VVSDK Standard 1.0

Introduction

This document describes VVSDK. VVAudio's VVSDK implements a complete first order ambisonic signal chain including encode, rotate, and decode. This distribution of VVSDK includes these three modules, which implement the VVSDK API, and two programs, which use it. Note that, for intellectual property reasons, the encode module is included as binary only. The rotate and encode modules, as well as the test program vvcli, come with complete source.

VVSDK API

VVSDK includes two types of interface. Both interfaces invoke the same signal processing code, however, they vary in how the processing parameters get set.

Note: for detailed documentation of the VVSDK API, see the doxegen generated documentation included with the source code.

High Level

A single include file, VVAmbisonic.h, defines the high level interface to VVSDK. There are classes for encode, rotate, and decode, each of which has only the minimum methods needed to access all of a modules functions. The use of presets keeps the interfaces to the encode and decode modules simple.

Header File	Class
VVAmbisonic.h	VVTetra – tetrahedral mic encoder
	VVRotate – rotate/tilt/tumble, flip x/y/z
	VVDecode – linear decoder

Low Level

Each module also provides a low level interface, which allows setting each parameter individually in addition to the functions provided in the high level interface. Access to the individual parameters of the decoder allows the user to create custom speaker arrangements or change options like the use of near field compensation (NFC) or shelf filters.

Header File	Class

AVVTetra.h	AVVTetra – tetrahedral mic encoder
AVVRotate.h	AVVRotate – rotate/tilt/tumble, flip x/y/z
AVVDecode.h	VVDecode – linear decoder

Modules

Tetra – Tetrahedral Mic Encoder

VVTetra converts an A-format signal from a tetrahedral microphone into B-format. Custom calibration files for each mic, supplied in the convolution matrix format, are read and used to control the conversion.

The `calibrationFilePath` passed to the constructor determines the directory searched for calibration files. There may be up to eight calibration files of the form '*.txt' with corresponding subdirectories for the filter definitions.

The `programName` passed to the `setProgram` method corresponds to the name of the calibration file without the '.txt', e.g. file '2003.txt' would be accessed by `vvTetra.setProgram("2003")`. If the program name given is not valid, -1 is returned and the previous program setting is retained.

Since filters may or may not exist for any given combination of sampling rate and calibration, `canProcess()` should be called whenever the program is changed.

Rotate - Spatial Processing

VVRotate performs rotate, tilt and tumble as well as mirroring. Positive rotation of a point directly in front moves it to the left. Positive tilt moves the same point up. Positive tumble rotates the scene counterclockwise. In general, motions refer to a movement of the scene relative to the microphone.

Using the normal process call, for each audio block, the current settings are compared to the previous settings to determine if interpolation is required. If they are the same, then the previous settings are simply used. If they are different, then the total transformations for the new and old settings are each computed and linear interpolation is used to smoothly switch between them.

If even faster rotation rates are required, then the processFast call can be used. This call takes three extra audio buffers on input that contain the rotate, tilt and

tumble angles. The angles are checked for changes at every sample to and the rotation matrix recomputed if needed.

Decode – Linear Ambisonic Decoder

VVMic converts a B-Format signal into speaker feeds according to the chosen program using first order, linear ambisonic processing. It optionally includes shelf filters and near field compensation. If desired, a matrix pseudo-inverse algorithm can used to determine the decode matrix from the speaker positions.

VVMic includes the following presets:

Program Name	Channels	Description
XY	2	90 degree cardioids
ORTF-ish	2	109 degree hypercardioids
180-cards	2	180 degree cardioids
Blumlein	2	90 degree figure 8's
Square	4	RF, LF, LS, RS
5.1	6	L R C Lfe Ls Rs
Hexagon	6	Counterclockwise starting at -30 degrees
Octagon	8	Counterclockwise starting at -22.5 degrees
Cube	8	Upper square then lower square
Rectangle	4	Sqrt(3) ratio rectangle

Programs

VVCLI – Command Line Interface

The program VVCLI serves both as a demonstration of the high level interface to VVSDK and as a general purpose ambisonic processing tool. VVCLI uses libsndfile for reading and writing sound files and uses PortAudio to provide realtime output. Using VVCLI one could create a batch file to encode a series of A-format files to B-format or one could play a B-format file to the default output device.

VVCLI currently supports the following parameters:

Parameter	Value	Description
-help		list command line options
-ambi-fmt	<format>	ambisonic format, FuMa or ACNSN3D
-in-file	<filename>	input file, must be 4 channel
-out-file	<filename>	output file, in the same format as the input
-encode-path	<calibration dir>	directory to search for calibration files
-encode-pgm	<program name>	name of calibration to use like '2003'
-rotate	<degrees>	rotate scene left relative to mic
-tilt	<degrees>	tilt scene up relative to mic
-tumble	<degrees>	tumble scene counterclockwise relative to mic
-flip-X		mirror front to back
-flip-Y		mirror left to right
-flip-Z		mirror top to bottom
-decode-pgm	<program name>	decode program like 'XY' or 'Cube'
-spin-rate	<degrees>	rate of rotation per block
-multi-angle	<degrees>	outputs multiple files rotated by given angle
-frontal-emphasis	<0.0-1.0>	makes the output more directional
-master-gain	<dB>	final output gain in dB like -3 or 2.3

Build

VVSDK targets primarily Visual Studio 2015. The included solution with multiple projects will build 32 or 64 bit, release or debug versions as long as the required 3rd party libraries are in place (see below). Makefiles are also included which should build on OSX or linux.

3rd Party Software

The program VVCLI uses libsndfile, which is licensed under the Lesser GNU Public License or LGPL. The header files used are included, unmodified in this distribution with their original copyright and license information. Note that a commercial application using libsndfile must link to it in the form of a shared

library, i.e. not use static linking. In order to run VVCLI, the appropriate (32 or 64 bit) DLL must be visible on the path. See: http://www.mega-nerd.com/libsndfile/

Similarly, libsndfile uses pthreads for Win32, which is licensed under the Lesser GNU Public License or LGPL. The header files used are included, unmodified in this distribution with their original copyright and license information. See: http://www.sourceware.org/pthreads-win32

VVCLI uses PortAudio for realtime output. PortAudio is released under a plain MIT license. No PortAudio source code is being redistributed. See: http://portaudio.com .

The library VVAmbisonic uses the Armadillo linear algebra library. This software is licensed under the Mozilla Public License 2.0 (MPL). The source code is license details for this software are available from: http://arma.sourceforge.net

The library VVTetra uses FFTReal version 1.01 from Laurent de Soras. This software may be freely used for commercial purposes as long as credit is given in documentation. See: http://ldesoras.free.fr

The following chart shows the install directories expected by VVSDK. To reiterate, only Armadillo and Eigen are required for the library modules. PortAudio and libsndfile are only required for building VVCLI.

Library	Directory
Libsndfile (32 bit)	C:\Program Files (x86)\Mega-Nerd\libsndfile
Libsndfile (64 bit)	C:\Program Files\Mega-Nerd\libsndfile
PortAudio	C:\SDKs\portaudio
Armadillo	C:\SDKs\armadillo
Eigen	C:\SDKs\eigen