Ambisonics User Defined Opcodes for Csound

Martin Neukom
Zurich University of the Arts
Institute for Computer Music and
Sound Technology
martin.neukom@zhdk.ch

ABSTRACT

This text describes the implementation of Ambisonics as user defined opcodes (UDOs) for Csound. The presented package of UDOs includes a basic encoder and a decoder up to 8th order, an encoder with distance correction, an in-phase decoder, opcodes for the two-dimensional equivalent of Ambisonics for any order, opcodes for Ambisonics equivalent panning (AEP) and several utilities such as coordinate converters, Doppler effect and more. Finally the usage of the UDOs is explained in some examples.

1. INTRODUCTION

Ambisonics is a technique for three-dimensional sound recording, rendering and storage. The fundamentals of Ambisonics were developed in the 70es by M. A. Gerzon [1]. In the first decade of the 21st century the theory has been enhanced and formulas for encoding and decoding have been published (see e.g. J. Daniel, 2003 [2], [3] and [4, pp. 438]). A short introduction into the principles of Ambisonics is given in Chapter 2. The presented Csound UDOs include in-phase decoding [2, p. 186], distance encoding (discussion and references for example in [5] and [6]), a two-dimensional equivalent to Ambisonics [2, p. 153] and Ambisonics equivalent panning AEP [7][8]. Not yet implemented are near field compensation [3], hemispherical encoding and decoding [9], decoding for not-ideal loudspeaker arrangements [10], and more.

There have been different reasons to realize Ambisonics UDOs for Csound. The Csound opcodes *bformenc1* and *bformdec1* only support Ambisonics up to 3rd order, no enhancements and variations are implemented and decoding is restricted to a few standard speaker setups. Ambisonics is not easy to understand and to use. Thus I decided to write an introduction for the FLOSS manual [11] where theory, implementation and application are demonstrated step by step. The Csound UDOs are written in Csound language and can be understood and expanded by non-programmers without recompiling Csound.

The UDOs are saved in the text files *ambisonics_udos.txt*, *ambisonics2D_udos.txt*, *AEP_udos.txt* and *utilities.txt* and can be downloaded together with the Csound examples from the ICST homepage [12].

2. PRINCIPLES OF AMBISONICS

Ambisonics is a surround-system for encoding and rendering a three-dimensional sound field. In Ambisonics the position of a virtual sound source is encoded together with the sound itself in a multi channel sound file, the so-called B-format which is independent of the speaker set-up. The encoding can be carried out to an arbitrary degree of accuracy. The accuracy is given by the so-called order m of Ambisonics.

The formulas for ambisonic encoding are derived from the solution of the three-dimensional wave equation in the spherical coordinate system where a point is described by radius r, azimuth az and elevation el. A signal S is encoded by multiplying the signal with the first m spherical harmonics [2][3]. The zeroth order corresponds to the mono signal and needs one channel. In first order Ambisonics the portions of the sound field in the directions x, y and z are encoded in three more channels. The order of resolution m defines the accuracy of the encoding and the number $n = (m + 1)^2$ of channels in the B-format.

From a B-format file with n channels and a given set-up of at least n speakers the signals for the speakers can be calculated as a weighted sum of the B-format channels. The speaker signals for symmetrical setups of n speakers can be calculated from the B-format and the matrix of the B-format of the speaker signals. This symmetric solution is normally used, even if the speaker set-up is not exactly symmetric. (Ambisonics2D, distance encoding and Ambisonics equivalent panning are explained below with their implementation.)

3. IMPLEMENTATION

3.1 Prerequisites

The provided UDOs should be simple to understand, to use and to enhance. They do not use flags, parameters and options. The single channels of the B-format are not visible to the user. Thus, only a single coordinate system is needed. The different encoding and decoding types as Ambisonics, Ambisonics2D and AEP each have their own UDOs. The UDOs are modular, i.e. absorption, Doppler effect and signal correction for speaker arrays with irregular distances to the centre are not included in encoder or decoder but implemented in their own UDOs.

The B-format is not visible to the user but written to a *zak* space. Therefore, *zakinit* must be run before any instrument definition (in the orchestra file after the header), providing at least $n = (m + 1)^2$ channels for Ambisonics and $n = 2(m + 1)$ channels for Ambisonics2D. *zacl* clears the *za* space and is called after decoding or writing the B-format.

3.2 Ambisonics

The Ambisonics UDOs use semi-normalized spherical harmonics (the formulas for the spherical harmonics up to 11th order can be found in [13]). If the B-format is encoded or decoded with another program the same format must be used or the B-format must be converted. The syntax of the Ambisonics encoder is:

```
k0   ambi_encode   asnd, iorder, kazimuth, kelevation
```

The order *iorder* is constant, the angles azimuth and elevation are control functions and given in degrees (the output *k0* is 0). The following code sample shows the encoding up to second order. The B-format is stored in the *zak* space: *zawm asnd, 0* accumulates the mono signal *asnd* as channel *W* to the first channel of the *zak* space, *zawm kcos_el*ksin_az*asnd, 1* accumulates the *y*-component of *asnd* to the second channel etc.

```
opcode   ambi_encode, k, aikk
asnd,iorder,kaz,kel            xin
kaz = $M_PI*kaz/180
kel = $M_PI*kel/180
kcos_el = cos(kel)
ksin_el = sin(kel)
kcos_az = cos(kaz)
ksin_az = sin(kaz)
   zawm asnd,0                        ; W
   zawm kcos_el*ksin_az*asnd,1        ; Y = Y(1,-1)
   zawm ksin_el*asnd,2                ; Z = Y(1,0)
   zawm kcos_el*kcos_az*asnd,3        ; X = Y(1,1)
   if iorder < 2 goto end
i2 = sqrt(3)/2
kcos_el_p2 = kcos_el*kcos_el
ksin_el_p2 = ksin_el*ksin_el
kcos_2az = cos(2*kaz)
ksin_2az = sin(2*kaz)
kcos_2el = cos(2*kel)
ksin_2el = sin(2*kel)
   zawm i2*kcos_el_p2*ksin_2az*asnd,4       ; V = Y(2,-2)
   zawm i2*ksin_2el*ksin_az*asnd,5          ; S = Y(2,-1)
   zawm .5*(3*ksin_el_p2 - 1)*asnd,6        ; R = Y(2,0 )
   zawm i2*ksin_2el*kcos_az*asnd,7          ; S = Y(2,1)
   zawm i2*kcos_el_p2*kcos_2az*asnd,8       ; U = Y(2,2)
   if iorder < 3 goto end
...
```

The decoding is done in two steps. First the B-format is decoded for one speaker with an opcode called *ambi_decode1*. The formulas are the same as for encoding (with different gains for compensation of the semi-normalization) but the input angles of the speakers are constant. *zar(0)* reads channel 0 from the *zak* space.

```
opcode   ambi_decode1, a, iii
iorder,iaz,iel        xin
iaz = $M_PI*iaz/180
iel = $M_PI*iel/180
a0 = zar(0)
   if iorder > 0 goto c0
...
```

In the second step an overloaded opcode produces the signals for *n* speakers. The number of output signals determines which version of the opcode is used. The following code shows the opcode for two speakers.

```
opcode   ambi_decode,      aa,ii
iorder,ifn xin
   xout   ambi_decode1(iorder,table(1,ifn),table(2,ifn)),
          ambi_decode1(iorder,table(3,ifn),table(4,ifn))
endop
```

The opcodes *ambi_encode* and *ambi_decode* up to 8th order are saved in the text file *ambisonics_udos.txt*.

3.3 Ambisonics2D

If the virtual sound sources are arranged in a plane Ambisonics can be replaced by a two-dimensional analogy called Ambisonics2D in what follows [4]. The number of channels is $n = 2(m + 1)$. The position of a sound source in a plane (normally the horizontal plane) is given by two coordinates. In Cartesian coordinates (x, y) the listener is at the origin of the coordinate system (0, 0), and the x-coordinate points to the front, the y-coordinate to the left. The position of a sound source can also be given in polar coordinates by the azimuth angle between the line of vision of the listener (front) and the direction to the sound source, and by its distance r. The formulas for Ambisonics2D encoding and decoding are derived from the solution of the wave equation in the cylindrical coordinate system. A signal S is encoded by multiplying the signal with the first m cylindrical harmonics which are just the sines and cosines of the multiples of the angle *az*. The syntax of the Ambisonics2D encoder is:

```
k0   ambi2D_encode   asnd, iorder, kazimuth
```

The following code shows the encoding up to any order.

```
opcode   ambi2D_encode, k, aik
asnd,iorder,kaz    xin
kaz = $M_PI*kaz/180
kk          iorder
c1:
  zawm   cos(kk*kaz)*asnd,2*kk-1
  zawm   sin(kk*kaz)*asnd,2*kk
kk = kk-1
  if        kk > 0 goto c1
            zawm     asnd,0
            xout      0
endop
```

The syntax of the Ambisonics2D decoder is:

```
a1 [, a2]...[, a8] ambi2D_decode iorder, iaz1 [, iaz2]...[, iaz8]
```

where *a1* ... *a8* are the speaker signals and *iaz1* ... *iaz8* are the azimuth angles to the loudspeakers. The formulas for the decoder are the same as for the encoder with the only difference that the channel W is multiplied by 1/2. The opcodes *ambi2D_encode* and *ambi_decode2D* are saved in the text file *ambisonics2D_udos.txt*.

3.4 In-Phase Decoding

Since only a few of the theoretically infinite number of channels of the decomposition of the encoded sound waves are used, the resulting speaker signals are not ideal. The loudspeakers near the virtual sound source indeed receive the strongest signals whereas all other loudspeakers have weaker signals. Still they do not become continuously weaker with increasing distance to the sound source and some have negative amplitudes, that is, reversed phases (left figure below)

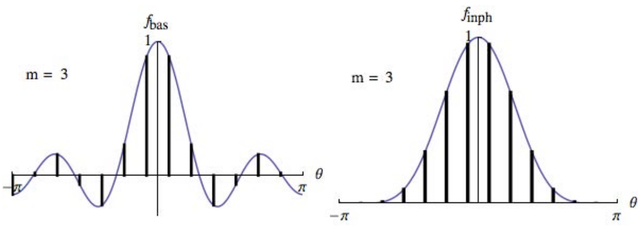

Figure 1. Basic and in-phase decoding level functions.

These side-effects can be avoided by weighting the B-format channels before being decoded. The weighting factors depend on the highest order used and the order of the particular channel being decoded [see e.g. 2]. The syntax of the in-phase decoders are:

```
a1 ... [, a8]  ambi_dec_inph      iorder, ifn
a1 ... [, a8]  ambi2D_dec_inph  iorder, iaz1] ... [, iaz8]
```

The following code sample shows the weighting factors stored in an array and the multiplication of the channels 1 to 3 before decoding of the Ambisonics in-phase decoder

```
opcode   ambi_dec1_inph, a, iii

iWeight3D[][] init   8,8
iWeight3D     array \
0.3333,0,0,0,0,0,0,0,
0.5,0.1,0,0,0,0,0,0,
0.6,0.2,0.0285714,0,0,0,0,0,
0.6667,0.2857,0.0714,0.0079,0,0,0,0,
0.7143,0.3571,0.119,0.0238,0.00216,0,0,0,
0.75,0.4167,0.1667,0.04545,0.00758,0.00058,0,0,
0.7778,0.4667,0.2121,0.0707,0.0163,0.0023,0.00016,0,
0.8,0.509,0.2545,0.098,0.028,0.0056,0.0007,0.00004
...
a0 = zar(0)
        if iorder > 0 goto c0
aout = a0
        goto end
c0:
a1 = iWeight3D[iorder-1][0]*zar(1)
a2 = iWeight3D[iorder-1][0]*zar(2)
a3 = iWeight3D[iorder-1][0]*zar(3)
...
```

3.5 Distance Encoding

In basic Ambisonics only the angle of incidence of the sound waves is encoded. In order to simulate distances and movements of sound sources, the signals have to be treated before being encoded. The main perceptual cues for the distance of a sound source are reduction of the amplitude, filtering due to the absorption of the air and the relation between direct and indirect sound. In order to simulate realistically moving sound sources the Doppler effect can be integrated. The reduction of the amplitude outside the unit circle (r = 1), absorption, reverb and Doppler effect are applied to the sound before encoding (UDOs for the Doppler effect and for a simple absorption are included in the file *ambi_utilities.txt*).

The increase of the amplitude of sounds inside the unit circle must be limited and special care must be taken if the position of a virtual sound source coincides with the origin of the coordinate system, which for example can happen when the position changes randomly or by uncontrolled manipulation with interfaces. The amplitude arriving at a listener is inverse proportional to the distance of the sound source. If the distance is larger than the unit circle (not necessarily the radius of the speaker setup, which does not need to be known when encoding sounds) we simply can divide the sound by the distance. With this calculation inside the unit circle the amplitude is amplified and becomes infinite when the distance becomes zero. Another problem arises when a virtual sound source passes the origin. The amplitude of the speaker signal in the direction of the movement suddenly becomes maximal and the signal of the opposite speaker suddenly becomes zero.

A simple solution for these problems is to limit the gain of the channel W inside the unit circle to 1 (*f1* in the figure below) and to fade out all other channels (*f2*). By fading out all channels except channel W the information about the direction of the sound source is lost and all speaker signals are the same and the sum of the speaker signals reaches its maximum when the distance is 0.

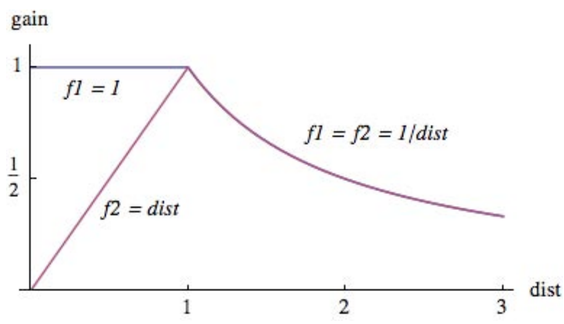

Figure 2. Amplitude functions: *f1* for channel W and *f2* for higher order channels.

We would prefer that gain functions are smoother at $d = 1$. Ideally, the functions should be differentiable and the slope of *f1* at distance $d = 0$ should be 0. For distances greater than 1 the functions should be approximately $1/d$. In addition the function *f1* should continuously grow with decreasing distance and reach its maximum at $d = 0$. The maximal gain must be 1. The function atan($d\pi/2$)/($d\pi/2$) fulfills these constraints. We create a function *f2* for the fading out of the other channels by multiplying *f1* with the factor $(1 - e^{-d})$.

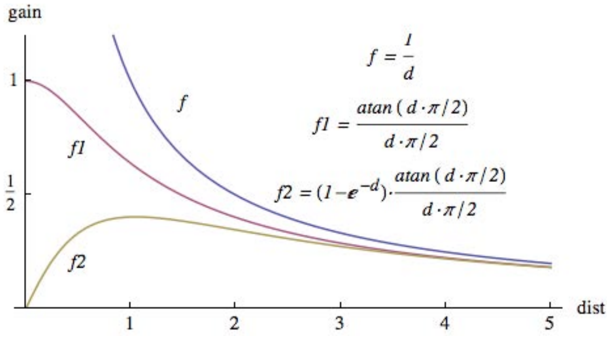

Figure 3. Smoother amplitude functions: *f1* for channel *W* and *f2* for higher order channels.

The UDO *ambi2D_enc_dist* encodes a sound at any order with distance correction. The inputs of the UDO are *asnd*, *iorder*, *kazimuth*, *kdistance*. If the distance becomes negative the azimuth angle is turned to its opposite (*kaz* += π) and the distance is taken positive.

```
opcode   ambi2D_enc_dist, k, aikk
asnd,iorder,kaz,kdist          xin
kaz = $M_PI*kaz/180
kaz =(kdist < 0 ? kaz + $M_PI : kaz)
kdist = abs(kdist)+0.0001
kgainW = taninv(kdist*1.5707963)/(kdist*1.5708)
kgainHO = (1 - exp(-kdist))
kk = iorder
asndW = kgainW*asnd
asndHO = kgainHO*asndW
c1:
  zawm  cos(kk*kaz)*asndHO,2*kk-1
  zawm  sin(kk*kaz)*asndHO,2*kk
kk = kk-1
   if       kk > 0 goto c1
   zawm asndW,0
   xout    0
endop
```

3.6 Ambisonics Equivalent Panning AEP

If we combine encoding and in-phase decoding, we obtain the following panning function (a gain function for a speaker depending on its distance to a virtual sound source)[7]:

$$P(\gamma, m) = (\tfrac{1}{2} + \tfrac{1}{2}\cos\gamma)^m \qquad (1)$$

where γ denotes the angle between a sound source and a speaker and *m* denotes the order. If the speakers are positioned on a unit sphere, the cosine of the angle γ is calculated as the scalar product of the vector to the sound source (x, y, z) and the vector to the speaker (x_s, y_s, z_s).

In contrast to Ambisonics the order indicated in the function does not have to be an integer. This means that the order can be continuously varied during decoding. The function can be used in both Ambisonics and Ambisonics2D.

This system of panning is called Ambisonics Equivalent Panning. It has the disadvantage of not producing a B-format representation, but its implementation is straightforward and the computation time is short and independent of the simulated Ambisonics order. Hence it is particularly useful for real-time applications, for panning in connection with sequencer programs and for experimentation with high and non-integral Ambisonic orders. The opcode *AEP1* calculates ambisonics equivalent panning for one speaker. The opcode *AEP* then uses *AEP1* to produce the signals for several speakers. In the text file *AEP_udos.txt AEP* is implemented for up to 16 speakers. The position of the speakers must be written to a function table. As the first parameter in the function table the maximal speaker distance must be given.

4. EXAMPLES

4.1 Basic Encoding

The first example shows basic encoding 4th order of a virtual sound source turning around the listener on a semi-circle from the front to the back of the listener. The B-format is written to the file *B_form1.wav*. Then the B-format (still stored in the *zak* space) is decoded to a regular eight-speaker setup in the horizontal plane given in the function table 17.

```
#include "ambisonics_udos.txt"
zakinit 25,1
instr 1
kaz    line   0,p3,180
asnd  rand 1
k0      ambi_encode    asnd,4,kaz,0
k0      ambi_write_B  "B_form1.wav",4,14
a1,a2,a3,a4,a5,a6,a7,a8   ambi_decode    4,17
outc a1,a2,a3,a4,a5,a6,a7,a8
zacl 0,24
endin
...
f 17 0 64 -2 0 0 0 45 0 90 0 135 0 180 0 225 0
```

Figure 4 shows the speaker signals of the first four speakers. Figure 5 shows the speaker signals for the same sound source but decoded in-phase.

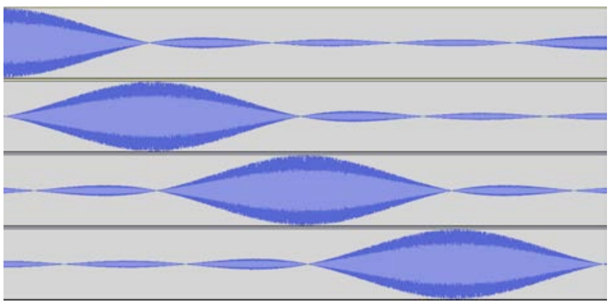

Figure 4. Signals of four speakers for a sound source turning around the listener with Ambisonics order 4.

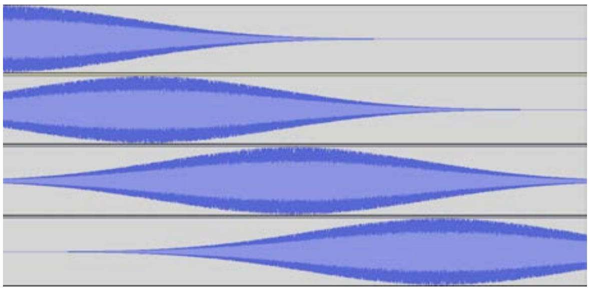

Figure 5. Signals of four speakers of a sound source turning around the listener. In-phase decoding 4th order.

4.2 Distance Encoding

The second example shows distance encoding in Ambisonices2D. A virtual sound source approaches the listener from the front (*az* = 0) goes through the origin and recedes in the opposite direction (*kdist line 2, p3, -2*).

```
#include "ambisonics2D_udos.txt"
zakinit 9,1
instr 1
kdist  line 2,p3,-2
asnd  rand 1
k0  ambi2D_enc_dist asnd,4,0,kdist
a1,a2,a3 ambi2D_dec_inph 4,0,120,-120
outc a1,a2,a3
zacl 0,8
endin
```

Figure 6 shows the speaker signals of three speakers positioned at *az* = 0, 120 and -120 degrees and in the fourth track the sum of them. After a fourth of the time the sound source reaches the unit circle. The amplitude of the first speaker signal now decreases and the amplitudes of the other speaker signals increase. In the middle of the file all amplitudes are the same and their sum is maximal.

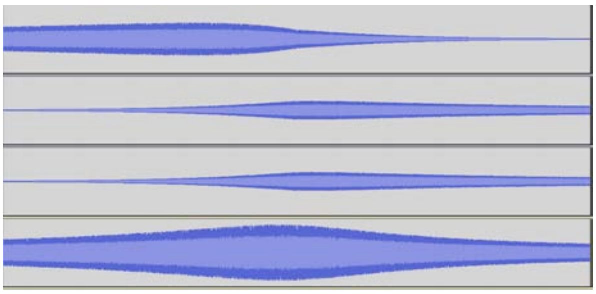

Figure 6. Speaker signals of three speakers positioned at *az* = 0, 120 and -120 degrees and the sum of them.

4.3 3D-Movement

In the third example a sound source moves in three dimensions. The coordinate functions are *x = 10sin(t), y = 10sin(.78t)* and *z = 10sin(.43t).* The UDO *xyz_to_aed* then transforms them to the spherical coordinates *kaz, kel* and *kdist* and the UDO *Doppler* simulates the Doppler effect. *ambi_decode* decodes for 8 speakers arranged in a cube (function table 17).

```
zakinit 16, 1
#include "ambisonics_udos.txt"
#include "ambisonics_utilities.txt"

instr 1
asnd       buzz       p4,p5,p6,1
kt         line       0,p3,p3
kaz,kel,kdist      xyz_to_aed \
           10*sin(kt),10*sin(.78*kt),10*sin(.43*kt)
adop       Doppler asnd,kdist
k0    ambi_enc_dist    adop,3,kaz,kel,kdist
a1,a2,a3,a4,a5,a6,a7,a8   ambi_decode    3,17
outc a1,a2,a3,a4,a5,a6,a7,a8
zacl 0,15
endin
...
f17 0 64 -2 0 -45 35.26 45 35.26 135 35.26 225 35.26 -45
-35.26 .7854 -35.26 135 -35.26 225 -35.26
i1 0 40 .5 300 40
```

4.4 AEP

In the last example the speaker signals for a regular octagon speaker setup (function table 17) of a sound source moving in the horizontal plain is calculated directly with the UDO *AEP*. Before applying the Doppler effect the UDO *Absorb*, a simple distance dependant low-pass filter simulates air absorption. A virtual sound source approaches the listener from the front (*az* = 0) goes through the origin and recedes in the opposite direction (*kdist line 2, p3, -2*).

```
#include "AEP_udos.txt"
#include "ambisonics_utilities.txt"

instr 1
ain        buzz       p4,p5,40,1
korder     line       1, p3, 17
kt         line       0,p3,p3
kx = 14*cos(0.61803*kt)
ky = 14*sin(kt)
kz         init       0
kdist      Dist       kx,ky
aabs       Absorb     ain,kdist
adop       Doppler .2*aabs,kdist
a1,a2,a3,a4,a5,a6,a7,a8  AEP  adop,korder,17,kx,ky,kz
outc       a1,a2,a3,a4,a5,a6,a7,a8
endin
...
f1 0 32768 10 1
f17 0 32 -2 1 .92 -.38 0 .92 .38 0 .38 .92 0 -.38 .92 0 -.92
.38 0 -.92 -.38 0 -.38 -.92 0 .38 -.92 0
i1 0 30 .8 300
```

5. CONCLUSION

The presented UDOs hopefully will be useful for spatial audio production and as a means for understanding, using and teaching Ambisonics. They have been implemented in such a way that they can be employed without detailed knowledge of the concepts of Ambisonics. In the future, enhancements, such as encoding and decoding higher than 8th order, conversion between semi-normalized and normalized spherical harmonics, conversion between different coordinate systems, near field compensation etc. will be implemented.

6. LISTING

The following listing shows the text files that must be included and the syntax of the implemented UDOs.

```
;#include "ambisonics_udos.txt"     (order <= 8)
k0   ambi_encode    asnd, iorder, kaz, kel
k0   ambi_enc_dist   asnd, iorder, kaz, kel, kdist
a1 [, a2] ... [, a8]   ambi_decode      iorder, ifn
a1 [, a2] ... [, a8]   ambi_dec_inph    iorder, ifn
f ifn  0 64 -2 p1 az1 el1 az2 el2 ...  (p1 is not used)
k0   ambi_write_B   "name", iorder, ifile_format
k0   ambi_read_B    "name", iorder (only <= 5)
kaz, kel, kdist   xyz_to_aed    kx, ky, kz

;#include "ambisonics2D_udos.txt"
k0   ambi2D_encode    asnd, iorder, kazimuth (any order)
k0   ambi2D_enc_dist  asnd, iorder, kaz, kdist
a1 [, a2] ... [, a8]   ambi2D_decode   iorder,
                       kaz1[, kaz2] ...[, kaz8]
a1 [, a2] ... [, a8]   ambi2D_dec_inph  iorder,
                       kaz1 [, kaz2] ... [, kaz8](order <= 12)
k0   ambi2D_write_B   "name", iorder, ifile_format
k0   ambi2D_read_B    "name", iorder (order <= 19)
kaz, kdist   xy_to_ad   kx, ky

#include "AEP_udos.txt" (any real order > 1)
a1 [, a2] ... [, a16]   AEP_xyz   asnd, korder, ifn,
                                  kx, ky, kz, kdist
f ifn 0 64 -2 max_speaker_dist x1 y1 z1 x2 y2 z2 ...
a1 [, a2] ... [, a8]   AEP  asnd, korder, ifn, kaz, kel, kdist
f ifn 0 64 -2 max_speaker_dist az1 el1 dist1 az2 el2 dist2
...

;#include "ambi_utilities.txt"
kdist      dist       kx, ky
kdist      dist       kx, ky, kz
ares       Doppler asnd, kdistance
ares       absorb    asnd, kdistance
kx, ky, kz            aed_to_xyz  kaz, kel, kdist
ix, iy, iz  aed_to_xyz   iaz, iel, idist
a1 [, a2] ... [, a16]  dist_corr a1 [, a2] ... [, a16], ifn
f ifn 0 32 -2 max_speaker_distance dist1, dist2, ... (in m)
```

7. REFERENCES

[1] M. A. Gerzon, “Periphony: With-height Sound Reproduction,” *Journal of the Audio Engineering Society*, Vol. 21 No. 1, 1973, pp. 2-10.

[2] J. Daniel, *Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia*. Ph.D. Thesis, University of Paris VI, France, 2000.

[3] J. Daniel ed. al., “Further Investigations of High Order Ambisonics and Wavefield Synthesis for Holophonic Sound Imaging,” in *AES 114st Convention*, Amsterdam, 2003.

[4] M. Neukom, *Signals, Systems and Sound Synthesis*. Peter Lang, 2013.

[5] B. G. Shinn-Cunningham, “Distance Cues for Virtual Auditory Space,” in *Proceedings of the First IEEE Pacific-Rim Conference on Multimedia*, Sydney, 2000, pp. 227-230.

[6] G. Kearney ed. al. “Perception in Interactive Virtual Acoustic Environments Using Higher Order Ambisonic Soundfields,” in *Proc. of the 2nd International Symposium on Ambisonics and Spherical Acoustics*, Paris, 2010.

[7] M. Neukom, “Ambisonic Panning,” in *AES 121st Convention*, New York, 2007.

[8] M. Neukom and J. C. Schacher, “Ambisonics Equivalent Panning,” in *Proceedings of the International Computer Music Conference*, Belfast, 2008.

[9] F. Zotter ed. al., “Ambisonic Decoding With and Without Mode-Matching: A Case Study Using the Hemisphere,” in *Proc. of the 2nd International Symposium on Ambisonics and Spherical Acoustics*, Paris, 2010.

[10] H. Pomberger ed. al., “An Ambisonics Format for Flexible Playback Layouts,” in *Proc. of the 1st Ambisonics Symposium*, Graz, 2009.

[11] http://en.flossmanuals.net/csound/ (accessed: 26. June 2014)

[12] http://www.icst.net/downloads (accessed: 26. June 2014)

[13] http://ambisonics.ch/ (accessed: 26. June 2014)