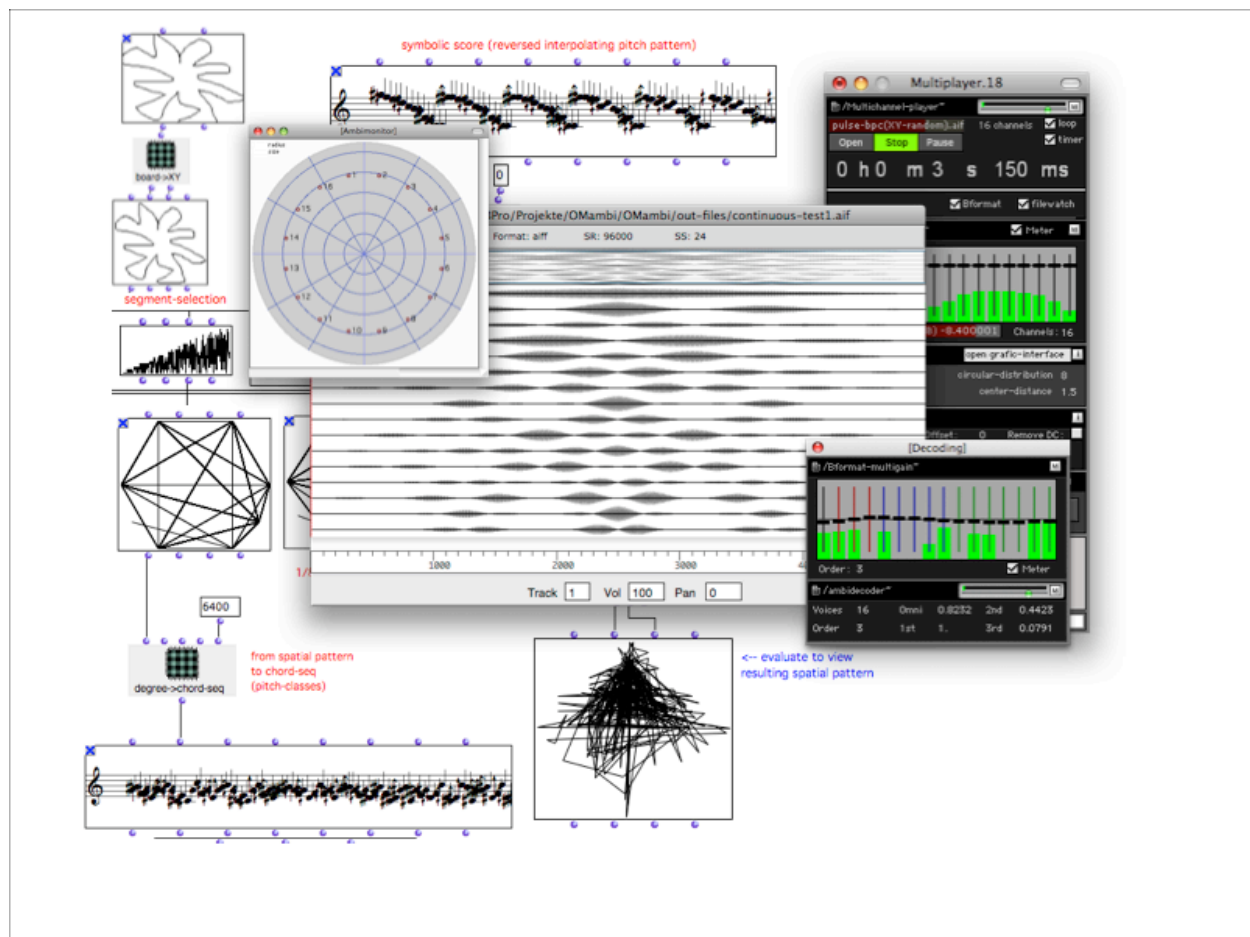


## Symbolic control of spatialization with OpenMusic

## Documentation & Tutorials

Marlon Schumacher, December 2008, Draft Version



---

# Table of Contents

|  |    |
|--|----|
| 1. Introduction.....                                   | 4  |
| 2. Requirements, Installation & Setup .....            | 5  |
| 3. Architecture.....                                   | 6  |
| 3.1. Principles of ambisonics .....                    | 6  |
| 3.2. Artificial Distance Cues .....                    | 6  |
| 3.3. Implementation .....                              | 7  |
| 4. Control Structure.....                              | 8  |
| 4.1 Matrix Representations .....                       | 8  |
| 4.2 Levels of control .....                            | 8  |
| 4.3 Implementation .....                               | 10 |
| 4.3.1 High-Level .....                                 | 10 |
| 4.3.2 Mid-Level .....                                  | 11 |
| 4.3.3 Low-Level.....                                   | 12 |
| 4.3.3 Continuous Control.....                          | 12 |
| 5. Ambisonic Decoder .....                             | 12 |
| 5.1 Components .....                                   | 13 |
| 5.2 Quickstart.....                                    | 15 |
| 6. Tutorials .....                                     | 15 |
| 6.1. Basic.....  | 15 |
| 6.1.1 Tutorial 1: spatial-risset-bell .....            | 16 |
| 6.1.2 Tutorial 2: additive-spatial-spread .....        | 16 |
| 6.1.3 Tutorial 3: fog-stochastic .....                 | 17 |
| 6.1.4 Tutorial 4: smpl-basic .....                     | 18 |
| 6.1.5 Tutorial 5: smpl-granular .....                  | 18 |
| 6.1.6 Tutorial 6: distance-cues .....                  | 19 |
| 6.1.7 Tutorial 7: smpl-bpc.....                        | 20 |
| 6.1.8 Tutorial 8: smpl-spatial-dafx.....               | 21 |
| 6.1.9a Tutorial 9a: smpl-chordseq.....                 | 22 |
| 6.1.9b Tutorial 9b: smpl-segmentation .....            | 22 |
| 6.2. Advanced .....                                    | 23 |
| 6.2.1 Tutorial 1: continuous-bpc-trajectory .....      | 23 |
| 6.2.2 Tutorial 2: granular-board-trajectory .....      | 24 |
| 6.2.3 Tutorial 3: granular-time-space-stretching ..... | 24 |
| 6.2.4 Tutorial 4: continuous-risset-bell.....          | 25 |
| 6.2.5 Tutorial 5: parsing-fun-fineberg .....           | 26 |
| 7. Building OMambi classes .....                       | 27 |
| 7.1 Csound orchestras and code snippets .....          | 27 |
| 7.2 Merging synthesizers & ambisonic encoders .....    | 27 |
| 8. Future Developments .....                           | 34 |

---

|  |    |
|--|----|
| 9. Appendix.....   | 34 |
| 9.1 Coordinate systems.....  | 34 |
| 9.2 List of matrix inlets & descriptions for discrete control.....   | 35 |
| 9.2.1 Ambiench (high-level control).....                             | 35 |
| 9.2.2 Ambiencm (mid-level control) .....                             | 36 |
| 9.2.2 Ambiencl (low-level control).....                              | 36 |
| 9.3 List of matrix inlets & descriptions for continuous control..... | 36 |
| 9.3.1 Ambiench (high-level control).....                             | 36 |
| 9.3.2 Ambiencm (mid-level control) .....                             | 37 |
| 9.3.3 Ambiencl (low-level control).....                              | 38 |
| 9.4 Overview of implemented OMambi-classes .....                     | 39 |

---

# 1. Introduction

OMambi is a library for symbolic control of spatialization with OpenMusic. It is implemented as a set of omChroma classes for higher order ambisonics providing control of spatialization from different levels of abstraction.

Although care has been taken to keep the tutorials as simple as possible, this documentation presupposes a certain degree of familiarity with Csound, OpenMusic and omChroma; Please refer to the respective documentations for detailed instructions on the use of the above software. It is recommended to be familiar with the basic concepts of the omChroma system such as the matrix, typed slots and keywords. The tutorials are intended as an introduction to the concepts for controlling spatialization in an environment for computer assisted composition like OpenMusic without trying to be exhaustive in terms of control possibilities or suggesting any particular control paradigm.

Ambisonic Classes have been implemented for three types of synthesis, namely additive, fog (formantic granular waveform) and sample-based, please see the Appendix for an overview of available classes. OMambi can be user-extended as described in chapter 7 of this documentation.

For ambisonic decoding and interactive setting of parameters an external [Max/MSP](#)<sup>1</sup>-application is provided, based on extended and custom built high-level modules complying with the [jamoma](#)<sup>2</sup> framework.

## Informations on the structure of this text:

Chapter 2 provides instructions for installation of this software. If you are already familiar with how to install and configure OpenMusic/omChroma, Csound and Max/MSP you can skip this part.

Chapter 3 is a very brief introduction into the underlying principles of higher-order-ambisonics and describes the dsp-architecture of OMambi.

Chapter 4 describes the general concepts behind the control structures of OMambi.

Chapter 5 is a concise documentation of the Max/MSP-based application for ambisonic decoding.

Chapter 6 contains Tutorials subdivided into two parts. The Basic tutorials present different methods for controlling spatialization with discrete source-positions. Concepts of spectral vs. granular spatialization are introduced. The Advanced tutorials address the continuous control of spatialization parameters, drawing of Trajectories, Time/Space-manipulation and the parsing-fun.

Chapter 7 provides instructions on how to extend the library with new classes.

Chapter 8 discusses future developments and perspectives.

---

<sup>1</sup> <http://www.cycling74.com/products/max5>

<sup>2</sup> <http://jamoma.org/>

---

## 2. Requirements, Installation & Setup

### Requirements:

This software has been tested with OpenMusic6.04, omChroma4.0 and Csound5.09. The Multiplayer is a standalone application. You can find OpenMusic/omChroma [here](#)<sup>1</sup>, Csound is available [here](#)<sup>2</sup>, OMambi can be downloaded from [here](#)<sup>3</sup>.

### Instructions:

Mount OMambi.dmg and copy the contents into a folder anywhere on your harddrive. After launching OpenMusic you will be asked for a workspace: just point to the above folder.

Of course it is also possible to use the OMambi-classes with another workspace:

Copy the folder “/user/OMambi” from the OMambi-workspace into the directory “/user” inside the workspace you would like to use. Start OpenMusic with your desired workspace and select “Windows/Library” from OpenMusic’s menubar. In the opened window expand the trunk “user” by clicking on the arrow left of it. Now double-click on the lower half of the red trunk called “omambi” to load the classes as shown in the figure below. You can instantiate a class by dragging it onto a patch or typing its name into an empty box in a OpenMusic patch.



### Setup:

Before starting to work with OMambi it is recommended to check OpenMusic’s Preferences for “Audio Settings” and “External Sound Processing”. Please refer to the omChroma documentation for detailed instructions on how to set the preferences. OMambi will work with the default settings<sup>4</sup>.

---

<sup>1</sup> <http://recherche.ircam.fr/equipes/repmus/OpenMusic/>

<sup>2</sup> <http://www.csounds.com/>

<sup>3</sup> <http://www.music.mcgill.ca/~marlon/software/OMambi/OMambi1.0.dmg>

<sup>4</sup> NB: By default omChroma renders soundfiles at 96kHz/24Bit resolution. Accordingly, the B-format files (16channels) generated by OMambi can become quite large (one second of audio corresponds roughly to 4.6MB)

---

## 3. Architecture

OMambi uses the ambisonic approach for spatialization. Although a detailed description of ambisonics is beyond the scope of this documentation the most relevant principles with regards to their control with OMambi shall be briefly described here, a comprehensive list of references can be found [here](http://www.york.ac.uk/inst/mustech/3d_audio/ambrefs.htm)<sup>1</sup>.

### 3.1. Principles of ambisonics

Ambisonics originated as a spatial sound recording technique using a special microphone described by M.Gerzon and Cooper in the early 1970s with the goal of storing spatial sound information with a few audio channels. The spatial information of the recorded or synthesized sound is encoded together with the sound itself in a specific number of channels, the *B-Format*, which is independent of the final speaker setup. With higher order ambisonics (HOA), a 2-dimensional (planar) or 3-dimensional (periphonic) soundfield can be reproduced with varying accuracy determined by the available number of speakers for reproduction as well as the *order* of the encoded signal: the higher the order, the more accurate the spatialization, 0<sup>th</sup>-order equals full monophony (omnidirectional). A great advantage of ambisonics is that the encoding and decoding process are independent of each other, which allows to store an “abstract” description of a soundfield in a portable B-Format file. The self-compatibility of ambisonics<sup>2</sup> makes it an ideal choice for spatial compositions ought to be reproduced with different loudspeaker setups in different listening spaces. Another benefit is that the spatial information is stored as an audio-signal, which is open to manipulation using digital signal processing techniques as with any other signal (e.g. soundfield-effects). The underlying equations used for HOA are based on several simplifications, one of them is that virtual sound sources and loudspeakers emit plane waves. Accordingly, the B-Format itself doesn’t contain any distance-information, auditory cues to create the perception of source distance need to be added to the signal before the encoding process as described in section 3.2.

### 3.2. Artificial Distance Cues

OMambi implements a series of dsp-units to provide for perceptual cues supporting the impression of distance of a sound source: A second-order butterworth lowpass filter is used to simulate the effect of air-absorption, that is loss of high-frequency energy of a soundwave travelling through air. The filtered signal is processed by a gain-unit to account for the attenuation of the signal as a function of distance, i.e. with increasing distance the sound source becomes more silent. Depending on the speed of sound and the distance of a sound source from the listener, i.e. the length of the propagation path, the time-of-flight varies accordingly. For moving sound sources this causes a pitch-shifting-effect, known as the doppler-effect. A variable

---

<sup>1</sup> [http://www.york.ac.uk/inst/mustech/3d\\_audio/ambrefs.htm](http://www.york.ac.uk/inst/mustech/3d_audio/ambrefs.htm)

<sup>2</sup> Higher orders can be decoded to lower order reproduction systems and vice versa

time delay is implemented to simulate this effect before the signal is finally encoded into B-Format. Most commonly parameters of the above units are controlled as functions of distance, however, as we shall see in chapter 4 all distance-cues can be controlled arbitrarily.

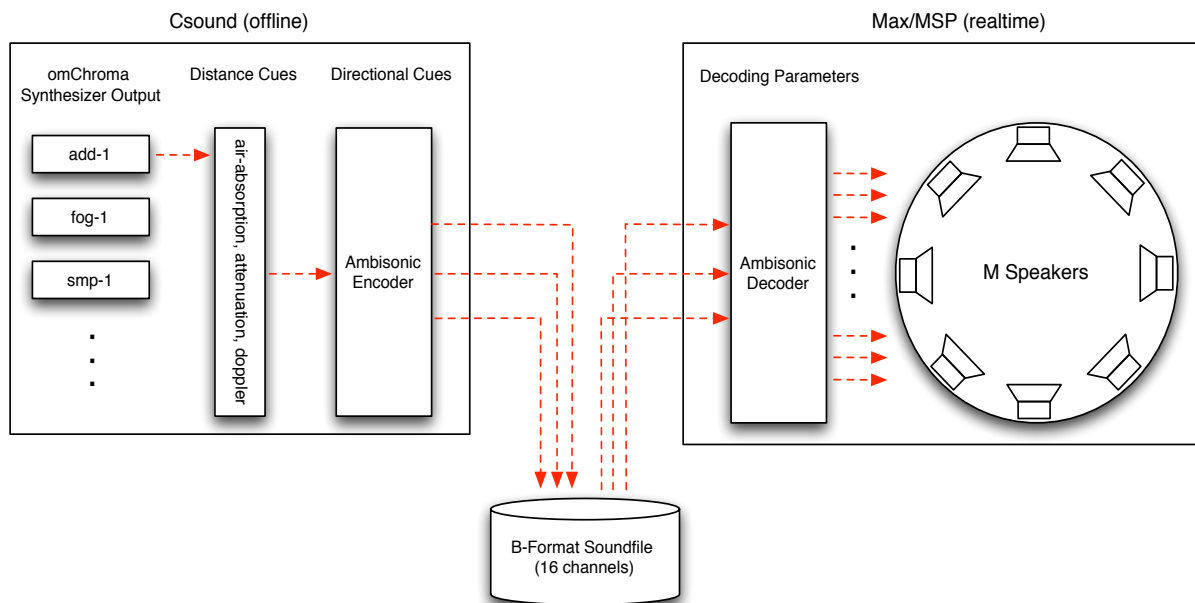
### 3.3. Implementation

OMambi is implemented following a general dsp-architecture shown in the figure below. Red arrows represent audio signals. The encoding and decoding process is performed independently by two dedicated applications.

On the left we see the Csound orchestra subdivided into 3 processing-modules. The leftmost module represents an existing synthesizer of the omChroma system, which is the sound source. The monophonic output of the synthesizer is first processed by dsp-units to account for distance cues (described in section 3.2) before it is encoded into 3<sup>rd</sup>-order periphonic Bformat<sup>1</sup>, and stored to disk as a 16-channel audio file. This is an offline-process controlled from within OpenMusic.

The decoding of this B-format file is carried out by a MaxMSP-based application in realtime. This allows for interactive tweaking of decoding parameters as described in chapter 5. The B-format file can theoretically be decoded to an arbitrary number of speakers<sup>2</sup>, however in the current implementation it is limited to a maximum of 32.

NB: Some classes have been implemented, which allow combine en- and decoding within the Csound orchestra for common speaker-layouts, like e.g. quadrophonic, 5.1 and stereo (stereo-files can be played back directly within OpenMusic). These classes are not used in the tutorials but can be found in the folder “/classes” in the OMambi workspace. Templates for building new classes for ambisonic en- and decoding within the Csound-orchestra are provided, please see Chapter 7 for instructions on how to build new OMambi-classes.



<sup>1</sup> For encoding into B-Format the Csound opcode “Bformenc1” by Richard Furse, Bruce Wiggins and Fons Adriaensen is used.

<sup>2</sup> as long as larger than  $(\text{order} \cdot 2) + 1$  in 2D, or  $(\text{order} + 1)^2$  in 3D

---

## 4. Control Structure

### 4.1 Matrix Representations

OMambi uses a matricial representation of control data –the omChroma matrix- and higher-level structures for the creation and manipulation of such matrices. An instance of an omChroma matrix represents a musical “event”. An event can be for example a single call of an instance of a dsp-program, a sequence or group of instances, or groups of sequences of instances, and so on and so forth. In other words, the semantics of an event can be arbitrarily defined<sup>1</sup>. Please note that a matrix may as well contain other matrices as cells, see e.g. class array-of-array in omChroma. Another important feature is that time is treated as a synthesis parameter which can be represented and controlled within the matrix as any other parameter<sup>2</sup>. As we shall see later this control-model provides a very powerful, efficient and ergonomic way of generating and controlling large data-sets. Due to the dynamic allocation of instances of a dsp-patch all parameters can be specified for each sound source/instance separately<sup>3</sup>. This contrasts to many spatialization systems in which certain settings are globally applied to the spatial sound renderer. Please refer to the omChroma documentation for more informations on the omChroma matrix.

### 4.2 Levels of control

Similar to the issue of controlling sound synthesis processes, spatialization algorithms can be controlled from different levels of abstraction. For example, similar to the perception of the *pitch* of a sound with a complex partial structure, spatial audition is a multidimensional phenomenon which depends on various perceptual cues. Generally speaking, spatial sound renderers try to fool the human auditory system by providing perceptual cues giving the impression that a *virtual* or phantom sound source is located at some *real* point in space (where it isn't).

Analogous to the synthesis of a harmonic sound which can be controlled from different levels of abstraction (for example by using a wavetable containing a complex waveform, versus a fixed bank of sine-oscillators, versus the multiple instantiation of a single sine-oscillator) perceptual cues that constitute the human spatial audition can be controlled in similar ways.

The musical use of spatialization is manifold; creative applications often manipulate perceptual cues in a way that does not correspond to the simulation of a physical sound source in physical space, e.g. by exaggerating certain perceptual cues and/or not employing other ones at all –as is for example often the case for doppler-effects. For this reason, and to provide a maximum of flexibility, OMambi allows the control of perceptual cues from different levels of abstraction. The figure below shows a diagram of OMambi's control structure with the three implemented levels which represent a continuum from efficiency to flexibility.

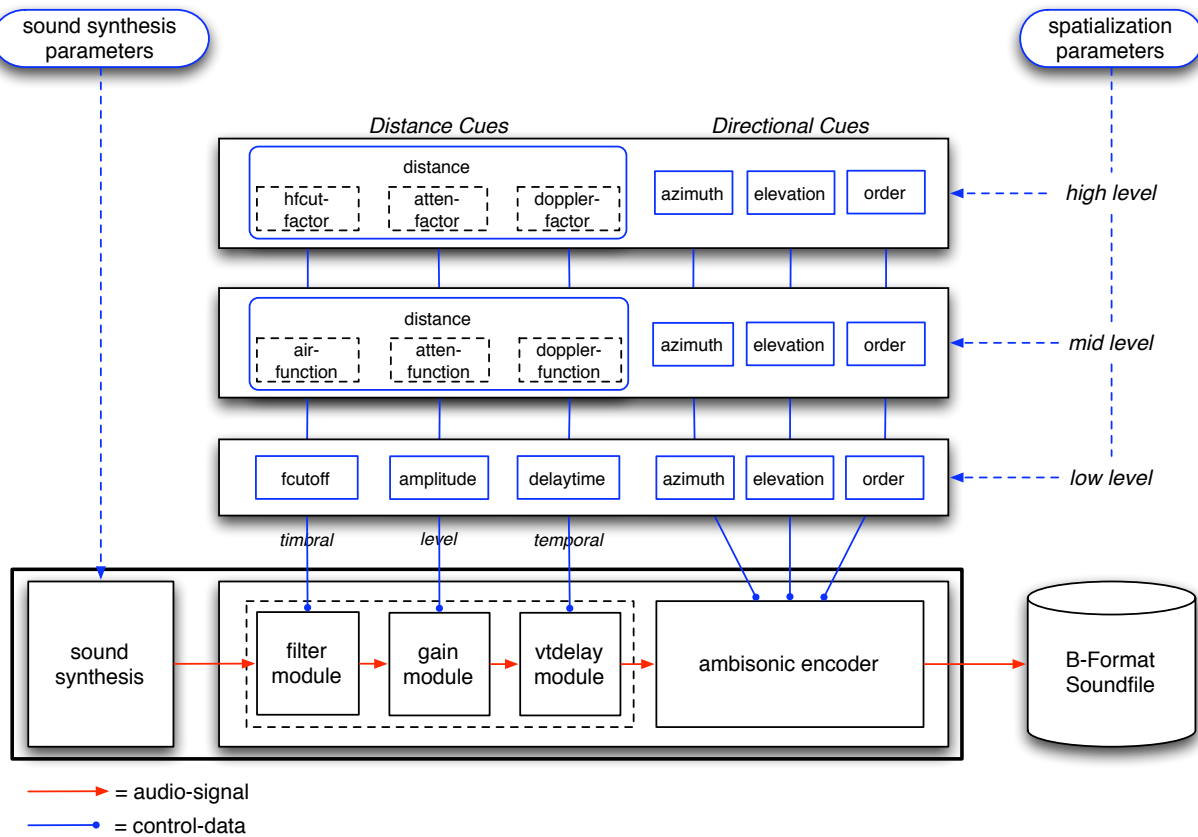
---

<sup>1</sup> This might be regarded analogouts to a note (as the smallest possible entity), a temporal sequence of notes (a melody), a group of notes (a chord), a chord-sequence, etc..

<sup>2</sup> It is worth mentioning here that global time can also be dynamically controlled via csound-“t-statements”.

<sup>3</sup> i.e. each sound source represents a ‘clone’ of the whole spatialization system





The black rectangle at the bottom represents the dsp-patch implemented in Csound (the Csound-orchestra). The (continuous) blue boxes represent control parameters. The dashed blue boxes represent parameters to tweak the functions controlling low-level parameters. For each sound synthesis module there are three OMambi-classes, corresponding to the different control-levels shown in the figure.

The control structure of the sound-synthesis module is left untouched (blue arrow on the left), i.e. the slots for the OMambi classes are the same as for the original omChroma class, please refer to the omChroma documentation for descriptions of the sound synthesizers. In order to control spatialization parameters from different levels, three control structures are available. Ambisonics is by its theory (assuming plane waves) a 2-dimensional spatialization system encoding the position of a sound source on a sphere, without any information on the sound source's distance. While directional cues<sup>1</sup> are controlled the same way for all 3 control-levels, the low-level parameters for controlling cues for distance-perception, i.e. the air-absorption filter, the gain-unit and the variable time delay can be controlled differently:

The classes for High-Level control are the most efficient in terms of dsp: The Csound-orchestra employs a set of built-in functions to calculate low-level parameters as a function of distance.

<sup>1</sup> Directional cues represent the 2-dimensional information of azimuth & elevation angle, plus the order of encoding

---

The behaviour of these internal functions can be tweaked via coefficients. However, there is no access to modify the mapping or editing the underlying equations.

For Mid-Level control the relationship between sound source distance and low-level parameters is also hard-wired into the Csound-orchestra. In this case, however, distance is used as an index for table-lookup of low-level parameters controlling the distance-cues of the signal before encoding. The lookup tables are provided in OpenMusic either locally via BPFs connected to the corresponding inlets, or globally as cs-tables to the function „synthesize“ -as is always the case for slots of type „cs-table“ in omChroma.

The Low-Level class allows the control of all low-level parameters directly. The equivalent functions to generate low-level parameters as used in the high-level Csound-orchestra can be found in the folder „/miscellaneous“ in the OMambi workspace. This implementation is the most flexible.

## 4.3 Implementation

### 4.3.1 High-Level

The inlets/keywords<sup>1</sup> of the matrix to control the position of a sound source in spherical coordinates are called „azimuth“, „elevation“ and „distanz“<sup>2</sup>. To control the level  $g$  of a sound source as a function of distance  $d$  two different functions are provided, for *inverse-proportional* and *exponential decrease*, respectively. The desired function can be selected via the inlet „atten-mode“, where „1“ corresponds to the inverse-proportional function and all other numbers to the exponential. The respective coefficients for tweaking the functions can be provided via the inlet „atten-factor“. See below for the implemented equations.

a) Inverse proportional decrease:

$$g_i = (d_i + (1-c_i))^{-q}$$

$c$  being the distance from the center/sweetspot to the speakers and  $q$  being the attenuation coefficient. By tweaking  $q$  the attenuation can be boosted or softened.

b) exponential decrease:

$$g_i = 10^{-k/20} * (d_i - c_i)$$

$c$  being the distance from the center/sweetspot and  $k$  being the attenuation coefficient. As with the inverse proportional decrease by changing  $k$  the attenuation is boosted or softened.

The theory of ambisonics assumes no physical distance between the microphones recording the periphonic soundfield. However, in real-life scenarios, there is always a distance between the

---

<sup>1</sup> These keywords correspond to the slots of the class. See omChroma1.0 Documentation.

<sup>2</sup> Since the English word “distance” is reserved for a compiled OM-function, the German equivalent “distanz” is used.

speakers used for reproduction. The speakers' membranes "represent" the counterpart of the microphone capsules used for recording the soundfield. The zone within the speaker-circle is referred to as the "center-zone", the radius is referred to as the "center-size". Theoretically speaking, a virtual sound source's position cannot be within this center-zone. However, for musical applications this is often a desideratum. Accordingly, the approach taken in OMambi decreases the encoding order within this center-zone linearly, reaching complete monophony at the very center. In order to compensate for the increase in loudness caused by the sound source's presence on all speakers, *and in order to avoid high amplifications for distances smaller than 1*, within the center-size a different attenuation law is applied:

$$g_i = (d_i * (1/c_i))^{-q} * (1-k) + k$$

**c** being the center-size, **q** being the attenuation coefficient and **k** being the maximum attenuation at the center in decibel. The attenuation coefficient **q** can be controlled via the inlet "center-curve", the maximum attenuation at the center **k** via the inlet "center-atten".

To control the cutoff-frequency **fc** of a 2nd-order butterworth filter as function of distance **d** to simulate the effect of air-absorption two functions are provided, *air-absorption-val* and *air-absorption-icst*, respectively. The desired function can be selected via the inlet „air-mode“, where „1“ corresponds to the *air-absorption-val* and all other numbers to *air-absorption-icst*. The respective coefficients for tweaking the functions can be provided via the inlet „hfcut-factor“. See below for the implemented equations.

a) *air-absorption-val*

$$fc(d) = -0.1668d^3 + 18.919d^2 - 785.71d + 15849$$

**d** being the distance from the center and **fc** being the cutoff-frequency in Hz.

b) *air-absorption-icst*

$$fc(d) = 22050 - d * k$$

**d** being the distance from the center, **k** being the air-absorption coefficient and **fc** being the cutoff-frequency in Hz. By tweaking **k** the effect of air-absorption can be softened or boosted.

The ambisonic encoding order can be controlled via the inlet "order" as a floating point number between 0 (=complete monophony) and 3.

### 4.3.2 Mid-Level

As with the high-level approach, there are three inlets to control the position of a sound source in spherical coordinates, again called „azimuth“, „elevation“ and „distanz“. However, this implementation uses a table lookup internally to control low-level parameters as a function of distance. Three lookup functions are implemented in the Csound-orchestra which use tables for lookup which are provided via the inlets „air-function“, „atten-function“ and „order-function“. The parameter „distanz“ is used as an index for table-lookup. For internal rescaling of lookup-indices the minimum and maximum value used for „distanz“ must be provided explicitly via the inlets „lookupmin“ and „lookupmax“. The ranges of the respective lookup-tables can be

---

provided via the inlets „airfmax“, „attenfmax“ and „orderfmax“. This can be done automatically in OpenMusic using the abstraction *get-list-ranges* which can be found in the folder /abstractions in the OMambi workspace. The “order-function” is relative to the maximum order given as a floating-point value via the inlet “order”.

### 4.3.3 Low-Level

For the low-level implementation the two-dimensional position of a sound source on a sphere is controlled via the inlets “azimuth” and “elevation”. The low-level parameters to control perceptual distance-cues are not implicitly calculated as a function of distance, but can be controlled explicitly. The inlet “hf-cutoff” controls the cutoff-frequency of the lowpass filter which can be used to simulate the effect of air-absorption. The inlet “attenuation” can be used to control the gain-factor (internally clipped to a range between 0 and 1) which can be used to account for distance-attenuation of a sound source. The inlets “1<sup>st</sup>-order-gain”, “2<sup>nd</sup>-order-gain” and “3<sup>rd</sup>-order-gain” provide explicit control of linear gains of the respective components of the resulting B-Format-files. Abstractions providing the equivalent functions as used by the high-level implementation internally can be found in the folder “/miscellaneous” in the OMambi workspace.

### 4.3.3 Continuous Control

For the classes allowing the continuous control of spatialization parameters, all parameters<sup>1</sup> are controlled via BPFs, i.e. the class’ slots are of type cs-table. These BPFs are internally translated into GEN07-functions (breakpoint-functions using linear interpolation) serving as wavetables for high-precision oscillators within the Csound-orchestra. For each of the envelopes the duration, as well as the minimum and maximum value can be provided<sup>2</sup>. The duration for one cycle of these envelopes can be set arbitrarily, e.g. for oscillations at audio-rate. Note, that this will most likely result in audible amplitude-modulations. If not specified, the envelope’s durations default to the duration of the azimuth-envelope, which defaults to the instance’s duration.

## 5. Ambisonic Decoder

The Decoding of the B-Format files is carried out by an external [Max/MSP](#)<sup>3</sup> based application, called “Multiplayer”. This application consists of a series of modified and custom-built dsp-modules complying with the [Jamoma](#)<sup>4</sup> framework. The figure below shows 3 different windows

---

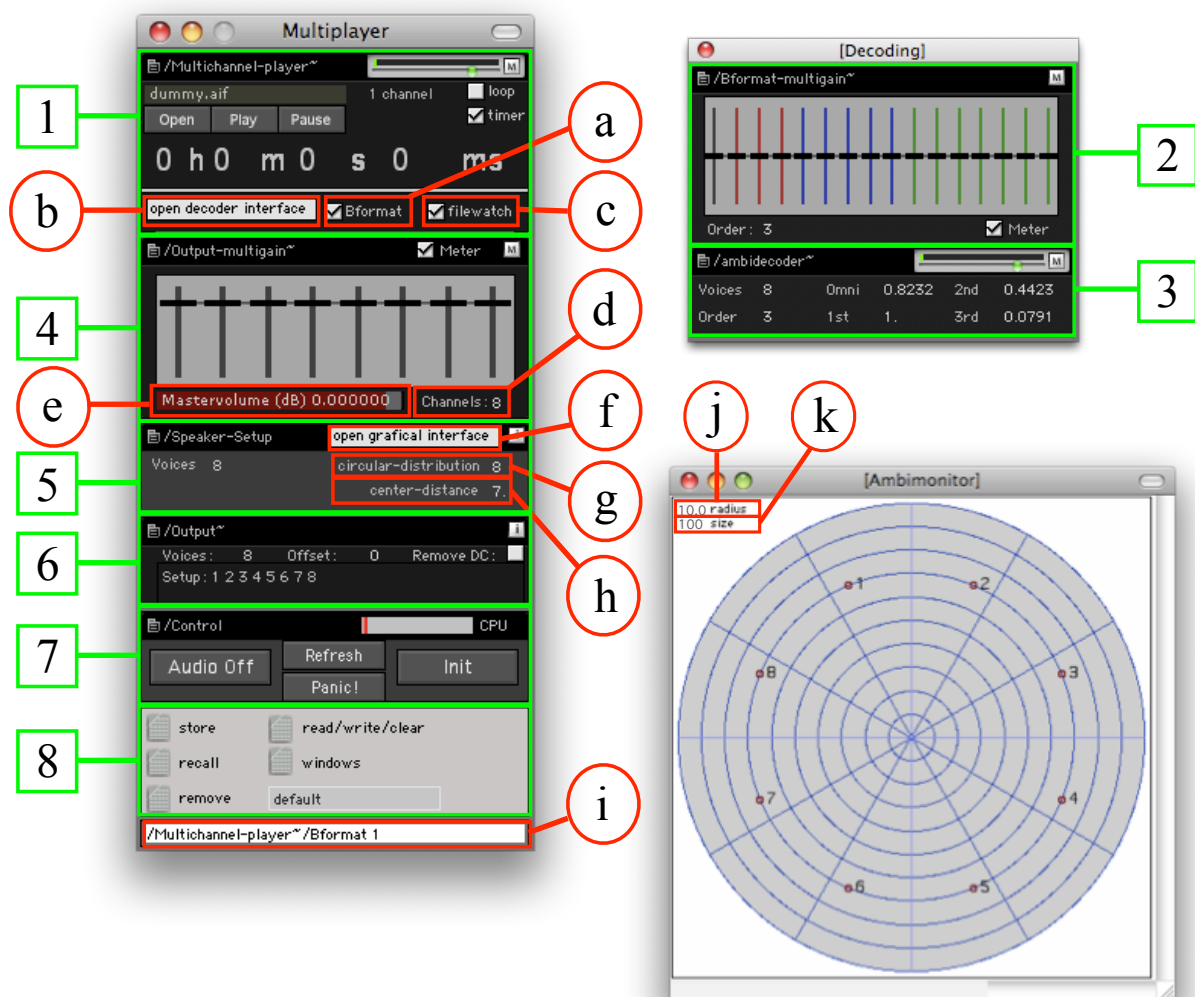
<sup>1</sup> Besides the selection of air-absorption and attenuation-functions for high-level control

<sup>2</sup> These parameters can be controlled either explicitly or implicitly using an abstraction to lookup the values from the BPFs.

<sup>3</sup> <http://www.cycling74.com>

<sup>4</sup> <http://www.jamoma.org>

of the application. Green squares identify modules, red-circles identify additional parameters not found in the original modules<sup>1</sup>.



## 5.1 Components

The following jamoma-modules are used within the application (in serial order):

- 1) jmod.ms.sur.multi.input~.mxt
- 2) jmod.sur.ambi.adjust~.mxt
- 3) jmod.sur.ambi.decode~.mxt
- 4) jmod.multigain~.mxt
- 5) jmod.ms.sur.speaker.setup.mxt
- 6) jmod.sur.output~.mxt
- 7) jmod.control.mxt

The [p.storage](#)<sup>1</sup> object is used for easy & flexible preset management and storage in xml-file-format.

<sup>1</sup> Due to the use of externals which haven't been ported to Max5 yet, the application is based on Max/MSP 4.6.3

---

Detailed description of the above software is beyond the scope of this document, please refer to the respective documentations on <http://jamoma.org>. The extended functionalities features used in the `multiplayer.app` are described here. Please note, that as with all jamoma-modules changes of parameters that cause the rebuilding of the dsp-chain (e.g. scripting auf dsp-objects) take effect only next time audio is switched on.

#### `jmod.ms.sur.multi.input~.mxt`

This module has been provided with the additional functionality of routing the multichannel-signal to two different outputs. Additionally, a “filewatch” feature has been implemented.

Checking the box next to “Bformat” **(a)** in the figure) routes the signal to the ambisonic decoder. When unchecked, the multichannel signal is routed directly to the `jmod.multigain~` module without any processing, which is useful for auditing regular multichannel files.

Clicking on “open decoder interface” **(b)** opens an additional window (shown on the top right of the picture), containing modules for ambisonic decoding (`jmod.sur.ambi.adjust~` and `jmod.sur.ambi.decode~`) the functionality of these modules has not been changed, i.e. they are the same as in the original jamoma-distribution and are documented there. `jmod.sur.ambi.adjust~` uses a the Max/MSP external “ambidecode~”, a single-band 3<sup>rd</sup>-order ambisonic encoder, documentation can be found [here](#)<sup>2</sup>.

Checking the box next to “filewatch” **(c)** activates the filewatch feature: Whenever the currently loaded file is changed by an external application (this is a common case, e.g. when working on a synthesis-process in OMambi and rendering a file with the same name), the application automatically re-loads this file.

#### `jmod.multigain~.mxt`

The `jmod.multigain~` module has been implemented for easy visual monitoring and adjustment of levels (linear gains) of individual channels of a multichannel audio stream. Use the number box **(d)**. If the audio stream contains more channels than the set number, the higher channel numbers are ignored. The graphical/numeric slider **(e)** can be used to control the volume of all channels simultaneously (in decibels), i.e. controlling the master volume.

#### `jmod.ms.sur.speaker.setup.mxt`

In order to provide an intuitive way for setting and tweaking of speaker positions, the original `jmod.sur.speaker.setup` has been extended with a graphical user-interface. Additionally, the functionality of the jamoma module `jmod.speaker.delay~` has been integrated into the module, i.e. time-delays to compensate for differing speaker distances are automatically applied without the need of additional user-interaction.

Further, it is now possible to automatically generate circular speaker setups in the horizontal plane following the convention starting with the first speaker left of the front-middle, and enumerating clockwise. Coordinate systems are described in the Appendix of this document. The number of speakers is set using the number box next to “circular distribution” (marked **(g)** in the

---

<sup>1</sup> <http://www.pelado.co.uk/2006/05/23/pstorage/>

<sup>2</sup> <http://www.icst.net/downloads/>

---

figure). The radius of the circle of speakers can be set via the number box next to “center-distance” (marked **(h)** in the figure). Please note that speaker-numbers exceeding the number of voices will be ignored.

Clicking on “open graphical interface” opens an additional window showing a bird’s eye perspective on the listening space. This interface uses the Max/MSP external “ambimonitor” for Max/MSP4.6.3, which can be found in the same software distribution as the external for ambisonic decoding mentioned above. Please refer to the same resource for documentation. The interface is fully integrated into the module’s namespace. The distance-units (blue circles) are normalized to meters. The “radius” parameter **(j)** sets the viewing radius for “zooming” in and out. The “size” parameters **(k)** resizes the whole window (in screen pixels).

## 5.2 Quickstart

A general remark: For ease of use the number of channels of all modules subsequent to *jmod.sur.ambi.decode~* are automatically adjusted to the number set for “voices”. If wished this can be overridden using controls of the respective modules.

1. Drag’n’drop a Bformat file from MacOS’ finder onto the time-display of the Multichannel-player and use the controls for playback as described in the jamoma documentation of the module *jmod.input~*.
2. Check the box next to “Bformat” **(a)** to route the signal from the Multichannel player to the decoder and –if desired- check the box next to “filewatch” **(c)** to activate the filewatch-feature. Click on “open decoder interface” **(b)** to open the window containing UI-elements for tweaking decoder settings, common presets are provided with the application (e.g. coefficients for in-phase decoding in different orders).
3. Choose one of the presets for decoding from the *jmod.sur.ambi.decode~*’s preset-menu or set the parameters manually. Set the number of desired output-channels for “voices” in module. Use the module *jmod.ms.sur.speaker.setup.mxt* to configure your speaker setup via the parameters “circular distribution” **(g)** and “center-distance” **(h)** and/or tweak them manually using the graphical interface **(f)**. As with all jamoma-modules settings can be stored as module-specific presets. Using the p.storage object **(8)** multiple settings of the entire application can be stored and later saved conveniently in an xml-file.

## 6. Tutorials

### 6.1. Basic

The purpose of the basic tutorials is to give an introduction into basic methods of controlling spatialization with OMambi. Classes for discrete control of spatialization parameters are presented and basic concepts of spectral vs. granular spatialization are introduced. For sake of clarity (and since there currently is no 3D-editor in OM) only 2D- (planar) spatialization is used.

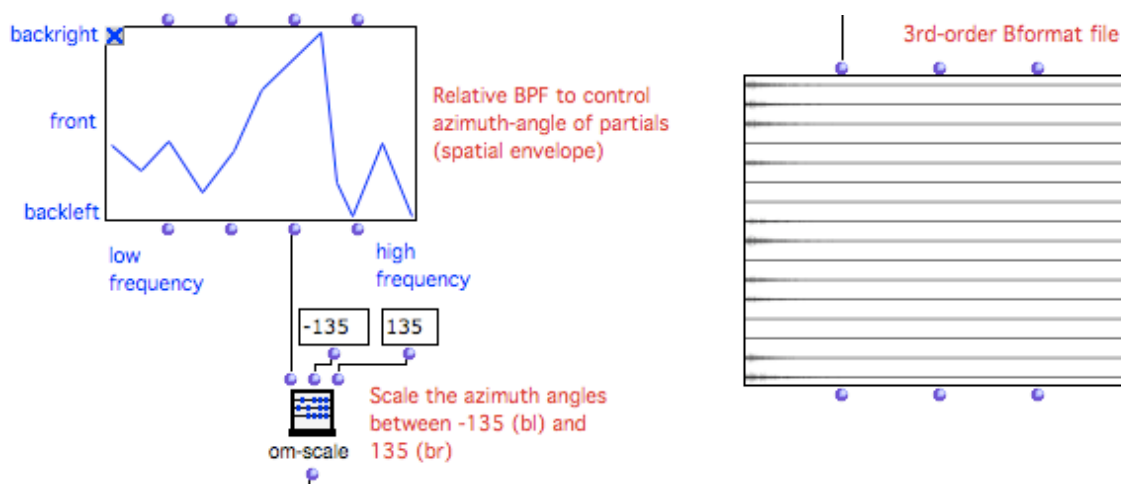
---

A note before starting:

OMambi renders periphonic 3<sup>rd</sup>-order B-Format files (16 audio channels), which are displayed properly in OM's sound-object. For playback/decoding of these files however, an external application is needed –e.g. the “multiplayer” which is described in chapter 5. The rendered soundfiles can be found in the directory set in OM's Preferences in “/Default Folders/Output Files”. By default the directory is “/out-files”, a sub-directory of your workspace.

### 6.1.1 Tutorial 1: spatial-risset-bell

Open the patch “1-spatial-risset-bell”. This first tutorial is a simple introductory patch using an adaption of the original omChroma tutorial “cs\_tut03” for additive synthesis of a bell-like sound taken from the Csound catalog. This is a simple example for spatial additive synthesis using the class *add-lambiencl-1*, which is an ambisonic version of the omChroma class *add-1*: Each partial is assigned a distinct panning position via a OpenMusic's breakpoint-function (BPF). Note, that the panning positions are not a function of frequency since the BPF is sampled across the number-of-rows of the matrix<sup>1</sup>, but a function of the frequency's index in the list-of-frequencies. Since the frequencies are sorted in an ascending fashion the BPF can be regarded as a 2D-plot representing frequency on the X-axis and azimuth angle on the Y-axis, similar to a “spatial envelope” as shown in the figure below. After evaluating the sound object at the bottom of the patch it should look like the right figure below. Note that –although displayed properly– the file can't be played back using the sound-object. Instead, use the multiplayer application for playback as described in chapter 5.



### 6.1.2 Tutorial 2: additive-spatial-spread

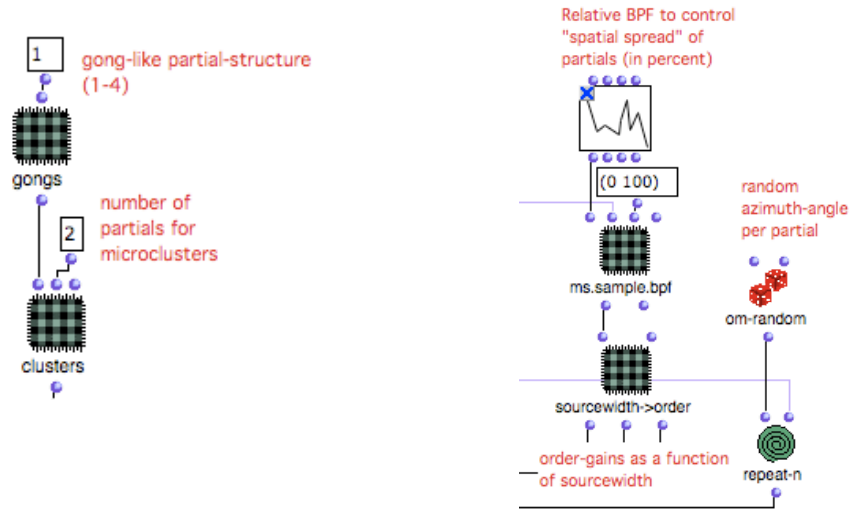
This tutorial allows for selecting a gong-like partial structure from a database via the argument to the abstraction “gong” at the top of the patch. The abstraction “clusters” generates micro-clusters

---

<sup>1</sup> this is the typical behaviour of the omChroma-matrix for a slot of type number

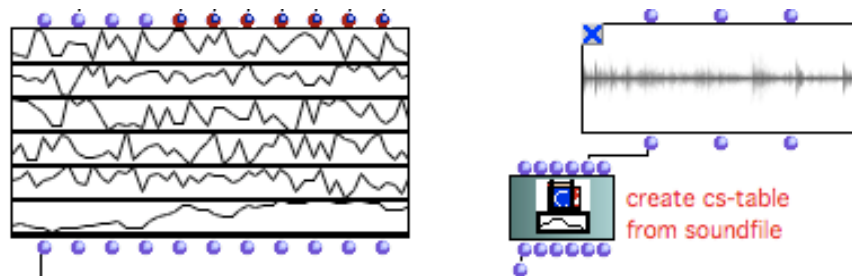


around each partial the number of partials for each cluster, as well as the frequency deviation can be set arbitrarily as shown in the left figure below. The azimuth-angle of each partial is given by a random-function. In addition to the control of panning-positions, this tutorial demonstrates an approach for controlling the spatial spread (or “sourcewidth”) of sound sources by controlling the gain of components of the respective ambisonic order on a scale from 0 to 100%: At 0<sup>th</sup> order, or a spatial spread of 100% the sound-source is omnidirectional<sup>1</sup>, whereas 0% corresponds to the highest possible accuracy of the system, that is 3<sup>rd</sup> order in OMambi. The right figure below shows how a relative BPF is used to control the ambisonic order as a function of “spatial spread” (one-to-many mapping) which is embedded in the subpatcher “sourcewidth->order”. Please note that this control structure is independent of the number of partials, as e.g. for spatial additive synthesis using thousands of partials.



### 6.1.3 Tutorial 3: fog-stochastic

Tutorial 3 demonstrates the use of various algorithmic (stochastic) functions to generate the control-data for a spatial fog-synthesizer<sup>2</sup>: the class *fog-lambiencl-1*. Here the azimuth-angle of each grain is controlled via a random-walk with a maximum step size of 45. degrees. The matrix object below shows a nice visualization of the various stochastic functions used to control the class' parameters, the lowest row is the random walk. Also note, that the source soundfile used for fog-synthesis can be provided using the leftmost outlet of OM's sound object, however, must be converted into a global cs-table which is written into the csound score which is read by the Csound-orchestra.



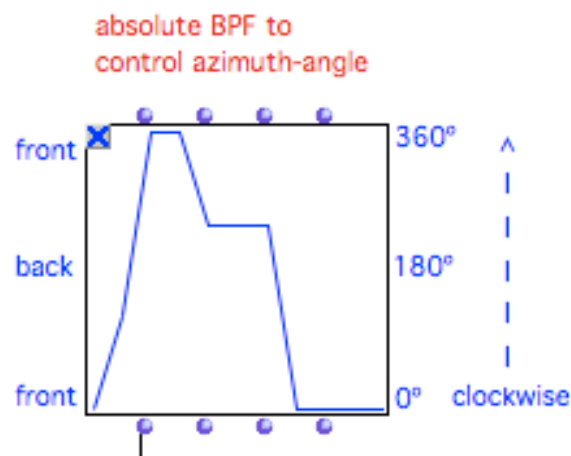
<sup>1</sup> Please note that this has an increase in overall amplitude as a side effect, since more loudspeakers are contributing.

<sup>2</sup> “fog” stands for “forme d’onde granulaire”, an extended type of formantic waveform synthesis using arbitrary soundfiles instead of sinusoids.

---

### 6.1.4 Tutorial 4: *smpl-basic*

This tutorial is a basic example for sample-based spatialization, i.e. panning of a sound file using the class *smpl-lambiencl-1* which is controlled exclusively with absolute BPFs: Please note that in contrast to Tutorial 1 the BPF used to control azimuth is an absolute BPF which is not rescaled and contains values from 0 to 360 (representing azimuth-angle in degrees). Thus the Y-axis represents a clockwise circle starting in front of the listener. The soundfile can be provided using an absolute or relative path, or by connecting the leftmost outlet of the sound-object with the slot called “filename”.

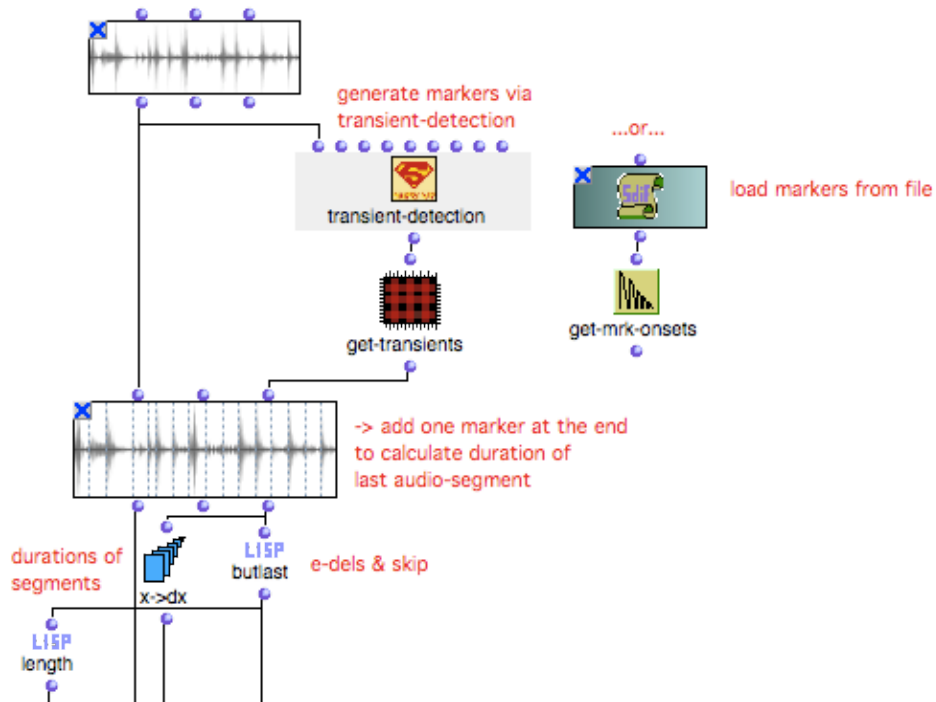


### 6.1.5 Tutorial 5: *smpl-granular*

In this tutorial we'll see how OMambi seamlessly integrates with OpenMusic's audio-processing functionalities such as SuperVP and OMsounds. An audiofile (a drumloop) is analyzed for transients using OM-SuperVP to generate markers. Alternatively, markers can be loaded from an SDIF-file. By retrieving the marker-positions from the sound-objects rightmost outlet, entry-delays, skips (offsets within the soundfile) and durations are calculated. For each of the audio segments (audio between two successive markers) a random panning-position (azimuth between -180 and 180 degrees) is calculated, resulting in a rhythmic spatial distribution of drumsounds. This patch can be regarded as a simple example for “granular spatialization”<sup>1</sup>. Please note that an additional marker should be added at the end of the analyzed sound file in order to calculate the duration of the last segment as shown in the figure below.

---

<sup>1</sup> The term “granular spatialization” is used to describe a technique in which the spatial position of each grain is controlled independently.



### 6.1.6 Tutorial 6: distance-cues

Up to this tutorial OMambi was used merely for one-dimensional panning purposes. For rendering of sound source positions with different distances, additional perceptual cues need to be provided. This tutorial presents three classes for the control of distance-cues from different levels of abstractions as described in chapter 4. In this tutorial all three classes produce exactly the same results. The audio material used for this tutorial is a transient-rich cymbal sound.

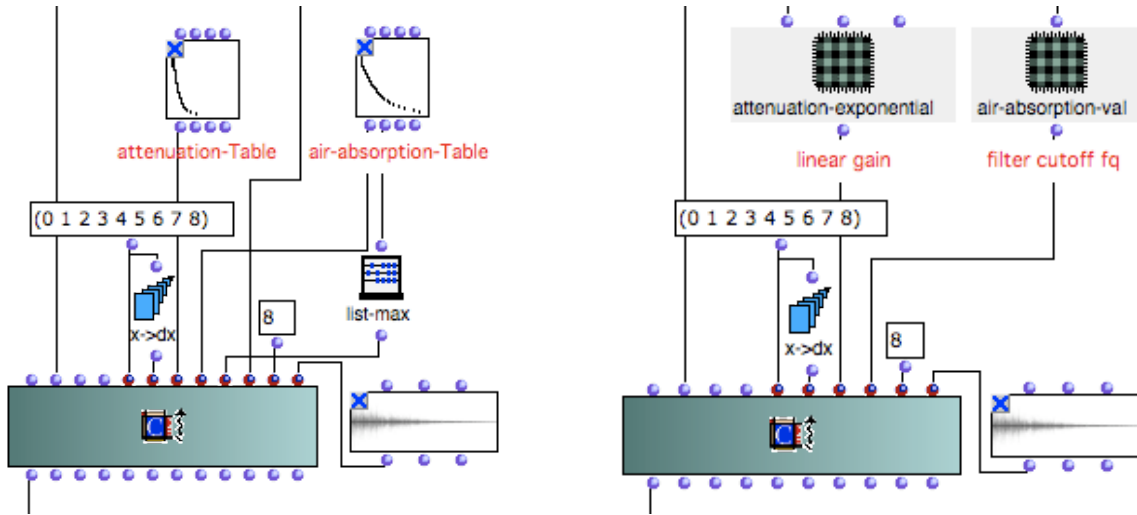
The high-level class calculates low-level parameters to control distance-cues internally as a function of distance. Two functions for distance-attenuation, as well as for air-absorption can be selected. The behaviour of these functions can be tweaked via two respective coefficients. Chapter 4 gives detailed descriptions of this class' control-structure, see the appendix for a list and a description of the matrix' inlets.

The mid-level class also uses internal functions to generate low-level parameters as a function of distance, thus analogous to the high-level class this relationship is hard-wired into the instrument. However, the behaviour of these functions can be arbitrarily defined: External function-tables -that is BPFs in OpenMusic- are used to lookup the respective low-level parameters as a function of distance. This provides an additional level of flexibility. Please refer to section 6.3.2 for instructions on the generation of BPFs from user-defined functions.

The low-level class provides access to all underlying low-level parameters for controlling distance-cues explicitly. In this tutorial user-defined functions (the abstractions "attenuation-

exponential” and “air-absorption”<sup>1)</sup> are employed to generate control-data for low-level parameters as a function of distance. Please refer to section 6.3.1 for more information on distance-functions. This is both the least efficient and the most flexible solution.

Due to the dynamic allocation of instances all parameters can be changed for each sound source separately independently of the control-level. The figure below shows the 2 BPFs generated from the two functions used for the low-level class for distances from 0 – 50 meters. Note, that for the mid-level implementation the maximum distance is automatically calculated using OM’s compiled function “list-max” .



### 6.1.7 Tutorial 7: smpl-bpc

This Tutorial shows a graphical, 2-dimensional representation of sound source positions in cartesian coordinates using a *BPC-factory*<sup>2</sup>. Each breakpoint in the BPC corresponds to a position in the horizontal (X/Y) Plane. The conversion utility “bpc2ad” is used to convert the representation in cartesian x/y coordinates into polar azimuth/distance coordinates<sup>3</sup>. The sound material is the same as in Tutorial 6. In addition, this tutorial shows yet another example for the control of distance-cues using a low-level class: Lookup-tables (instances of BPFs) are used within the OpenMusic patch for control of the low-level parameters of a low-level class. This is equivalent to the method in the mid-level class. Please note that time delays must be accounted for as positive offsets for entry-delays (the abstraction “distance-delay”). The figure below shows a manually vs. an algorithmically created BPC and the abstractions used for table-lookup in OM.

<sup>1</sup> These are exactly the same functions as implemented in the csound-orchestras/OMambi-classes for high-level control.

<sup>2</sup> OpenMusic’s breakpoint-curve editor

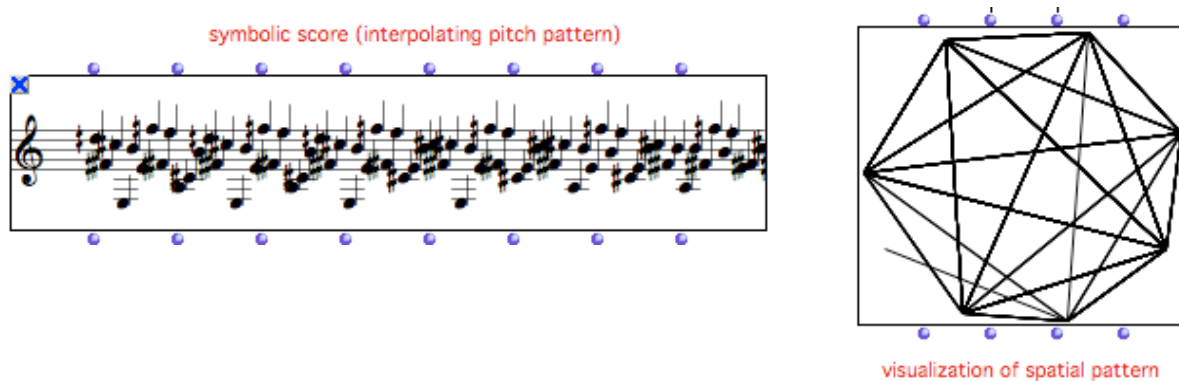
<sup>3</sup> The folder “/abstractions” in the OMambi-workspace contains various utilities for the control of spatialization, like conversions, mappings, etc.



---

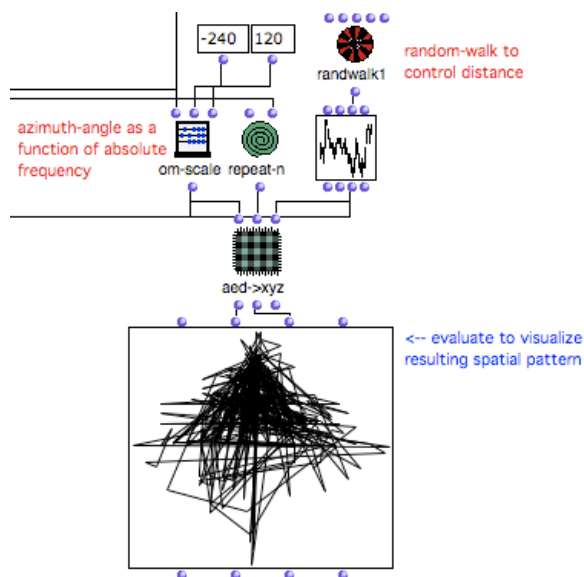
## 6.1.9a Tutorial 9a: smpl-chordseq

Since OpenMusic is a CAC-environment using symbolic representations of musical objects in common music notation it makes sense to use these representations for the control of spatialization. The following 2 tutorials are examples for the use of a *chord-seq* -a symbolic representation of a musical score- as a control structure for granular spatialization. The sound material is the same as in the previous tutorial. For each note in the chord-seq audio segments are successively “picked” from the source soundfile. This tutorial nicely demonstrates the benefit of a feature of the omChroma-matrix which re-iterates a list from the beginning in a loop, if the list’s elements are less then the matrix’s components. This is the case for the lists provided at the inlets “skip” and “duration”. Each segment is transposed as a function of the corresponding note’s pitch. The panning positions (azimuth-angles) of each audio segment are determined as a function of pitch-classes with 1/8 tone resolution (i.e. there are 48 pitch-classes in an octave). This results in a conversion of the pitch-pattern in the chord-seq to a spatial pattern. Additionally, the segment’s durations are compressed/stretched according to the respective transposition-factor, which creates temporal variations and overlapping sounds, giving a very lively, natural feel to the spatialized sound pattern. The two figures below show the chord-seq (the pitch pattern) and the resulting spatial pattern.



## 6.1.9b Tutorial 9b: smpl-segmentation

This tutorial presents a different method for controlling azimuth-angles as a function of pitch using absolute frequencies (in Hz). In addition, a random-walk is employed to control the sound source distances, resulting in a much more complex spatial pattern which is shown in the figure below. The orientation of the pattern can be controlled by tweaking the arguments of the *om-scale* object. Further, in this tutorial different patterns for picking audio segments from the source soundfile can be selected.



## 6.2. Advanced

The advanced tutorials present more sophisticated applications using continuous control of spatialization parameters. Drawing of Trajectories, Time/Space-Stretching and the concept of the parsing-fun are presented.

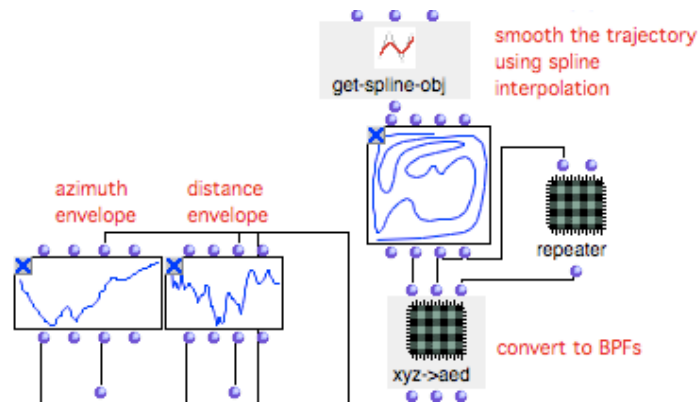
### 6.2.1 Tutorial 1: continuous-bpc-trajectory

This tutorial introduces the control of sound source positions using BPCs to describe a continuous trajectory, in a certain perspective analogous to a “spatial glissando”: A manually drawn BPC is smoothed out using the function *get-spline-object* via spline interpolation to remove the edges<sup>1</sup>. The resulting smoothed BPC represents the trajectory of a sound source in cartesian coordinates on the horizontal plane. The abstraction “xyz->aed” converts this representation from cartesian coordinates into polar coordinates<sup>2</sup>. The BPC’s X/Y coordinates are converted into 2 BPFs, the values for azimuth and distance are represented on the respective Y-axes, the X-axes are a relative representation of the total duration of the sound file, i.e. the BPFs are relative azimuth/distance envelopes over the duration of the instance which is given by the soundfile. Since the two-dimensional BPC doesn’t contain any temporal information, the list of Y-coordinates in the BPFs is assigned to equidistant values on the X-axis, i.e the x-deltas are constant. Accordingly, the speed of the sound source travelling along the trajectory is proportional to the euclidean distance between two successive breakpoints in the original BPC, or inversely expressed the time of travel between two points is a constant determined by the duration of the soundfile divided by the number of breakpoints. Note, that this is an arbitrary choice. For example, by extracting coefficients via the euclidean distance between two points in the BPC and applying these coefficients to the x-coordinates of the BPFs when converting, a

<sup>1</sup> Depending on the desired smoothness this function can be tweaked using the parameters (“resolution” and “degree”).

<sup>2</sup> Note that since there is no elevation-data in the BPC, a value of “0” is repeated for as many breakpoints as the trajectory contains. This is carried out by the abstraction “repeater”.

steady travel speed along the trajectory could be achieved. The figure below shows the 2 BPFs obtained from the smoothed BPC.



### 6.2.2 Tutorial 2: granular-board-trajectory

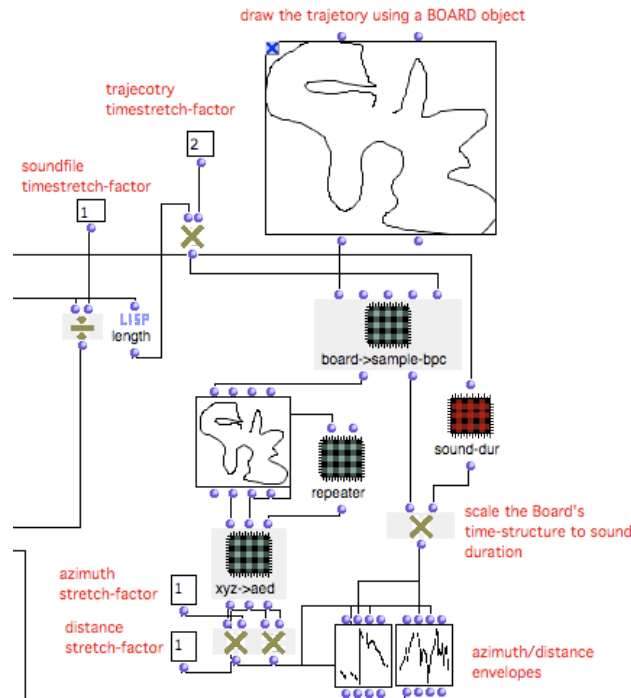
This tutorial introduces a different concept for the control of a sound source travelling along a trajectory: A granular representation using many discrete positions, in a certain perspective analogous to a “spatial arpeggio”. Additionally, a new interface object is presented to draw the trajectory: the *Board* object. Please note that this tutorial uses the class *simpl-lambiench-1* which doesn’t provide for continuous control of spatialization parameters. The subpatcher “e-dels” generates a list of entry-delays with a stepsize of 50ms until the source soundfile’s total duration is reached. The duration of each grain corresponds to the deltas between two successive entry-delays. A windowing-function is obtained by adding 25ms to the duration of each grain within the grain is faded-out using a sigmoid function which overlaps with the subsequent grain which has a 25ms sigmoid-fade-in. Analogous to the BPC in the previous tutorial the Board represents the sound source’s spatial trajectory in the horizontal (x/y) plane. It is rescaled and converted into a BPC and then resampled into a sequence of discrete positions at a rate determined by the soundfile’s duration divided by the grainsize.

### 6.2.3 Tutorial 3: granular-time-space-stretching

This tutorial shows a simple approach to manipulate the spatio-temporal morphology of a trajectory via granular time-stretching of either the sound-file, or the trajectory (temporal and spatial). In contrast to the 2-dimensional data contained in a BPC, the Board object represents time (the “speed” of drawing-motions) on a normalized scale (0-1) as a third dimension. This temporal structure is rescaled to the duration of the soundfile within the subpatcher “sound-dur”. Analogous to the previous tutorial the Board is converted to BPFs. The difference here is, however, that the X-values of the azimuth/ distance envelopes are not equidistant but given by the temporal structure of the trajectory in the Board object. By increasing or decreasing the number of samples taken from the trajectory, the incremental step (“spatial samplerate”) is changed since the trajectory is always sampled from beginning till end. Since the e-dels of the grains aren’t affected this results in a temporal stretch of the spatial trajectory. Note that the

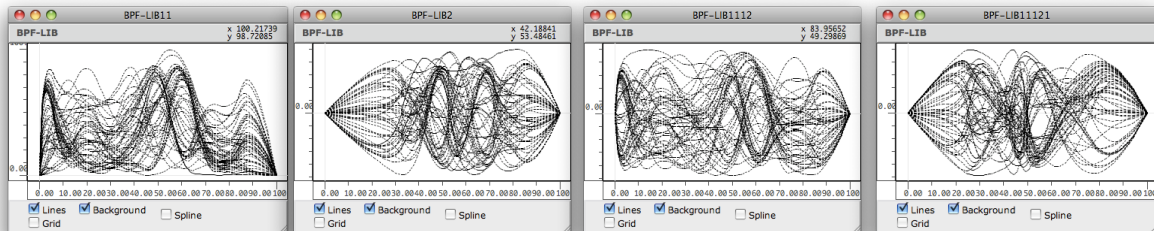


omChroma-matrix ignores elements exceeding the number-of-components and repeats a list from the beginning in case its elements are less than components. This control-paradigm is analogous to the continuous control of sound source positions with OMambi using BPFs. The soundfile, as well as the trajectory's temporal and spatial structure can be stretched independently of each other as shown in the figure below.



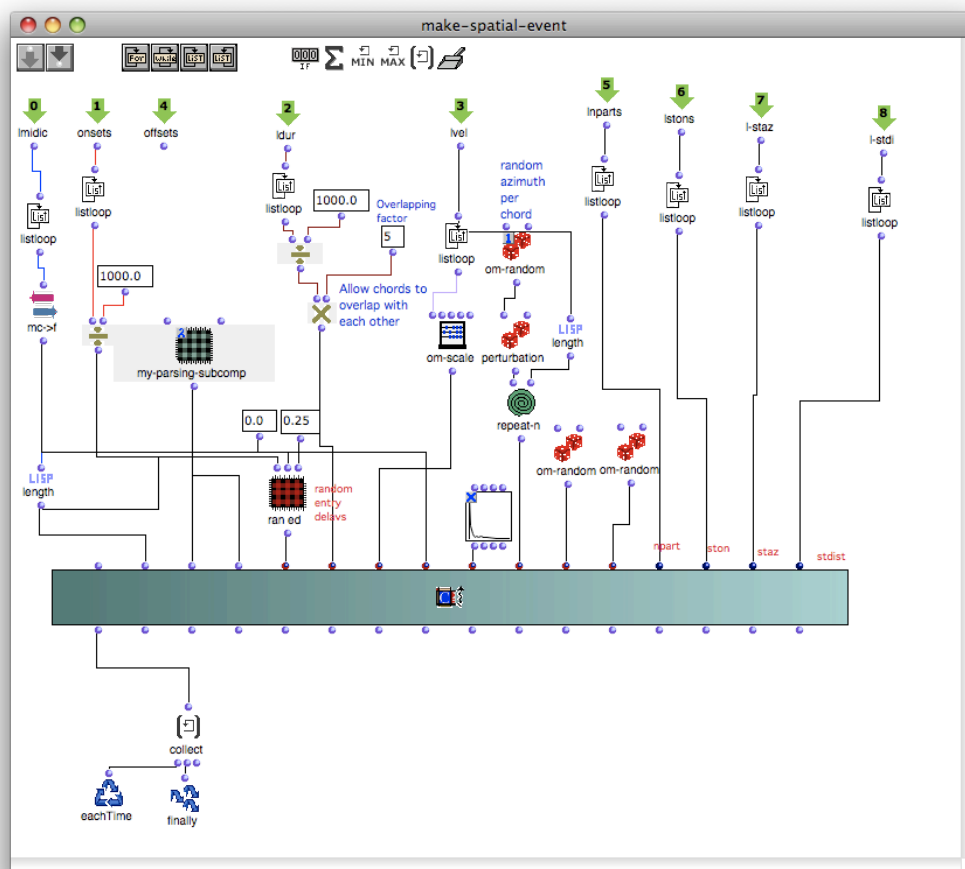
## 6.2.4 Tutorial 4: continuous-risset-bell

This tutorial is an adaptation of the Basic Tutorial 1 in section 6.1.1. Relative BPFs are employed to control a sound's spectro-spatial morphology, that is the temporal evolution of partials in space. Amplitude-envelopes, frequency-envelopes, azimuth-envelopes and distance-envelopes of the sound's partials are controlled via 4 main BPFs. The abstraction "perturb-bpf->bpf-lib" generates a perturbed variation of the original BPF for each partial. The first and last Y-values, however, remain unchanged (since otherwise in some cases clicks might occur, e.g. when used for example for amplitude-envelopes). The loop *get-splines* is used to smooth the BPFs. Note that the relative BPFs are re-scaled to the default ranges set for the respective parameters in the class definition. For each evaluation of the patch a slightly different spectro-spatial morphology of the bell-sound is obtained, yet every unique realization belongs to the same "family" sounds. The figure below shows 4 BPF-libraries obtained from the same evaluation of the patch for amplitude-envelopes, frequency-envelopes, azimuth-envelopes & distance-envelopes (left-to-right order).



### 6.2.5 Tutorial 5: parsing-fun-fineberg

This tutorial is an adaptation of the original omChroma tutorial `cs2_tut07` to demonstrate the use of a parsing-fun for the control of spatialization parameters in a similar fashion as in the original tutorial partial frequencies are controlled. Please refer to the omChroma tutorial for a general description of the concept of the “parsing-fun”. As in the omChroma example the basic material is a chord-seq used in an ensemble piece of Joshua Fineberg. A new chord-seq is generated by time-stretching and transposing the original chord-seq. For each of the chords in this new chord-seq one matrix is generated inside the abstraction “`cs-ambi-events-list`”. Each chord is assigned a random position in 2D-space with an azimuth-angle between -180 and 180 degrees and a distance between 0.5 and 5 meters. Also the encoding order is given as a random number between 0.5 and 3, which translates into perceived “spatial spread”. For each note within this chord an azimuth-value with a slight deviation (max 20%) from the chord’s main position is calculated. The abstraction “`my-parsing-subcomponent`” generates a micro-cluster of other partials around each note’s frequencies, with random e-dels between 0 and 0.25 seconds. The frequential deviation is determined by a BPF which is sampled over the number of matrices (or in this case: chords). The BPF’s range is given by the “`min stonatura`” and “`max stonatura`” parameters at the top-level of the patch. In a similar fashion for each of the cluster’s frequencies a new spatial position is computed, the BPFs ranges, that is the max. spatial scatter of the cluster’s-partial is given by “`max azimuth deviation`” and “`max distance deviation`” at the top-level of the patch. Again, for each evaluation of the patch a different sound will be obtained. The figure below shows the loop “`make spatial event`” containing the matrix inside the abstraction “`cs-ambi-events-list`”.



---

## 7. Building OMambi classes

### 7.1 Csound-orchestras and code-snippets

The Csound source code used within the OMambi-classes can be found in the folder “/resources/sourcecode/OMambi” in the OMambi workspace. The subfolders “/additive”, “/granular” and “/sample-based” contain the respective Csound code. When combining synthesizers with ambisonic encoders large parts of the code are redundant and can be re-used. Thus, it is very economic to work with code-snippets which can be copied and pasted using a text editor. The subfolder “/Template” contains the code-snippets which were used to implement the default OMambi classes. These code-snippets can be used to create new classes as described in the following section.

### 7.2 Merging synthesizers & ambisonic encoders

This chapter describes how to build your own OMambi classes in 5 steps. Please note that in future releases of OMambi an object-oriented approach will be taken in which spatial synthesizers are created automatically via multiple inheritance from sound synthesis- and spatialization-superclasses, respectively. See Chapter 8 on Future Developments.

1. Choose the omChroma synthesizer you would like to use as a sound source for ambisonic spatialization. You can basically use any given synthesizer as long as its output monophonic (otherwise it needs to be modified slightly). Lets choose the omChroma-class buzz-1 for sake of clarity. You can find the Csound sourcecode of the omChroma classes in “/userlibrary/chroma-classes/sources/classes/basic/mono” within the folder containing your OpenMusic application, by default this is “/Applications/OM6.x.x”. Open the csound code with a text-editor. The Csound code should look somewhat like this:

```
;=====
;          BUZZ-1.ORB
; DYNAMIC SPECTRUM OSCILLATOR (FROM ACCCI, 43_21_1.ORB) / MONO
; AMPLITUDE ENVELOPE WITH POSCIL
;=====

; Timbre:    Various controlled noise spectra
; Synthesis: (g)buzz
;           POSCILI envelopes
; Coded:     jpg 8/92, modified ms 9/02, 8/08

; NB: NEW STRUCTURE FOR THE AMPLITUDES FROM AUGUST 2008!
;   Positive value > 0.0 : linear amplitude (>0.0-1000.0)
;   0.0 or negative value : amplitude in dB (0 = maximum value)

; The apparently arbitrary amplitude range (0-1000, rather than 0-1)
;   avoids Lisp printing small values with exponential notation

; Replaced oscili with poscil (precise oscillator), ms 8/08
; Default SR = 96000, recommended precision: 24 bits
;-----

;   p1      = instrument number
```

---

```

;   p2   = action time [sec]
;   p3   = duration [sec]
;   p4   = maximum amplitude [linear, >0.0-1000.0 or dB, <= 0.0]
;   p5   = fundamental frequency [Hz]
;   p6   = amplitude envelope [GEN]
;   p7   = lowest harmonic present in the buzz [int]
;   p8   = % of maximum possible harmonic present [0-1]
;   p9   = multiplier in the series of amp coeff for the buzz [0-1]
;   p10  = envelope for the multiplier [GEN]
;-----
; COMPULSORY GEN FUNCTIONS
;   f5   large cosine
;-----

; CLASS: BUZZ-1

; GLOBAL KEYWORDS (default values within parentheses):
;   NUMROWS      : amount of rows (components) in the event (1)
;   ACTION-TIME  : start time of the whole event [sec] (0.0)
;   USER-FUN    : user-defined parsing function (nil)

; LOCAL KEYWORDS:
;   E-DELS :    entry delays [sec] (0.0)
;   DURS   :    duration [sec] (1.0)
;   AMP    :    amplitude [lin, >0.0-1000.0 or dB <- 0.0] (-6.0)
;   F0     :    fundamental frequency [Hz] (220.0)
;   AENV   :    function number for the amplitude envelope [GEN] (triangle)
;   BZL    :    lowest harmonic present in the buzz [integer] (1)
;   BZH    :    % of maximum possible harmonic present [0-1] (1)
;   BZM    :    multiplier in the series of amp coeff [0-1] (0.95)
;   BZMENV :    function number for the buzz envelope [GEN] (triangle)
;*****

sr    = 96000
kr    = 96000
ksmps = 1
nchnls = 1

;0dbfs = 32767 ; 16 bits
0dbfs = 8388697 ; 24 bits

instr 1; *****
idur      = p3
idurosc   = 1/p3
iamp      = (p4 > 0.0 ? (p4*0.001*0dbfs) : (ampdbfs (p4)))
ifq       = p5
iaenv     = p6
inn       = sr/2/ifq      ; total possible number of harmonics present
inn       = int (inn * p8) ; % of possible total
ilh       = p7            ; lowest harmonic present
ifn       = 5             ; stored cosine function
ibzmul    = p9            ; multiplier
ibzmenv   = p10           ; envelope for the multiplier

      kenv      poscil      iamp, idurosc, iaenv      ; amp envelope
      kratio poscil      ibzmul, idurosc, ibzmenv ; kratio envelope

      asrc      gbuzz      kenv, ifq, inn, ilh, kratio, ifn
      out      asrc

endin

```

---

2. Some minor modifications must be made before we can use this Csound orchestra to create an OMambi class from it. Two lines of code must be changed now (marked in red in the code above):

- 
- a) Look for the last line of code before the word “endin”, usually the last line but one (or search for the word “out” with a search function, make sure that it is not a comment).

```
out      asrc
```

Replace the word “out” with this string “asound =” and delete the word “endin”, such that the last line of the code looks like this:

```
asound = asrc
```

- b) Look for a line of code that looks like this:

```
nchnls = 1
```

It must be changed to this:

```
nchnls = 16
```

You can save this file now as a synthesizer-Template for future use, e.g. in the folder “/resources/sourcecode/OMambi/Templates/synthesis”

3. Now open the folder “/resources/sourcecode/OMambi/Templates/ambisonics/encoding” in the OMambi workspace. Select the code-snippet you would like to use according to the preferred control-level. Lets’ choose “Ambisonics(discrete-LL)” for sake of clarity. Open it in a text-editor. It should look somewhat like this:

```
; Ambisonics *****

asound = asound + 0.000001 * 0.000001      ; avoid underflows

; Assign Variables for spatialization-----
; sourceposition
iazimuth = p10                          ;azimuth (degrees)
ielevation = p11                        ;elevation (degrees)
; Filter cutoff frequency
icutoff = p12                          ;cutoff freq (Hz)
; attenuation
iatten = p13                          ;linear gain factor
; order-gains
iordg1 = p14                          ;linear gain factor
iordg2 = p15                          ;linear gain factor
iordg3 = p16                          ;linear gain factor

; Distance-Cues -----
; Air Absorption
icutoffl limit icutoff, 0, 22048
asound_lp butterlp asound, icutoff

; Distance Rolloff
iattenl limit iatten, 0, 1
asound_lp_at = asound_lp * iattenl

; Ambisonic Panning / Generate B-Format *****
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1
asound_lp_at, iazimuth, ielevation

; write audio out (apply order gains)
outx aw, ax * iordg1, ay * iordg1, az * iordg1, ar * iordg2, as * iordg2, at
* iordg2, au * iordg2, av * iordg2, ak * iordg3, al * iordg3, am * iordg3, an *
iordg3, ao * iordg3, ap * iordg3, aq * iordg3

endin
```

4. Copy the entire text and paste it below the last line of the synthesis orchestra opened before. The resulting code should look like this:

```

;=====
;          BUZZ-1.ORB
; DYNAMIC SPECTRUM OSCILLATOR (FROM ACCCI, 43_21_1.ORB) / MONO
; AMPLITUDE ENVELOPE WITH POSCIL
;=====

; Timbre:    Various controlled noise spectra
; Synthesis: (g)buzz
;           POSCILI envelopes
; Coded:     jpg 8/92, modified ms 9/02, 8/08

; NB: NEW STRUCTURE FOR THE AMPLITUDES FROM AUGUST 2008!
;   Positive value > 0.0 : linear amplitude (>0.0-1000.0)
;   0.0 or negative value : amplitude in dB (0 = maximum value)

; The apparently arbitrary amplitude range (0-1000, rather than 0-1)
;   avoids Lisp printing small values with exponential notation

; Replaced oscili with poscil (precise oscillator), ms 8/08
; Default SR = 96000, recommended precision: 24 bits
;-----

;   p1    = instrument number
;   p2    = action time [sec]
;   p3    = duration [sec]
;   p4    = maximum amplitude [linear, >0.0-1000.0 or dB, <= 0.0]
;   p5    = fundamental frequency [Hz]
;   p6    = amplitude envelope [GEN]
;   p7    = lowest harmonic present in the buzz [int]
;   p8    = % of maximum possible harmonic present [0-1]
;   p9    = multiplier in the series of amp coeff for the buzz [0-1]
;   p10   = envelope for the multiplier [GEN]
;-----
; COMPULSORY GEN FUNCTIONS
;   f5    large cosine
;-----

; CLASS: BUZZ-1

; GLOBAL KEYWORDS (default values within parentheses):
;   NUMROWS      : amount of rows (components) in the event (1)
;   ACTION-TIME  : start time of the whole event [sec] (0.0)
;   USER-FUN    : user-defined parsing function (nil)

; LOCAL KEYWORDS:
;   E-DELS : entry delays [sec] (0.0)
;   DURS   : duration [sec] (1.0)
;   AMP    : amplitude [lin, >0.0-1000.0 or dB <- 0.0] (-6.0)
;   F0     : fundamental frequency [Hz] (220.0)
;   AENV   : function number for the amplitude envelope [GEN] (triangle)
;   BZL    : lowest harmonic present in the buzz [integer] (1)
;   BZH    : % of maximum possible harmonic present [0-1] (1)
;   BZM    : multiplier in the series of amp coeff [0-1] (0.95)
;   BZMENV : function number for the buzz envelope [GEN] (triangle)
;*****

sr    = 96000
kr    = 96000
ksmps = 1
nchnls = 16

;0dbfs = 32767 ; 16 bits

```

---

```
0dbfs = 8388697 ; 24 bits
```

```
instr 1; *****
idur      = p3
idurosc   = 1/p3
iamp      = (p4 > 0.0 ? (p4*0.001*0dbfs) : (ampdbfs (p4)))
ifq       = p5
iaenv     = p6
inn       = sr/2/ifq      ; total possible number of harmonics present
inn       = int (inn * p8) ; % of possible total
ilh       = p7            ; lowest harmonic present
ifn       = 5            ; stored cosine function
ibzmul    = p9            ; multiplier
ibzmenv   = p10           ; envelope for the multiplier

kenv      poscil      iamp, idurosc, iaenv      ; amp envelope
kratio    poscil      ibzmul, idurosc, ibzmenv   ; kratio envelope

asrc      gbuzz      kenv,ifq,inn,ilh,kratio,ifn
asound = asrc

; Ambisonics *****

asound = asound + 0.000001 * 0.000001      ; avoid underflows

; Assign Variables for spatialization-----
; sourceposition
iazimuth = p10                ;azimuth (degrees)
ielevation = p11              ;elevation (degrees)
; Filter cutoff frequency
icutoff = p12                  ;cutoff freq (Hz)
; attenuation
iatten = p13                   ;linear gain factor
; order-gains
iordg1 = p14                   ;linear gain factor
iordg2 = p15                   ;linear gain factor
iordg3 = p16                   ;linear gain factor

; Distance-Cues -----
; Air Absorption
icutoffl limit icutoff, 0, 22048
asound_lp butterlp asound, icutoff

; Distance Rolloff
iattenl limit iatten, 0, 1
asound_lp_at = asound_lp * iattenl

; Ambisonic Panning / Generate B-Format *****
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1
asound_lp_at, iazimuth, ielevation

; write audio out (apply order gains)
outx aw, ax * iordg1, ay * iordg1, az * iordg1, ar * iordg2, as * iordg2, at
* iordg2, au * iordg2, av * iordg2, ak * iordg3, al * iordg3, am * iordg3, an *
iordg3, ao * iordg3, ap * iordg3, aq * iordg3

endin
```

5. Now you must look for the p-field with the highest number in the sound synthesis part of the code (which is “p10” in this example) and enumerate the p-fields of the ambisonics part in ascending order starting from this number +1 (10+1). After doing this the Csound code should look like this:

```

;=====
;          BUZZ-1.ORB
; DYNAMIC SPECTRUM OSCILLATOR (FROM ACCCI, 43_21_1.ORB) / MONO
; AMPLITUDE ENVELOPE WITH POSCIL
;=====

; Timbre:    Various controlled noise spectra
; Synthesis: (g)buzz
;           POSCILI envelopes
; Coded:     jpg 8/92, modified ms 9/02, 8/08

; NB: NEW STRUCTURE FOR THE AMPLITUDES FROM AUGUST 2008!
;   Positive value > 0.0 : linear amplitude (>0.0-1000.0)
;   0.0 or negative value : amplitude in dB (0 = maximum value)

; The apparently arbitrary amplitude range (0-1000, rather than 0-1)
;   avoids Lisp printing small values with exponential notation

; Replaced oscili with poscil (precise oscillator), ms 8/08
; Default SR = 96000, recommended precision: 24 bits
;-----

;   p1  = instrument number
;   p2  = action time [sec]
;   p3  = duration [sec]
;   p4  = maximum amplitude [linear, >0.0-1000.0 or dB, <= 0.0]
;   p5  = fundamental frequency [Hz]
;   p6  = amplitude envelope [GEN]
;   p7  = lowest harmonic present in the buzz [int]
;   p8  = % of maximum possible harmonic present [0-1]
;   p9  = multiplier in the series of amp coeff for the buzz [0-1]
;   p10 = envelope for the multiplier [GEN]
;-----
; COMPULSORY GEN FUNCTIONS
;   f5   large cosine
;-----

; CLASS: BUZZ-1

; GLOBAL KEYWORDS (default values within parentheses):
;   NUMROWS      : amount of rows (components) in the event (1)
;   ACTION-TIME  : start time of the whole event [sec] (0.0)
;   USER-FUN     : user-defined parsing function (nil)

; LOCAL KEYWORDS:
;   E-DELS :    entry delays [sec] (0.0)
;   DURS   :    duration [sec] (1.0)
;   AMP    :    amplitude [lin, >0.0-1000.0 or dB <- 0.0] (-6.0)
;   F0     :    fundamental frequency [Hz] (220.0)
;   AENV   :    function number for the amplitude envelope [GEN] (triangle)
;   BZL    :    lowest harmonic present in the buzz [integer] (1)
;   BZH    :    % of maximum possible harmonic present [0-1] (1)
;   BZM    :    multiplier in the series of amp coeff [0-1] (0.95)
;   BZMENU :    function number for the buzz envelope [GEN] (triangle)
;*****

sr    = 96000
kr    = 96000
ksmps = 16
nchnls = 1

;0dbfs = 32767 ; 16 bits
0dbfs = 8388607 ; 24 bits

instr 1; *****
idur    = p3
idurosc = 1/p3
iamp    = (p4 > 0.0 ? (p4*0.001*0dbfs) : (ampdbfs (p4)))

```



---

```

ifq      = p5
iaenv    = p6
inn      = sr/2/ifq      ; total possible number of harmonics present
inn      = int (inn * p8) ; % of possible total
ilh      = p7            ; lowest harmonic present
ifn      = 5             ; stored cosine function
ibzmul   = p9             ; multiplier
ibzmenv  = p10            ; envelope for the multiplier

kenv     poscil   iamp, idurosc, iaenv      ; amp envelope
kratio   poscil   ibzmul, idurosc, ibzmenv  ; kratio envelope

asrc     gbuzz    kenv,ifq,inn,ilh,kratio,ifn
asound = asrc

; Ambisonics *****

asound = asound + 0.000001 * 0.000001      ; avoid underflows

; Assign Variables for spatialization-----
; sourceposition
iazimuth = p11                        ;azimuth (degrees)
ielevation = p12                      ;elevation (degrees)
; Filter cutoff frequency
icutoff = p13                        ;cutoff freq (Hz)
; attenuation
iatten = p14                         ;linear gain factor
; order-gains
iordg1 = p15                         ;linear gain factor
iordg2 = p16                         ;linear gain factor
iordg3 = p17                         ;linear gain factor

; Distance-Cues -----
; Air Absorption
icutoffl limit icutoff, 0, 22048
asound_lp butterlp asound, icutoff

; Distance Rolloff
iattenl limit iatten, 0, 1
asound_lp_at = asound_lp * iattenl

; Ambisonic Panning / Generate B-Format *****
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1
asound_lp_at, iazimuth, ielevation

; write audio out (apply order gains)
outx aw, ax * iordg1, ay * iordg1, az * iordg1, ar * iordg2, as * iordg2, at
* iordg2, au * iordg2, av * iordg2, ak * iordg3, al * iordg3, am * iordg3, an *
iordg3, ao * iordg3, ap * iordg3, aq * iordg3

endin

```

---

Your Csound-orchestra is now ready to build an Omambi class from it! If you wish you can add comments to describe the p-fields. The procedure is the same for all the templates. You can save the file to disk with an arbitrary file name, however you should preferably use the suggested naming convention to keep an overview of omChroma/OMambi classes. To build an OMambi class from the Csound-orchestra you can either manually write a class in LISP or use the box “get-instrument” in OpenMusic as described thoroughly in the documentation of omChroma.

---

## 8. Future Developments

Increasing the order of the ambisonic encoding improves the spatial resolution of the Bformat file. However, with increasing order the number of audio channels needed for the Bformat file raises exponentially. Instead of a brute force increase of the encoding order, the quality of spatial sound rendering should be improved by other means: E.g. via a room-model to account for early reflections/reverberation, or a Bformat convolution reverb. It is also planned to implement source-directivity (periphonic). Further implementing distinct classes for 2-dimensional and 3-dimensional encoding would avoid wasting audio channels for Bformat components which are not used.

When combining sound synthesizers with ambisonic encoders (6 possible encoders for each sound synthesizer) the number of possible combinations explodes ( $Z^{6n}$ )<sup>1</sup> and the programming of each combination soon becomes intractable even for a small number of classes. It is planned to switch to an object-oriented approach for automatic creation of spatial sound synthesizers via multiple inheritance from sound synthesis- and spatialization-superclasses, respectively. In this process the csound-code used for sound synthesis classes in omChroma and for ambisonic spatialization in OMambi will be merged by a LISP-function which defines a new spatial sound synthesis class. This operation currently needs to be done manually, as described in Chapter 7.

The Max/MSP external used for decoding of the Bformat files will be extended to allow for decoding up to 5<sup>th</sup> order. Further, arbitrary re-mapping of Bformat-components to decoder-inputs is planned to allow the decoding of Bformat files with arbitrary arrangements of components. In order to avoid switching between two applications, a protocol is currently being established to control the decoding application from within OpenMusic. The ambisonic decoder will be extended to allow for decoding up to 5<sup>th</sup> order.

## 9. Appendix

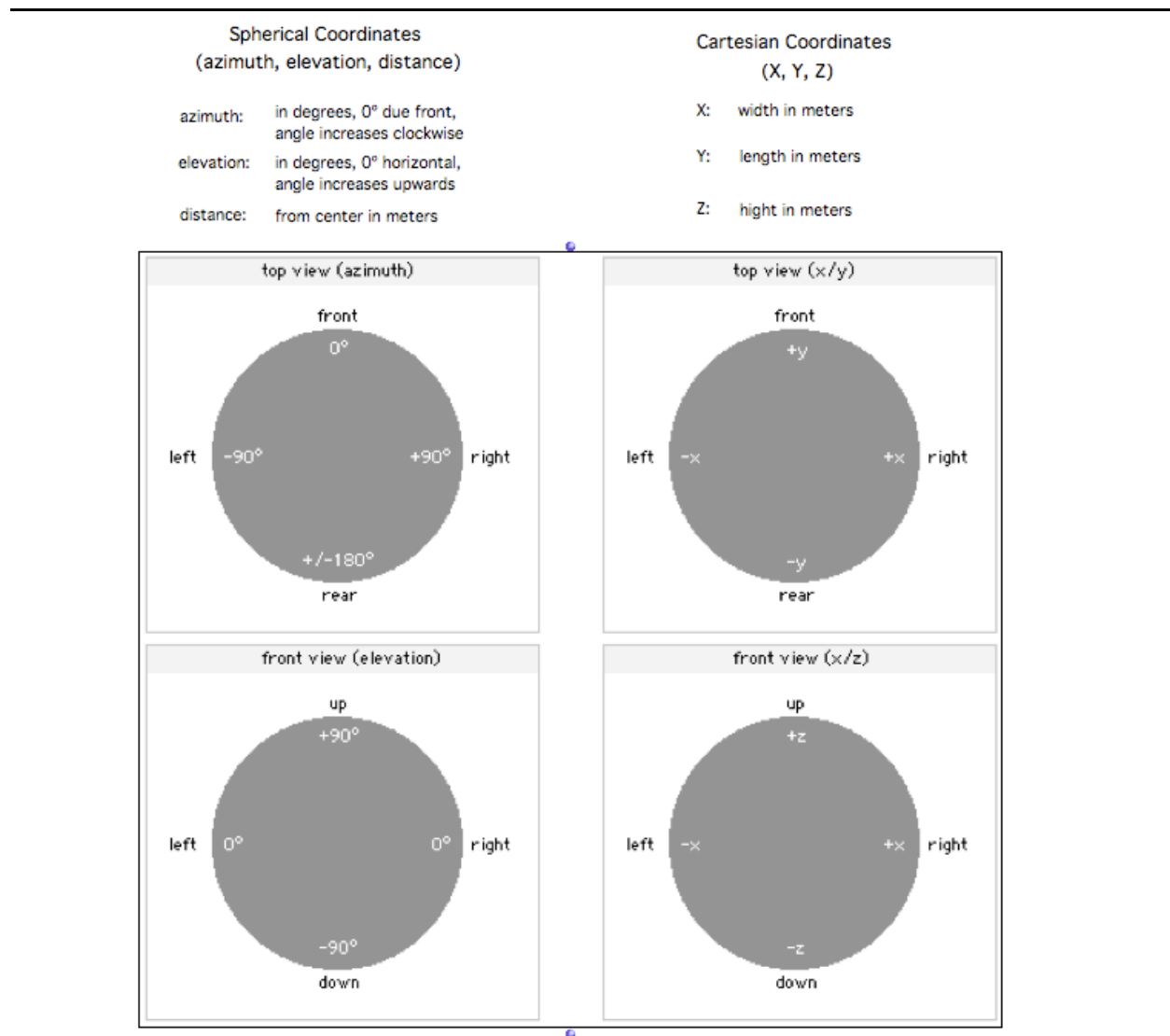
### 9.1 Coordinate systems

OMambi uses a head-centered<sup>2</sup>, right-handed spherical coordinate system based on navigation/topology conventions, in compliance with SpatDIF, the "Spatial Sound Description Interchange Format". For further information on SpatDIF see [www.SpatDIF.org](http://www.SpatDIF.org). Tools for conversion between spherical and cartesian systems are provided in the folder /miscellaneous.

---

<sup>1</sup> see e.g. the Appendix for a tree-diagram of currently implemented classes based on 3 sound synthesizers.

<sup>2</sup> The system's reference point is the sweetspot at the center of the loudspeakers, and source positions are given relative to this position, where the listener is supposed to be.



## 9.2 List of matrix inlets & descriptions for discrete control

### 9.2.1 Ambiench (high-evel control)

| INLET               | DESCRIPTION                                 | TYPE   |
|---------------------|---|--------|
| <i>azimuth</i>      | Azimuth-angle in degrees                    | Number |
| <i>elevation</i>    | Elevation-angle in degrees                  | Number |
| <i>distanz</i>      | Distance in meters                          | Number |
| <i>hfcut-factor</i> | Coefficient for air-absorption-function     | Number |
| <i>atten-factor</i> | Coefficient for attenuation-function        | Number |
| <i>center-size</i>  | Size of center-zone                         | Number |
| <i>center-curve</i> | Coefficient for center-attenuation-function | Number |
| <i>center-atten</i> | Attenuation at center (in decibel)          | Number |

|                    |                         |        |
|--------------------|-------------------------|--------|
| <i>airabs-mode</i> | Air-absorption-function | Number |
| <i>atten-mode</i>  | Attenuation-function    | Number |
| <i>order</i>       | Encoding Order          | Number |

### 9.2.2 Ambienm (mid-level control)

| INLET                 | DESCRIPTION  | TYPE     |
|-----------------------|--|----------|
| <i>azimuth</i>        | Azimuth-angle in degrees                               | Number   |
| <i>elevation</i>      | Elevation-angle in degrees                             | Number   |
| <i>distanz</i>        | Distance in meters                                     | Number   |
| <i>air-function</i>   | Table used for lookup of cutoff-freq of lowpass-filter | Cs-table |
| <i>airf-max</i>       | Maximum value of table used for air-function           | Number   |
| <i>atten-function</i> | Table used for lookup of linear gain-factor            | Cs-table |
| <i>attenf-max</i>     | Maximum value of table used for air-function (0-1)     | Number   |
| <i>order-function</i> | Table used for lookup of encoding-order                | Cs-table |
| <i>orderfmax</i>      | Maximum value of table used for encoding-order         | Number   |
| <i>order</i>          | Encoding-order (0-3)                                   | Number   |
| <i>lookupmin</i>      | Min. distance (index for lookup-tables)                | Number   |
| <i>lookupmax</i>      | Max. distance (index for lookup-tables)                | Number   |

### 9.2.2 Ambienl (low-level control)

| INLET                            | DESCRIPTION   | TYPE   |
|----------------------------------|---|--------|
| <i>azimuth</i>                   | Azimuth-angle in degrees                                  | Number |
| <i>elevation</i>                 | Elevation-angle in degrees                                | Number |
| <i>hf-cutoff</i>                 | Cutoff-Frequency of 2 <sup>nd</sup> -order-lowpass filter | Number |
| <i>attenuation</i>               | Linear gain-factor (0-1)                                  | Number |
| <i>1<sup>st</sup>-order-gain</i> | Linear gain-factor for 1 <sup>st</sup> -order-components  | Number |
| <i>2<sup>nd</sup>-order-gain</i> | Linear gain-factor for 2 <sup>nd</sup> -order-components  | Number |
| <i>3<sup>rd</sup>-order-gain</i> | Linear gain-factor for 3 <sup>rd</sup> -order-components  | Number |

## 9.3 List of matrix inlets & descriptions for continuous control

### 9.3.1 Ambienh (high-level control)

| INLET               | DESCRIPTION   | TYPE     |
|---------------------|---|----------|
| <i>azimin</i>       | Minimum value of azi-envelope                       | Number   |
| <i>azimax</i>       | Maximum value of azi-envelope                       | Number   |
| <i>azidur</i>       | Duration of azi-envelope (in seconds)               | Number   |
| <i>azi-envelope</i> | Table/BPF used as envelope for azimuth (in degrees) | Cs-table |

|                          |  |          |
|--------------------------|--|----------|
| <i>elemin</i>            | Minimum value of ele-envelope  | Number   |
| <i>elemax</i>            | Maximum value of ele-envelope  | Number   |
| <i>eledur</i>            | Duration of ele-envelope (in seconds)                                  | Number   |
| <i>ele-envelope</i>      | Table/BPF used as envelope for elevation (in degrees)                  | Cs-table |
| <i>distmin</i>           | Minimum value of dist-envelope   | Number   |
| <i>distmax</i>           | Maximum value of dist-envelope   | Number   |
| <i>distdur</i>           | Duration of dist-envelope (in seconds)                                 | Number   |
| <i>dist-envelope</i>     | Table/BPF used as envelope for distanz (in meters)                     | Cs-table |
| <i>hfcutfmin</i>         | Minimum value of hfcutf-envelope                                       | Number   |
| <i>hfcutfmax</i>         | Maximum value of hfcutf-envelope                                       | Number   |
| <i>hfcutfdur</i>         | Duration of hfcutf-envelope (in seconds)                               | Number   |
| <i>hfcutf-envelope</i>   | Table/BPF used as envelope for coefficient for air-absorption function | Cs-table |
| <i>attenfmin</i>         | Minimum value of attenf-envelope                                       | Number   |
| <i>attenfmax</i>         | Maximum value of attenf-envelope                                       | Number   |
| <i>attenfdur</i>         | Duration of attenf-envelope (in seconds)                               | Number   |
| <i>attenf-envelope</i>   | Table/BPF used as envelope for coefficient for attenuation factor      | Cs-table |
| <i>dopplerfmin</i>       | Minimum value of dopplerf-envelope                                     | Number   |
| <i>dopplerfmax</i>       | Maximum value of dopplerf-envelope                                     | Number   |
| <i>dopplerfdur</i>       | Duration of dopplerf-envelope (in seconds)                             | Number   |
| <i>dopplerf-envelope</i> | Table/BPF used as envelope for coefficient for dopplerfunction         | Cs-table |
| <i>ordmin</i>            | Minimum value of ord-envelope  | Number   |
| <i>ordmax</i>            | Maximum value of ord-envelope  | Number   |
| <i>orddur</i>            | Duration of ord-envelope (in seconds)                                  | Number   |
| <i>ord-envelope</i>      | Table/BPF used as envelope for encoding order                          | Cs-table |
| <i>center-size</i>       | size of center-zone  | Number   |
| <i>center-curve</i>      | Coefficient for center-attenuation-function                            | Number   |
| <i>center-atten</i>      | Attenuation at center (in decibel)                                     | Number   |
| <i>airabs-mode</i>       | Air-absorption-function  | Number   |
| <i>atten-mode</i>        | Attenuation-function   | Number   |

### 9.3.2 Ambienm (mid-level control)

| <b>INLET</b>        | <b>DESCRIPTION</b>                                    | <b>TYPE</b> |
|---------------------|---|-------------|
| <i>azimin</i>       | Minimum value of azi-envelope                         | Number      |
| <i>azimax</i>       | Maximum value of azi-envelope                         | Number      |
| <i>azidur</i>       | Duration of azi-envelope (in seconds)                 | Number      |
| <i>azi-envelope</i> | Table/BPF used as envelope for azimuth (in degrees)   | Cs-table    |
| <i>elemin</i>       | Minimum value of ele-envelope                         | Number      |
| <i>elemax</i>       | Maximum value of ele-envelope                         | Number      |
| <i>eledur</i>       | Duration of ele-envelope (in seconds)                 | Number      |
| <i>ele-envelope</i> | Table/BPF used as envelope for elevation (in degrees) | Cs-table    |
| <i>distmin</i>      | Minimum value of dist-envelope                        | Number      |
| <i>distmax</i>      | Maximum value of dist-envelope                        | Number      |

|                         |  |          |
|-------------------------|--|----------|
| <i>distdur</i>          | Duration of dist-envelope (in seconds)                 | Number   |
| <i>dist-envelope</i>    | Table/BPF used as envelope for distanz (in meters)     | Cs-table |
| <i>ordmin</i>           | Minimum value of ord-envelope                          | Number   |
| <i>ordmax</i>           | Maximum value of ord-envelope                          | Number   |
| <i>orddur</i>           | Duration of ord-envelope (in seconds)                  | Number   |
| <i>ord-envelope</i>     | Table/BPF used as envelope for encoding order          | Cs-table |
| <i>air-function</i>     | Table used for lookup of cutoff-freq of lowpass-filter | Cs-table |
| <i>airf-max</i>         | Maximum value of table used for air-function           | Number   |
| <i>atten-function</i>   | Table used for lookup of linear gain-factor            | Cs-table |
| <i>attenf-max</i>       | Maximum value of table used for air-function (0-1)     | Number   |
| <i>doppler-function</i> | Table used for lookup of time-delay                    | Cs-table |
| <i>dopplerfmax</i>      | Maximum value of table used for time-delay (in ms)     | Number   |
| <i>order-function</i>   | Table used for lookup of encoding-order                | Cs-table |
| <i>orderfmax</i>        | Maximum value of table used for encoding order (0-3)   | Number   |
| <i>lookupmin</i>        | Min. distance (index for lookup-tables)                | Number   |
| <i>lookupmax</i>        | Max. distance (index for lookup-tables)                | Number   |

### 9.3.3 Ambiencl (low-level control)

| <b>INLET</b>            | <b>DESCRIPTION</b>   | <b>TYPE</b> |
|-------------------------|--|-------------|
| <i>azimin</i>           | Minimum value of azi-envelope  | Number      |
| <i>azimax</i>           | Maximum value of azi-envelope  | Number      |
| <i>azidur</i>           | Duration of azi-envelope (in seconds)  | Number      |
| <i>azi-envelope</i>     | Table/BPF used as envelope for azimuth (in degrees)                                      | Cs-table    |
| <i>elemin</i>           | Minimum value of ele-envelope  | Number      |
| <i>elemax</i>           | Maximum value of ele-envelope  | Number      |
| <i>eledur</i>           | Duration of ele-envelope (in seconds)  | Number      |
| <i>ele-envelope</i>     | Table/BPF used as envelope for elevation (in degrees)                                    | Cs-table    |
| <i>hfcutmin</i>         | Minimum value of hfcut-envelope  | Number      |
| <i>hfcutmax</i>         | Maximum value of hfcut-envelope  | Number      |
| <i>hfcutdur</i>         | Duration of hfcut-envelope (in seconds)  | Number      |
| <i>hfcut-envelope</i>   | Table/BPF used as envelope for cutoff-frequency of 2 <sup>nd</sup> -order lowpass filter | Cs-table    |
| <i>attenmin</i>         | Minimum value of atten-envelope  | Number      |
| <i>attenmax</i>         | Maximum value of atten-envelope  | Number      |
| <i>attendur</i>         | Duration of atten-envelope (in seconds)  | Number      |
| <i>atten-envelope</i>   | Table/BPF used as envelope for linear gain-factor  | Cs-table    |
| <i>dopplermin</i>       | Minimum value of doppler-envelope  | Number      |
| <i>dopplermax</i>       | Maximum value of doppler-envelope  | Number      |
| <i>dopplerdur</i>       | Duration of doppler-envelope (in seconds)  | Number      |
| <i>doppler-envelope</i> | Table/BPF used as envelope for time-delay  | Cs-table    |
| <i>ordermin</i>         | Minimum value of order-envelope  | Number      |
| <i>ordermax</i>         | Maximum value of order-envelope  | Number      |
| <i>orderdur</i>         | Duration of order-envelope (0-3)   | Number      |
| <i>order-envelope</i>   | Table/BPF used as envelope for encoding order  | Cs-table    |

## 9.4 Overview of implemented OMambi-classes

The tree-diagram below shows the currently implemented OMambi-classes which are named following the scheme <name-of-omChroma-synthesizer><OMambi-class>”-1”

