List of Functions in CDP Component Packages
~ Release 7 (17/3/2018)~

CDP Component Packages

CDP PACKAGES	GROUPS/CATEGORIES ([TD] = Time Domain, [SD] = Spectral Domain)		
CDP-FOCUS	BLUR [SD]	FOCUS [SD]	
CDP-PITCH	MODIFY [TD]	REPITCH [SD]	
CDP-TEXTURE	TEXTURE [TD]		
CDP-X	DISTORT [TD]	GRAIN [TD]	STRANGE [SD]
CDP-UTILS-1	ENVEL [TD]	ENVNU [TD]	

List of Functions in Groups and Packages

Function names in square brackets are separate programs. The others are sub-modules of a Group program.

CDP-FOCUS – focusing and blurring

BLUR | FOCUS

CDP Processing Functions to BLUR sonic material (BLUR)

BLUR Blur spectral data over time
NOISE Add noise to spectrum

BLUR BLUR – Time-average the spectrum

Usage

blur blur *infile outfile blurring*

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

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

Understanding the BLUR BLUR Process

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

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

Musical Applications

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

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

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

BLUR NOISE – Put noise in the spectrum

Usage

blur noise *infile outfile noise*

Parameters

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

noise may vary over time

Understanding the BLUR NOISE Process

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

Musical Applications

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

CDP Processing Functions to FOCUS on sonic material (FOCUS)

ACCU	sustain each spectral band, until louder data appears in that band
STEP	Step-frame through a sound by freezing the spectrum at regular time intervals

FOCUS ACCU – Sustain (accumulate) each spectral band, until louder data appears in that band

This function has been supersed and extended in Release 7 by SUPERACCU.

Usage

focus accu *inanalysisfile outanalysisfile* **-d***decay* **-g***glis*

Typical command line:

focus accu raspdt.ana raspdtacc.ana -d0.75 -g-0.9

Parameters

inanalysisfile – input analysis file
outanalysisfile – output analysis file
-d*decay* sustained channel data decays by a factor of *decay* each second

(Possible Range: 0.001000 to 1.0. Default is 1.0. – Suggested effective Range: 0.000001 to 0.5)

-g*glis* sustained channel data glissandos at *glis* octaves per second

(Approximate Range: -11.7 to 11.7. Default is 0)

Understanding the FOCUS ACCU Process

Frequencies are sustained into subsequent windows. The overall effect is one of sustaining, but one which also makes the spectrum more complex.

The *gliss* parameter appears to change the result the most. It produces glissandos within the spectrum of the sound. Very effective slow glissandos are produced when *gliss* is near 0, e.g., -0.9 or 0.1. At 0.5, there are several glissandos, at 1, they are fairly fast, and at 10 it becomes a wash.

The range allowed for *gliss* is really very extreme, especially in the negative range below zero. Values e.g., of -5.0 or less lead to the sound settling down to a steady, sustained pitch after an initial short glisssando. **Recommended effective range is from -0.5 to 10**.

The *delay* parameter is important for making the glissandos more perceptible. A low value of e.g., 0.1 is really very dry, whereas something like 0.75 brings out the glissandos nicely, with a kind of reverberant aura.

Musical Applications

The main type of output from this function creates a supple, internally modulating sound, moving slowly at the low end of the *gliss* range and increasingly quickly as one moves towards the high end.

With a clearly pitched input, the results are aurally clearer, but otherwise similar. TIMESTRETCH applied to these results produces nicely slow-moving timbral and pitch changes.

Also see: HILITE ARPEG. Churning, bubbly textures can also be achieved with **BLUR SCATTER** (high values for *keep*) and **HILITE TRACE** (low value for *channels*).

FOCUS STEP – Step-frame through sound by freezing spectrum at regular time intervals

Usage

focus step *inanalysisfile outanalysisfile timestep*

Typical command line:
focus step raspdt.ana raspdtstep.ana 0.1

Parameters

inanalysisfile – input analysis file
outanalysisfile – output analysis file
timestep duration of the steps. Must be >= the duration of two analysis frames. The value here is rounded internally to a multiple of analysis frame time.

Understanding the FOCUS STEP Process

This is an automatic FREEZE of the spectrum at intervals of *timestep*, similar to the familiar 'sample-hold' effect. The main thing here is that the time intervals are *regular*. Thus the spectrum of the sound, which normally is constantly changing, is frozen with the values at the beginning of each *timestep*.

> The effect with fairly large *timesteps* can be a relatively slow-moving sample-hold effect. But as the duration of the *timestep* was shortened, the complex source developed regular 'jangling' effect, while the clearly pitched source only changed its tone.
>
> The 'jangling' effect seems to occur with *timestep* at about 0.1 or even 0.05. At 0.25, the sample-hold is quite noticeable, rather slow at 0.5 and probably too slow at 1.0 – but this setting could be used to produce sections of steady-state material that can then be CUT and used elsewhere.
>
> *Timestep* can be very small. The Phase Vocoder reference manual explains that the value shown by DIRSF for the sample rate of an analysis file is actually frame rate: the number of analysis frames per second, e.g. 172 or 344 etc. (The sample rate and value for **-N** both alter the frame rate.) So if you divide 1 sec by the frame rate, you will learn the duration of one frame: e.g., 1/344 = 0.0029 (2.9ms).
> As *timestep* gets larger, timbral alteration increases.

Musical Applications

It is recommended that you start exploring this function by using very small values for *timestep*. This should produce a granulation effect. But note that it is doing this without creating enveloped 'grains' as does MODIFY BRASSAGE. It is working directly on the spectrum of the sound.

CDP-PITCH – transposition, pitch-warping, harmony, tuning, loudness, echo &pan

CDP Processing Functions to MODIFY sounds (MODIFY)

BRASSAGE	Granular reconstitution of a soundfile
RADICAL	Radical changes to the sound
SPEED	Change the speed and pitch of the source sound

MODIFY BRASSAGE – Granular reconstitution of soundfile

Usage | Modes | Parameters | Understanding | Musical Applications

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* [**-d**]

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

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

Usage | Modes | Parameters | Understanding | Musical Applications

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!

MODIFY SPEED – Change speed (& pitch) of sound

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

Usage | Modes | Parameters | Understanding | Musical Applications

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 5th
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 2nd is nominally later than the 1st, 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 5th (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.

CDP Processing Functions to REPITCH (further modify) pitch data (REPITCH)

TRANSPOSE Transpose spectrum (spectral envelope also moves)

REPITCH TRANSPOSE – Transpose spectrum (spectral envelope also moves)

Usage

repitch transpose 1–3 *infile outfile transpos* [**-l***minfrq*] [**-h***maxfrq*] [**-x**]
repitch transpose 4 *infile transpos outfile* [**-l***minfrq*] [**-h***maxfrq*] [**-x**]

Modes

1 Transposition as a frequency ratio
2 Transposition in (fractions of) octaves
3 Transposition in (fractions of) semitones
4 Transposition as a binary data file (**.trn**)

Parameters

infile and *outfile* are analysis files
transpos for modes **1–3** a transposition factor or time varying breakpoint file in a format according with the mode being used, but note that Mode 4 must be used if the transposition data is in a binary format (.trn). The binary or breakpoint input files can be made by REPITCH COMBINE or REPITCH COMBINEB, and a breakpoint file can also be created 'by hand' in a text editor.
-l*minfrq* – minimum frequency, below which data is filtered out
-h*maxfrq* – maximum frequency, above which data is filtered out
-x – fuller spectrum

Frequency ratio, octave or semitone transpositions may vary over time.

Understanding the REPITCH TRANSPOSE Process

REPITCH TRANSPOSE does fixed or time varying transposition in the spectral domain. It accepts a variety of breakpoint file inputs as well as a binary data file. The spectral envelope moves with the transposition (as opposed to TRANSPOSEF, where it does not), meaning that the timbres associated with formants are going to change.

Musical Applications

The process of transposition can be approached in various ways:

- Transpose a file by a constant amount.
- Create a transposition breakpoint file by hand and apply it to any sound.
- Extract a pitch contour with REPITCH GETPITCH, saving as a breakpoint (.brk) file and use this output to combine with another pitch contour or

transposition file in COMBINE/B to form another (time varying) transposition file to apply to an *infile*.

- Extract pitch contours from two files with REPITCH GETPITCH. Then create a transposition file with REPITCH COMBINE/COMBINEB Mode 1 which contains the information needed to change the pitch contour of the first file into that of the second. Then apply this transposition data to the first file here with TRANSPOSE in order to effect the transposition.
- Similarly the output of COMBINE/COMBINEB Mode 3 can be used as the transposition input to TRANSPOSE. Summing transposition data can be an interestingly serendipitous (unpredictable) exercise.
- Modify *pitchfile* data in some way, save as a transposition file and use as an input to TRANSPOSE/F. The following REPITCH pitchfile modification functions have a transposition file output mode: APPROX, EXAG, INVERT, QUANTISE, RANDOMISE, SMOOTH and VIBRATO.

The results of these transpositions can be used to alter timbre, especially with TRANSPOSE. Also, some use of transposition could form an important step when realising a morph: to draw one sound closer to that of another in frequency region or timbre, or to create a aurally transitional stage in the morphing process.

CDP-TEXTURE – texture-builder with harmonic/set options

TEXTURE

CDP Processing Functions to build complex and supple TEXTURES, often on harmonic lattices (TEXTURE)

SIMPLE	Create textures from single events (Also see TEXMCHAN and NEWTEX)

TEXTURE SIMPLE – A texture of events shaped by random selections from parameter ranges, with one or more input sounds; a Harmonic Field/Set may be used

Parameters	Notedata File	Understanding	Two fields/sets	Applications
TEXMCHAN – Special Notes on differences in Multichannel version				

Usage

texture simple mode *infile [infile2] outfile notedata outdur packing scatter tgrid sndfirst sndlast mingain maxgain mindur maxdur minpich maxpich*
[**-a***atten*] [**-p***pposition*] [**-s***spread*] [**-r***seed*] [**-w**]

Modes

1 On a given harmonic field
2 On changing harmonic fields
3 On a given harmonic set
4 On changing harmonic sets
5 None (Neutral)

Parameters

infile – input soundfile to use as source material
infile2 ... – optional soundfile(s) to use as additional inputs
outfile – output soundfile
notedata – **textfile, containing:**

1. assumed MIDI 'pitch' of each input sound, specified on the 1st line. In Mode **5** this is all that is needed.
2. In Modes **1 - 4** there is also a NOTELIST to define a Harmonic Field/Set, specified thus:
   - **#***N* (where *N* is the number of pitches in the notelist).
   - This is followed by *N* lines in the format:
     time (secs) *infile_no pitch* (MIDI) *amp* (MIDI) *dur* (secs)
   - *Amp* and *dur* are inactive.
   - Any times within the field must increase (or remain the same, during chords)

Form: **MPV/–/–/–**		Form: **MPV/HF-S/–/–**	
Mode 5 format	**Comments**	**Modes 1-4 format**	**Comments**
60	MIDI pitch value(s)	60	MIDI pitch value(s)
		#N	No. of lines to follow
		time instr pitch amp dur	List of Harmonic Field/Set

outdur – minimum duration of the *outfile*
packing – (average) time in seconds between event onsets
scatter – randomisation in seconds of event onsets (Range: 0 to 10)
tgrid – minimum step in milliseconds or quantised timegrid (for group start times) (Range: 0 to 10000; Default: 0)
sndfirst, sndlast – first and last soundfiles to use from a list of soundfiles for input (Range: 1 to the number of sounds). Note that this is a time-varying parameter:

- Enter 1 for both *sndfirst* and *sndlast* to use just one input soundfile.
- When you use more than one sound, the names of these input soundfiles are given to the program. The note data file must provide a reference pitch level for each of these sounds. If you just enter, e.g., 1 for *sndfirst* and 3 for *sndlast*, the program will select one of the three input soundfiles at random for each event in the texture.
- To specify when each soundfile is to be used in creating the output texture, a text file is needed to provide a *time sound_number* pair of values, the numbers corresponding to the order in which the soundfiles have been submitted to the program.
- For one file at a time changing at different times in the output (the times given are as in the output soundfile), use the same file for both *sndfirst* and *sndlast*, e.g.,

```
sndfirst  sndlast
 0.0  1   0.0  1
10.0  2  10.0  2
18.8  3  18.8  3
```

- For a changing number of files, such as expanding the number of files being used at any one time (illustrated below), use a lower number at a given time for *sndfirst*, and a higher number at the same time (or an overlapping time) for *sndlast*.

```
sndfirst  sndlast
 0.0  1   0.0  1
10.0  1  10.0  2
18.8  1  18.8  3
```

Note that the rate at which each soundfile is employed will slow down as more are invoked over the same time period.

mingain, maxgain – minimum and maximum level of input sounds (Range: 1 to 127; Default: 64 and 64)

mindur, maxdur – minimum and maximum duration of events in texture
minpich, maxpich – minimum and maximum pitch (MIDI note value)
-a*atten* – overall attenuation of the output
-p*position* – centre of sound output image (Range: 0 to 1, where 0 is Left and 1 is Right; Default: 0.5)
-s*spread* – spatial spread of texture events (Range: 0 to 1, where 1 is full spread)
-r*seed* – the same *seed* number will produce the same result on rerun (Default: 0, where 0 is different result each time)
-w – always play whole input sound, ignoring duration values

Understanding the TEXTURE SIMPLE Process

Getting started
Please make sure you have gone through the background information before continuing.

This Reference Document for the TEXTURE suite contains numerous worked examples which you can hear directly from this manual. They are stored in **\txsnds**. You can also go to the **\txpack** folder, where you will find all the files you need run to create the sound examples yourselves. There are two versions: one for the Command Line and *Soundshaper*environments and one for the *Sound Loom* environment.

What you need from the **txpack** are the note data and breakpoint files, as well as the two source sounds, *horn.wav* and *marimba.wav*, **or source sounds of your choice**. The parameter information is stored in **Presets** supplied with *Soundshaper* and **Patches** supplied with *Sound Loom*. In a Command Line environment, you can run the batch (.bat) files supplied in **txpack**.

Location of the source sounds and other files:

- *Soundshaper* – In OPTIONS > SETTINGS go to **File** in the top left corner and open **txpack.cfg**. This Will take you to **\txpack**, which contains the source sounds, note data and other files needed to run the examples. Also load the **Presets** file **Presets.dat**, which is the default file supplied with the Program).
- *Sound Loom* – select **txpack** as your working directory, **Grab** everything and place the files on the **Workspace**. Now use **Choose Files** to select the appropriate sooundfile(s) for processing. The Patches go in the **_cdpatch** folder, and should be there already in your distribution.
- Command line environment – 'change directory' to **\txpack** and run the various **batch** files, e.g., **simplexs.bat** for the TEXTURE SIMPLE examples.

The complete list of files is given in *Readtxtu.txt*. The list at the date of this file is as follows:

Note data files:

ndfsiml.txt *ndfsim2.txt*
ndfsim3a.txt *ndfsim3b.txt*
ndfsim4a.txt *ndfsim4b.txt*
ndfsim5.txt *ndfsim6.txt*
ndfdec1.txt *ndfdec2a.txt* *ndfdec2b.txt* *ndfdec3.txt*
ndfmot1.txt *ndfmot2.txt* *ndfmot3.txt* *ndfmot4.txt*
ndforn1.txt *ndforn2.txt* *ndforn3.txt* *ndforn4.txt* *ndforn5.txt*
ndftim1.txt *ndftim12.txt* *ndftim3.txt*
ndftgr2.txt
ndftmo1.txt *ndftmo2.txt*

Breakpoint files:

packchng.brk simplpak.brk grppack.brk
gprlo.brk gprhi.brk
dm5gprhi.brk dx3gprhi.brk
Names for extra breakpoint files for ORNATE examples,
to make yourself. These are described in ORNATE section:
oex3mulo.brk oex3muhi.brk
oex3mulb.brk oex3muhb.brk
oex3skip.brk oex3skpa.brk oex3skpb.brk

Please check your *Soundshaper* and/or *Sound Loom* reference manuals on handling Presets and Patches. If you are a *Sound Loom* user, for your convenience, we also supply *ctextusl.htm*, a version of this document which is *Sound Loom* specific, with all parameters as named in *Sound Loom*. Also note that *Sound Loom* Version 5.6 and above can run batch files.

First example
Mode **5** is a good place to begin. In this Mode, the note data file contains only one value: the (real or arbitrary) MIDI pitch of the input soundfile. *There is no further data because there is no list of notes.* All the note events are generated from the information of the command parameters.

To get a feel for the 'lie of the land', we can run TEXTURE SIMPLE in Mode **5** with the following parameters (*ndfsim1.txt* has only the number 60 in it). The *infile* is the horn.wav sound used in the *GrainMill* Tutorial (2.9 sec). The 'b' example has time varying packing, as defined in *packchng.brk*: 0 0.025
3 0.1
6 0.05
9 0.25
12 0.025

SIMPLE Examples 1a/b (Presets/Patches **simplex1a** and **simplex1b**):
outdur (12) packing (0.25) scatter (0) tgrid (0) sndfirst (1) sndlast (1) mingain (36) maxgain (84) mindur (0.2) maxdur (1.5) minpich (60) maxpich (60) – in command line format:
texture simple 5 horn **simplex1a** ndfsim1.txt 12 0.25 0 0 1 1 36 84 0.2 1.5 60 60
texture simple 5 horn **simplex1b** ndfsim1.txt 12 packchng.txt 0 0 1 1 36 84 0.2 1.5 60 60

You will hear a regularly occuring randomised selection of mostly overlapping note events, some longer and some shorter, repeating the pitch Middle C (randomly) in both channels – note the Left-Right movement between speakers. Try making *maxpich* 61, and then 64. With *maxpich* still at 64, try reducing the *packing* to 0.025.

All the examples for TEXTURE SIMPLE can be made at once by running *simplexs.bat* from the DOS prompt. The soundfiles produced can be deleted with *simpldel.bat*.

Now let's start to look more closely at the salient parameters:

- *packing* determines the density of the texture by setting the time onsets of the note events. In the above example, 0.25 sec sets 4 events per second, i.e., semiquavers at crotchet (quarter note) = 60). These events will occur evenly, because both *scatter* and *tgrid* are set to zero.
- *mindur* & *maxdur* set the range within which the duration of each note event will be randomly chosen. If only one duration is wanted, set both of these to the same value.
- *minpich* & *maxpich* set the range within which the pitch of each note event (in MIDI note values) will be randomly chosen. If only one pitch is wanted, set both of these to the same value. Given a pitch range greater than just one pitch, the program selects pitches according to a random function. **NB:** The specification of MIDI pitch values is not limited to integers. Thus a pitch range of 60 60.75 will result in microtonal detuning.

OK, now let's note and experiment carefully with the two additional timing parameters: *scatter* and *tgrid*.

•The onset timing can be somewhat randomised by utilising the *scatter* parameter. For example, if in the above example *scatter* is set to 0.07 and *tgrid* is still 0, you will hear how the previous evenness is somewhat displaced.
•If you then set *tgrid* to 330ms (*scatter* still at 0.07), a quantisation grid of about a 3rd of a second is set. In a 'snap to grid' fashion, this will draw the still somewhat randomised note events towards these 3rd of a second 'grid lines'. Because quantisation sets a **time grid**, when there are more events, they will tend to bunch at the grid points, some virtually simultaneous such that they sound like one event.
•If *scatter* is then set to 0 (with *packing* still at 0.25 and *tgrid* at 330, i.e., if the onset randomisation factor is eliminated, the note events will much more regularly occur on/near the 3rdof a second time grid points.

The following table summarises some basic combinations:

TEXTURE SIMPLE Mode 5 Examples

Running some changes on the initial example				
outdur	***packing***	***scatter***	***tgrid***	**Result**
12	0.25	0	0	48 events will occur regularly on the ¼ second
12	0.25	0.07	0	48 event onsets will be displaced, becoming a little uneven rhythmically
12	0.25	0.07	330	48 events somewhat irregularly placed will gather at the 3rd of a second quantisation time grid points – but some will be doubled.
12	0.25	0	330	48 events will gather at the the 3rd of a second quantisation time grid points
Some other generic illustrations				
outdur	***packing***	***scatter***	***tgrid***	**Result**
12	1	0	500	12 events will occur on the second: quantisation is 'pulled' to the 1 second 'boundary'
12	1	1	500	12 events on the second or half-second: e.g., 1, 2, 2.5, 4 etc.
12	1	0	2000	6 double events every 2 seconds: *packing* demands 12 events: *tgrid* forces them to a 2 sec grid
12	0.5	0	330	24 events will occur. They will be shifted to the 3rd of a second grid.
12	0.1	0	330	The many events will group at the 3rd of a second grid.
12	0.1	0	150	The slighter quantisation here will reduce the bunching of the previous example

You are recommended to experiment with TEXTURE SIMPLE Mode **5** for a while to see how easy it is to create random textures of varying density, and then to get a firm grasp of how *packing*, *scatter* and *tgrid* affect each other. This will provide you with a firm foundation with which to make full use of the more complex facilities which the TEXTURE Group provides.

Harmonic fields or sets – handling the pitches
TEXTURE SIMPLE can also deal with harmonic fields or sets. **Fields** transpose the specified pitch set into all (MIDI) octaves. **Sets** use only the specified pitches. These form pitch lattices onto which the various note

events are placed. The first thing to understand here is how the pitches specified in the command parameters relate to those specified in the note data file.

But first, try this example of a fixed pitch set (Mode **3**). The note data file *ndfsim2.txt* has two pitches, both starting at time 0: 60 and 67.

```
60
#2
0 1 60 0 0
0 1 67 0 0
```

You will hear, due to the rapid repetitions, a textured interval of a perfect 5th.

SIMPLE Example 2 (Preset/Patch **simplex2**):
outdur (12) packing (0.025) scatter (0) tgrid (0) sndfirst (1) sndlast (1) mingain (36) maxgain (84) mindur (0.2) maxdur (1.5) minpich (60) maxpich (67) – in command line format:
texture simple 3 horn **simplex2** ndfsim2.txt 12 0.025 0 0 1 1 36 84 0.2 1.5 60 67

- **The rule of thumb is that the program first generates the MIDI note values within the range specified by the parameters, and *then* looks at the pitches listed in your note data file and adjusts the former to fit onto the pitch grid specified by the latter.** (Reminder: The 'pitch parameters' referred to are those entered on the command line or with a graphic interface. The note data file is a text file prepared beforehand.)
- Thus the range of pitch values specified with the *minpich* and *maxpich* parameters is very important:
  - If the parameter pitch range is the same as that in the note data file, all notes will lie within the pitch range.
  - If the parameter pitch range is only one pitch, only one pitch will be produced, regardless of whether or not there are different pitches in the note data file.
  - If the parameter pitch range is greater than that of the range of the list of notes in the note data file, in Mode **1** pitches both inside *and outside* the note data file range will be produced, but keeping to the pitch grid, albeit in another octave. In Mode **3**, only the pitches in the note data file will sound, as any other pitches generated by the pitch parameters will be transposed to lie within the pitch set of the note data file.
  - If the parameter pitch range is within one octave and all the pitches in the note data file are the same, only one pitch will be produced.
  - If the parameter pitch range spans more than one octave and all the pitches in the note data file are the same, the one pitch will sound in different octave transpositions in Mode **1**(harmonic fields – all octaves), and only one pitch will sound in Mode **3** (harmonic sets – uses only the specified pitches).

- The above list of possibilities brings out the difference between Mode **1** and Mode **3**, namely that Mode **1** (Harmonic Fields) allows the pitch events listed in the note data file to sound in all octaves (within the parameter pitch range), whereas Mode **3** (Pitch Sets) forces all pitches to lie on the precise pitch grid specified in the note data file. However, **note that the parameter pitch range needs to be wide enough to accommodate the range in the note data file.** If, for example, the paramter pitch range is a perfect 5^{th} and the note data range is a 12^{th}, the note data pitches which lie outside the perfect 5^{th} will have nowhere to go, and will therefore sound inside the perfect 5^{th}.
- **Note that in TEXTURE SIMPLE, the velocity and the duration fields in the note data file are ignored – this information is taken from the command parameters. Zeros are placed in these fields in the examples in order to emphasise this point.**

Harmonic fields or sets – handling the durations
We have already seen how the timing of the events is random within constraints, with the density (and number of events) controlled by the *packing* parameter, while *scatter* can somewhat randomise the time onsets of these events.

In TEXTURE SIMPLE, the duration parameter in the note data file is inactive and may be zero (or any other value – it is ignored.) When this parameter is active, the interaction between the command and note event durations listed in the note data file works in a similar way as that for pitch.

- Thus, the parameter duration range (*mindur* and *maxdur*) needs to be wide enough to accommodate the shortest and longest duration listed in the note data file.
- If the parameter duration range is smaller than the range of durations in the note data file, the durations of the note events will be constrained within this smaller range.
- However, if the parameter duration range is wider than the duration range of the note events in the note data file, you will find that the note events will use this wider range.
- So with durations, it may usually be best to match the duration ranges of the parameter data and the note data file data.

Examples for harmonic sets and fields

The following example illustrates Mode **3**. The notelist uses the pitches 60-67-72-76 (i.e., C-G-C-E': root-fifth-octave-tenth). The parameter pitch range is set as 60 to 76 to encompass the range in the note data file. Because Mode **3** is for pitch sets, we will hear only the 4 pitches listed in the note data file, spread out in a randomised order over the specified output duration. Here is the note data file for this (*ndfsim3a.txt*):

```
60
#4
0 1 60 0 0
0 1 67 0 0
0 1 72 0 0
0 1 76 0 0
```

Ndfsim3b.txt is the same, except the times (first column) are 0 4 7 and 11. You can run these notedata files with the following command lines:

SIMPLE Examples 3a/b/c(Presets/Patches **simplex3a,b &c**):
outdur (12) packing (0.25) scatter (0) tgrid (0) sndfirst (1) sndlast (1) mingain (36) maxgain (84) mindur (0.2) maxdur (1.5) minpich (60) maxpich (76) – in command line format:
texture simple 3 horn **simplex3a** ndfsim3a.txt 12 0.25 0 0 1 1 36 84 0.2 1.5 60 76
texture simple 2 horn **simplex3b** ndfsim3b.txt 12 0.25 0 0 1 1 36 84 0.2 1.5 60 76
texture simple 2 horn **simplex3c** ndfsim3b.txt 12 0.25 0 0 1 1 36 84 0.2 1.5 36 96

Let's look at what the various parameter options chosen mean – reminder: the parameters after the notedata file are (EX 3a):
outdur (12) packing (0.25) scatter (0) tgrid (0) sndfirst (1) sndlast (1) mingain (64) maxgain (64) mindur (0.2) maxdur (1.5) minpich (60) maxpich (76)

- The duration of the output soundfile is 12 seconds
- The *packing* (i.e., density of – time between) – the note events is ¼ sec. If you give larger and smaller values for this parameter, you will see that the program will generate fewer or more note events. (Try a *packing* of 0.025.) No timings are given in the notelist: all are set at 0.
- *Scatter* is set at 0. This means that the note events will form regular divisions of the *outdur*. The start time of each event will be increasingly randomised as the value for *scatter* is increased – note the large range of 0 to 10 available for *scatter*. Try adding some scatter.
- *Tgrid* is also set to 0.
- The first and last sounds are both given as '1'. This means that there is only one input sound. If there are two to be used, you would put 1 2. The program reads your list of input soundfiles and assigns numbers in the order in which they occur in command

parameter for input soundfiles. If you have more than one *infile*, you need additional entries for the real or arbitrary pitch of each in the note data file (on the first line, separated by spaces).

•The gain parameters set a MIDI 'velocity' range, in this case all notes will have an equal mid-range amplitude.

•The *mindur maxdur* range is set as '0.2 1.5', to match the range in the note data file.

- •When harmonic fields or sets are used, the duration (sustain) parameter in the note data file has no effect and can be set to zero. **This means that in TEXTURE SIMPLE and TEXTURE GROUPED, the duration parameter in the note data file is not used in any of the 5 modes.** The note durations are taken from the parameter duration range, unless you flag it to be ignored (**-w** flag), whereupon it always uses the full duration of the input soundfile for each note events.
- •When decorations and ornaments are used (see the relevant programs), the durations in the note data file must be set and have precedence over the parameter duration range.
- •Also note that notelength is constrained by the duration of your input sound. TEXTURE **cannot** lengthen the input sound – it can only shorten it.
- •Also, as you transpose your sound up, it will get shorter (time-domain transposition), which also constrains how long it can be.
- •When the duration parameter range takes precedence over the durations specified in the notelist, be sure that the shortest is given as *mindur* and the longest as *maxdur* to ensure that full desired range is employed.

In the above example, also note that the 1.5 second durations are longer than the ¼ second *packing* parameter, so we hear the notes overlap in the output texture – a rather nice effect.

•The last parameter set used here specifies the pitch range. As noted above, this also affects what is specified in the notelist, so if you want the notelist pitches to be used, be sure that the lowest is given as *minpich* and the highest as *maxpich*.

•New possibilities emerge when the *time* for each note event is specified in the note data file. Try editing *ndatseta.txt* so that the four note events begin at 0.0, 4, 7 and 11 sec. respectively (save as ndatsetb.txt).

•Now run the same again in Mode **2**. You will hear pitches on the harmonic lattice being drawn in from different octaves.

You should now be able to experiment successfully with TEXTURE SIMPLE.

Two Fields/Sets

The next step is to use **two harmonic fields or sets**. *Ndfsim4a.txt* presents a revised *ndfsim3b.txt* for use with Modes **2** or **4** of the program: 'changing harmonic fields/sets'. There are 8 note events in two 'sets'. The first set starts at time 0, and in the second set, which starts at 6 sec., we change to the minor tenth (75). In the output file, we hear the harmony change from major to minor the second set at the 6 second mark. The 'b' example illustrates how a much tighter packing results in a textured chord.

```
[ndfsim4a.txt   ndfsim4b.txt]
60              60
#8              #8
0 1 60 0 0      0  1 60 0 0
0 1 67 0 0      0  1 67 0 0
0 1 72 0 0      0  1 72 0 0
0 1 76 0 0      0  1 76 0 0
6 1 60 0 0      6  1 60 0 0
6 1 67 0 0      8  1 67 0 0
6 1 72 0 0      10 1 72 0 0
6 1 75 0 0      10 1 75 0 0
```

SIMPLE Examples 4a/b/c (Presets/Patches **simplex4a,b, &c**):
outdur (12) packing (0.25) scatter (0) tgrid (0) sndfirst (1) sndlast (1) mingain (36) maxgain (84) mindur (0.2) maxdur (1.5) minpich (60) maxpich (76) – in command line format:
texture simple 4 horn **simplex4a** ndfsim4a.txt 12 0.25 0 0 1 1 36 84 0.2 1.5 60 76
texture simple 4 horn **simplex4b** ndfsim4a.txt 12 0.025 0 0 1 1 36 84 0.2 1.5 60 76
texture simple 4 horn **simplex4c** ndfsim4b.txt 12 0.025 0 0 1 1 36 84 0.2 1.5 60 76

The 'c' example tells us more about how the timing works by staggering the times of the 2nd four note events in *ndfsim4a.txt*: i.e., starting at times 6, 8, 10 and 10 seconds respectively, to create *ndfsim4b.txt*. To do this, we again need to invoke Mode **4** (changing harmonic sets). Now we hear the first four notes, the notes of the 1st set, playing in random order for the first 6 seconds. Then, at time 6 seconds, we hear mainly the 5th note event (with some overlaps from the previous set), at time 8 seconds, the 6th note event, and at time 10 seconds, we hear the 7th and 8th notes in rapid succession, sounding a minor third – except that we will hear multiple iterations of those notes according to the setting of the *packing* parameter. Again the close *packing* at 0.025 sec creates a highly textured surface.

Were we to use Mode **3** with *ndfsim4b.txt*, the different start times would be disregarded and all the note events would play in random order throughout.

The above information tells us that we can create a single melodic line by using Mode 4 and setting a different start time for a series of note events. Some overlapping will occur if the notes are of different durations. *Packing* of onsets closer than the duration of any of the note events will texture the lines, while *packing* of onsets spaced wider than the duration of any of the note events will produce gaps.

So try this 'melodic' example for TEXTURE SIMPLE. The note data file *ndfsim5.txt* contains:

```
60
#10
0.0  1 60 0 0
1.0  1 67 0 0
3.0  1 66 0 0
3.5  1 62 0 0
4.5  1 64 0 0
6.0  1 69 0 0
7.5  1 66 0 0
8.5  1 60 0 0
9.5  1 62 0 0
10.0 1 67 0 0
```

SIMPLE Example 5 (Preset/Patch **simplex5**):
outdur (12) packing (0.4) scatter (0.3) tgrid (0) sndfirst (1) sndlast (1) mingain (36) maxgain (84) mindur (0.2) maxdur (1.5) minpich (60) maxpich (69) – in command line format:
texture simple 4 horn **simplex5** ndfsim5.txt 12 0.4 0.3 0 1 1 36 84 0.2 1.5 60 69 -w

Note that the *packing* is set to be just under the shortest time between two note events, so the melodic notes come through without too much texturing, following the note sequence in *ndfsim5.txt*: C-G-F#-D-E-A-F#-C-D-G. The **-w** flag makes the note duration range redundant, because when this is set the whole duration of the *infile* is used – i.e., the whole length of the horn sound for each note of the *outfile*. In this case, this produces a great deal of overlapping notes – hence the 'resonance'.

Our final topic is **time-varying texture shapes** – i.e., envelope shapes with sloping edges. **Example 6** creates two of these with the files *gprlo.brk* and *gprhi.brk*. The first begins at C-60 and expands up to 70 and down to 49 at 5 seconds, returning to C-60 at 10 sec. Then the second shape starts at 10.01 seconds at 78 high and 58 low and compressing towards the centre G-67 by 18.0 sec. The last portion repeats the G. I've added time-varying packing to create more density where the texture shapes are widest. These are the files:

```
[ndfsim6.txt    simplpak.brk    gprlo.brk       gprhi.brk]
60              0     0.25      0     60        0     60
#22             5     0.1       5     49        5     70
0.0  1 49 0 0   10    0.3       10    60        10    60
0.0  1 52 0 0   10.01 0.05      10.01 58        10.01 78
```

```
0.0  1 54 0 0   18    0.15      18    67        18    67
0.0  1 55 0 0   20    0.25      20    67        20    67
0.0  1 58 0 0
0.0  1 60 0 0
0.0  1 61 0 0
0.0  1 64 0 0
0.0  1 66 0 0
0.0  1 67 0 0
0.0  1 70 0 0
10.0 1 58 0 0
10.0 1 60 0 0
10.0 1 61 0 0
10.0 1 64 0 0
10.0 1 66 0 0
10.0 1 67 0 0
10.0 1 70 0 0
10.0 1 72 0 0
10.0 1 73 0 0
10.0 1 76 0 0
10.0 1 78 0 0
```

And the parameters are set as follows:

SIMPLE Example 6 (Preset/Patch **simplex6**):
outdur (12) packing (simplpak.txt) scatter (0) tgrid (0) sndfirst (1) sndlast (1) mingain (36) maxgain (84) mindur (0.2) maxdur (1.5) minpich (gprlo.brk) maxpich (gprhi.brk) – in command line format:
texture simple 3 horn **simplex6** ndfsim6.txt 21 simplpak.brk 0 0 1 1 36 84 0.2 1.5 gprlo.brk gprhi.brk

Musical Applications

We can already see that this is an extremely powerful set of programs, which can undoubtedly be used in endlessly imaginative ways. For the moment, let us simply outline what we have learned so far. With the TEXTURE program group, we can create:

•a melodic line, with gaps and/or overlaps.

METHOD: use Mode **4**, specify specific start times for each note event in the *ndata* file, and set the *packing* to be about the same as the shortest time between notes. Note durations in the note data file can be zero (they are disregarded with harmonic fields or sets); note overlaps result from making the successive start times shorter than at least some of the note durations set with the parameter duration range. With the **-w** flag set, there may be even more overlapping if the sound's duration is longer than the*packing* setting, with possibilities for resonances and harmonies emerging from the melody.

•a melodic line, with a grainy, textured surface.

METHOD: use Mode **4** and specify specific start times in the *ndata* file and set *packing* to be very short, certainly much shorter than the time between note events. A very close *packing* (e.g., 0.01) may cause amplitude overload. You can apply the gain reduction recommended by the program, or perhaps reduce the *maxdur* value to lessen the amount of note overlap.

•a texture of randomly selected notes drawn from a defined list of notes – i.e., these can, for example, form a sonorous chord when the *packing* is tight.

METHOD: use Mode **3**, set all start times in the *ndata* file to 0 and specify pitches as desired. A *mindur* – *maxdur* range will produce note overlaps (unless *maxdur* is less than the time gap between events).

•a texture which moves between two harmonic fields (Mode **2** all octaves) or sets (Mode **4** only the pitches specified).

METHOD: in the *ndata* file, start all note events of the first set at one time, and all note events of the second set at another, later, time.

•a texture with envelope shapes, with pitches drawn from a defined harmonic set; time-varying packing can help make the shapes audible by changing the note density.

METHOD: define harmonic sets, which may differ at different times. Then create time-varying group pitch range files to define the envelope shapes.

CDP-X – more extreme forms of distortion, extension & scrambling

CDP Processing Functions to apply wavecycle DISTORTION to sounds (DISTORT)

INTERPOLATE	Timestretch file by repeating 'wavecycles' and interpolating between them
PITCH	Pitchwarp 'wavecycles' of sound
REPEAT	Timestretch file by repeating 'wavecycles'

DISTORT INTERPOLATE –Time-stretch file by repeating 'wavecycles' and interpolating between them

Usage

distort interpolate *infile outfile multiplier* [**-s***skipcycles*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
multiplier – (integer) number of times each 'wavecycle' repeats

multiplier may vary over time.

-s*skipcycles* – (integer) number of 'wavecycles' to skip at start of file

Understanding the DISTORT INTERPOLATE Process

With this process, the shape of a 'wavecycle' is transformed into that of the next over *multiplier* repetitions. Note that this is waveshape-based interpolation, not a spectral interpolation, and that the length of the 'wavecycle' is also transformed by the process.

The effect of the transformation is drastic, leading to a strongly granular *outfile*. The length of the *outfile* increases in step with the value of *multiplier*, as does the apparent pitchiness.

Musical Applications

The interpolation process adds a modulatory quality to the output, so that the successive wavecycles gliss and bend as they flow into one another. Even so, as *multiplier* increases, the perception of separate 'grains', i.e., 'wavecycles' increases. A value of 32, for example, changes the sound to a strange stream of modulating tones.

DISTORT PITCH – Pitchwarp 'wavecycles' of sound

Usage

distort pitch *infile outfile octvary* [**-c***cyclelen*] [**-s***skipcycles*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
octvary – maximum possible transposition up or down in (fractions of) octaves (Range > 0.0 to 8.0)

> Note that the pitch of **each** 'wavecycle' is varied by a random amount within the range of *octvary* octaves up to *octvary* octaves down: i.e., the value for *octvary* covers a total up/down range of 2 * *octvary*.

-c*cyclelen* – mamimum number of 'wavecycles' between the generation of transposition values (Range: > 1, Default: 64)

> *octvary* and *cyclelen* may vary over time.

-s*skipcycles* – (integer) number of 'wavecycles' to skip at start of file

Understanding the DISTORT PITCH Process

The random up/down movement of the 'wavecycles' within the total *octvary* range produces a great deal of bending of the sound, especially if the original alters its pitch a good deal. It is better, therefore, to start with relatively small values for *octvary* – e.g., less than 1 – so that you start to use this function with some degree of control over the results.

> The full power of DISTORT PITCH doesn't really come into its own until time-varying parameters are used, especially for *cyclecnt*. Large values for the latter will serve to slow down the rate of change.

Musical Applications

DISTORT PITCH is useful for creating 'flexitones' (to coin a term) – with distortion, of course.

DISTORT REPEAT – Timestretch soundfile by repeating 'wavecycles'

Usage

distort repeat *infile outfile multiplier* [**-c***cyclecnt*] [**-s***skipcycles*]

Example command line:

distort repeat infile outfile 5 -c3 -s20

Parameters

infile – input soundfile to process (mono only)
outfile – soundfile output after processing
multiplier – number of times (integer) each 'wavecycle' (group) repeats
-c*cyclecnt* – number of 'wavecycles' (integer) in repeated groups
-s*skipcycles* – number of 'wavecyles' (integer) to skip at start of file

multiplier and *cyclecnt* may vary over time.

Understanding the DISTORT REPEAT Process

The repetition of the 'wavecycles' stretches out the sound, making it both longer and more granular in texture. This granularity is increased if (increasingly larger) groups of *cyclecnt* 'wavecycles' are used: then the whole group repeats *multiplier* times.

Musical Applications

DISTORT REPEAT produces long, grainy (distorted) sounds. The sense of stretching out the original is very apparent.

A significant application of DISTORT REPEAT is that, by increasing the *cyclecnt* factor, one crosses the pitch-perception boundary: that is, starting with a noisy sound in which all the wavecycles are randomly different, one ends up with, for example, 7 repetitions of the **same** wavecycle, followed by 7 of another and so on – and each of these comprise sufficient repetitions for us to hear **pitch**. Thus the noise source becomes a string of pitch beads, each of arbitrary timbre. With a *cyclecnt* of, for example, 128, one can even get a slowish random melody.

CDP Processing Functions to alter the disposition of GRAINS of sounds (GRAIN)

TIMEWARP Stretch (or shrink) the duration of a grainy sound, without stretching the grains themselves

GRAIN TIMEWARP – Stretch (or shrink) the duration of a grainy sound without stretching the grains themselves

Usage

grain timewarp *infile outfile timestretch_ratio* [**-b***len*] [**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Parameters

infile – input soundfile to process
outfile – output soundfile
timestretch_ratio – the degree of stretching or shrinking of the intergrain time

- *timestretch_ratio* may vary over time, using a data file containing *time timestretch_ratio* value pairs
- this ratio is a floating point multipler
- a value of 2 doubles the intergrain time
- a value of 0.5 halves the intergrain time
- the maximum value for a multiplier is 1000
- the minimum value for a multiplier is 0.001000

- If any intergrain time is reduced the minimum time allowed for a grain (0.032 sec), it will be reset to this minimum grain time.

-b*len* – maximum time between grains (Range: 1 to duration of *infile*)
-l*gate* – required signal level for grain to be seen (Range: 0 to 1; the Default is 0.3)

gate may vary over time.

-h*minhole* – minimum duration of holes between grains (Minimum allowed value is 0.032 – the Default)
-t*winsize* – gate level tracks the signal level, as found with a window size of *winsize* milliseconds (Range: 0.0 to duration of *infile*)

0.0 turns off tracking

-x – ignore the last grain in the source

Understanding the GRAIN TIMEWARP Process

Using a numerical value for *timestretch_ratio* applies a constant multiplier to all the intergrain times. Thus the grains come constantly faster or slower, depending on the multiplier.

In the process the onset times of the grains are timewarped, **but not the grains themselves**. This is akin, for example, to playing the same melody faster on the same instrument. It therefore differs from other timewarping processes, which also warp the internal architecture of the sounds themselves.

Use of a breakpoint file adds considerable flexibility to the process because the function will interpolate gradual changes between different ratios at different times. For example, moving from a multiplier of 1.0 to 2.0 over the time of the *infile* will produce a gradually decelerando, or from 1.0 to 0.5, a gradual accelerando. Similarly, speeds can be stepped by applying different ratios at marginally different times (it isn't logical to apply two different ratios at precisely the same time).

As an example of stepped changes, the following breakpoint file doubles the gap time for the first half of a 1.8 second *infile*, halves the (original) gap time for 0.6 seconds, returning to the longer gaps for the last part of the sound. Between these times, the timing of the grains remains constant.

```
[time timestretch_ratio]
0.0   2.0
0.9   2.0
1.01  0.5
1.60  0.5
1.61  2.0
```

Musical Applications

This is an important tool for creating all kinds of rhythmic patterns.

CDP-UTILS-1 – CDP Time-Domain Editing Functions

ENVEL

CDP Utilities to manipulate amplitude ENVELOPES (ENVEL)

WARP Warp the envelope of a soundfile

ENVEL WARP – Warp the envelope of a soundfile

Usage

envel warp 1–12 *sndfile outsndfile wsize various_params*
envel warp 13 *sndfile outsndfile rampfile wsize various_params*
envel warp 14–15 *sndfile outsndfile wsize various_params*

wisize – the duration of the enveloping window, in milliseconds

Modes

The 15 Modes of ENVEL RESHAPE, REPLOT & WARP

Modes 1 – 4	Modes 5 – 8	Modes 9 – 12	Modes 13 – 15
1 NORMALISE	**5** LIFT	**9** INVERT	**13** TRIGGER
2 REVERSE	**6** TIMESTRETCH	**10** LIMIT	**14** CEILING
3 EXAGGERATE	**7** FLATTEN	**11** CORRUGATE	**15** DUCKED
4 ATTENUATE	**8** GATE	**12** EXPAND	

The Modes and their Parameters

1 NORMALISE

Expand envelope so that the highest envelope point is at the maximum possible amplitude value.

No other extra parameters are needed.

2 REVERSE

Reverse the envelope in time.

No other extra parameters are needed.

3 EXAGGERATE

Exaggerate the envelope contour. One other parameter:

exaggerate – Range: > 0.0
< 1, low values are boosted

= 1, no change
> 1, high values are boosted
exaggerate may vary over time

4 ATTENUATE

Reduce the amplitude of an envelope. One other parameter:

attenuation – multiplier by which to scale down the envelope. Range: 0 to 1
attenuation may vary over time

5 LIFT

Raise the envelope by a fixed amount. One other parameter:

lift – amount to add to each amplitude value. Range: 0 to 1
lift may vary over time

6 TIMESTRETCH

Stretch or compress an envelope in time. One other parameter:

timestretch – multiplier by which to stretch the envelope. Range: > 0.0
< 1.0 compresses the envelope in time
= 1.0 no change
> 1.0 stretches the envelope in time

7 FLATTEN

Even out the envelope contour. One other parameter:

flatten – number of envelope values over which to average. Range: 1 to 5000.
flatten may vary over time
Note that there is an envelope value for every segment of sound, and that this may be considerably more widely spaced than, for example, the control points specified by *ksmps* in a Csound orchestra (often set to 15 samples).

8 GATE

Zero lower portion of envelope. Two other parameters:

gate – levels less than *gate* are set to 0. Range: 0 to 1.
smoothing – excises segments with an amplitude less than *smoothing*. Range: 0 to 32767.
smoothing < 0 turns off the smoothing effect.
gate may vary over time

9 INVERT

Turn envelope upside down: negative values become positive, and positive values become negative. Two other parameters:

gate – levels less than *gate* are set to 0. Range: 0 to < *mirror*.
mirror – reflection point: all values other than those below *gate*, both above and below *mirror*, are inverted to the other side of *mirror*. Range: > *gate* to 1.
gate and *mirror* may vary over time

10 LIMIT

Squeeze down amplitude into a specified range. Two other parameters:

limit – upper limit of range; amplitude values which were above *limit* are scaled down such that 'maxamp' becomes *limit*. Range: > *threshold* to 1.
threshold – lower limit of range; only values above *threshold* are affected. Range: 0 to *limit*.
limit and *threshold* may vary over time

11 CORRUGATE

Tighten the peaks into close-set ridges by taking troughs (downward dips) in the envelope to zero. Two other parameters:

trofdel – number of windows to set to zero per trough. Range: 1 to < *peak_separation*.
peak_separation – minimum number of windows per peak. Range: 2 to 32767.
trofdel and *peak_separation* may vary over time

12 EXPAND

Inflate the envelope. Three other parameters:

gate – levels below *gate* are set to 0. Range: 0 to *threshold*.
threshold – lower limit of range: amplitude values that were above *gate* and below *threshold* are scaled up such that the minimum level becomes *threshold*. Range: *gate* to 1.
smoothing – excises low-level segments which are less than *smoothing* windows in length. Range: 0 to 32767. 0 turns off the smoothing effect.
gate and *threshold* may vary over time

13 TRIGGER

Create a new envelope of sudden **on** bursts, triggered by the rate of rise of the current envelope. Four other parameters:

rampfile – your new breakpoint file for the (triggered) bursts
gate – the average level must be above *gate* before triggering can occur. Range: 0 to 1.
rise – minimum loudness-step before triggering can occur. Range: 0 to 1.
dur – maximum duration over which the *rise* can take place before triggering can occur. Range: must be >= the envelope window duration.
gate may vary over time

14 CEILING

Force envelope up to its maximum level, **everywhere**. No other extra parameters needed.

15 DUCKED

Create 'ducking' envelope: envelope kept below a set level. Two other parameters are needed:

gate – highest level allowed for the envelope. Range: 0 to 1.
threshold – amplitude level the envelope must reach before 'ducking' is applied. Range: 0 to 1.
gate and *threshold* may vary over time

- **With envelope Warping:** When the input envelope level exceeds *threshold*, the output envelope is reduced to *gate* level. Elsewhere it is unchanged.
- **With envelope Reshaping or Replotting:** When the input envelope level exceeds *threshold*, the output envelope takes *gate* level. Elsewhere, it gives unity gain (no change).

- This function can be used to create an envelope to apply to another sound which can then be mixed with the sound from which the original envelope was extracted.

Understanding the ENVEL RESHAPE, REPLOT & WARP Processes

The envelope shaping routines for these three functions are identical. Only the output to which the changes are applied differs.

Movement in time creates shapes. Amplitude change plays a key role in performance articulation and in the identification of sounds. When we hear the amplitude 'grow' as a soprano sings a glorious long tone, we are aware of its amplitude 'envelope'. When we hear the sharp 'report' of a snare drum rimshot, we are aware of the 'attack transient' of the sound: i.e., its very rapid rise in amplitude. Voices rise and fall in both pitch and amplitude.

ENVEL WARP is a way of working with the shape produced by the envelope over the course of its whole duration. If we think of it as made of rubber, we can imagine it being pulled and stretched and squeezed.

Musical Applications

Amplitude envelope is an essential part of musical 'articulation' and has a great bearing on emotional content. These functions can be used to increase contrasts and make the music 'vivid', or to reduce the contrasts and 'tone down' the emotional effect.

On the other hand, the amplitude shape can be handled as an abstract form which can be extracted from a sound, remoulded in some way, and then applied to another sound or to some other musical data. This introduces a point of connection between different parts of the musical material. For example, we might hear an amplitude shape in the form of a series of vocal bursts. Then this same/similar shape may become audible in the form of a pitch transient (pitch glissando). The musical principle is 'sameness and difference': creating unity amidst disparate material by introducing elements of sameness.

Last Updated 17 Mar. 2018
Documentation: Archer Endrich Revisions: Robert Fraser
Quick documentation for CDP Plugin for Reaper – 27 Jul. 2018: Walter Cesarini
All observations & ideas for improvement appreciated
Composers Desktop Project Ltd

Email: cdpse9@gmail.com