTION:** These processes will produce a deliberately coarse sound. 'Rough' sound material is sometimes useful as a serendipitous input to other processes.
- **Mode 5 RING MODULATION:** achieves a timbral alteration which makes it sound thin and hollow, depending on the *modulating_frq*.
- **Mode 6 CROSS MODULATION:** another wierd and wonderful effect which the CDP System allows you to explore!

End of MODIFY RADICAL

MODIFY REVECHO – Create reverberation, echo or resonance around the sound

Usage

modify revecho 1 *infile outfile delay mix feedback tail* [**-p***prescale*] [**-i**]
modify revecho 2 *infile outfile delay mix feedback lfomod lfofreq lfophase lfodelay tail* [**-p***prescale*] [**-s***seed*]
modify revecho 3 *infile outfile* [**-g***gain*] [**-r***roll_off*] [**-s***size*] [**-e***count*]

Modes

1 STANDARD DELAY: with feedback & mix (0 = dry) of original and delayed signal
2 VARYING DELAY: with low frequency oscillator varying the delay time
3 STADIUM ECHO: create stadium P.A. type echoes

Parameters

infile – soundfile to process
outfile – output soundfile
delay – delay time, **in milliseconds**
mix – amount of delayed signal in final mix: 0 gives 'dry' result (Range: 0 to 1)
feedback – produces resonances related to delay time (with short times) (Range: -1.0 to 1.0)
tail – time to allow decayed signal to decay to zero (Range: -1.0 to 1.0)
-p*prescale* – prescales input level, to avoid overload
-i – inverts the dry signal (for phasing effects)
lfomod – depth of the delay-variation sweep (Range: 0 to 1)
lfofreq – frequency of the delay-variation sweep (negative values give random oscillations)
lfophase – start-phase of the delay-variation sweep (Range: 0 to 1)
lfodelay – time in seconds before the delay-variation sweep begins
-s*seed* – non-zero value gives reproducible output (with the same seed) where random oscillations are used (see *lfofreq*)
-g*gain* – to apply to input signal (Default: 0.645654)
-r*roll-off* – rate of loss of level across stadium (Default: 1)
-s*size* – multiplies average time between echoes (the Default time between echoes of 0.1 sec)
-e*count* – number of stadium echoes (Default and max: 23)

Understanding the MODIFY REVECHO Process

Mode **1** provides a fixed delay time. If a smooth echoey effect is wanted, be careful to keep *mix* on the low side. Remember that *delay* is given **in milliseconds**; after 60ms or so, one begins to hear a strong reverberation; after 100 ms the reverberation begins to 'bounce', and by 200 ms one begins to hear distinct echoes. *Delay* times greater than 1000 will cause repetitions of (all or much of) the sound. **Don't forget to lengthen the *tail* to match the *delay* time.**

The *mix* parameter, in adding in the delayed signal, increases the reverberant effect. With a short delay time, this can sound like the reverberation which occurs in an enclosed space. *Feedback* has a similar effect, becoming very pronounced towards a value of 1.0 – rather like the 'feedback' which occurs when a microphone is placed facing a loudspeaker.

REVECHO Mode 1 – Mapping out the Parameter Space

Name & Mode	*infile & outfile*	*delay*	*mix*	*feedback*	*tail*
modify revecho 1	balsam be1	35	0.3	0.5	0.1
modify revecho 1	balsam be2	35	0.8	0.5	0.1
modify revecho 1	balsam be3	35	0.3	0.8	0.1
modify revecho 1	balsam be4	100	0.5	0.5	0.1
modify revecho 1	balsam be5	200	0.5	0.5	0.2
modify revecho 1	balsam be6	500	0.5	0.5	0.5

Musical Applications

The operation of a standard delay line in Mode **1** ranges from modest reverberation to echo effects. The reverberations, however, tend easily towards rough edges, so for smoother reverbs, look to the programs listed below.

In Mode **2**, lower values for *lfofreq* produce a more noticeable wave motion, while higher values add to the 'bounce' of reverberation.

Mode **3** creates a stereo *outfile*, bouncing the signal between speakers with a very prominent delay factor. The idea is that you hear the signal bouncing around the 'stadium'. The defaults work quite nicely, but you can intensify the effect by multiplying the delay time with the *size* parameter. Note that this is a multiple, so values less than 0 (multiplied with the default time of 0.1 sec) will decrease the delay time. Beyond *size* = 2, you are likely to run into insufficient buffer space, so if you really want longer delay times, you will have to increase the buffer size with 'set CDP_MEMORY_BBSIZE=...' (bear in mind the RAM capacity of your machine). (The default buffer size is 1 Mbyte and the units are in 'k': e.g., a BBSIZE of 3000 units is 3 Mbyte).

ALSO SEE: REVERB, **RMREVERB**, **TAPDELAY** and **NEWDELAY**.

End of MODIFY REVECHO

MODIFY SAUSAGE – Brassage on several sources

In **Release 6,** various multi-channel options became available in this update. See the *space* and *channel* parameters.
In **Release 7**, also see **NEWTEX** in the GRAIN function group, which also has multi-channel output options.

Usage

modify sausage *infile [infile2 ...] outfile velocity density hvelocity hdensity grainsize pitchshift amp space bsplice esplice hgrainsize hpitchshift hamp hspace hbsplice hesplice* [**-r***range*] [**-j***jitter*] [**-l***outlength*] [**-c***channel*] [**-x**] [**-n**]

Parameters

infile – soundfile to be processed, normally, but not necessarily, mono (see *space*)
infile2 ... – additional input soundfiles to process
outfile – output soundfile written after processing
velocity – speed of advance in *infile*, relative to *outfile* (Range: >= 0)

This is the inverse of a *timestretch* (i.e., 1/n: higher values make the output shorter, lower values make it longer). This permits an infinite timestretch.

density – amount of grain overlap (Range: > 0)

Values < 1 leaves intergrain silence, i.e., gaps. Extremely small values will cease to perform predictably.

grainsize – size of the grains in milliseconds (Range: must be > 2 * the length of the splices; Default: 50ms)
pitchshift – transposition of grains in + or - (fractions of) semitones
amp – gain applied to the grains (Range: 0 to 1; Default: 1.0)

Use only if you want amplitude to vary over a range &/or in time.

bsplice – length of start-splices on grains in ms (Default: 5)
esplice – length of end-splices on grains in ms (Default: 5)
space – set stereo position in *outfile* 0 = L, 1 = R (Range: 0 to *N*)

Various multi-channel options are available:

- If the *space* flag is used on a **stereo** input, the program mixes it to mono before acting.

- In order to spread the output of the brassage process over the multi-channel stage, both the *space* and *hspace* parameters should be set (use Mode **7**), with **different** values. These can now take values from zero to *N*, the maximum channel count specified for the output file. The output of the BRASSAGE process will be spread out between the two output channels specified. (Values below 1 correspond to positions between channel 1 and the highest output channel).

- For audibly equivalent results to a similar process with normal stereo output, a higher density value must be set.

range – of search for next grain, before *infile* 'now' (Default: 0 ms)
jitter – randomisation of grain position; Range 0 to 1 (Default: 0.5)
outlength – maximum *outfile* length (if end of data is not reached)

Set to zero (the Default) for this parameter to be ignored. But if *velocity* is 0 **anywhere**, *outlength* must be given.

channel – extract and work on just one channel of a multi-channel input (Range: 1 to *N*)

Various multi-channel options are available:

- Set *channel* to **0** (the Default) for this parameter to be ignored, or for a **multichannel** *infile* to be mixed to MONO before processing.

- Set *channel* to a **positive** integer (*N*), e.g., 3, the N^{th} channel of the multi-channel file is selected for processing.

- Set *channel* to a **negative** integer value (-*N*), to produce a soundfile. For example with the setting **-c-*8***, an 8-channel output soundfile will be produced.

Also see the *space* parameter above. *-d* –
-x – do exponential splices (Default: linear)
-n – no interpolation for pitch values (quick but dirty)

The presence of the upper limit parameters *hvelocity*, *hdensity*, *hgrainsize*, *hpitchshift*, *hamp*, *hbsplice*, *hesplice*, and *hspace* make it possible to specify a range of values. When set, the program chooses random values for that parameter **for each grain** between the lower and upper limits. For example, if *pitchshift* is -3 and *hpitchshift* is 4, the pitch of each grain will be somewhere within the interval of a perfect 5^{th}. **NB:** Furthermore, any of these parameters may vary in time (provide the name of a *time value* breakpoint file).

All parameters – except *outlength* and *channel* may vary over time.

Understanding the MODIFY SAUSAGE Process

MODIFY SAUSAGE is in fact a slight variant on MODIFY BRASSAGE in which the latter is adapted to accept more than one *infile*. This means that the 'brassage' operations will cycle around the input soundfiles, producing a more complex sonic texture.

For Reference manual details, see MODIFY BRASSAGE above.

NB: This program is not the same as the pre-Release 4 CDP program, SAUSAGE. The older SAUSAGE has the useful option to have the grains from the different sources appear in regular rotation or in random order. It can also cycle round a list of pitch (i.e., grain transposition) values. Because of these unique features, the older SAUSAGE program has been recompiled for Release 4 and included under 'CDP EXTRAS'. It is invoked on the command line simply by 'sausage' and has its own HTML manual of the same name.

Musical Applications

MODIFY SAUSAGE should be used when granular transformations need to work with several input soundfiles.

End of MODIFY SAUSAGE

MODIFY SCALEDPAN – Distribute sound in stereo space, scaling pan data to soundfile duration

Usage

modify scaledpan *infile outfile pan* [**-p***prescale*]

Parameters

infile – input soundfile, MONO only
outfile – output soundfile
pan – pan data breakpoint file: positions sound in a stereo field, from -1 (Left) to 1 (Right), or beyond
-p*prescale* – gain reduction factor applied to input level in order to avoid clipping (Default: 0.7)

Understanding the MODIFY SCALEDPAN Process

Sometimes it is useful to apply a process to many sounds, slightly modifying that process to suit the different durations of the source files. This is especially useful when using BULK PROCESSING on the *Sound Loom*, so that new data files do not need to be written for every single input source.

MODIFY SCALEDPAN enables you to take an existing pan file, for example with a panning pattern you use often, and apply it to a new sound. With **bulk processing** you could apply it to hundreds of sounds of slightly different duration.

Musical Applications

In breakpoint files, it can be important that the final value come at or near the end of the input soundfile. This can be particularly significant in PAN operations. Besides enabling you to reuse a favourite pan file with new sounds easily, the scaling also ensures (automatically) that the panning is properly timed with the duration of the *infile*.

End of MODIFY SCALEDPAN

MODIFY SHUDDER – Shudder a stereo file

Usage

modify shudder *infile1 outfilename starttime frq scatter stereo_spread mindepth maxdepth minwidth maxwidth*

Parameters

infile1 – input stereo soundfile
outfilename – note that a '0' will replace the last character of your given outfile name
starttime – time when the shuddering will begin
frq – the average frequency of the shuddering
scatter – randomises the shudder events in **time**. (Range: 0 to 1)
stereo_spread – positions the shudder events in space. (Range: 0 to 1)
mindepth maxdepth – amplitude of the shudders – each gets a random value between MIN and MAX
minwidth maxwidth – the duration of the shudder events in milliseconds – each gets a random value between MIN and MAX

Name of *outfile* must not end with a '1'

Understanding the MODIFY SHUDDER Process

MODIFY SHUDDER imposes tremulations on a stereo file, randomised both in time **and space**, so that it appears to shudder. It is used in the shuddering noise band which then 'speaks' in Trevor Wishart's *Imago*.

Note the relationship between *frq* and *width*: the frequency per second of the shudders and the width of soundfile segment that is affected. Thus there could be one shudder per second involving 500ms (0.5 sec) of sound, or one shudder per second involving 100ms (0.1 sec), etc. *Depth* and *width* MIN and MAX can be the same value.

Musical Applications

Although not unlike tremolo and vibrato, these 'shudders' are designed to be more gestural, more dramatic in character.

Suggestion: experiment with larger values for *count* (e.g., 10 or more), high values for (amplitude) *depth* (e.g., 0.9 - 1.0), and smaller values for *width* (e.g., 0.07 - 0.1). Then try slower shudders, such as *count* = 2 and *width* (both MIN and MAX) = 0.5.

End of MODIFY SHUDDER

MODIFY SPACE – Spatialise, or alter the spatialisation of, a soundfile

Usage

modify space 1 *infile outfile pan* **-p***prescale*
modify space 2 *infile outfile*
modify space 3 *inpantextfile outpantextfile*
modify space 4 *infile outfile narrowing*

Modes

1 PAN: Position or move mono sound in a stereo field
2 MIRROR: Invert stereo positions in a stereo field (i.e., the two channels swap sides, left going to right, and right to left)
3 MIRRORPAN: Invert stereo positions in a pan data file
4 NARROW: Narrow the stereo image of a sound

Parameters

infile – input soundfile to modify; Mode**1** accepts a mono input and produces a stereo output. The other modes accept only a stereo input.
inpantextfile – a text file containing *time pan_position* data
outfile – resultant spatialised soundfile
outpantextfile – a text file containing *time pan_position* data produced by the program, with the pan positions reversed
pan – floating point value to specify location between the pair of stereo speakers, or breakpoint file of *time pan* pairs. Range: -1.0 to 1.0; 0.0 is centre.
-p*prescale* – gain factor multiplier with which to adjust the input level (Default: 0.7)
narrowing – make the stereo image less wide:

- **1** leaves the stereo image where it is
- **.5** narrows the stereo image by half
- **0** converts the stereo image to mono
- negative values work similarly, but also invert the stereo image

Understanding the MODIFY SPACE Process

Each Mode performs different but related operations on a sound:

- Mode **1** PAN: Adjusts the amplitude levels of the two channels so as to create spatial illusions. A *time pan* breakpoint file is used to create movement.
- Mode **2** MIRROR: the data in the two channels swaps sides
- Mode **3** MIRRORPAN: this Mode acts on a breakpoint file, reversing the pan data. Thus, if the pan was from Left to Right, (-1 to 1), it would become Right to Left (1 to -1).
- Mode **4** NARROW: a way to restrict the spatial location of a sound in the horizontal plane

Musical Applications

These functions are part of the basic toolbox.

- Mode **1** PAN: control of sound in 3-D space has been a basic procedure when composing with 'sound' material, especially when these are drawn from nature/the environment. The assumption here is that you are thinking about the placement and movement of sound as an integral part of your compositional process. In this case, we are dealing with building these procedures into the sound itself. There is a further aspect external to the sound itself, achieved by 'mixing' (superimposing sounds and writing a new soundfile) and 'diffusion' (sound placement during playback achieved by manipulating the pan controls on a mixer).
- Mode **2** MIRROR: a utility for a quick Left/Right swap
- Mode **3** MIRRORPAN: a utility to achieve Left/Right swaps within breakpoint files
- Mode **4** NARROW: this program allows you to narrow the spread of a stereo field

Eventually, multi-channel facilities will open up the possibilities of spatial movement enormously. Until that point, everything has to be done in stereo and mixed to create more complicated, e.g., 4-channel effects. For example, if working with a 4-channels and 4 speakers positioned 1 -2 in front and 3 - 4 behind, oblique movement from 2 to 4 can be achieved by a Left to Right pan within the sound, and then using the mixing desk to move that sound from speaker 2 to speaker 4.

End of MODIFY SPACE

MODIFY SPACEFORM – Create a sinusoidal spatial distribution data file

Usage

modify spaceform *outpanfile cyclelen width dur quantisation phase*

Parameters

outpanfile – output pan datafile produced by the program
cyclelen – the duration of one complete sinusoidal pan cycle
width – the width of the pan (from 0 to 1, full width)
dur – the duration of the output file
quantisation – the time step between successive space-position specifications
phase – the angular position at which the pan starts. 0 is full left and 360 is full right.

cyclelen and *phase* may vary over time

Understanding the MODIFY SPACEFORM Function

MODIFY SPACEFORM produces a pan breakpoint file (.brk would be the correct extension) that can be used with **PAN** or **MODIFY SCALEDPAN**. Note that the duration of the file, i.e., the last time in the file, can be specified.

This function in effect swings the sound back and forth between the speakers in a stereo field. The speed at which it does so is controlled by *cyclelen* and the location between the speakers is handled by *width*. If *width* is:

- **1**, the full stereo field is used
- **0**, no panning takes place and the sound is heard equally (without movement) from both Left and Right speakers.
- **0.5**, the swing is from halfway between the Left speaker and Center to halfway between the Right speaker and Center, i.e., from -0.5 to + 0.5

The *quantisation* parameter determines the number of intermediate pan positions there will be during each swing around the loop. 0.1 appeared to produce a good result. The *phase* sets where in the stereo field the sound will be heard at the beginning of each repeat cycle, from whence the movement will emerge.

Note the sophistication that can be brought to the spatial movement when breakpoint files for *cyclelen* and *phase* are employed.

Musical Applications

A slowly cycling panning motion can be created, generating a sense of an object rotating in space, e.g. a marble rolling around the rim of a huger glass bowl in front of you. This spatial image can be enhanced by making the level increase on the left-to-right motion, and decrease on the right-to-left motion (or vice cersa), suggesting that the sound moves towards, then away from you, as well as from right to left. This perception can be enhanced by adding treble-cut filtering and/or subtle reverb to the 'more distant' part of the motion. More generally speaking, sound streams in a piece can be differentiated and characterised by their spatial motion.

Also see **MODIFY SPACE** (PAN is Mode **1**) and **MODIFY SCALEDPAN**.

End of MODIFY SPACEFORM

MODIFY SPEED – Change speed (& pitch) of sound

Release 6: The multi-channel equivalent of MODIFY SPEED is **STRANS MULTI**.

Usage

TRANSPOSITION:
(NB! - Transposition is always relative to the **original** speed of the soundfile)
modify speed 1 *infile outfile speed* [**-o**]
modify speed 2 *infile outfile semitone-transpos* [**-o**]

INFORMATION:
modify speed 3 *infile inbrkfile* [**-o**]
modify speed 4 *infile inbrkfile* [**-o**]

ACCELERATION / DECELERATION:
modify speed 5 *infile outfile accel goaltime* [**-s***starttime*]

VIBRATO:
modify speed 6 *infile outfile vibrate vibdepth*

Modes

1 Vary speed & pitch of a sound, constant or time-varying
2 Vary speed & pitch by a constant or time-varying (fractional) number of semitones
3 Get information on varying speed in a time-changing manner
4 Get information on time-variable speed change in semitones
5 Accelerate or decelerate a sound
6 Add vibrato to a sound

Parameters

infile – input soundfile to process
outfile – resultant soundfile
speed – transposition value (ratio) expressed as a floating point multiplier. See Chart of Ratios covering up and down 2 octaves. **NB:** use **1.0** for no transposition.
semitone-transpos – transposition value in positive or negative number of semitones; e.g., 12 raises the sound by an octave, and -12 lowers it by an octave. **NB:** use **0** (semitones) for no transposition.

Both *speed* and *semitone-transpos* may vary over time.

accel – multiplication of speed to be reached by *goaltime* – i.e., a transposition ratio
goaltime – time in *outfile* at which the accelerated speed is to be reached.

- If the *infile* does not end there, it continues to accelerate.
- If the *infile* finishes before *goaltime* is reached, the *outfile* won't reach the specified acceleration value.

starttime – time in *infile / outfile* at which the acceleration begins
vibrate – the rate of vibrato shaking in cyles-per-second (Range: 0.0 to 120.0)
vibdepth – vibrato depth (pitch shift from centre) in [possibly fractional] semitones (Range: 0.0 to 96.0)

Vibrate and *vibdepth* can vary over time.

-o – breakpoint times are read as times in the *outfile*. The Default is to read them as times in the *infile*

Understanding the MODIFY SPEED Process

Speed modification processes change the **duration and the pitch** of the sound together. Thus a faster speed causes a higher pitch, a slower speed a lower pitch.

MODIFY SPEED offers a range of functions which affect the speed of the soundfile. Perhaps it will be most often used for transposition. Modes **1** and **2** both accept either single values or the names of time-varying breakpoint files with *time transposition* pairs.

The single values act as constants and transpose the whole soundfile up or down by the given amount. In the breakpoint files, transposition can be **almost instantaneous** (almost same time, different transposition value), or **gradual**, creating glissandi (different time, different transposition value). **No transposition** between times is a third possibility (different time, same transposition value). These three possibilities are illustrated in the table below:

Time-varying transposition (using ratios)

time	***speed***	**Comments**
0.0	1.0	No transposition: start at the original pitch
1.0	1.498	Over 1 sec., gliss upwards through a Pefect 5^{th}
3.0	1.498	Hold this new level for 2 sec.
3.0001	0.5	At 3 sec., (almost) instantly drop one 8^{ve} below the original pitch, i.e., to 19 semitones below the previous position
6.0	1.0	Spend the next 3 sec. glissing back to the original pitch

The program will not accept exactly simultaneous values, giving a message to the effect that the times 'are not in increasing order'. To get around this, add a tiny bit to the second time – as in the example – so that the 2^{nd} is nominally later than the 1^{st}, but virtually simultaneous.

IMPORTANT! Note that transposition changes are always **relative to the original speed** of the soundfile, not its current output speed. Thus, in the example above, the soundfile glisses up a perfect 5^{th} (from *speed* 1.0 to *speed* 1.498). With the next ratio, 0.5, the soundfile will drop to half of its **original** speed, and consequently to the octave below its **original** pitch.

Sometimes, you may need certain transpositions to occur at specific times in the *outfile*. The **-o** flag makes this possible. Thus, changing the speed of a file will alter it's duration, and, especially when the speed change itself varies in time (using a breakpoint file), it will be difficult to determine at what time events in the output sound will appear. For example, if you specify a *speed* of 0.5 at *time* 2.0 with the default mode, the speed of the file will reach 0.5 after 2 seconds of the INPUT file have passed – but because the speed of the file has been changed, this will **not** be at 2.0 seconds in the OUTPUT file. **Therefore, if you want the *speed* to reach 0.5 at *time* 2.0 in the OUTPUT file, you should use the -o flag**.

Musical Applications

Transposition which also changes the speed, and therefore the pitch, of the soundfile greatly alters the character of the sound. It is often very interesting to hear what a sound will be like 1, 2 or even 3 octaves below its original pitch. Deep, rich tones can be achieved in this way. These tones can slowly rise or descend if created with a time-varying breakpoint file e.g., moving an octave up or down over the time of the whole sound (airplane takeoff sounds, etc.).

The graphic program *Brkedit* (PC systems) can create exponential or logarithmic breakpoint data, so glissandi in MODIFY SPEED can increase or decrease in speed as well as move in a steady (linear) manner. Alternatively, this can be done with Mode **5**.

Vocal material is very sensitive to pitch changes, so upwards transposition of this type will produce fast, squeaky voices, like the mice trio in *Babe*, and downwards transposition will produce slow, drawn-out ponderous voices.

The vibrato created in Mode **6** is a frequency modulation. Given the very wide ranges allowed, this function is immensely powerful. A slow *vibrate* with a large *vibdepth* will swing the original sound wildly – increase *vibrate* and it really 'flaps in the breeze' (like a flag in the wind). A fast *vibrate* with a reasonably tight *vibdepth*, e.g., a minor 3rd, will produce a fluttering effect.

Altogether, a great program to explore and use to push beyond accepted conventions.

NB For transposition which alters the pitch without altering the duration of the sound, use **REPITCH TRANSPOSE** (spectral envelope moves) or **REPITCH TRANSPOSEF** (original spectral envelope is retained).

Also see: STRANS MULTI, the multi-channel version of this function.

End of MODIFY SPEED

MODIFY STACK – Create a mix that stacks transposed versions of source on top of one another

Usage

modify stack *infile outfile transpos count lean attack-offset gain dur* [**-s**]

Parameters

infile – soundfile to stack
outfile – resultant stacked soundfile
transpos – when numeric, *transpos* is the transposition in semitones between successive copies, OR, as a file, it is a set of transposition values, one for each stack component
count – the number of copies in the stack
lean – the loudness of the highest component, relative to the lowest (may be > 1)
attack-offset – adjusts the **time** at which the attack of each sound occurs; called *attack-time* in *Sound Loom*.
gain – an overall amplifying gain factor on the output sound; some reduction (less than 1) may be needed. (It may also be > 1.) Range: 0.1 to 10
dur – how much of the output to make (a proportion, from 0 to 1)
-s – see the relative levels of the layers in the stack

Understanding the MODIFY STACK Process

New sounds can be developed from existing sources by stacking transposed copes of the original on top of each other. This is particularly successful where the source sound has a clearly defined attack at or near the start of the sound and then dies away gradually. If the copies are **lined up in such a way that their *attacks* coincide exactly**, the new sound will psycho-acoustically merge into a single sound event with harmonic content (rather than a chord made up of several sounds). The meaning of this is very precise: one is lining up the soundfiles at their *perceived* beginning, i.e., its **attack**, not the start of each soundfile. This is achieved by adjusting the start times of the transposed soundfiles.

Trevor Wishart has clarified what happens with this program as follows:

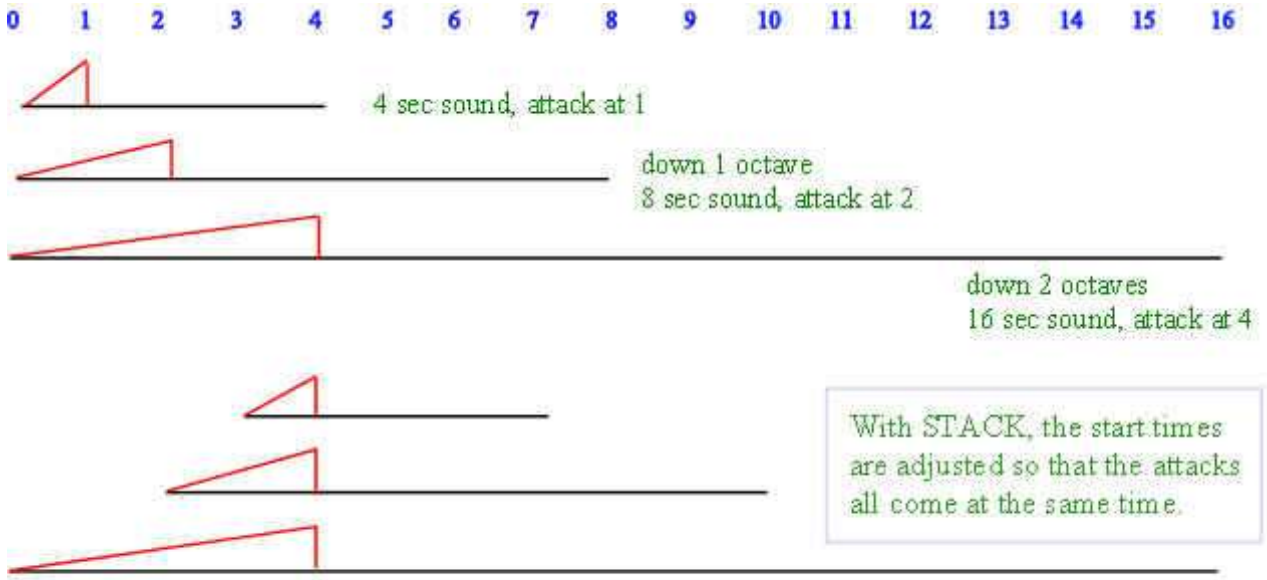
> "For example, suppose you start with a source with its attack at 0.5 sec, and your stack transpositions are 0, 12, 24 semitones (i.e. zero, 1 octave up, and 2 octaves up). If you tell the process that the attack is at zero (which it isn't!!!) then the transposed sounds in the output will be lined up to synchronise at zero time, but you will actually hear the event attacks at .125 (24 semitones up), .25 (12 semitones up), and 0.5 seconds: because the transposed-up sounds get shortened in time, the attacks of the higher transpositions occur earlier in the output sound. **If, however, you declare, correctly, that the attack is at 0.5 seconds in the source, then all the sounds will line up with their attacks at 0.5 seconds in the output.**
>
> "In this example, where all the transpositions are upwards, the attack in the output is still at 0.5 secs as described.

"On the other hand, if we used a stack with 2 DOWNWARD transpositions by octaves(0, -12, -24), in order to REACH the attack of the most-slowed-down sound we have now to wait 0.5 x 4 = 2 seconds. Telling the program that the attack (in the original sound) is at 0.5 seconds will STILL line up all the attacks correctly, but they won't happen until 2 seconds into the output sound, as you can't reach the attack of the slowest sound before 2 seconds have elapsed.

"So this parameter is intended to tell the process where the attack in the original sound really is. However, you can get interesting effects by giving a slightly incorrect value, sometimes producing rapid pitch-slide attacks in the output."

It is important, therefore, that you accurately identify the time-location of the attack of the sound and work with this information, either completely fusing the sounds by making their attacks coincide, or introducing slight timing differences for various musical reasons, such as a sense of acceleration within a sound. Thus you can intentionally *misinform* the program about where the attack is in order to get different kinds of output. If you give an attack time that is only slightly wrong, this can produce interesting new attack characteristics in the original. If the attack time given is completely incorrect, the outcome could be anything.

In the illustration below, the attack comes at 1 second after the start of the sound. A downwards transposition by 1 octave doubles the length of the sound, causing the attack to come at 2. Downwards transposition by 2 octaves quadruples the length of the sound, causing the attack to come at 4.



MODIFY STACK - aligns attacks

- If we tell MODIFY STACK that the attack comes at the beginning of the sound (*attack* = 0), we get the upper diagram, with the time between attacks slowing down.

- If we tell the program (correctly) that the attack is at 1 second, then all the attacks are aligned at the 4 second mark, as in the lower diagram.

The *dur* (duration) parameter is the proportion of the sound you want to keep. 1 keeps all of it, 0, none of it, and 0.5, half of it. This is to enable you to truncate the output when very low transpositions would make the output excessively long.

Octave transpositions are particularly effective at 'strengthening' a sound source, but any harmonic structure may be used. Alternatively, the attacks may be very slightly staggered to generate downward (or upward) sweeping glissandi, or very tiny transpositions may be used introducing delay effects into the output.

Musical Applications

MODIFY STACK is a very important program. It enables you to construct 'larger' versions of sounds by superimposing transposed versions on top of one another. Furthermore, it does this in such a way as to maximise the fusion of the layered sound by synchronising the attacks. This relates to a basic technique of orchestral scoring (where it is called 'doubling'). It is a form of automatic mixing involving one soundfile.

Because of the way the program functions, this is, however, quite a different process from transposing and mixing sounds to produce chords. (Doing this in the Spectral Domain, where transposition can be done without affecting duration, still has a place: see, for example, the duplication possibilites in **SPECTWIN**.) Here, if the attack points are made to coincide, **the new sound-object is an integral whole**, rather than a 'chord' made up of the transposed stack components: that is, the components fuse and cannot be separated in perception, as they can with a chord played, for example, on a piano keyboard.

Also note that, as a Time Domain process, the higher transpositions end sooner than the lower ones. Use Spectral Domain transpositon along with SUBMIX MIX to assemble transposed versions of the same sound when equal length is important.

Very low transpositions could produce an extremely long output. The *dur* parameter enables you to specify a much shorter outfile length. You can usually get an idea of the harmonic content of the output from the first few seconds.

Also see: **SPECTWIN**

End of MODIFY STACK

VERGES – Play source, with specified brief moments glissing up or down

Usage

verges verges *infile outfile times* [**-t***transp*] [**-e***exp*] [**-d***dur*] [**-n**] [**-b** |**-s**]

Parameters

infile –input soundfile.
outfile – output soundfile.
times – datafile of approx. times to introduce the glissando.
-t*transp* – semitone transposition at start of verge (Range: -24 to +24 semitones; default: 5)
-e*exp* – slope of glissandi; higher values gliss faster. (Range: 1 [default] to 8)
-d*dur* – duration of glissandi. (Range: 20 to 1000 mS; default: 100 mS)
-n – use times given. (Default: looks for peak closest to time in source.)
-b – boost the level of the verges.
-s – suppress all input except verges (boost verges).

Understanding the VERGES Process

VERGES introduces a brief glissando up or down at the times specified. The normal operation would be to extract a set of times matching elements in the source. The duration *dur*, degree of transposition (*transp*) and speed of the glissando *exp* can all be set.

Musical Applications

...

End of VERGES

Documentation: Archer Endrich, revised R.Fraser