Csound Ambisonics UDOs

- **Usage of the ambisonics UDOs:**

The channels of the B-format are stored in a zak space. Call *zakinit* only once and put it outside any instrument definition, in the orchestra file after the header. *zacl* clears the za space and is called after decoding. The B format of order *n* can be decoded in any order <= *n*.
The text files “ambisonics_udos.txt”, “ambisonics2D_udos.txt”, “AEP_udos.txt” must be located in the same folder as the csd files or included with full path.

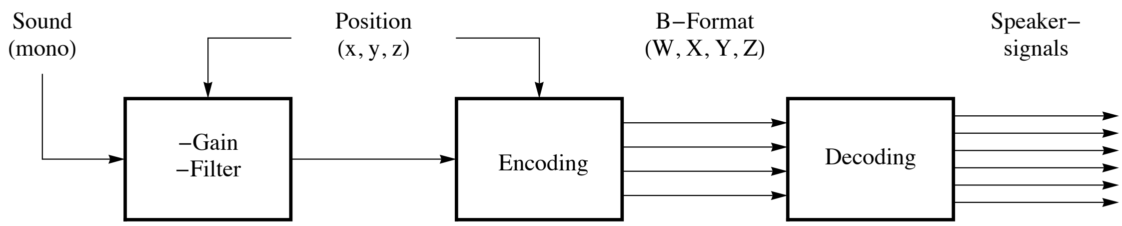
zakinit isizea, isizek (isizea = (order + 1)^2 in ambisonics (3D); isizea = 2·order + 1 in ambi2D; isizek = 1)

```
;#include "ambisonics_udos.txt"       (order <= 8)
k0                  ambi_encode       asnd, iorder, kazimuth, kelevation      (azimuth, elevation in degrees)
k0                  ambi_enc_dist     asnd, iorder, kazimuth, kelevation, kdistance
a1 [, a2] ... [, a8]   ambi_decode       iorder, ifn
a1 [, a2] ... [, a8]   ambi_dec_inph     iorder, ifn
      f ifn  0  n  -2 p1 az1 el1 az2 el2 ...   (n is a power of 2 greater than 3·number_of_spekers + 1) (p1 is not used)
k0                  ambi_write_B      "name", iorder, ifile_format    (ifile_format see fout in the csound help)
k0                  ambi_read_B       "name", iorder (only <= 5)
kaz, kel, kdist     xyz_to_aed        kx, ky, kz

;#include "ambisonics2D_udos.txt"     (any order)
k0                  ambi2D_encode     asnd, iorder, kazimuth                  (azimuth in degrees)
k0                  ambi2D_enc_dist   asnd, iorder, kazimuth, kdistance
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 order integer or fractional)
a1 [, a2] ... [, a16]  AEP_xyz    asnd, korder, ifn, kx, ky, kz, kdistance
      f ifn  0  64  -2  max_speaker_distance x1 y1 z1 x2 y2 z2 ...
a1 [, a2] ... [, a8]   AEP        asnd, korder, ifn, kazimuth, kelevation, kdistance (azimuth, elevation in degrees)
      f ifn  0  64  -2  max_speaker_distance az1 el1 dist1 az2 el2 dist2 ...  (azimuth, elevation in degrees)


#ambi_utilities
kdist               dist              kx, ky
kdist               dist              kx, ky, kz
ares                Doppler           asnd, kdistance
ares                absorb            asnd, kdistance
kx, ky, kz          aed_to_xyz        kazimuth, kelevation, kdistance
ix, iy, iz          aed_to_xyz        iazimuth, ielevation, idistance
a1 [, a2] ... [, a16]  dist_corr         a1 [, a2] ... [, a16], ifn
      f ifn  0  32  -2  max_speaker_distance dist1, dist2, ... (distances in m)
irad                radiani           idegree
krad                radian            kdegree
arad                radian            adegree
idegree             degreei           irad
kdegree             degree            krad
adegree             degree            arad
```

Introduction

In the following introduction we will explain the principles of ambisonics step by step and write an opcode for every step. The opcodes above combine all the functionalities described. Since the two-dimensional analogy to Ambisonics is easier to understand and to try out with a simple equipment we will explain it first and at full length.

Ambisonics is a technique of three-dimensional sound projection. The information about the recorded or synthesized sound field is encoded and stored in several channels, taking no account of the arrangement of the loudspeakers for reproduction. The encoding of a signal's spatial information can be more or less precise, depending on the so-called order of the algorithm used. Order zero corresponds to the monophonic signal and requires only one channel for storage and reproduction. In *first-order Ambisonics*, three further channels are used to encode the portions of the sound field in the three orthogonal directions x, y and z. These four channels constitute the so-called *first-order B-format*. When Ambisonics is used for artificial spatialization of recorded or synthesized sound, the encoding can be of an arbitrarily high order. The higher orders cannot be interpreted as easily as orders zero and one.
In a two-dimensional analogy to Ambisonics (called *Ambisonics2D* in what follows), only sound waves in the horizontal plane are encoded.
The loudspeaker feeds are obtained by decoding the B-format signal. The resulting panning is amplitude panning, and only the direction to the sound source is taken into account.
The illustration below shows the principle of Ambisonics. First a sound is generated and its position determined. The amplitude and spectrum are adjusted to simulate distance, the latter using a low-pass filter. Then the Ambisonic encoding is computed using the sound's coordinates. Encoding mth order B-format requires $n = (m + 1)^2$ channels ($n = 2m + 1$ channels in Ambisonics2D). By decoding the B-format one can obtain the signals for any number (>= n) of loudspeakers in any arrangement. Best results are achieved with symmetrical speaker arrangements.
If the B-format does not need to be recorded the speaker signals can be calculated at low costs and arbitrary order using so-called ambisonics equivalent panning (AEP).



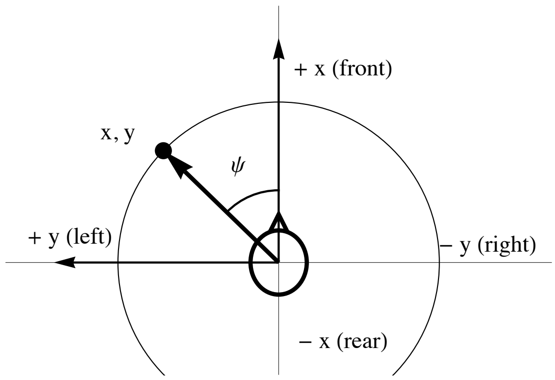
Ambisonics2D

We will first explain the encoding process in Ambisonics2D. The position of a sound source in 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 the sound source can also be given in polar coordinates by the angle ψ between the line of vision of the listener (front) and the direction to the sound source, and by their distance r. Cartesian coordinates can be converted to polar coordinates by the formulas

$$r = \sqrt{x^2 + y^2} \text{ and } \psi = \arctan(x, y),$$

polar to Cartesian coordinates by

$$x = r \cdot \cos(\psi) \text{ and } y = r \cdot \sin(\psi).$$



The 0th order B-Format of a signal S of a sound source on the unit circle is just the monosignal: $W_0 = W = S$. The first order B-Format contains two additional channels: $W_{1,1} = X = S\cdot\cos(\psi) = S\cdot x$ and $W_{1,2} = Y = S\cdot\sin(\psi) = S\cdot y$, i.e. the product of the Signal S with the sine and the cosine of the direction ψ of the sound source. The B-Format higher order contains two additional channels per order m: $W_{m,1} = S\cdot\cos(m\psi)$ and $W_{m,2} = S\cdot\sin(m\psi)$.

$$
\begin{array}{ll}
W_0 = S & \\
W_{1,1} = X = S\cdot\cos(\psi) = S\cdot x & W_{1,2} = Y = S\cdot\sin(\psi) = S\cdot y \\
W_{2,1} = S\cdot\cos(2\psi) & W_{2,2} = S\cdot\sin(2\psi) \\
\ldots & \\
W_{m,1} = S\cdot\cos(m\psi) & W_{m,2} = S\cdot\sin(m\psi)
\end{array}
$$

From the $n = 2m + 1$ B-Format channels the loudspeaker signals p_i of n loudspeakers which are set up symmetrically on a circle (with angle φ_i) are:

$$p_i = \frac{1}{n}(W_0 + 2W_{1,1}\cos(\varphi_i) + 2W_{1,2}\sin(\varphi_i) + 2W_{2,1}\cos(2\varphi_i) + 2W_{2,2}\sin(2\varphi_i) + \ldots)$$

$$= \frac{2}{n}(\frac{1}{2} W_0 + W_{1,1}\cos(\varphi_i) + W_{1,2}\sin(\varphi_i) + W_{2,1}\cos(2\varphi_i) + W_{2,2}\sin(2\varphi_i) + \ldots)$$

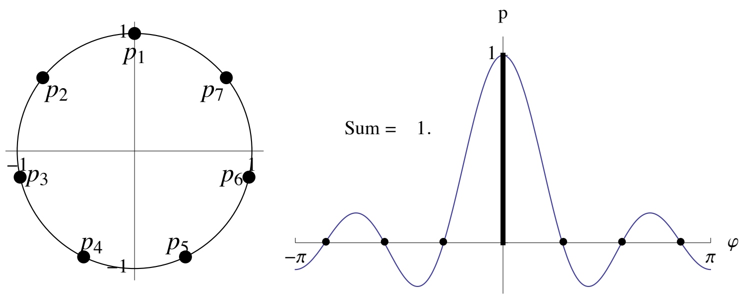
(If more than n speakers are used, we can use the same formula)

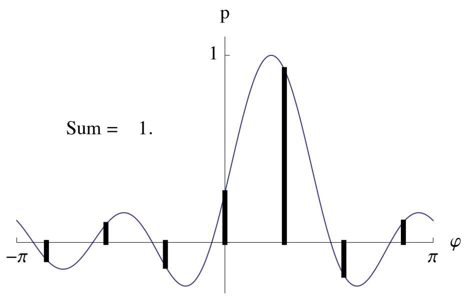
In the Csound example udo_ambisonics2D_1.csd the opcode ambi2D_encode_1a produces the 3 channels W, X and Y (a0, a11, a12) from an input sound and the angle ψ (azmuth kaz), the opcode ambi2D_decode_1_8 decodes them to 8 speaker signals a1, a2, ..., a8. The inputs of the decoder are the 3 channels a0, a11, a12 and the 8 angles of the speakers. (→ udo_ambisonics2D_1)

The B-format of all events of all instruments can be summed before decoding. Thus in the example udo_ambisonics2D_2.csd we create a zak space with 21 channels (zakinit 21, 1) for the 2D B-format up to 10th order where the encoded signals are accumulated. The opcode ambi2D_encode_3 shows how to produce the 7 B-format channels a0, a11, a12, ..., a32 for third order. The opcode ambi2D_encode_n produces the 2(n+1) channels a0, a11, a12, ..., n32 for any order n (needs zakinit 2(n+1), 1). The opcode ambi2D_decode_basic is an overloaded function i.e. it decodes to n speaker signals depending on the number of in- and outputs given (in this example only for 1 or 2 speakers). Any number of instruments can play arbitrary often. Instrument 10 decodes for the first 4 speakers of a 18 speaker setup.

- **In-phase Decoding**

The left figure below shows a symmetrical arrangement of 7 loudspeakers. If the virtual sound source is precisely in the direction of a loudspeaker, only this loudspeaker gets a signal (center figure). If the virtual sound source is between two loudspeakers, these loudspeakers receive the strongest signals, all other loudspeakers have weaker signals, some with negative amplitude, that is, reversed phase (right figure).





To avoid having loudspeaker sounds that are far away from the virtual sound source and to ensure that negative amplitudes (inverted phase) do not arise, the B-format channels can be weighted before being decoded. The weighting factors depend on the highest order used (M) and the order of the particular channel being decoded (m).

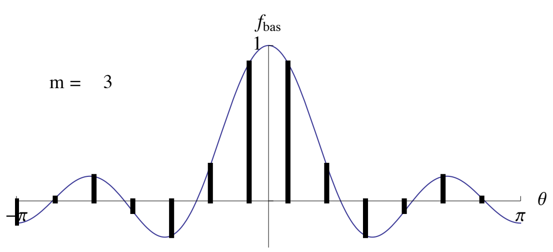
$$g_m = \frac{(M!)^2}{(M+m)!\cdot(\mathrm{M}-\mathrm{m})!}$$

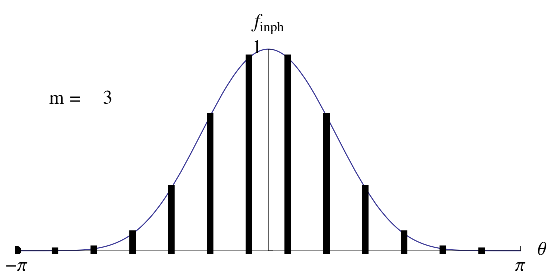
M			g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8
1		1	0.5							
2		1	0.666667	0.166667						
3		1	0.75	0.3	0.05					
4		1	0.8	0.4	0.114286	0.0142857				
5		1	0.833333	0.47619	0.178571	0.0396825	0.00396825			
6		1	0.857143	0.535714	0.238095	0.0714286	0.012987	0.00108225		
7		1	0.875	0.583333	0.291667	0.1060601	0.0265152	0.00407925	0.000291375	
8		1	0.888889	0.622222	0.339394	0.141414	0.043512	0.009324	0.0012432	0.0000777

The decoded signal can be normalized with the factor $g_{norm}(M) = \frac{(2M+1)!}{4^M\,(M!)^2}$

M	1	2	3	4	5	6	7	8
$g_{norm}(M)$	1	0.75	0.625	0.546875	0.492188	0.451172	0.418945	0.392761

The illustration below shows a third-order B-format signal decoded to 13 loudspeakers first uncorrected (so-called *basic decoding*, left), then corrected by weighting (so-called *in-phase decoding*, right).

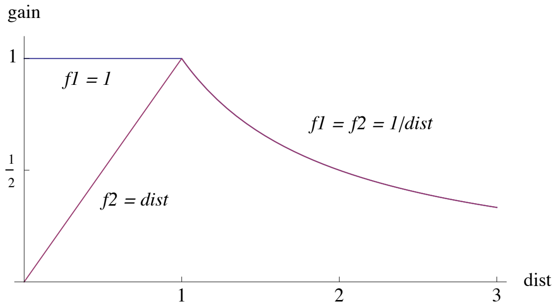




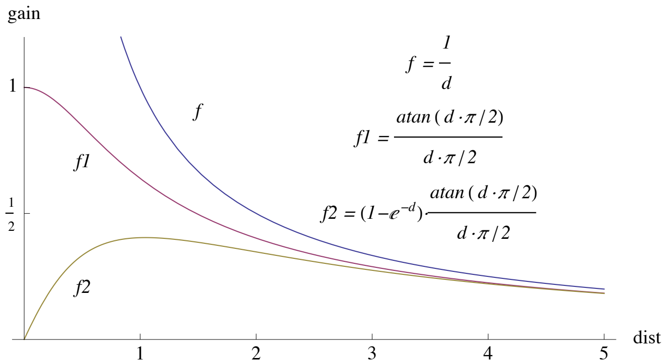
Example udo_ambisonics2D_3.csd shows in-phase decoding. The weights and norms up to 12th order are safed in the arrays iWeight2D[][] and iNorm2D[] respectively. Instrument 11 decodes third order for 4 speakers in a square.

■ Distance

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 absorbtion of the air and the relation between direct and indirect sound. We will implement the first two of these cues. 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.



Now, we are looking for gain functions that are smoother at $d = 1$. 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 $\mathrm{atan}(c \cdot d \cdot \pi/2)/(c \cdot d \cdot \pi/2)$ fulfills these constraints. We create a function *f2* for the fading out of the other channels by multiplying *f1* with the factor $(1 - e^{-d})$.



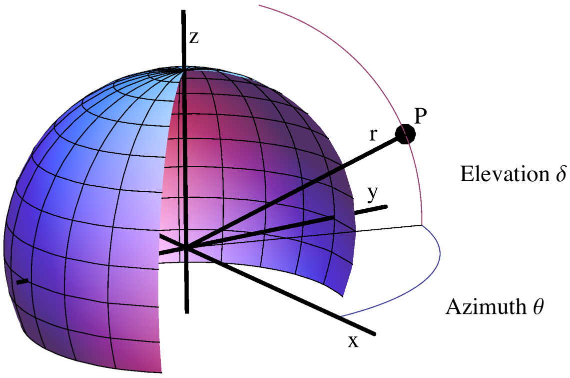
..... with parameters

In example udo_ambisonics2D_4 the UDO ambi2D_enc_dist_n 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 taken positive.

In order to simulate the absorption of the air we introduce a very simple lowpass filter with a distance depending cutoff frequency. We produce a Doppler-shift with a distance dependent delay of the sound. Now, we have to determine our unit since the delay of the sound wave is calculated as distance devided by sound velocity. In our example udo_ambisonic-s2D_5.csd we set the unit to 1 meter. These procedures are performed before the encoding. In instrument 1 the movement of the sound source is defined in Cartesian coordinates. The UDO xy_to_ad transformes them into polar coordinates. The B-format channels can be written to a sound file with the opcode fout. The UDO write_ambi2D_2 writes the channels up to second order into a sound file.

■ Ambisonics (3D)

The position of a point in space can be given by its Cartesian coordinates x, y and z or by its spherical coordinates the radial distance r from the origin of the coordinate system, the elevation δ (which lies between $-\pi$ and π) and the azimuth angle θ.



The formulas for transforming coordinates are as follows:

$$x = r\cdot\cos(\delta)\cos(\theta) \qquad y = r\cdot\cos(\delta)\sin(\theta) \qquad z = r\cdot\sin(\delta)$$

$$r = \sqrt{x^2 + y^2 + z^2} \qquad \theta = \arctan(y/x) \qquad \delta = \operatorname{arccot}\left(\frac{\sqrt{x^2+y^2}}{z}\right)$$

The channels of the Ambisonic B-format are computed as the product of the sounds themselves and the so-called spherical harmonics representing the direction to the virtual sound sources. The spherical harmonics can be normalized in various ways. We shall use the so-called s*emi-normalized spherical harmonics*. The following table shows the encoding functions up to the third order as function of azimuth an elevation $Y_{mn}(\theta, \delta)$ and as function of x, y and z $Y_{mn}(x, y, z)$ for sound sources on the unit sphere. The decoding formulas for symmetrical speaker setups are the same.

m	n	$Y_{mn}(\theta, \delta)$	$Y_{mn}(x, y, z)$
1	0	$\text{Sin}[\delta]$	z
	1	$\text{Cos}[\delta]\,\text{Cos}[\theta]$	x
	−1	$\text{Cos}[\delta]\,\text{Sin}[\theta]$	y
2	0	$\frac{1}{2}\left(-1 + 3\,\text{Sin}[\delta]^2\right)$	$\frac{1}{2}\left(-1 + 3 z^2\right)$
	1	$\frac{1}{2}\sqrt{3}\,\text{Cos}[\theta]\,\text{Sin}[2\,\delta]$	$\sqrt{3}\, x z$
	−1	$\frac{1}{2}\sqrt{3}\,\text{Sin}[2\,\delta]\,\text{Sin}[\theta]$	$\sqrt{3}\, y z$
	2	$\frac{1}{2}\sqrt{3}\,\text{Cos}[\delta]^2\,\text{Cos}[2\,\theta]$	$\frac{1}{2}\left(\sqrt{3}\, x^2 - \sqrt{3}\, y^2\right)$
	−2	$\sqrt{3}\,\text{Cos}[\delta]^2\,\text{Cos}[\theta]\,\text{Sin}[\theta]$	$\sqrt{3}\, x y$
3	0	$\frac{1}{8}(3\,\text{Sin}[\delta] - 5\,\text{Sin}[3\,\delta])$	$\frac{1}{2} z\left(-3 + 5 z^2\right)$
	1	$\frac{1}{8}\sqrt{\frac{3}{2}}\,(\text{Cos}[\delta] - 5\,\text{Cos}[3\,\delta])\,\text{Cos}[\theta]$	$\frac{1}{4}\left(-\sqrt{6}\, x + 5\sqrt{6}\, x z^2\right)$
	−1	$\frac{1}{8}\sqrt{\frac{3}{2}}\,(\text{Cos}[\delta] - 5\,\text{Cos}[3\,\delta])\,\text{Sin}[\theta]$	$\frac{1}{4}\left(-\sqrt{6}\, y + 5\sqrt{6}\, y z^2\right)$
	2	$\frac{1}{2}\sqrt{15}\,\text{Cos}[\delta]^2\,\text{Cos}[2\,\theta]\,\text{Sin}[\delta]$	$\frac{1}{2}\left(\sqrt{15}\, z - 2\sqrt{15}\, y^2 z - \sqrt{15}\, z^3\right)$
	−2	$\sqrt{15}\,\text{Cos}[\delta]^2\,\text{Cos}[\theta]\,\text{Sin}[\delta]\,\text{Sin}[\theta]$	$\sqrt{15}\, x y z$
	3	$\frac{1}{2}\sqrt{\frac{5}{2}}\,\text{Cos}[\delta]^3\,\text{Cos}[3\,\theta]$	$\frac{1}{4}\left(\sqrt{10}\, x^3 - 3\sqrt{10}\, x y^2\right)$
	−3	$\frac{1}{2}\sqrt{\frac{5}{2}}\,\text{Cos}[\delta]^3\,\text{Sin}[3\,\theta]$	$\frac{1}{4}\left(3\sqrt{10}\, x^2 y - \sqrt{10}\, y^3\right)$

In the first 3 of the following examples we will not produce sound but display in number boxes the amplitude of 3 speakers at positions (1, 0, 0), (0, 1, 0) and (0, 0, 1) in Cartesian coordinates. The position of the sound source can be changed with the two scroll numbers. The example udo_ambisonics_1.csd shows encoding up to second order. The decoding is done in two steps. First we decode the B-format for one speaker. In the second step, we create a overloaded opcode for n speakers. The number of output signals determines which version of the opcode is used. The opcodes ambi_encode and ambi_decode up to 8th order are saved in the text file "ambisonics_udos.txt".

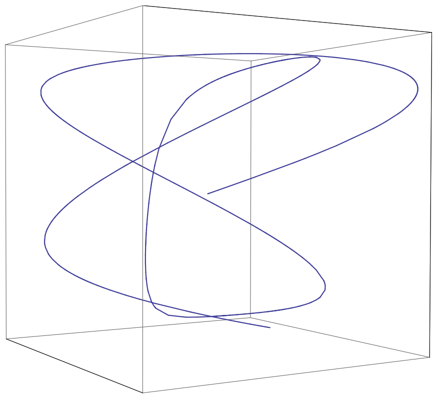
Example udo_ambisonics_2.csd shows in-phase decoding. The weights up to 8th order are safed in the arrays iWeight3D[][].

The weighting factors for in-phase decoding of Ambisonics3D are:

M			g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8
1		1	0.333333							
2		1	0.5	0.1						
3		1	0.6	0.2	0.0285714					
4		1	0.666667	0.285714	0.0714286	0.00793651				
5		1	0.714286	0.357143	0.119048	0.0238095	0.0021645			
6		1	0.75	0.416667	0.166667	0.0454545	0.00757576	0.000582751		
7		1	0.777778	0.466667	0.212121	0.0707071	0.016317	0.002331	0.0001554	
8		1	0.8	0.509091	0.254545	0.0979021	0.027972	0.00559441	0.000699301	0.000041135

Example udo_ambisonics_3.csd shows distance encoding.

In example udo_ambisonics_4.csd a buzzer with the three-dimensional trajectory shown below is encoded in third order and decoded for a speaker setup in a cube (f17).



■ 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)

$$P(\gamma, m) = \left(\frac{1}{2} + \frac{1}{2}\cos\gamma\right)^m$$

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 Ambisonics order simulated. 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 in the example udo_AEP.csd shows the calculation of 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 ist implemented for up to 16 speakers. The position of the speakers must be written in a function table. As the first parameter in the function table the maximal speaker distance must be given.

■ Utilities

dist computes the disance from the origin (0, 0) or (0, 0, 0) to a point (x, y) or (x, y, z)
```
kdist  dist   kx, ky
kdist  dist   kx, ky, kz
```

Doppler simulates the Doppler-shift
```
ares   Doppler    asnd, kdistance
```

absorb is a very simple simulation of the frequency dependant absorption
```
ares   absorb     asnd, kdistance
```

aed_to_xyz converts polar coordinates to Cartesian coordinates
```
kx, ky, kz   aed_to_xyz   kazimuth, kelevation, kdistance
ix, iy, iz   aed_to_xyz   iazimuth, ielevation, idistance
```

dist_corr induces a delay and reduction of the speaker signals relativ to the most distant speaker.
```
a1 [, a2] ... [, a16]   dist_corr   a1 [, a2] ... [, a16], ifn
      f ifn  0  32  -2  max_speaker_distance dist1, dist2, ... (distances in m)
```

radian (radiani) converts degrees to radian

irad	radiani	idegree
krad	radian	kdegree
arad	radian	adegree

degree (degreei) converts radian to degrees

idegree	degreei	irad
kdegree	degree	krad
adegree	degree	arad

- **Standard speaker setups**