CDP Composers' Desktop Project

CDP GRAIN Functions

(with Command Line Usage)

Granular Functions for soundfiles

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

ALIGN
Synchronise grain onsets in 2nd grainy sound with those in the 1st
ASSESS
Estimate best gate value for grain extraction
COUNT
Count grains found in a sound (at given *gate* and *minhole* values)
DUPLICATE
Duplicate grains in a grainy sound
[GRAINEX]
Find grains in a sound and extend the area that contains them
GREV
Find and manipulate 'grains', using envelope troughs and zero-crossings
[NEWTEX]
Generate a texture of grains made from a source sound or sounds
NOISE_EXTEND
Find and time-stretch noise components in a sound
FIND
Locate timings of grain-onsets in a grainy sound
OMIT
Omit a proportion of grains from a grainy sound
REMOTIF
Change pitch and rhythm of grains in a grainy sound
REORDER
Reorder grains in a grainy sound
REPITCH
Repitch grains in a grainy sound
REPOSITION
Reposition grain onsets in a grainy sound
RERHYTHM
Change rhythm of grains in a grainy sound
REVERSE
Reverse order of grains in a grainy sound without reversing the grains themselves

R_EXTEND
'Time-stretch' natural sounds like the rolled 'rrr' in speech
TIMEWARP
Stretch (or shrink) the duration of a grainy sound without stretching the grains themselves
[WRAPPAGE]
Granular reconstitution of one or more soundfiles over multi-channel space

SEE:

MODIFY BRASSAGE

Granular reconstitution of a soundfile

MODIFY SAUSAGE

Granular reconstitution of several soundfiles scrambled together

GRAINMILL

Graphic program for BRASSAGE.

Note that a sound can be quickly granulated with MODIFY BRASSAGE Mode 5:

```
   modify brassage 5 infile outfile density
```

Use a *density* value of 1.0 or less, which will introduce gaps between the grains.

GRAIN ALIGN – Synchronise grain onsets in 2nd grainy sound with those in the 1st

Usage

grain align *infile1 infile2 outfile offset gate2* [**-b***len*][**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Parameters

infile1 – 1st input soundfile: provides grain onset times
infile2 – 2nd input soundfile: provides the actual grains to be re-timed
outfile – output soundfile
offset – add this value to **all** grain timings
gate2 – minimum signal level to register grain in *infile2* (Range: 0 to 1)
-b*len* – maximum time between grains (Range: 1 to duration of *infile*)
-l*gate* – minimum signal level to register grain in *infile1* (Range: 0 to 1, Default 0.3)
-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 *infile1*)

0.0 turns off tracking

-x – ignore the last grain in the source

gate and *gate2* may vary over time.

Understanding the GRAIN ALIGN Process

The timing of the grains of *infile2* is made to match those of *infile1*.

It may be unwise not to specify a value for the optional parameter **-l***gate* (gate level for *infile1*).

Musical Applications

If you achieve an effective grain timing for two sounds, GRAIN ALIGN makes it possible to create a subtle correspondence between the two soundfiles.

For example, the fine attack-structure of the two sounds can be synchronised, so that they appear to pulse in parallel.

End of GRAIN ALIGN

GRAIN ASSESS – Estimate best gate value for grain extraction

Usage

grain assess *infile*

Parameters

infile – input soundfile in which to estimate a useful gate value

Understanding the GRAIN ASSESS Function

GRAIN ASSESS is a simple utility to provide useful information when using the GRAIN set of programs. You just provide an input soundfile and it scans the file, displaying a report such as:

```
Maximum grains found = 21 at gate value 0.117100 and windowlen 50ms
```

The program is therefore assessing which *gate* value and which *windowlength* value will result in the most grains. You can then use this information in other GRAIN programs.

Musical Applications

For example, even as simple a program as GRAIN COUNT gives you the option to specify a *gate* value. GRAIN ASSESS provides the value that will produce the most grains, and you can take it from there. *Windowlength* also affects the number of grains found. If it is too long, it may count several grains as one, or if too short, it may not find some of the grains.

End of GRAIN ASSESS

GRAIN COUNT – Count grains found in a sound (at given *gate* and *minhole* values)

Usage

grain count *infile* [**-b***len*][**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Parameters

infile – input soundfile in which to count the grains
-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 COUNT Function

GRAIN COUNT displays the message: "*N* grains found at this gate level."

Musical Applications

GRAIN COUNT provides a check on the granularity of a file, and also on the gate level at which a given number of grains is operational.

End of GRAIN COUNT

GRAIN DUPLICATE – Duplicate grains in a grainy sound

Usage

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

Parameters

infile – input soundfile
N – number of repetitions of each grain
-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)

N and *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 DUPLICATE Process

This process makes *N* copies of each grain, proceding through the *infile* grain by grain.

You are recommended to use GRAIN COUNT with the *gate* flag to determine which gate level works best with the file (produces a good granulation of the source). This *gate* level should be used with GRAIN DUPLICATE. Not using it appears to produce anomalous results, such as no grain duplications and a long period of silence.

Musical Applications

The overall effect is that of rapid-fire repeats or a stuttering effect, depending on the source.

End of GRAIN DUPLICATE

GRAIN FIND – Locate timings of grain-onsets in a grainy sound

Usage

grain find *infile out-textfile* [**-b***len*] [**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Parameters

infile – input soundfile in which to find the grain-onsets
out-textfile – output textfile containing a list of times in seconds at which each grain begins
-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 FIND Function

This function locates and writes to a text file the time in seconds at which each grain begins. Again, the effective *gate* level determined with GRAIN COUNT should be used.

Musical Applications

Examination of this text file provides a way to check on the results of the various GRAIN functions which affect the grain timings. The data can be used to retime the grains in another grainy sound. See GRAIN REPOSITION.

End of GRAIN FIND

GRAINEX EXTEND – Find grains in a sound, and extend the area that surrounds them

Usage

grainex extend *infile outfile wsiz trof plus stt end*

Example command line to find and extend grain regions:

```
grainex extend in.wav out.wav 30 0.5 2.5 1.5 3.5
OR:

                          WSIZ TROF PLUS STT END
grainex extend inf outf   5    0.5  5    0   1
```

Parameters

infile – input soundfile in which to find the grains
outfile – output soundfile
wsiz – size of window in milliseconds, which determines the size of the grains to find (Range: 1.81406 to file-length ÷ 3ms)
trof – acceptable trough depth, relative to adjacent peaks (Range: > 0 to < 1 – *Soundshaper* default: 0.5 – see Note below)
(? *gpcnt* – number of grains to treat as a unit in the operations)
plus – how much duration to add to the source (Range: 0.000002 to 3600 secs)
stt – time of start of grain material within the source (Range: 0.0 to file-length in seconds)
end – time of end of grain material within the source (Range: 0 to file-length in seconds and > *stt*)

Understanding the GRAINEX EXTEND Process

For GRAINEX, as with the other GRAIN functions, you will need a reasonably grainy sound as an input, otherwise it won't be able to find any grains and you will probably see the message "Insufficient valid troughs in the file".

GRAINEX extends an area containing grains, the start and end times being set by the user. The grains are found by envelope troughs and zero-crossings.

When the above example command line was run with the input file *count.wav*, when the output sound got to the number 'three', it was repeated several times before continuing on with the rest of the count up to the number ten. The input sound was 8.066 seconds long, and the output was 18.866 seconds long, i.e., considerably more than the 2.5 seconds specified.

NOTE
It is possible to set *trof* too low. Being relative to the peak level, if the signal doesn't drop that far in level, you may get the Error Message: "Insufficient grains to proceed" or "INSUFFICIENT PEAKS IN THE FILE AREA SPECIFIED".

End of GRAINEX EXTEND

GRAIN GREV – Find and manipulate 'grains', using envelope troughs and zero-crossings

Usage

grain grev 1 *infile outfile wsiz trof gpcnt*
grain grev 2 *infile outfile wsiz trof gpcnt repets*
grain grev 3-4 *infile outfile wsiz trof gpcnt keep outof*
grain grev 5 *infile outfile wsiz trof gpcnt tstretch*
grain grev 6 *infile out_timesfile wsiz trof gpcnt*
grain grev 7 *infile outfile in_timesfile wsiz trof gpcnt*

Example command line:

```
grain grev 2 infile outfile 10 0.5 4 3
```

Modes

1 REVERSE – the grain-units isolated are played in reverse order
2 REPEAT – *gpcnt*-sized units are repeated
3 DELETE – remove the specified number of units
4 OMIT – replace specified number of units with silence
5 TIMESTRETCH – the time expansion is happening in the troughs: the grains of sound are not themselves stretched
6 GET – writes grain time-positions to a textfile
7 PUT – places grains in time according to textfile

Parameters

infile – input soundfile in which to count the grains
outfile – output soundfile produced by the program
out_timesfile – textfile produced by the program, containing times of grain positions
in_timesfile – textfile containg times at which the program is to place grains
wsiz – windowsize in milliseconds, determining the size of the grains to find
trof – the acceptable trough height, relative to adjacent peaks. Range: greater than 0 to less than 1 (> 0 Range < 1)
gpcnt – ('groupcount') the number of grains to treat as a unit in the operations
repets – the number of repetitions of each unit
keep – *outof* – the number of units to keep, e.g., 3 out of 5
tstretch – the amount to timestretch the output (grains NOT stretched) Range: 0.01 to 100)

gpcnt, *repets*, *keep* and *tstretch* can vary over time.

Understanding the GRAIN GREV Process

This program was written to assist in separating the large-scale 'grains' in a stream of sound, e.g. the syllables of speech. It can be a useful alternative to SFEDIT SYLLABLES, which is for the precise manual editing of syllabic units.

The other 'grain' programs rely on a gate which searches for moments when the input sound falls below a certain level, and then splices the sound at those points. This works fine when the sound grains are fairly consistent (e.g. a sequence of pizzicato sounds).

However, the syllables of speech are usually quite different from one another. So the gating procedure finds some good places to cut, and misses others. The 'grev' process does not depend on setting a gate level. It simply looks for troughs (low points) in the envelope (at a suitable timescale), then within those troughs, searches for a zero-crossing at the lowest point of the signal in the trough.

Once it finds these it can separate the grains with zero-length splices. The key thing is to set the envelope window about three times smaller than the features (e.g. syllables) you want to tease out.

Musical Applications

Designed for spoken words, this program enables you to play with syllabic units with remarkable efficiency. Although similar to others in the GRAIN set, it gives direct control over the syllabic content of speech.

Mode **2** reveals how this program works. The example command line given above has these parameters: *wsiz* = 10 ms, *trof* = 0.5 (half-way can be a useful place to start with a parameter), *gpcnt* = 4 (four grains to a group), and *repets* = 3 (repeat the 4-grain-units 3 times). When run, we find that the syllables of the spoken text are, for the most part, **clearly isolated** and repeat 3 times each without overlaps. Thus the program is working effectively with the spaces between the syllables.

That this is the case is also shown by Mode **5**. When run with the same parameter values (the last parameter being *tstretch* instead of *repets*, we find that the syllables do not repeat but are more spaced out in time.

Modes **3** and **4** operate as expected, deleting grains or replacing them with silence. Mode **1** plays back the isolated grains in reverse order, i.e., you hear syllabic fragments from the end of the file first and proceed back to the beginning – but the text fragments themselves are 'forward' as normal. Higher values for *wsiz* and *gpcnt* result in longer passages of the text treated as a unit, while smaller values fragment the text more. It's a very odd effect.

Mode **6** provides a way to write grain-position times (shaped by the other parameters) to a file. This file can then be used as input in Mode **7** with, e.g., another input sound or with different values for the other parameters, thus achieving results that are partly shaped and partly serendipitous.

Altogether, GRAIN GREV is a powerful tool with which to manipulate speech material. On a deeper level, it can be used to musicalise speech and make formal connections with other shapes in your composition.

End of GRAIN GREV

NEWTEX – Create a texture of grains made from a source or sounds

Usage

newtex newtex 1 *insndfile outsndfile transposes dur chans maxrange step spacetype* [**-s***splice*] [**-n***number*] [**-x**]
 EXTRA OPTIONAL FLAGS (if *spacetype*>0):
 [**-r***rotspeed*] [**-j**] [**-e***from* **-E***time*] [**-c***to* **-C***time*]

newtex newtex 2 *insndfile1 insndfile2 [insndfile3 ...] outsndfile dur chans maxrange step spacetype delay* [**-s***splice*] [**-n***number*] [**-x**]
 EXTRA OPTIONAL FLAGS (if *spacetype*>0):
 [**-r***rotspeed*] [**-j**] [**-e***from* **-E***time*] [**-c***to* **-C***time*]

newtex newtex 3 *insndfile1 [insndfile2 ...] outsndfile transposes dur chans maxrange step spacetype* [**-s***splice*] [**-n***number*] [**-x**]
 EXTRA OPTIONAL FLAGS (if *spacetype*>0):
 [**-r***rotspeed*] [**-j**] [**-e***from* **-E***time*] [**-c***to* **-C***time*]

Example command line to create a combo grain texture :

```
newtex newtex 3 insndfile.wav outsndfile.wav 30 4 3 0.1 0 0 1.5 0.35 -s36
```

i.e.:

```
DUR  CHANS  MAXRANGE  STEP  SPACETYPE  LOC  AMB  GSTEP SPLICE
30   4      3         0.1   0          0    1.5  0.35    36
```

This example takes short segments and scatters them randomly across the channels. Splice is quite long to give the segments a softer attack. It works well in stereo too, but 4 or more channels are worth generating to get a variety of segmentations. (The channels can be split, possibly processed and then re-combined or re-mixed in various ways.)

Modes

1 The transpositions of *insndfile* are spread over N octaves and spatially, and fade in and out randomly.
2 *Insndfile* is read at its original rate (i.e., no transpositions), spread spatially, and fades in and out randomly.
3 *Insndfile* is read as 'drunken walks', spread spatially, and fades in and out randomly.

Parameters

insndfile(s) – input soundfile or soundfiles, depending on which Mode is used
outsndfile – the output textured soundfile

transposes – textfile containing a list of transposition **ratios** and relative levels, against time.

- The data is a text file of lines of data.
- Every line must have the same number of entries.
- The first entry on each line is a time.
- Times must start at zero and increase.
- **All EVEN-numbered entries** are transposition levels.
- Transpositions must increase from entry to entry.
- **All other ODD-numbered entries** are transposition *levels*.
- Levels should have values between **-1** and **1**.
- Octave (transposition) values invert the phase of the source.

dur – the duration of the output sound
chans – the number of output channels
maxrange – In **Mode 1** *maxrange* is the range in octaves of the transpositions of the input.
In **Mode 2** *maxrange* is the number of simultaneous soundings of any source soundfile.
step – the average time between changes to the stream-content of the output.
splice – the splice-lengths in milliseconds for component entry and exit
number – the number of components chosen for each event
delay – the time delay between identical components
loc – the *locus* (time-location) from which to read sound segments.
NB: It is an important feature that *loc* is time-varying.
amb – the 'ambitus', i.e., the restricted time-range in milliseconds around the *locus* where reads can begin.
NB: It is an important feature that *amb* is time-varying.
- TIP: Compare *locus* and *ambitus* with the parameters of the same name in **EXTEND DRUNK**.

gstep – the maximum size of the random steps between one read start and the next read.
-x – 'Xclusive': change all components, as far as possible, from event to event
-j – 'Jump': all components are assigned to the same location for any one event, and then it jumps to the next location
spacetype – the type of output spatialisation:
0 Spatial position set at random.
For 8-channel output only (assumes an 8-speaker 'ring' formation):

1. Positions alternate between Left and Right sides, but are otherwise random.
2. Positions alternate between Front and Back, but are otherwise random.
3. Rotating clockwise or anticlockwise.
4. Random permutations of all 8 channels.
5. As [4] plus all possible pairs of channels.
6. As [4] plus all possible meaningful small and large triangles.
7. As [4] plus square, diamond and all-at-once.
   In types 4 to 7, all members of the permutation are used before the next permutation starts.
8. Alternate between all-left and all-right.
9. Alternate between all-front and all-back.
10. Alternate between all-square and all-diamond.
11. Rotate triangle formed by loudspeakers 2-apart clockwise.
12. Rotate triangle formed by loudspeakers 3-apart clockwise.
13. Rotate triangle formed by loudspeakers 2-apart anticlockwise.
14. Rotate triangle formed by loudspeakers 3-apart anticlockwise.

EXTRA OPTIONAL FLAGS (only if *spacetype*>0):
rotspeed – rotation speed for certain spatialisation types
-e*from* **-E***time*– 'Emerge': the sound emerges from channel *from* over time *time* at start
-c*from* **-C***time*– 'Converge': the sound converges from channel *to* over time *time* at end

- Flags with NO parameters must be placed AFTER any flags WITH parameters, on the command line.
- *maxrange*, *step*, *number*, *loc*, *amb* and *gstep*, can vary over time.

Understanding the NEWTEX Process

NEWTEX is a powerhouse of a program that combines aspects of MODIFY BRASSAGE, EXTEND DRUNK and TEXTURE. It generates a (time-varying) texture from segments cut from a source sound or sounds.

End of NEWTEX NEWTEX

GRAIN NOISE_EXTEND – Find and timestretch noise component in a sound

Usage

grain noise_extend *infile outfile duration minfrq mindur maxdur* [**-x**]

Parameters

infile – input soundfile in which to count the grains
outfile – output soundfile produced by the program
duration – duration of the noise part of the output soundfile
minfrq – the lowest 'frequency' in Hz acceptable as noise. Range: 1000 to 22050Hz. (Try 6000Hz)
mindur – the minimum duration of signal in milliseconds acceptable as noise source. Range: 0 to 50 milliseconds
maxdur – the maximum duration of signal in seconds acceptable as noise source. Range: 0.0 to length of input soundfile (in seconds)
-x – keep only the extended noise. The Default is to keep the rest of the input source sound as well.

Understanding the GRAIN NOISE_EXTEND Process

This process is complementary to **GRAIN R_EXTEND**. It searches for sybillants in speech (or noise materials in any input sound) then allows them (and only them) to be sustained.

Note that different types of values are involved with the *mindur* and *maxdur* parameters. The minimum duration is given in milliseconds, but the maximum duration, because you're thinking in terms of the length of the soundfile, is given in seconds.

The **-x** option enables you to save only the extended noise component, discarding the rest of the file. When **-x** is not invoked, you hear the whole soundfile (up to the *maxdur* you specified) with that first (extended) noise component in its original position in the soundfile.

Musical Applications

This program acually looks for and extends (only) the first noise component that it finds at the given parameter specifications.

The extended noise component, especially if saved on its own via the **-x** option, can be a useful bit of source material for other sound transformations.

End of GRAIN NOISE_EXTEND

GRAIN OMIT – Omit a proportion of grains in a grainy sound

Usage

grain omit *infile outfile keep out-of* [**-b***len*] [**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Parameters

infile – input soundfile to process
outfile – output soundfile
keep – number of grains to keep from each set of *out-of* grains

keep may vary over time, but must not exceed *out-of*

out-of – *keep* grains retained from start of each set of *out-of* grains
-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 OMIT Process

The key here is the *out-of* parameter. This defines the size of the group which serves as the unit of operations: out of every unit, *keep* grains are retained. Thus, if 5 out of 10 are retained, 50% of the file is omitted, if 7 out of 10 are retained, 3 out of the 10 are omitted (30%go), if 4 out of 10 are retained, 6 out of the 10 are omitted (60% go).

What is significant about the *out-of* parameter is that its size affects the result, Thus, a large value for *out-of* with a smaller value for *keep* , e.g., 20 and 10, will break up the file much more than it would if *out-of* were smaller: e.g., 4 and 2 – even though the proportion of omitted material is the same (50%). This is because larger continuous chunks of the source (10 grains rather than 2 grains) are lost in the former case.

Musical Applications

GRAIN OMIT provides a useful way to contract sound material without altering the sonic substance (i.e. the grains are not themselves, time-contracted).

End of GRAIN OMIT

GRAIN REMOTIF – Change pitch and rhythm of grains in a grainy sound

Usage

grain remotif mode *infile outfile transpmultfile* [**-b***len*] [**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Modes

1 Transform each grain in turn, without repeating any grains: on reaching the end of the *transpmultfile* data, cycle back to its start.
2 Transform grain in each specified way before proceeding to the next grain.

Parameters

infile – input soundfile to process
outfile – output soundfile
transpmultfile – is a file containing *transposition time* multiplier pairs.

- transpositions are given as positive or negative semitone shifts.
- the maximum transposition is 4 octaves up or down
- time multipliers change the duration between one grain onset and the next grain onset.
- the maximum value for a multiplier is 1000
- the minimum value for a multiplier is 0.001000
- If any inter-grain 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 REMOTIF Process

Implementing this process is really very easy. The *transpmultfile* does not have to match the number of grains in the *infile*. You just need to design the pitch and time shapes into which you want the grains to flow. The pitch transpositions are placed in the left column as positive or negative semitones, and the time-multipliers are placed in the right column, > 1 to make the gap longer, < 1 to make it shorter.

Here is an example *transpmultfile* that rises, drops down below the original pitch level, and then rises back to the starting point, with the grains getting shorter and then longer again.

```
[transposition timing]
 (semitones)    (multiplier)
0              1.0
1              0.9
2              0.8
3              0.7
2              0.6
1              0.5
0              0.4
-1             0.5
-2             0.6
-3             0.7
-2             0.8
-1             0.9
0              1.0
```

The difference between the modes is dramatic.

- In Mode **1** a single line of the *transpmultfile* is applied to a single grain, the next line to the next grain etc., cycling around the textfile until reaching the last grain of *infile*. Any difference in duration between *infile* and *outfile* is due to the timing difference between the grains. You can make the shape occur once by having as many lines in the textfile as there are grains in the *infile*.

- In Mode **2** ALL of the lines of the *transpmultfile* are applied to each grain in turn. Thus every grain repeats for as many times as there are lines in the textfile, and one hears the whole shape defined in the textfile repeated as many times as there are grains in the *infile*. This makes the *outfile* much longer than the *infile*.

Be careful with Mode **2**, because an over-long textfile will create a very long *outfile* (and be very repetitive).

Musical Applications

GRAIN REMOTIF can be used to apply a gross shape to the grains of the *infile*, with controlled accelerandi or other timing changes. This shape can occur once, cycle around the textfile with one line per grain (Mode **1**), or the shape can occur once per grain: each grain repeats with transposition and timing differences as many times as there are lines in the textfile (Mode **2**).

End of GRAIN REMOTIF

GRAIN REORDER – Reorder grains in a grainy sound

Usage

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

Parameters

infile – input soundfile to process
outfile – output soundfile
code – a string such as **adb:c** indicating how grains are to be reordered.

The example means use grains 1 (**a**), 4 (**d**) and 2 (**b**) in sequence, then
begin this grain-jumping pattern again,
but start at grain 3 (**c**).

-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 REORDER Process

This is so much fun. Comparable with **DISTORT SHUFFLE**, GRAIN REORDER provides another way for controlled fragmentation. The illustration *code* used in the Usage causes a fair amount of fragmentation. But a *code* such as **abcdefg:b** steps through 7 grains in their normal sequence, then goes back to the 2nd grain and proceeds to the 8th etc., moving gradually through the whole *infile* in this way. The *code* **gfedcba:b** does the same thing, but places the grains in reverse order.

Musical Applications

So you can see that you can play with the shaping possibilities to your heart's content, possibly making subtle correlations between the shapes made by the grains and patterns made elsewhere in the music.

End of GRAIN REORDER

GRAIN REPITCH – Repitch grains in a grainy sound

Usage

grain repitch mode *infile outfile transpfile* [**-b***len*] [**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Modes

1 Repitch each grain in turn, without repeating any grains; on reaching the end of the transposition list, cycle back to its start.
2 Play grain at each transposed pitch, before proceeding to the next grain.

Parameters

infile – input soundfile to process
outfile – output soundfile
transpfile – a file listing transpositions given as positive or negative semitone shifts. The maximum transposition is 4 octaves up or down.
-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 REPITCH Process

This function is like GRAIN REMOTIF but only deals with transposition of the grains. Again, the transpositions are handled in terms of positive or negative numbers of semitones, so it is very simple to create the required *transpfile*. The transposition **pattern / contour** in the *transpfile* is transferred to the grains, such as narrow and wavy, or wide and jagged. This pattern can also serve as an important gestural or formal unifier in the composition.

The textfile can be written on a single line, with the values separated by spaces. Newlines are ignored. For example:

```
0 1 2 3 2 1 0 -1 -2 -3 -2 -1 0
```

There may be some slight change in duration, proceeding from the source to the *outfile*, due to the effect of transposition on the final grain. Otherwise the grain onset timings should remain exactly as in the source sound.

The role of the two Modes is the same as in GRAIN REMOTIF, either taking each grain in turn, or applying all the transpositions in the textfile to each grain, repeating each grain as many times as there are transpositions.

Be careful with Mode **2**, because an overly long textfile will create a very long *outfile* (and be very repetitive). However, the pattern is very clear, and a part of it may be especially interesting and able to be CUT and used elsewhere.

Musical Applications

An ideal tool for creating staccato pitch figurations with clear contour shapes or even gestural potential.

End of GRAIN REPITCH

GRAIN REPOSITION – Reposition grain onsets in a grainy sound

Usage

grain reposition *infile outfile timefile offset* [**-b***len*] [**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Parameters

infile – input grainy soundfile
outfile – output soundfile with grain timing re-patterned
timefile – must contain a list of grain-onset times in seconds. If any inter-grain time is reduced the minimum time allowed for a grain (0.032 sec), it will be reset to this minimum grain time.
offset – add this value in seconds to **all** grain timings – in effect, begin to apply the *timefile* pattern at this point in the *infile*.
-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 REPOSITION Process

This process re-times the start-times of the grains in the soundfile according to the pattern of times specified in *timefile*. The gaps between grain onsets can become larger or smaller in the *timefile*, but all onset times must increase – otherwise you would be trying to go backwards: e.g., 2.3 2.5 is OK, but 2.3 1.8 is not.

The *offset* parameter determines when after the start of the soundfile the *timefile* pattern is applied. If, for example, the *offset* is 2 (sec.) and there are grains before the 2 second point, you will hear these sound as normal, and then you will hear grains after the 2 second point patterned by the *timefile*

It is useful to run GRAIN FIND first, because it finds any grains in the soundfile and writes their start-times to a textfile. You can then edit this textfile to create a *timefile* with your own timing pattern.

Musical Applications

GRAIN REPOSITION enables you to impose a rhythmic structure on the grains in a soundfile. Considerably more subtle effects are also possible. Trevor (Wishart) writes that it "provides a way to create a correlation between the fine attack structure and some other musical events. For example, the repositioning data can be taken from another grainy sound (using GRAIN FIND) or any other timing data, allowing the fine attack structure of a grainy sound to be synchronised or coordinated with another sound or sound sequence."

Here's a simple procedure:

- A *timefile* for the grains in a soundfile is quickly made with GRAIN FIND.
- Then GRAIN REPOSITION is run twice, e.g., with an *offset* of 0.25 the first time, and 0.33 the second time.
- The two files can be mixed with **SUBMIX MERGE** and
- You can then audition the result, hearing the double occurrence of the grains.

End of GRAIN REPOSITION

GRAIN RERHYTHM – Change rhythm of grains in a grainy sound

Usage

grain rerhythm mode *infile outfile multfile* [**-b***len*] [**-l***gate*] [**-h***minhole*] [**-t***winsize*] [**-x**]

Modes

1 Lengthen or shorten each grain in turn, without repeating any grains; on reaching the end of the time-multipliers list, cycle back to its start.
2 Play grain at each specified retiming, before proceeding to the next grain.

Parameters

infile – input soundfile to process
outfile – output soundfile
multfile – a file of listing duration-multipliers to change the duration between one grain onset and the next grain onset.

- the maximum value for a multiplier is 1000
- the minimum value for a multiplier is 0.001000
- If any inter-grain 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 RERHYTHM Process

GRAIN RERHYTHM works in the same way as REMOTIF AND REPITCH, except applied to the timing of the grains only. Thus they retain their original pitch but begin at different times, according to the data in *multfile*. It may be that the grain lengths appear to change, because they begin to overlap one another: i.e., the distance between grain onsets changes, but the lengths of the grains are not meant to change.

The two Modes also work in the same way, either applying the data in the textfile to the grains sequentially, or repeating grains, applying all the data in the textfile to each grain in turn.

Musical Applications

As pitch is not altered, the sense of movement is more directly perceived. The flow of the grains can be made to flex in supple or in strongly contrasting ways.

If there is a stuttering or rapidfire quality, it may be caused by the process not adequately separating the grains. If a group of grains are not seen as separate, they will behave as a single grain. It is essential with grain processes to ensure that all the grains are clearly separated (i.e., set the appropriate gate level). In some cases it will not be possible to get a totally satisfactory result (everything depends on the nature of the source). A little cosmetic editing may be required.

Also see **GRAIN TIMEWARP**.

End of GRAIN RERHYTHM

GRAIN REVERSE – Reverse order of grains in a grainy sound without reversing the grains themselves

Usage

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

Parameters

infile – input soundfile to process
outfile – output soundfile
-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 REVERSE Process

Unlike MODIFY RADICAL Mode **1** (which plays the source sound **itself** backwards, and thus can radically modify it), this process plays each original grain in the normal way (not reversed in time), but **reversed in order**. This is much like the idea of retrograding a melodic motif in traditional music (where the sounds themselves are not reversed).

The original source is thus always clearly recognisable in the output sound.

Musical Applications

The forwards orientation of each grain means that the attack transient of the grains is retained. Even through the grains are generally very TW: Even though (Rather than 'even through') short, it is a tribute to the acuteness of our hearing that their forwards orientation makes such a difference in the recognisability of the original. GRAIN REVERSE, therefore, provides a wonderfully subtle variant of the original.

End of GRAIN REVERSE

GRAIN R_EXTEND – 'Time-stretch' natural sounds like the rolled 'rrr' in speech

Usage

grain r_extend 1 *infile outfile stt end ts pr rep get asc psc* [**-x**]
grain r_extend 2-3 *infile outfile gate usz ts pr rep get asc psc skp at T by* [**-s**] [**-e**]

Modes

1 Mark where the iterative part of the sound is located.
2-3 The program attempts to find the iterative part, using envelope tracing.

Parameters

infile – input soundfile containing iterative material
outfile – output soundfile with the iterations extended
stt – (Mode **1**) - time of start of iterated material within source soundfile
end – (Mode **1**) - time of end of iterated material within source soundfile
gate – (Mode **2**) - minimum level in source soundfile for envelope tracing to kick in. (Range: 0 to 1)
usz – (Mode **2**) - size of unit searched for, in milliseconds (15 for rolled 'rr')
ts – ('time-stretch') - multiplier that specifies how much to time-stretch the marked (or found) material
pr – guesstimate of the pitch-range of the iteration in the source, in octaves, or parts of octaves (try *pr* = 1). This is the pitch range of the (low-frequency) iteration, NOT the resonant frequency. E.g., for a rolled 'rrr', the frequency of the 'rrr' itself (even if unvoiced), and NOT the pitch of any sung note (if 'rrr' is voiced). (Range: 0 to 4)
rep – ('repeats') - the iterated material is extended by reusing individual segments in a randomised pattern. In this pattern, segment A may occur next to an identical copy of segment A, or not – see more detailed explanation below. (Range: 1 to 32767). Recommendation: try *rep* = 1 or *rep* = 2.
get – guesstimate of the number of iterations you expect to find (in the segment you've selected in the source – listen to the source). This helps to improve the accuracy of the search for iterated segment boundaries. In Mode **2**, this is the **minimum** number of repeats you expect to find.
asc – random amplitude variation of the output segments. The value for *asc* multiplies the original segment amplitude by a random value in the range 1 to 1-*asc*. (Range of *asc*: 0 to 1)
psc – random pitch variation of the output segments. The value for *psc* transposes the pitch by a random value in the range **-***psc* to **+***psc* semitones. (Range of *psc*: 0 to 24)
skp – the number of found (iterative) units to skip before processing
at T – iterative ritard, after *at* seconds to event-separation *T*
by – ritard end point reached after a further *by* seconds; ***by* = 0 gives no ritard**
[**-x**] – (Mode **1**) - keep rrr-extension only
[**-s**] – (Mode **2**) - keep sound *before* the extended material
[**-e**] – (Mode **2**) - keep sound *after* the extended material

Understanding the GRAIN R_EXTEND Process

The key here is to listen to the source sound to ascertain where iterative material starts and ends. 'Iterative material' means a rapid pulsation as in a vocal rolled 'r', vocal grit generally, rattling train windows, etc. GRAIN R_EXTEND enables you to prolong this material, with subtle variants in order to make the flow supple.

Notice that ***rep*** introduces a varying degree of randomised order in the repetitions of the grain units, the randomisation increasing with higher values. *Rep* has to do with how the grain sequence is generated by controlling how many identical grains can be adjacent.

- **Case 1:**
  1. If you have 4 grains `1, 2, 3, 4`, these are played back in a random order, e.g., `3 - 1 - 4 - 2`.
  2. Then another random order of the 4 is generated, e.g., `2 - 4 - 3 - 1`, etc., until the (time-stretched) output duration is filled up.
  3. When such permutations are put together, you might get a repetition of an element at the boundary between them, such as, using the two sequences above: `3 - 1 - 4 - 2 - 2 - 4 - 3 - 1`.
  4. If in this case *rep* were to be set to 1, this sequence would not be permitted, i.e., the second adjacent '2' would be discarded.
  5. **Thus *rep* determines how many such repetitions you allow.**
- **Case 2:**
  1. If you set *rep* to 2, instead of permutating `1, 2, 3, 4`, we permutate `1, 1, 2, 2, 3, 3, 4, 4`, which generates a sequence in which **any** element **might** repeat: e.g., `4 - 1 - 4 - 3 - 3 - 1 - 2 - 2` in which the 2 and the 3 are repeated.
  2. However, you could still get a second permutation such as `2 - 1 - 3 - 4 - 3 - 1 - 4 - 2`, which, when juxtaposed with the first permutation, would place 3 2's adjacent to each other: `4 - 1 - 4 - 3 - 3 - 1 - 2 - 2 - 2 - 1 - 3 - 4 - 3 - 1 - 4 - 2`.
  3. This would not be permitted if *rep* = 2, and the 3rd 2 would be discarded. You can see therefore, how *rep* is functioning.
  4. **Thus *rep* is a function of the maximum internal 'loopiness' of the random sequence.**

Musical Applications

GRAIN R_EXTEND provides some middle ground between larger-scale segmentation techniques and smaller-scale granulation techniques. You can also focus on a particular part of a sound and prolong it with variants without having to cut and resplice the material.

End of GRAIN R_EXTEND

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.

End of GRAIN TIMEWARP

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

Usage

wrappage *infile* [*infile2 ...*] *outfile centre outchans spread depth*
velocity hvelocity density hdensity grainsize hgrainsize pitchshift hpitchshift amp hamp bsplice hbsplice esplice hesplice
range jitter outlength
[**-b***mult*] [**-e**] [**-o**]

Example command line to create multi-channel brassage:

```
wrappage wrappage horn omahum seven hosmc 4.5 8 4 5 etc.
```

Parameters

infile – input mono soundfile
infile2 ... – optional second and subsequent input soundfiles
outfile – output multi-channel soundfile – 2 or more (filename must not end with a '1')
centre – central position of the output sound field image in the *outfile* (Range 0 to *outchans*). Values < 1 for *centre* are positions between the last channel and the first channel. *Centre* can be a numeric constant, or a breakpoint text file of *time centre direction* triples. *Direction* has two settings: **1** = clockwise, and **-1** = anticlockwise; this is the direction of movement of *centre* from its previous position to its next position.
outchans – the number of channels in the output soundfile.
spread – the width (from far Left to far Right) of spatialisation around *centre*.
depth – the number of channels with sound, behind the leading edges of *spread*. The *depth* parameter is not allowed to be larger than 2 * *spread*. If it is, the program automatically truncates it to the maximum allowed value.

velocity – the speed of advance within the *infile(s)*, relative to outfile. This value must be >= 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.

density – grain overlap. Values > 0 and < 1 leave intergrain silence. Extremely small values don't perform predictably.
grainsize – grainsize in milliseconds (must be > 2 * splicelen). (Default 50)
pitchshift – pitchshift in + or - (fractions of) semitones.
amp – gain on grains (Range 0 to 1) (Default 1.0). Use if amplitude variation is required (over a range &/or in time).
bsplice – grain-startsplice length,in milliseconds (Default 5)
esplice – grain-endsplice length,in milliseconds (Default 5)

range – the length of time in milliseconds before 'now' in the *infile* in which the search for the next grain will take place (Default 0: stay at 'now')
jitter – randomisation of grain position (Range 0-1) (Default 0.5)
outlength – maximum *outfile* duration in seconds (if end of data is not reached). Set to zero (Default) to ignore. **NB:** if *velocity* at any point is 0, a value for *outlength* **must** be given.

- *hvelocity, hdensity, grainsize, hpitchshift, hamp, hbplsice, hesplice* – allow a range of values to be specified for any of the cooresponding parameters above. For example, with both *pitchshift* and *hpitchshift* set, a **random value** for *pitchshift* is chosen somewhere between these lower and upper limits.
- **NB** *pitchshift* and *hpitchshift* can both vary through time, i.e., use a breakpoint text file containing *time pitchshift* value pairs. The same is true of the other low-high parameter pairs.
- All parameters except *outlength* and *outchan* can vary through time.

EXTRA OPTIONAL FLAGS:
-b*mult* – enlarges the output buffers *mult* times. This is to accommodate long silences which may appear in the output.
-e – use exponential splices (Default: linear)
-o – make parameters other than *velocity* relative to time in the **output** soundfile. By default, these other parameters are read relative to time in the soundfile, but *velocity* is AWAYS read relative to time in the file.

Understanding the WRAPPAGE Process

How is a 'moving' multi-channel output is achieved?
T Wishart writes:
The *centre* parameter of WRAPPAGE defines spatial position: i.e., the channel number at which the output sound field is *centred*. This *centre* can be made to move from one output channel to another by providing a data file of *time centre direction* value-triples. For example:

```
time centre direction
0    2       1
2    4      -1
5    2       1
```

The *centre* of the image starts (*time* 0 on line 1) at channel 2, and moves clockwise: the last '1' on line 1 indicates clockwise motion. At 2 seconds (line 2) it reaches channel 4, and then begins to move anticlockwise (the '-1' on line 2), until, at 5 seconds (line 3) it arrives at channel 2.

The *spread* parameter of WRAPPAGE defines spatial spread: the width of the sound image, i.e., how many adjacent channels, or loudspeakers across which it spreads around this *centre*. Thus, for example, an image *centred* at 4 with a *spread* of 3 (in an 8-channel output space) would be spread across channels 3, 4 and 5.

The moving image of 'wrappage' works in exactly the same way as that in MCHANPAN, but is in no way dependent on it. WRAPPAGE produces the spatial motion itself. There is a difference, because MCHANPAN Mode 1 is moving a **single** (mono) source around the multi-channel space whereas WRAPPAGE is generating a sound-image of textured sounds which can not only move, but can change in width (as it moves or even when its' *centre* is stationary. The wrappage sound image could fill the entire multi-channel space (centre it anywhere and, in an *N*-channel space, make the *spread N*), and this image could even rotate (keep the *spread* at maximum and move the *centre*). However, unless the sound elements of the texture were themselves long, this rotation might not be noticed: i.e., with very short sounds, it would not be possible to tell whether the whole texture was rotating or whether the individual elements were simply occuring in different positions, sounding much like a non-rotating texture of the same materials.

Musical Applications

This program is essentially the same as **MODIFY BRASSAGE** in that it produces granulated soundfile output. The difference lies in its multi-channel facilities, particularly the movement patterns that can be achieved with a breakpoint file input for the *centre* parameter: defining with value triples the *time*, the *centre* position of the sound image (the channel), and the *direction* of movement. A multi-channel output soundfile of *outchans* number of channels can be created.

End of WRAPPAGE