What *is* Chumbi?
An API Reference for ChucK's Ambisonics Package

Everett M. CARPENTER

November 19, 2025

Created by Everett M. Carpenter while completing a
Bachelor of Science at Rensselaer Polytechnic Institute.

Abstract

This document serves as active reference for the ChucK package titled Chumbi. Chumbi serves as a hub for ambisonic processors within ChucK, hosting encoders, decoders, soundfield utilities, and microphone signal converters.

1 Introduction

At the beginning of my time at Rensselaer Polytechnic Institute (RPI), I began using the ChucK programming language. ChucK is a programming language which grants programmers the ability to control the flow of time ([**?**]). I quickly developed many synthesizers, samplers, audio effects, etc.. While constructing these, I begun researching optimal ways of controlling loudspeaker arrays. As a result of this research, I became increasingly interested in ambisonics. Utilizing resources such as Institute of Electronic Music and Acoustics (IEM) and the Audio Engineering Society (AES), I read myself into the acoustic and mathematical theory of ambisonics. Inside ChucK, I assembled a very immature system for ambisonic processing utilizing gain altering unit generators (UGens). If I recall correctly, this system was initially titled ”ChucK-bisonics” and utilized a custom ChucK class called ”AmbiMath” to calculate spherical harmonic values. If it were to be maintained, it was important that this code leaves the higher level ChucK programming language. So, it was rewritten in C++ and brought into ChucK via the Chugin framework ([**?**]). Due to the project being rewritten in C++, performance was fast and reliable. Now, Chumbi hangs out inside of ChucK's package manager ChuMP ([**?**]), where I continue to update it. That is the basis of this document; an ever-growing manual of Chumbi to allow new (and old) users to reference it. It will contain examples using Chumbi, as well as all the API reference you could need.

2 Encoders

As of November 19, 2025, Chumbi only provides one ambisonic encoder, EncodeN. EncodeN operates at orders 1-5, has one input, and outputs $(N + 1)^2$ channels. The following defines all member functions of EncodeN.

2.1 EncodeN Class

`geti()` **Signature:** `float geti(int index)`
Description: Get the spherical harmonic value at the specified index.
Parameters: `index` (int): Index of spherical harmonic (0-based)

`seti()` **Signature:** `void seti(float sh, int index)`
Description: Sets the spherical harmonic value at the specified index.
Parameters: `sh` (float): Value to set
`index` (int): Target index

`coeff()` **Signature:** `float[] coeff()`
Description: Get all spherical harmonics actively being used.
Parameters: None

`pos()` **Signature:** `float[] pos()`
Description: Same as coeff()
Parameters: None

`pos()` **Signature:** `void pos(float azimuth, float zenith)`
Description: Set the position of the encoder.
Parameters: `azimuth` (float): In degrees, 0 is straight forward
`zenith` (float): In degrees, 90 is directly above and -90 is below.

`azi()` **Signature:** `float azi()`
Description: Get the azimuth value currently being used.
Parameters: None

`azi()` **Signature:** `void azi(float azimuth)`
Description: Set the azimuth value to be used.
Parameters: `azimuth` (float): In degrees, 0 is straight forward.

`zeni()` **Signature:** `float zeni()`
Description: Get the zenith value currently being used.
Parameters: None

`zeni()` **Signature:** `void zeni(float zenith)`
Description: Set the zenith value to be used.
Parameters: `zenith` (float): In degrees, 0 is horizontal, 90 is above, -90 is below.

3 Decoders

As of November 19, 2025, Chumbi only provides 2ambisonic decoders. They operate at orders 1-5, they have $(N + 1)^2$ inputs and outputs. Their outputs can be interpreted as loudspeaker signals, with inputs as B-Format signals. The following are member functions inherited by all decoders in Chumbi.

3.1 DecodeN Class

`placement()` **Signature:** `float[2][] placement()`

- **Description:** Get all spherical harmonics actively being used.
- **Parameters:** None

`placement()` **Signature:** `void placement(float[2][])`

- **Description:** Set the speaker placements to be used.
- **Parameters:** `speakerAngles` (float[2][]): Pairs of speaker azimuth and zenith angles.

`weights()` **Signature:** `float[] weights()`

- **Description:** Get all weights actively being used.
- **Parameters:** None

`weights()` **Signature:** `void weights(float[])`

- **Description:** Set all weights to be used.
- **Parameters:** `weights` (float[]): Set weighting system to be used while decoding.