

ECMC Csound Library

Ambisonic processing modules

Allan Schindler

(This documentation last updated Dec. 11, 2006)

1. Introduction

The ECMC Csound *ambisonic processing modules* enable users to append ambisonic B-format processing modules to the end of instruments in any Csound orchestra file instruments and to add commented p-fields for these modules to the end of *score11* input files for these instruments. These *ambisonic processing modules* are appended to existing Csound orchestra instruments and to existing *score11* input files, somewhat like *PD* "abstractions" or "subpatches," by means of two ECMC scripts:

☞ **getscb** ("get a score template that includes **b** processing modules") creates a new *score11* input file with added p-field parameters for B-format processing of the output of the instrument, and with p-fields for an added reverberation instrument.

getscb functions much like the Eastman Csound Library *getsc* command, but in addition to displaying the score template for an ECMC library instrument such as *tsamp* or *gran*, it also includes additional p-fields for ambisonic processing of the output of the instrument. *getscb* can also be used to append ambisonic processing p-fields to *score11* input files for your own Csound instruments, so long as the source files include a macro called *NEWPAR* and do not include an *end;* statement.

☞ **mkob** ("make an orchestra file that includes **b** processing modules") creates a Csound orchestra file with B-format processing added to the end of each instrument, and adds a global reverberation instrument to the end of the orchestra file. *mkob* can be used in place of *mko* to create orchestra files that include ECMC library instruments such as *gran* and *marimba*. It also can be used to append B-format processing modules to the end of your own instruments, so long as these instrument files include a *NEWPAR* macro and do not end with an *endin* statement.

getscb and *mkob* thus are used together to create *score11* input files and companion Csound orchestra files that include B-format ambisonic processing.

Three procession modules are available for use with *getscb* and *mkob*:

(1) **3df** (abbreviated "*f*") adds B-format processing in which the spatial localization of each "note" (or "event") in the score remains fixed in a stationary location. The notes within a score can have different localizations, but the location of each individual note remains stationary.

(2) **3dm** (abbreviated "*m*") is used to apply movement from one location to another within individual notes or events, and/or to introduce a small or large degree of random movement for some or all of the notes.

(3) **3dverb** is a global reverberator for all instruments and notes in the score.

3dverb is always included in orchestra and score files that employ the ECMC Csound ambisonic processing modules. A score file for a particular instrument must employ either *3df* or *3dm* for localizing each note in the score. However, a complete score file that includes more than one instrument, or more than one score for a single instrument, can append *3df* to one instrument and *3dm* to another instrument.

The steps involved in using the ECMC Csound ambisonic processing modules are:

1. Create a score template file that includes p-fields for the instrument(s) you will be using and also includes p-fields for the ambisonic processing modules. Alternatively, if you already have a working score file, it is possible to retrofit this score file, using *getscb* to append ambisonic processing modules to this score.
2. Edit the score and get it working to your satisfaction without ambisonic processing, using the normal *mko* command to create a mono or stereo orchestra file to compile the score file into a soundfile. (Do not try to accomplish too much all at once. Break your work down into manageable tasks, getting the pitch, rhythm, tempo and so on of your score working to your satisfaction, then spatializing these sounds.)
3. Once you are happy with the basic score file, turn your attention to sound spatialization, editing the score file to fill in values for the ambisonic post-processing parameters.
4. Use *mkob* to obtain an orchestra file for the score that includes ambisonic post-processing modules. Compile your orchestra and score files into a decoded stereo or quad soundfile.
5. Continue editing and improving the ambisonic post-processing until you are happy with the spatial localization.
6. When you are satisfied with both the basic score and its ambisonic spatialization, create a final version of the soundfile, which can be decoded to stereo or quad or, by running *mkob* again with different options, a B-format master.

1.1. Using *getscb* to create a score11 input file that includes ambisonic processing modules

To use *getscb* to obtain a score template that includes ambisonic processing modules for any of the following Eastman Csound Library instruments

bigsamp	cbsn	gran	resyn	trpt
bigtsamp	celesta	marimba	samp	tsamp
bsn	drums	phavoc	sf2to1	
carillon	fmod	plunk	sf	

simply type in the name of the Library instrument followed either by an

- **f** (or else by *3df*) to append p-fields for the *3df* module; or else by an
- **m** (or by *3dm*) to append p-fields for the *3dm* module to the score template

At the end of the template, p-fields for an added global reverberation instrument also will be added.

Examples:

(1) *getscb gran f*

Result: A score template for Library instrument *gran* is displayed that includes p-fields for the *3df* module and for the global reverberation instrument.

(2) *getscb tsamp m* Result: A score template for Library instrument *tsamp* is displayed that includes p-fields for the *3dm* module and for the global reverberation instrument.

(3) *getscb tsamp f samp m*

Result: The display includes a score template for instrument *tsamp* that incorporates p-fields for *3df*, then a template for *samp* that incorporates p-fields for *3dm*, and finally p-fields for the global reverberation instrument.

1.1.1. Using *getscb* with *score11* input files for your own instruments

Before you can use *getscb* to add ambisonic processing p-fields to *score11* files for your own Csound instruments, or to pre-existing *score11* files such as the *marimba1* or *granflute* examples, you must take a few seconds to prepare these files:

The source score file MUST include a *NEWPAR* macro definition, which specifies the first new (currently unused) p-field for the instrument. A *NEWPAR* macro statement looks like this:

```
define([NEWPAR],XX)dnl
```

where *XX* is an integer. If your instrument currently uses 12 p-fields, you probably would want the first p-field for the ambisonic module to be *p13*.

The ECMC utility *newpar* simplifies that creation of *NEWPAR* macro statements. Typing
newpar 13

will display the line

```
define([NEWPAR],13)dnl
```

which you can copy and paste into your score file before any ambisonic p-fields. Type *newpar* with no arguments for a full usage summary.

Note: You will need a corresponding *NEWPAR* statement within the orchestra file for the instrument. However, you do not need to insert a *NEWPAR* macro statement when creating scores and orchestra files for ECMC Csound Library instruments such as *tsamp* or *gran* with *getscb* and *mkob*. *NEWPAR* definitions already have been built into the score templates and orchestra files for these instruments. The *NEWPAR* values for these instruments are

bigsamp 43	cbsn 20	gran 11	resyn 16	trpt 23
bigtsamp 43	celesta 14	marimba 11	samp 21	tsamp 21
bsn 19	drums 22	phavoc 13	sf2to1 9	
carillon 13	fmod 16	plunk 23	sf 9	

If a score file does not contain a *NEWPAR* definition, *getscb* will abort.

1.2. Using *3df*

An example score call *3df1*, which illustrates usage of the *3df* module, can be displayed by typing:

```
getex 3df1
```

A compiled soundfile in B format created by this example score is available in the */sflib/x* directory. To decode and play this soundfile type

```
play -b2 /sflib/x/3df1.wxyz or else play -b4 /sflib/x/3df1.wxyz
```

The *3df* module, in which each score note remains stationary in position, is naturally considerably easier to use than *3dm*. Typing

```
getscb gran f
```

will produce the following score template; lines for the *3df* module are shown here in boldface:

```
*f90 0 64 -2 3 48 -1 -1 -1 7777 < Room definition
< 7777 reseed can be changed { 7777 reseeds from current time}
< enable DIST  rand ref.lev. EQ  EQlev  Q  EQmode
  1    6.13  .05 .42  3829  .6  .67  2  < ceiling
  1    2.071 .05 .38  3105  .5  .72  2  < floor
  1   11.278 .05 .46  4417  .8  .63  2  < front
  1   13.475 .05 .41  3763  .8  .61  2  < back
  1    8.319 .05 .45  4124  .8  .65  2  < right
  1    8.437 .05 .44  4017  .8  .65  2 ; < left; end of f90
```

```

    < score file for instrument >> gran <<
define([NEWPAR],11)dnl
instr gran
p3
    < duty factor: determines overlap of grains
    < normal range is between 100.02 and 100.2 {producing .02-.2 second overlaps}
du
    < p4 cross-fade time for both fade-in & fade-out ;
    < p4 normally = c. 1/2 duty factor fraction { 1/2 the grain overlap time}
p4
    < p5 = output amplitude : raw amp {10.1-32767} or multiplier {.001 - 10.}
    < default 0 = same amplitude as input soundfile
p5
p6          < p6 = gen1 function number
    < p7 = length of soundfile to be used, in seconds {duration} OR # of samples
p7
    < p8 = skip time into gen1 func : if positive p8 = skip time in seconds
    < if negative {between -.001 and -.999}, p8 = % of soundfile skipped
p8
    < p9 = pitch multiplier {default 0 = no pitch change}
p9
    < for stereo output only p10 = pan location {1. = hard left, 0 = hard right}
p10
include(/usr/local/turnkey/cslib/amb/pfields.h)dnl
< ----- Fixed ambisonic sound localization -----
PAR1          < func. num. for room parameters; def. 90
    < PAR2 is a 3 way switch: if 0 xyz all cart.; if PAR2 = 1 xyz all polar
    < if PAR2 = -1 xy polar and z cart. w/ sqrt of z added to PAR3 dist.
PAR2
    < PAR3 : CART: x in meters: front + /rear -
    < PAR3 : POLAR: xy angle: 0 = hard left, 90 = straight ahead,
    < 180 = hard right, 270 = straight rear
PAR3
    < PAR4 : CART: y in meters: left + / right -
    < POLAR: xyz distance of sound source
PAR4
    < PAR5 : if PAR2 = 0 or -1 CART. z is in meters: + = up,- = down; 0 = SKIP z
    < if PAR2= 1 POLAR: elevation angle; 0-180 up, 180-360 down; ; 0 = SKIP z
    < polar z affects x and y if not 0 or 180; cart. z added to polar PAR4 dist.
PAR5
    < PAR6 : reverb gain multiplier; ord. c. .6; < 1. less reverb, > 1 more rev.
PAR6
    < PAR7 : reverb brightness multiplier: ord. c. .6;, .001 - .99 = less rev.,
PAR7          < > 1.0 = more reverb
PAR8 1; < 0 = no noise; 1 = add small amnt. of noise for better perf.
end; < --- End of score for gran / 3df score ---

i99 0 0 1; < global reverb instr for w; also output x,y and z
te 60;
ampfac 1;
p3
du 1.;

```

```

< p4 - p6 affect reverberators but are optional : 0 = 1 ,
p4      < rev gain multiplier
p5      < reverb time multiplier c. .5 - 2.
p6      < reverb "brightness," high freq. diffusion c. .5 - 2.
p7      < optional small reverb delay added to that of source instr
end;
<<< getscb gran f : mkob {OUTFLAG} gran 3df >>>

```

Note that at the end of this display *getscb* includes a comment reminding you how you created this template, and in certain cases also will suggest the *mkob* command to create a corresponding orchestra file for this score. PP At the top of the score *getscb* has inserted a *function* (which Csound uses to create a table of numbers loaded into RAM) that defines a rather generic room in which the sounds will be placed. Often you will not need to edit this *f90* ("function number 90") function definition, but sometimes you may wish to change it. The function defines the distance from the listening position to the ceiling to be 6.13 meters, the distance to the floor to be 2.071 meters, the distance to the front wall to be 11.278 meters, the distance to the rear wall to be 13.475 meters, the distance to the right wall as 8.319 meters and the distance to the left wall to be 8.437 meters. You can change these values if you wish to increase or decrease the size of the room, which will affect sound diffusion and reverberation time. Avoid simple ratios between these dimensions.

Changing the "1" in the first (*Enable*) column in front of these numbers to a 0 would turn off reflections from this wall. There is almost never any reason to do this.

The third (*random*) column, with .05 values, introduces a small random element into the reverberation calculations. The fourth (*reflectivity level*) column, with values of .41, .38, .46 and so on, determines the reflectivity of each surface, with values between 0 and 1.; higher values produce greater diffusion (more echos). Values between about .25 and .6 are generally useful.

The values in the *EQ* column (e.g. 3829, 3105, etc.) are read into an EQ network and determine the "brightness" of the room. Values between about 1600 and 6000 are generally useful. I rarely change the *EQlev*, *Q* and *EQmode* values in the table. Advanced users who would like more information on all of the parameters within this function definition can consult the documentation for the *spat3d* opcode within the Csound reference manual. Note that although you can fine tune the reverberation time, diffusion and brightness of the room by changing the values within this table, this can be tedious, and you can more easily change some of the reverberant qualities by means of p-fields provided later in the template.

The actual p-fields for the *3df* module are *PAR1*, *PAR2*, *PAR3* and so on through *PAR8*. Since *NEWPAR* is defined to be *11* for instrument *gran*, *PAR1* ("ambisonic parameter 1") will become *p11* when this file is run through *score11*, *PAR2* will become *p12*, and *PAR8* will become *p18*.

PAR1 specifies the number of the function that defines the room parameters. If left blank (the norm) or set to 0, the default value for *PAR1* is 90, a pointer to the *f90* room definition function at the top of the score. Note that it is possible, although unusual, to create alternative room definition functions, and thus to place different notes of the score into different rooms.

PAR2 is a flag, or 3-position switch. If left blank or set to 0, the *x*, *y* and *z* values will be specified as Cartesian coordinates (exactly like those used with *vspace*) in the following three p-fields. If *PAR2* is set to 1, the localization of the sound is expressed instead in terms of ambisonic *polar* coordinates: an azimuth (*x/y*) angle between 0 and 360 degrees, an angle of elevation for *z*, and a total distance value, expressed in meters. In this tutorial I will not delve into the use of polar coordinates. See me if you would like to try to use them, since the polar coordinate angles used in

3df and *3dm* differ from standard ambisonic usage.

Assuming the *PAR2* is set to 0:

PAR3 provides an *x* (front-rear) coordinate for each note, in meters; positive values are in front of the listening position, negative values are behind the listener;

PAR4 provides a *y* (left-right) coordinate for each note, in meters; positive values are to the left of the listening position, negative values are to the right;

PAR5 provides a *z* (up-down) coordinate for each note, in meters; positive values are above the listening position, negative values are below ear level. You can leave *PAR5* blank if you are certain that you will never want to decode for 8 audio channels or for some other rig that can convey height information.

Values between 1.0 (meters) and about 4.0 or so meters are most common for *PAR3*, *PAR4* and *PAR5*, but of course you are free to experiment. Do not place a coordinate outside the room boundaries, however. With the following values:

PAR3 mo 4. 3. 4. -2. -3.; < x

PAR4 .5 1. 1.6 .5 1.7 2.6; < y

PAR5 nu 2./1.5/1./-1./-1.5/-2.; < z

- *x*: the sound sources will move linearly over 4 beats from locations somewhere in between 3 and 4 meters in front of the listener to locations somewhere in between 2 and 3 feet behind the listener
- *y*: half of the sounds will be between 1.0 and 1.6 meters to the left, and half of the notes will be positioned between 1.7 and 2.6 meters to the left
- the first note will be 2 meters above ear level, the second note 1.5 meters above, the third note 1 meter up, the fourth note 1.0 meter below ear level, and so on.

PAR6 enables one to adjust the amplitude of the late reflections for each note. Values greater than 1.0 will increase the amount of reverberation, values less than 1.0 will decrease the reverberation amplitude. If left blank, or set to 0, *PAR6* will have a default value of 1.0.

PAR7 enables one to adjust the brightness of the late reflections for each note. Values greater than 1.0 will increase the brightness, values less than 1.0 will decrease the brightness. If left blank, or set to 0, *PAR7* will have a default value of 1.0.

The default room defined in *f90* is fairly bright and reverberant, so it is typical to employ values somewhere between .3 and .9 for both *PAR6* and *PAR7*.

PAR8 is rarely needed but occasionally handy. Normally leave it blank or set it to 0.

If set to 1, the algorithm will add a tiny amount of noise to the signal. Intel processors suffer from a defect called *denormals*. Calculation of very small values (such as some room reflections) can, under certain circumstances, slow the computation of a job to a crawl. If you notice your job running extremely slowly, place a 1 in *PAR8*.

Important note: You will often find that the output amplitude created by *3df* or *3dm* is very low, unless the sound sources are very close to the listening position. It is usually necessary to include or add a global *ampfac* value to the score file, such as

ampfac 3.0 (am 3.0)

This will multiply all output amplitudes by 3.0. *ampfac*s between 1.5 and 7.0, or even higher depending upon the distance of your sound sources, are usually necessary. Example score *3df1* employs an *ampfac* of 7.2

Global reverberation instrument: 3dreverb

Instrument number 99 (*i99*), a global reverberation instrument that is added to the end of all scores for *3df* and *3dm*, provides late reflection reverberation for all notes produced by all instruments, and also outputs the *x*, *y* and *z* signals. (Early reflections are calculated within the source instruments). Instrument 99 plays only one "note." It should be turned on along with or prior to the beginning of the first note in the score, and its event should end about .5 to .8 seconds after the final note in the score has ended, to allow the reverberation time to decay to zero.

To determine the correct *p3* duration for this instrument, run the source score through *score11*, look at the bottom of the resulting *sout* file, find the starting time of the last note, add to this time the duration of the note, then add about .5 seconds to this time. Alternatively, if you already have compiled a soundfile without ambisonic post-processing from the source score, use *sfdur* or *sfinfo* to find the duration of this soundfile, add about .5 seconds to this duration, and type this duration into the *p3* p-field for *instrument 99*.

If you wish, all the the other parameters for this instrument — *p4* through *p7* — can be left blank. These fields enable you to tweak the global reverberation of all notes — to increase or decrease the reverberant amplitude, the reverberation time, and the brightness of the reverberation. Although this also can be accomplished by changing the values in the *f90* function table at the top of the score, it usually is more easier to "massage" the reverberation here instead.

p4 is a gain multiplier for the *reverberation amplitude*. Values greater than 1.0 increase the "wetness" (reverberant amplitude) of the "wet/dry" mix, while values less than 1.0 decrease the amount of reverberation. Values between about .7 and 1.3 are most common.

p5 is a multiplier for the *reverberation time* (how long, in seconds, it takes the reverberation to decrease by 60 dB). Values greater than 1.0 increase the reverberation time (which is calculated from the room dimensions and location of the sound sources), while values less than 1.0 decrease the reverberation time, effectively reducing the size of the room. Values between about .8 and 1.2 are most common.

p6 alters the "*brightness*" of the reverberation. Values greater than 1.0 increase the brightness (as though the walls are altered so that they are constructed from a harder substance), while values less than 1.0 decrease the brightness, "adding drapes and carpet" to, or darkening, the room. Values between about .8 and 1.2 are most common.

p7 can be used to add a small amount of additional delay to the late reflections, which sometimes can seem to increase the size of the room. Values between about .015 and .04 are most common, although this p-field generally is left blank.

1.3. Using *mkob* to create orchestra files with *3df* and *3dm* module inserts

Next we will consider how to create an orchestra file in order to compile a soundfile using a score with a *3df* insert. You should create a few scores that include *3df* and compile them into soundfiles before attempting to use the more complex *3dm* modules.

The command line syntax for *mkob* is:

```
mkob [-] [OUTFORMAT] [SR] [kr KR] instr 3dinsert [instr2 3dinsert2] [instrN 3dinsertN]
```

This allows us to set an output format, a sampling rate, a Csound control rate (if you are not satisfied with the default), and then one or more source instrument files. Each source instrument file name must be followed either by an *f* (to append Csound code for *3df* to the instrument) or else by an *m* (to append Csound code for *3dm* to the instrument). The source instrument file(s) can be either from the ECMC Csound Library (e.g. *samp* or *gran*) or else one of your own instruments, so long as it is properly prepared.

Valid *output formats* are

- **B** for encoded b-format (.wxyz); this is the default soundfile output that the orchestra file will create, and so the -B flag is never used.
- **U** or **UHJ** : this will create a decoded stereo UHJ output soundfile
- **2** : this also will create a standard, decoded stereo WAVE soundfile, but will "throw away" *x*; I recommend using UHJ format for stereo instead.
- **4** : this will create decoded quad output.
- **1** : mono output, without reverberation; used only occasionally for initial tests, never for

final versions of a soundfile.

Most often, we will choose "4" (quad) or "U" (UHJ) for testing a score, and then, when we are ready to create a final version, will redo the *mkob* and select B-format output

Sampling rate : The default output sampling rate is 44100. Valid sampling rate arguments are:

88200 or 88 : sets the sampling rate to 88200

48000 or 48 : sets the sampling rate to 48000

44100 or 44 : sets the sampling rate to 44100 (these arguments are rarely used, since this is the default sampling rate)

32000 or 32 : sets the sampling rate to 32000 (rarely used)

22050 or 22 : sets the sampling rate to 22050 (rarely used)

Control rate: The default Csound control rate will be set to 1/10 the sampling rate. This can be changed by inserting a *kr* flag on the command line followed by the desired control rate. This is rarely necessary.

Each *instr* argument is the name of a file with a source instrument, and each *3dinsert* argument is either an *f* or an *m*.

Some examples:

(1) *mkob U samp2 f* or *mk3d0 U samp2 3df*

Result: An orchestra file is created with decoded UHJ stereo output in which a *3df* module is appended to Library instrument *samp*. A global reverberation instrument (*instrument 99*) also is appended to the orchestra file. The sampling rate is 44100.

(2) *mkob 96 samp2 f* or *mk3d0 96 samp2 3df*

The same orchestra file is re-made, this time with encoded B-format output and a sampling rate of 96000.

(3) *mkob 96 kr 9600 U tsamp m gran f*

Result: An orchestra file is made that sets the sampling rate to 96000, the control rate to 9600, and the soundfile output to UHJ stereo. A *3dm* module is appended to Library instrument *tsamp*, while a *3df* module is appended to Library instrument *gran*.

(4) *mkob sackbut m*

Result: A *3dm* module and global reverberation instrument are appended to the user's instrument file *sackbut*. The soundfile output will be in B format and the sampling rate will be 44100.

In the last example, the user's *sackbut* file will need to include a *NEWPAR* macro statement that matches the corresponding statement in the score file. The *sackbut* file should include only one instrument definition. Additionally, the audio output signal to be processed by *3df* or *3dm* must be named *a1*. If your instrument currently defines the output audio signal to be something like *asignal*, you can include a simple assignment statement such as

$$a1 = asignal$$

at the end of the instrument. Also, it is often a good idea to remove the *endin* statement at the end of your instrument. This is often not necessary — *mkob* will try to strip it out, but occasionally it will miss.

Once you have created a score file and an orchestra file you can compile the output soundfile with Csound in the usual way.

1.4. Using *3dm* to move sounds around in space

The *3dm* module shares many features in common with the *3df* module, but in addition allows one to specify time varying spatial movement for each note in the score between two *x* locations, two *y* locations and two *z* locations. Additionally, a random movement component can be applied to *x*, to *y* and/or to *z*. Because of these additional features, *3dm* includes many more parameter fields — 27 — than *3df*, which has only 8 parameters. The *3dm* template also includes

several additional pre-defined function definitions that apply various types of shapes to the movement. *3dm* thus is more complicated to use, and Csound compilation of *3dm* jobs often run considerably slower than compilation of *3df* jobs; the calculations of early reflections for moving sound sources, and for doppler pitch shifts and changing brightness filter coefficients (sounds contain proportionately more high frequency energy as they move closer to our ears) all require additional and rather complex computation beyond the basic ambisonic formulae employed in the *3df* module.

To append modules for *3dm* and for the necessary global reverberator instrument to a new score for Library instrument *samp* we would type:

```
getscb samp
```

This will display the template below and on the following pages. For clarity, a portion of the basic *samp* template for p-fields 4 through 19 have been omitted here, and I have numbered the lines of the template. Portions of the template that are identical, or nearly identical, to the *3df* score template are shown in italics. The basic template for Library instrument *samp* is shown in small type. Portions of the template that are unique to *3dm* are shown in boldface.

```

1  *f90 0 64 -2 3 48 -1 -1 -1 7777 < Room definition
2  < 7777 reseed can be changed { 7777 reseeds from current time}
3  < enable DIST rand ref.lev. EQ EQlev Q EQmode
4  1 6.13 .05 .42 3829 .6 .67 2 < ceiling
5  1 2.071 .05 .38 3105 .5 .72 2 < floor
6  1 11.278 .05 .46 4417 .8 .63 2 < front
7  1 13.475 .05 .41 3763 .8 .61 2 < back
8  1 8.319 .05 .45 4124 .8 .65 2 < right
9  1 8.437 .05 .44 4017 .8 .65 2 ; < left; end of f90
10 < score file for instrument >> samp <<
11 * f100 0 1024 10 1.; < SINE WAVE for vibrato
12 * f50 0 65 7 0 64 1.; < linear change from value 1 to value 2
13 * f52 0 65 7 0 32 1. 32 0; < linear change from value 1 to val 2 to val 1
14 * f60 0 65 5 .01 64 1.; < exponential change from value 1 to value 2
15 *f57 0 64 7 0 31 0 1 1. 31 1. 1 0 ; < square wave for trill
16 < f58 creates 3 note trill {above & below center pitch}
17 *f58 0 64 7 0 21 0 1 1. 20 1. 1 -1 20 -1 1 0 ;
18
19 ampfac 1;
20 define([NEWPAR],21)dnl
21 SAMP START < mono input soundfiles
22 p3
23 du
    (parameters 4 through 19 of the original samp template omitted here)
24 < p20 = optional center freq. for filters : pch or cps : Default = p4 pitch
25 p20
26 < functions for moving sources
27 < val. 1 to val. 2
28 * f50 0 65 7 0 64 1.; < linear change from value 1 to value 2
29 * f51 0 65 7 0 4 .02 56 .98 4 1.; < lin change w/ sloped ends
30 * f52 0 65 5 .01 64 1.; < exponential change from value 1 to value 2
31 * f53 0 65 5 .005 3 .02 58 .98 3 1.; <expo change w/ sloped ends

```

```

32 * f54 0 65 8 0 16 0 .2 0 31.8 1 .1 1 ; < 1/2 of bell curve
33 * f55 0 1025 9 .25 1 0; < 1st quarter of sine wave
34 < symmetrical pyramid functions: val. 1 to val. 2 back to val. 1
35 * f60 0 65 7 0 32 1. 32 0; < linear change from value 1 to val 2 to val 1
36 * f61 0 65 7 0 2 .02 29 1. 1 1 29 .02 3 0; < lin pyramid, sloped ends
37 * f62 0 65 5 .01 32 1. 32 .01; < expo. change from value 1 to value 2 to val 1
38 * f63 0 65 5 .005 29 .97 2 1. 1 1 2 .97 30 .005; < expo. pyramid with sloped middle
39 * f64 0 65 8 0 16 0 .1 0 15.9 1 15.9 0 .1 0 16 0; < symmetric bell curve
40 * f65 0 1025 9 .5 1 0; < 1st half of sine wave
41 * f100 0 1025 10 1. 1 0 ; < sine wave: val. 1 to val. 2 to -val. 2 to val. 1
42 < ---- end of functions for moving x,y & z values
43 include(/usr/local/turnkey/cslib/amb/pfields.h)dnl
44 < ----3dm Time varying ambisonic sound localization ----
45 PAR1 < func. num. for room parameters; def. 90
46 < PAR2 is a 3 way switch: if 0 xyz all cart.; if PAR2 = 1 xyz all polar
47 < if PAR2 = -1 xy polar and z cart. w/ sqrt of z added to PAR3 dist.
48 PAR2
49 < for fixed locations put -999 in PAR4 or PAR5, PAR11 or PAR12, PAR18 or PAR19
50 < -----
51 < Note: PAR4,PAR11,PAR18: to add to or subtract values from PAR3, PAR10 or
52 < PAR17 use values > 1000 {add} or < 1000 {subtract}
53 < -----
54 < flags for all time values < { PAR5, PAR6, PAR12, PAR13, PAR19, PAR20}
55 < > 100 = val - 100. * p3 ; negative = abs * p3 to read the func
56 < -----
57 < == = PAR3 - PAR9 : < CARTESIAN: x in meters: front + /rear - =====>
58 < POLAR: xy angle: 0 = hard left, 90 = straight ahead,
59 < 180 = hard right, 270 = straight rear
60 < PAR3 x coord. 1 or xy angle 1; PAR4 x coord. 2 or xy angle 2
61 PAR3 < x1 or xy ang. 1
62 PAR4 -999; < x2 or xy ang. 2; see Note above
63 PAR5 < dur for PAR3 : 0 = .001, 999=fi xed
64 PAR6 < dur for change btw PAR3 & PAR4
65 PAR7 < func for change {def. = 51}
66 PAR8 < rd amt. {POL: degrees, CART. meters}
67 PAR9 < rd rate
68 < == = PAR10 - PAR16 : CART: y coord. in meters: left + /right - or =====>
69 < POLAR: xyz or xy distance in meters
70 < PAR10 y1 or xy dist. 1; PAR11 y2 or xy dist. 2
71 PAR10 < y or dist. 1
72 PAR11 -999; < y or dist. 2:see Note above
73 PAR12 < dur for PAR10:0 = .001;-999=fi xed
74 PAR13 < dur for change btw PAR10 & PAR11
75 PAR14 < func for change {def. = 51}
76 PAR15 < rd in meters
77 PAR16 < rd rate
78 < == = PAR17 - PAR23 : z :if PAR2 = 0 or -1 cart.; if PAR2 = 1 polar
79 < PAR17 & PAR18 : z1 & z2: cart: + up, - down; polar 0-180 up,180-360 down
80 PAR17 < z 1

```

```

81  PAR18 -999;          < z 2;0=0;-999=fi xed;see Note above
82  PAR19              < dur for z dist 1 {PAR17}:0=.001;-999=fi xed
83  PAR20              < dur for change btw z1 & z2
84  PAR21              < func for change {def. = 51}
85  PAR22              < rd amt. {POL: degrees,CART. meters}
86  PAR23              < rd rate
87  < - - - - -
88  PAR24              < oversample ratio, 1 - 8, def. 2, 8 best
89  < PAR25 : reverb gain multiplier;ord. c. .6; < 1. less reverb, > 1 more rev.
90  PAR25
91  < PAR26 : reverb brightness multiplier: ord. c. .6, .001 - .99 = less rev.,
92  PAR26              < > 1.0 = more reverb
93  PAR27              < 0 = no noise; 1 = add small amnt. of noise for better perf.
94  end; < --- End of score for samp / 3dm.s score ---
95  i99 0 0 1; < GLOBAL REVERB INSTR ; also output x,y and z
96  te 60;
97  ampfac 1;
98  p3
99  du 1.;
100 < p4 - p6 affect reverberators but are optional : 0 = 1 ,
101 p4      < rev gain multiplier
102 p5      < reverb time multiplier c. .5 - 2.
103 p6      < reverb "brightness," high freq. diffusion c. .5 - 2.
104 p7      < optional small reverb delay added to that of source instr
105 end;
106 <<< getscb samp m ; mkob samp m>>>

```

Italicized elements that this template shares in common with the *3df* template include:

- (1) An identical function definition that describes the room (*f90*, lines 1 through 9). The room dimensions, reflectivity (diffuseness) and brightness values can be changed in this table (if you know what you are doing), but similar results often can be obtained more easily by adjustments in
 - *PAR25* (line 90) and *p4* (line 100) in the global reverberation instrument, which alter the amplitude of the calculated reverberation
 - *PAR26* (line 92) and *p6* (line 103) in the global reverberation instrument, which alter the brightness of the calculated reverberation
 - *p5* and, to a lesser degree, *p7* (lines 102 and 104), which alter the calculated reverberation time and late reflection delay and thus, subjectively, the size and diffusion qualities of the room
- (2) *PAR1* (line 45) points to the function number that defines the room, which by default, if this p-field is left blank, is 90 (*f90* at the top of the score file).
- (3) *PAR2* (line 48) determines whether Cartesian or polar coordinates will be used to specify *x*, *y* and *z*. If *PAR2* is left blank or set to 0, the default, Cartesian coordinates will be used, with *x*, *y* and *z* parameter values expressed in meters.
- (4) *PAR25* (line 90, corresponding to *PAR6* in the *3df* module) enables one to adjust the gain of the reverberation. Values greater than 1.0 increase the amount of reverberation while values less than 1.0 decrease the reverberation level.
- (5) *PAR26* (line 92, corresponding to *PAR7* in the *3df* module) allows us to adjust the brightness of the reverberation, "brightening" the room (if *PAR26* is > 1.0) or "darkening" the hall (if *PAR26*

is < 1.0).

(6) *PAR27* (line 93), if set to 1, will add a tiny amount of noise, which under certain circumstances will solve the demormal problem on Intel processors.

Features that are unique to the *3dm* score template:

(1) Specification of the time varying *x*, *y* and *z* variables for every score note each require seven p-fields rather than the single p-field found in the *3df* template.

(2) *PAR24*, which sets an oversampling ratio, adjusts the quality versus speed of the computation. Leave this p-field blank, or set it to 2 (the default value) while doing test runs and perfecting your score. When you are ready to create the final version of your output soundfile, change the *PAR24* to 8. This will provide the best audio quality for moving sound sources, but at considerable cost in computation time. Even on a fast machine such *madking*, you will not be able to compile most soundfiles in real time without glitches when *PAR24* is set to 8.

1.4.1. Specification of *x*, *y* and *z* coordinates in 3dm : *PAR3* through *PAR23*

The subroutines for specifying time varying values

for *x* (*PAR3* through *PAR9*, lines 61 through 67),

for *y* (*PAR10* through *PAR16*, lines 71 through 77),

and for *z* (*PAR17* through *PAR23*, lines 80 through 86)

each work in an identical manner.

The first parameter within each of these groups (*PAR3* for *x*, *PAR10* for *y* and *PAR17* for *z*) specifies an initial (beginning) value. The second parameter within each of these groups (*PAR4* for *x*, *PAR11* for *y* and *PAR18* for *z*) specifies a second value for *x*, for *y* or for *z*. The default value that has been filled in for each of these parameters, 999, is a flag, which causes the *x*, *y* or *z* value to remain fixed throughout the note, as with *3df*.¹ Sometimes we may want *x* to remain fixed while *y* moves, but we cannot leave the parameter for the second *x* value blank because this would specify movement to a second *x* location of 0. Change the default 999 flag value to some other value to create movement along the *x*, *y* or *z* plane from the initial value to the second value.

Example: (In this example, and in those that follow, it is assumed that *PAR2* has been left blank or set to 0, so that Cartesian coordinates are being used.) Let's assume the following p-field values:

PAR3 -4.; < *x1* or *xy* ang. 1

PAR4 2.; < *x2* or *xy* ang. 2

PAR10 1.5; < *y* or *dist.* 1

PAR11 .5; < *y* or *dist.*

PAR17 -1.33; < *z* 1 or *z* ang. 1

PAR18 0; < *z* 2; -999=fixed

At some point during the note the sound will begin moving gradually from its initial, rear *x* location 4 meters behind the listener to a new location 2 meters in front of the listener. The sound also will move at some point from its initial *y* location 1.5 meters to the left of the listener to a new location 1/2 meter to the left. Additionally, if the resulting soundfile is decoded for 8 channels, the sound will gradually move upward from an initial location 1.33 meters below ear level to a location parallel to the listener's ears. However, *x*, *y* and *z* need not move together, nor at the same speed, nor follow the same shape.

The next three parameters for *x*, for *y* and for *z* specify exactly how the sound will move between the initial and second values. We will consider *x* first, but the parameters for *y* and for *z*

¹ This also means that any random deviation (discussed later) specified for the coordinate will be ignored.

work in an identical manner. For x , $PAR5$ indicates an initial time duration during which x will remain fixed at its initial value (-4 in the example above). $PAR6$ specifies a duration during which x will move to its second value, or through this second value to some final location. For the remainder of the note (any portion of the total $p3$ duration of the note that remains after $PAR5$ and $PAR6$ are subtracted from $p3$), the x vector of the sound will remain fixed at its concluding value.

If $PAR5$ and/or $PAR6$ are positive they designate a duration in seconds. If $PAR5$ and/or $PAR6$ are a negative value between -.001 and -.999, they specify a decimal *percentage* of the total note duration. See the following example for clarification.

The next parameter, $PAR7$, specifies a pre-defined function number (table of numbers) that "draws a shape" for the movement between values 1 and 2. Several pre-defined functions are included on lines 28 through 41 of the template for this purpose.² Postscript graphical displays of the pre-defined functions 50 through 55, 60 through 65, and 100 (sine wave) can be viewed on any web browser that supports postscript display (e.g. on any computer that includes Adobe's *Acrobat Reader*) at

<http://ecmc.rochester.edu/ecmc/docs/movefuncs.eps>

Function numbers 50 ($f50$, line 28) through 55 (line 33) all define movement from value 1 to value 2, but according to different shapes:

$f50$ defines simple, linear straight-line (ramp) movement from value 1 to value 2; this is the default function for x , y and z movement, if $PAR7$ (x), $PAR14$ (y) or $PAR21$ (z) are left blank or set to 0.

$f51$ is similar to function 50, but the beginning and end of the ramp are tapered, or sloped, so that the movement begins and ends more gradually, rather than abruptly

$f52$ defines exponential movement from value 1 to value 2, so that the movement begins slowly, accelerates and then stops abruptly; most of the change comes near the end

$f53$ is similar to $f52$, but with a tapered (or sloped) beginning and end, so that the motion ends more gradually

$f54$ defines the first half of a bell-shaped curve, where the motion of value 1 to value 2 is most rapid in the middle and slower at both ends

$f55$ defines the first quarter of a sine wave, where the motion from value 1 to value 2 is most rapid at the beginning and slows down near the end

Function numbers 60 ($f60$, line 35) through 65 (line 40), and function number 100 (line 41) all define symmetrical, mirror-like movement from value 1 to value 2 and then back again to value 1, but according to different shapes:

$f60$ creates simple, straight-line linear movement from value 1 to value 2 back to value 1

$f61$ is similar to $f60$, but the beginning and end of the ramp movement are tapered, or sloped, so that motion begins and ends more gradually

$f62$ is an exponential pyramid; motion is rapid in the middle (approaching and leaving the value 2 location) and much slower at the beginning and end

$f63$ defines an exponential pyramid, similar to $f62$, but the middle of the function is sloped so that it does not "turn the corner" so abruptly

$f64$ creates a smooth bell-shaped curve, in which the motion is most rapid in the middle and slower at the two ends

$f65$ is the first half of a sine wave; motion is most rapid 1/4 and 3/4 of the way through the function and slower in the middle

$f100$ creates one full cycle of a sine wave; here the shape of movement will be more complex, moving from value 1 to value 2 back to value 1, then to minus value 2 and finally

² Users who know how to code in Csound can create their own alternative functions to precisely define desired shapes for movement and to use instead of, or in addition to, the predefined "move" functions included in the template.

back to value 1.

Two example score files illustrating the use of the "move" functions: *3dm1* and *3dm2*

ECMC Library example scores *3dm1* and *3dm2* are very boring, didactic examples to view or to listen to, but they illustrate the usage of these "move" functions and the resulting "shapes" of the spatial motion. You can view these example scores by typing

getex 3dm1 or *getex 3dm2*

and you can play the resulting compiled B-format soundfiles in the *sftb/x* directory by typing

play -b2 /sftb/x/3dm1.wxyz or *play -b2 /sftb/x/3dm2.wxyz*

To play these soundfiles with quad decoding change the *-b2* flag or *-b4*.

Example score *3dm1* creates 6 output notes — a Gmajor scale played by violin samples — and example score *3dm2* creates 6 output notes. In all of the notes for both scores, *x* and *z* remain constant, with all notes positioned 1.5 meters in front of the listener (*x*) and at ear level (no values are given for the *z* parameters). Motion is confined to the left-right *y* plane, which makes it easiest to hear to effect of each of the "move" functions.

(Excerpt from ECMC Csound Library example score file *3dm1*.)

```
PAR3 1.5;          < x1 : 1.5 meters to the front
PAR4 -999;         < x2 : same value as x1, 1.5 meters forward
(x parameters PAR6, PAR7, PAR8 and PAR9 are left blank and are omitted here)
< PAR10 y1 or xy dist. 1; PAR11 y2 or xy dist. 2
PAR10 -2;         < y or dist. 1
PAR11 2;          < y or dist. 2:see Note above
PAR12 .5;         < dur for PAR10:0 = .001;-999=fixed
PAR13 2.0;        < dur for change btw PAR10 & PAR11
PAR14 nu 50/51/52/53/54/55; < func for change {def. = 51}
PAR15             < rd in meters
PAR16             < rd rate
< = = = PAR17 - PAR23 : z :if PAR2 = 0 or -1 cart.; if PAR2 = 1 polar < PAR17 &
PAR18 : z1 & z2: cart: + up, - down; polar 0-180 up,180-360 down
PAR17             < z 1 : z is not used
PAR18 -999;       < z 2;0=0;-999=fixed
```

Example score file *3dm2*, and its soundfile compilation */sftb/x/3dm2.wxyz*, is identical to example *3dm1* in most respects, except that "move" functions 60 through 65 are illustrated, so that the sounds move from *y* location 1 (2 meters to the right) to *y* location 2 (2 meters to the left) and then back to the original location (2 meters to the right). Some of the functions create continuous, smooth motion, while in others the sound "turns the corner sharply" when doubling back to the original location.

These same "move" functions can be applied to *x* (*PAR7*) and to *z* (*PAR21*) to produce similar types of movement in the front-rear *x* plane and in the up-down *z* plane. If *PAR7*, *PAR14* or *PAR21* are left blank, the default function used is *f51* — linear motion with sloped ends, producing smooth motion from value 1 to value 2.

Random deviation applied to the x, y and z coordinates

Two optional additional parameters are included in each of the *x*, *y* and *z* subroutines that introduce a small or large amount of random movement. These p-files are

PAR8 and *PAR9* (lines 66-67) for *x*

PAR15 and *PAR16* (lines 76-77) for *y*

and *PAR22* and *PAR23* (lines 85-86) for *z*

The first parameter in each of these groups (*PAR8*, *PAR15* and *PAR22*) specifies the *amount*,

in meters, by which the x,y or z coordinate can vary randomly, in either direction, from its current position; the second parameter (*PAR9*, *PAR16* and *PAR23*) specifies the *rate*, in seconds, at which this random jitter will occur.

Example score *3dm3* illustrates the use of random deviation applied to spatial localization. This example illustrates random deviation applied to x,y and z movement it is best played with quad or (better) 8 channel decoding The first two "notes" are similar — a nan-bang roll moves from 5 meters forward to 5.5 meters to the rear (*PAR3* & *PAR4*) and is 1.8 meters to the right (*PAR10* & *PAR11*), 1. meter below ear level (*PAR17*). There is no random spatial deviation in the first note, but the left-right y coordinate varies randomly by +/- 1.2 meters 3.3 times per second in the second note. The third note presents an insect swarm with random deviation applied to x, y and z .

Example scores *3dm1*, *3dm2* and *3dm3* are didactic primers and not very interesting musically. The final example score for module *3dm*, example *3dm4*, while unlikely to find its way onto boxed collections of *Fifteen Greatest Classical Msterpieces of All Time*, may at least hold your musical interest for more than 3 or 4 seconds. *3dm4* is a "spatialization" of Library example score *tsamp2*, which loops three sounds. For your listening enjoyment this score has been compiled into soundfile `/sfb/x/3dm4.wxyz`, which can be decoded for quad or stereo playback, and the 8 channel decoded version `/sfb/x/oct.3dm4.wav`.