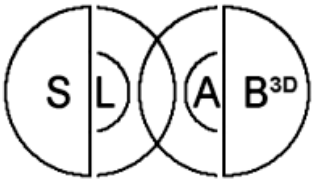
slab3d User Manual v6.8.2

Joel D. Miller



slab3d is a software-based real-time virtual acoustic environment (VAE) rendering system originally developed in the Spatial Auditory Displays Lab at NASA Ames Research Center. slab3d is now maintained externally as well.

Contents

Last Updated: 2017.05.12

Introduction

The slab3d release is a collection of applications, libraries, documentation, MATLAB utilities, and source code for virtual acoustic environment rendering and real-time audio signal processing. The list below provides a quick reference to some of the slab3d components.

- just curious about 3D-sound - see SLABScape
- rendering your own HRTFs - see slabtools and SLABScape
- writing experiment software - see SRAPI and slabcon
- writing virtual environment software - see SRAPI and SLABScape
- writing rendering algorithms - see Render Plugins and SLABScape
- writing signal generators - see Generator Plugins, SLABScape, and SLABSurface
- real-time audio signal processing - see slabwire and SLABWireDemo
- frequency-driven OpenGL displays - see SLABWireDemo
- network communications - see DISTrans, DISRec, AvadeServer, and SRAPI

Installation

Please see slab3d.sonisphere.com for information regarding slab3d system requirements, download, and installation.

Windows Sound Schemes
When listening to slab3d with headphones it is probably best to disable system sounds by selecting the "No Sounds" sound scheme under Control Panel | Sounds and Multimedia | Sounds | Scheme. Otherwise, an uncomfortably loud system sound might occur while listening to slab3d.

Speakers Property
Some sound peripherals process the output signal based on the type of display attached to the output (e.g., headphones versus desktop speakers). When using slab3d, it is best to select an unprocessed output path under Control Panel | Sounds and Multimedia | Audio | Advanced | Speakers | Speaker Setup.

ASIO
ASIO latency will most likely be less than DirectSound latency (e.g., ~4ms vs. ~23ms for a similar configuration). If your sound peripheral does not have an ASIO driver, you might be able to use the slab3d ASIO features by installing ASIO4ALL (www.asio4all.com). ASIO4ALL provides an ASIO interface wrapper for the standard Windows Driver Model (WDM) sound driver interface. ASIO4ALL v2.6 was successfully tested with the "Avance AC97 Audio" sound device. The impact of ASIO4ALL on slab3d latency has not been examined.

Citing slab3d

As a courtesy, the use of slab3d in published research should be acknowledged in the publication by citing the two slab3d home pages:

[1] http://slab3d.sonisphere.com/, http://humansystems.arc.nasa.gov/SLAB/

The preferred paper reference describing the slab3d release and its implementation:

[2] Miller, J. D. and Wenzel, E. M., "Recent Developments in SLAB: A Software-Based System for Interactive Spatial Sound Synthesis," Proceedings of the International Conference on Auditory Display, ICAD 2002, Kyoto, Japan, pp. 403-408, 2002.

Additional Documentation

This document and others are available in the release installation directory:

slab3d User Manual	<slab3d install dir>\doc\slab3d_user_manual.pdf
SRAPI Reference Manual	<slab3d install dir>\doc\ref\index.html
slabwire Reference Manual	<slab3d install dir>\doc\slabwire\index.html
slabtools User Manual	<slab3d install dir>\doc\slabtools_user_manual.pdf
slabtools Reference Manual	<slab3d install dir>\doc\slabtools\index.html

Websites:

slab3d home page	http://slab3d.sonisphere.com
slab3d SourceForge.net project page	http://sourceforge.net/projects/slab3d
NASA SLAB Home Page	http://humansystems.arc.nasa.gov/SLAB
NASA Open Source Software site	http://opensource.arc.nasa.gov

Papers: http://humansystems.arc.nasa.gov/groups/ACD/personnel_view.php?personnel_id=81

Acknowledgements

I would like to thank Beth Wenzel for her support, feedback, and for making slab3d possible, Martine Godfroy for helping make XScape and slabx possible, Jonathan Abel for early DSP and physical modeling assistance, Mark Anderson for programming assistance, Durand Begault for initial use and suggestions, Marlene Hernan, Robert Padilla, Robin Orans, Dave Encisco, Pat Moran, and Vance Dubberly for NASA release, NOSA, and sys admin efforts, the Air Force Research Laboratory for use and support, and spatial audiophiles everywhere for all of the papers, talks, discussions, compositions...

--joel

Developers

Joel Miller	lead designer and programmer
Jonathan Abel	provided physical modeling and signal processing MATLAB examples during the first year of development
Mark Anderson	developed or assisted with traklib Fastrak driver, SLABScape 3D View, BCF2000 interface, managed SRAPI
Mitch Clapp	provided SLABScape Direct3D head model and textures
Renee Goldschmid	developed five SLABWireDemo OpenGL displays, improved spectrum display
John Stewart	developed RT_DIS API and slabwire CSI_RT_DIS_Radio class

Open-Source Software

The following open-source software is used in slab3d development. Many thanks to the authors!

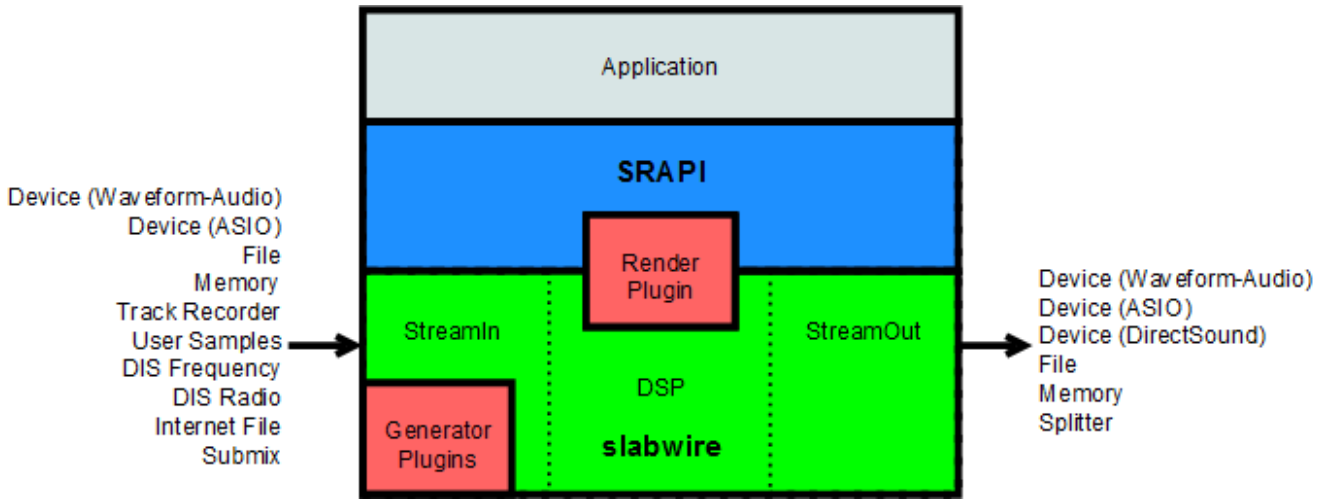
Software	How Used	Source
rtdis	low-latency DIS support	slab3d usrlib directory (from John Stewart, AFRL)
slabDISinterface	DIS support library	slab3d usrlib directory (permission granted by AFRL)
HighPrecisionTimer	AvadeClient Update Rate	https://github.com/mzboray/HighPrecisionTimer
ASIO	low-latency audio interface	http://www.steinberg.de/
STK	gstk generator plugin	http://ccrma.stanford.edu/software/stk/
Auditory Toolbox	cbe.m critical band filters	http://rvl4.ecn.purdue.edu/~malcolm/interval/1998-010/
fft.cpp	SLABWireDemo FFT	"A Programmer's Guide to Sound" by Tim Kientzle
GLUT	SLABWireDemo OpenGL displays	http://www.xmission.com/~nate/glut.html
Shortcut	SRAPI SetLinks()	http://www.codeguru.com
libresample	slabwcon, CResampler2	http://www-ccrma.stanford.edu/~jos/resample/Available_Software.html
VBAP	incomplete, rvbap.cpp	http://www.acoustics.hut.fi/software/vbap/Pure_Data/
doog2.m, gaussian.m	dog.m, mdog.m DOG generation	http://www.cs.berkeley.edu/~stellayu/code.html
wave_matlab	wmdemo.m wavelet transform	http://paos.colorado.edu/research/wavelets/
WaveLab	cwt.m wavelet transform scaling	http://www-stat.stanford.edu/~wavelab/
mp_fm2d.m	dustimg.m low-vis image	http://technion.ac.il/~pavel/comphy/
doxygen	documentation	http://www.stack.nl/~dimitri/doxygen/
Graphviz	documentation	http://www.graphviz.org/
Dia for Windows	documentation	http://dia-installer.de
m2html	documentation	http://www.artefact.tk/software/matlab/m2html/

Architecture, Libraries, and APIs

Architecture

The slab3d architecture is illustrated in the figure below.

slab3d Architecture



SRAPI API and Library: The SLAB Render API (SRAPI) provides a platform for the development of virtual acoustic environment (VAE) applications. Using SRAPI, the user can control rendering, allocate sound sources, configure frame-accurate callbacks, select sound output destinations, and specify acoustic scene and renderer parameters (e.g., sound source location, number of HRIR FIR taps). SRAPI also supports a Script and Modifier mechanism for the automatic updating of acoustic scene parameters.

Documentation:	SRAPI Reference Manual
Library:	<install dir>\lib\srapimr.lib (mr = multi-threaded, release)
Header Files:	<install dir>\include\
Source Code:	<install dir>\src\srapi\

slabwire API and Library: The slabwire library is used to configure a StreamIn-DSP-StreamOut sample processing chain. It is used internally by SRAPI but it can also be used as a stand-alone library. When using SRAPI, slabwire is used to control StreamIn sample streams and to develop Render Plugins and Generator Plugins. The SLABWireDemo application demonstrates the direct use of the slabwire API using the SoundIn-DSP-SoundOut framework. The slabwcon application demonstrates low-level slabwire use.

Documentation:	slabwire Reference Manual
Library:	<install dir>\lib\slabwmr.lib (mr = multi-threaded, release)
Header Files:	<install dir>\include\
Source Code:	<install dir>\src\slabw\

Render Plugin API: The VAE rendering engine is encapsulated in the Render Plugin (rplugin) "Spatial". rplugins can be swapped in and out while rendering allowing for the comparison of different rendering strategies. rplugins also allow for the construction of tailored displays for specific applications (e.g., a spatial communications system). See Included Plugins.

Documentation:	slabwire Reference Manual
Library:	Several render plugins are built into the srapi library. Additional render plugins can be placed in the SRAPI-based executable directory, e.g., <install dir>\bin\r*.dll.
Header Files:	<install dir>\include\srapi.h (for built-in render plugin use) and <install dir>\include\rplugin.h (for render plugin development)
Source Code:	<install dir>\src\r*\

Generator Plugin API: Signal generator sound sources are encapsulated in Generator Plugins (gplugins). This enables users to conveniently add additional signal generation algorithms to slab3d without modifying slab3d itself. See Included Plugins.

Documentation:	slabwire Reference Manual
Library:	Generator plugins are placed with the SRAPI-based executable, e.g., <install dir>\bin\g*.dll.
Header File:	<install dir>\include\gplugin.h
Source Code:	<install dir>\src\g*\

Static-state and Render-time, Static Settings and Dynamic Settings:

- static-state = state of SRAPI when not rendering
- render-time = state of SRAPI when rendering
- static settings = setting that must remain constant while rendering
- dynamic settings = settings that can be set at any time

Some static settings are sensitive to setting sequence, e.g.: SampleRate before StreamOuts before StreamIns. Outs and ins depend on the sample rate and ASIO-in depends on ASIO-out. The SRAPI Reference Manual states if a function is a static-state function. All other functions should be assumed to be dynamic functions. This differentiation primarily impacts the calling sequence (e.g., static, dynamic, render start, dynamic, render stop, static, ...) and error handling. There are similar constraints in slabwire.

SRAPI Sound Sources and StreamIns: An SRAPI "sound source" contains state information for a virtual environment sound emitter (Spatial render plugin) and a mixer channel strip (Mixer render plugin). The SRAPI SrcAlloc() function allocates a sound source and assigns it a user-specified ID. This ID is used to set source attributes via the Src*() functions. All sound sources must be allocated before calling RenderStart() to initiate rendering. The samples attached to a sound source are termed StreamIns and are allocated using the SiAlloc*() functions. StreamIns are attached to sound sources using the SrcStream() function. The various StreamIn types are shown in the figure above.

SRAPI Outputs: The SRAPI Out*() functions are used to define the destination of sample output. For the Spatial renderer, this is a binaural signal corresponding to the sampled sound pressure at the openings of the listener's ear canals. The various stream output types (aka StreamOuts) are shown in the figure above.

Support Libraries

usrlib libraries: Download information is available under Acknowledgements.

- **slabDISInterface** provides DIS (Distributed Interactive Simulation) packet support.
- **rtdis** provides an alternate DIS interface. It also includes a DIS transmitter.
- **traklib** is a Polhemus Fastrak electromagnetic tracker driver developed in the Spatial Auditory Displays Lab at NASA Ames Research Center. traklib is used by SLABScape to access the Fastrak.
- **STK** is "The Synthesis ToolKit in C++" and is used by the gstk Generator Plugin. See Included Plugins.

download libraries: Information for the download libraries is provided under Building slab3d.

- **ASIO** is a low-latency sound device interface.

Microsoft Visual C++ libraries: See the project settings for the various slab3d applications.

DirectX and XNA: See Building slab3d.

CSRAPI

The primary C++ interface used to access SRAPI is CSRAPI (srapi.h). CSRAPI is used to configure sound sources, input streams, and sound outputs and to control rendering. It is also used to specify the acoustic scene and renderer parameters.

The console app below demonstrates the minimum API necessary to render a sound source.

```
#include <windows.h>
#include "srapi.h"

int
main()
{
  CSRAPI  cSRAPI;
  if( cSRAPI.ErrorState() )
    return 1;  // error

  // specify Waveform-Audio device output
  cSRAPI.OutWave();

  // allocate a sound source, ID = 0
  cSRAPI.SrcAlloc( 0 )

  // allocate a looped wave file StreamIn, ID = 0
  if( cSRAPI.SiAllocFile( 0, "\\slab3d\\wavs\\voice.wav" ) != ERR_SLAB_NONE )
    return 1;  // error

  // attach ch 0 (third param) of stream 0 (second param) to
  // source 0 (first param)
  cSRAPI.SrcStream( 0, 0, 0 );

  // set source 0 attributes
  cSRAPI.SrcLocate( 0, 0.5, 0.5, 0.5 );  // x,y,z in meters
  cSRAPI.SrcGain( 0, 0.0 );              // dB
  cSRAPI.SrcEnable( 0, true );           // enable rendering

  // render source for 5 seconds
  cSRAPI.RenderStart();
  Sleep( 5000 );
  cSRAPI.RenderStop();
  return 0;  // success
}
```

CSRAPI Defaults

Render Plugin: The default render plugin used for auditory display rendering is the binaural HRTF-based renderer "Spatial" (Render ID RENDER_SPATIAL defined in slabdefs.h).

Render Plugin Directory: The render plugin directory is the executable directory.

Generator Plugin Directory: The default generator plugin directory is the executable directory. A generator plugin directory can also be specified when allocating a signal generator.

HRTF Database: The default HRTF database is <slab3d install dir>\hrtf\jdm.slh (sample rate = 44100 samples/s). This database is built into the srapi library (using hrtfhex.h generated by slabcon).

Setting	Default	Function
Sample Rate	44100 samples/s	SetSampleRate()
FIR Taps	128 taps for the direct path, 32 taps for reflections	SetFIRTaps()
Smooth Time	15ms parameter tracking filter time constant	SetSmoothTime()
Non-spatial Source Gain Offset	-24dB	SetNSOffset()
Source Location	origin, 0 x,y,z	SrcLocate()
Source Enable	not enabled	SrcEnable()
Source Reflection Enable	not enabled	SrcRefEnable()
Source Reflection Offset	disabled	SrcRefOffset()
Source Gain	0 scalar gain	SrcGainScalar()
Source Mute	not muted	SrcMute()
Source Radius	10cm	SrcRadius()
Source Spread Exponent	1	SrcRadius()
Room Depth (X)	4m	EnvPlanes()
Room Width (Y)	4m	EnvPlanes()
Room Height (Z)	4m, ceiling = 4m - dLstHeight, floor = -dLstHeight, dLstHeight = 1.8288m = 6'	EnvPlanes()
Wall Materials	none, i.e., perfect reflector	EnvMaterial()
Listener Position	head at origin, 0 x,y,z,yaw,pitch,roll	LstPosition()
Listener Sensor Offset	7.5" above interaural axis	LstSensorOffset()
Delay Line Length	500ms	SetDelayIn()
Level Meter Attack Time Constant	137ms	SetLevelTimes()
Level Meter Release Time Constant	137ms	SetLevelTimes()
Skip Silence Threshold	0.05 linear gain	SetSkipSilence()
Skip Silence Time	0.5s	SetSkipSilence()
Skip Silence Fade-In	68ms	SetSkipSilence()
Tick Updating	disabled, i.e., use frame updating	SetTick()

Some of the defaults in the table have corresponding defines in slabdefs.h. "Sample Rate" through "Listener Sensor Offset" are reset to their defaults via CSRAPI::SetDefaults().

Coordinate Systems

slab3d uses a right-handed, FLT (Front-Left-Top) coordinate system, i.e., if fingers curled from x (front) to y (left), thumb points to z (top).

Location		
+x front, through nose	+y left, through left ear	+z top, through top of head

Orientation (positive CCW looking down axis towards origin)		
-yaw to right +yaw to left	-pitch up +pitch down	+roll right -roll left

Polar		
+azimuth to right -azimuth to left	+elevation up -elevation down	+range forward -range backward

Convolvotron and Polhemus Azimuth and Elevation

These definitions are compatible with the Polhemus Isotrak and Fastrak head trackers and the Crystal River Engineering Convolvotron coordinate systems with the following exceptions:

slab3d azimuth = - Convolvotron and Polhemus azimuth
slab3d and Convolvotron elevation = - Polhemus elevation

Polhemus Transmitter and Receiver

The physical placement of the Polhemus transmitter and receiver often results in the following configuration (as specified on the Fastrak transmitter): +X forward, +Y right, +Z down. Since slab3d uses +Z up and +Y left, the signs of Y, Z, Yaw, and Pitch must be changed if your tracker driver follows this convention.

Microsoft XNA

XNA uses a right-handed right_x-top_y-back_z coordinate system with +yaw left, +pitch up, +roll left. The X-prefixed Managed SRAPI functions are provided for use with XNA.

Latitude-Longitude-Height (LLH) and Aircraft Rotations

latitude in degrees, +90N to -90S
longitude in degrees, -180W to +180E

Aircraft orientation angles rotate the slab3d coordinate system 180° about the x (roll, nose) axis, thus:

aircraft yaw = - slab3d yaw
aircraft pitch = - slab3d pitch

Render and Generator Plugins

All audio rendering in slab3d is performed with render plugins (rplugins). The base class of all rplugins is CRPlugIn. The default rplugins are built into the srapi library. Additional rplugins can be built as DLLs and placed in the SRAPI-based executable directory. All rplugins found in this directory will be read into memory when a CSRAPI object is allocated. The SRAPI Render Functions exist for querying and selecting rplugins. Through the CSRAPI class, rplugins are given access to SRAPI's scene parameters, input delay lines, and output stream. The rplugin DLL and CSRAPI interface is only supported by SRAPI; rplugins can be created and used as CRPlugIn subclasses in slabwire.

All signal generation in slab3d is performed with generator plugins (gplugins). Similar to rplugins, gplugins are subclassed from CGPlugIn, built as DLLs, and placed in the slab3d-based executable directory. A gplugin DLL directory can also be specified when allocating a generator. gplugins are completely implemented in slabwire and are thus accessible from both slabwire and SRAPI.

See Also: CSRAPI, CRPlugIn, CGPlugIn, and gslab.h in the SRAPI Reference Manual.

Starter Projects

The *rmyplugin* project is the starter project for creating user Render Plugins. The *gslab* project is the starter project for creating user Generator Plugins. Both projects contain functioning example code that illustrates where to place your new custom code.

Follow the steps below to create your own plugin using Microsoft Visual C++:

1. Create a new Win32 DLL project.
2. Delete the generated .cpp from the project and from the directory.
3. Copy *rmyplugin.cpp* (Render Plugin) or *gslab.cpp* (Generator Plugin) from slab3d's *src*\ directory to your new project directory. Rename it if you wish. Add it to the project.
4. Add the slab3d *include*\ and *include**slabw* directories to the Additional Include Directories compiler property.
5. For slab3d to find your DLL, it must reside in the slab3d-based executable's directory. Set the linker Output File property as follows:
    1. For debug DLLs, enter the name of your DLL appended to the corresponding executable's path. E.g., "<slab3d install dir>/bin/r*d.dll"
       where '*' denotes a name of your choosing, 'r' indicates the DLL is a "render" plugin, 'd' indicates the DLL is the debug version of the DLL.
    2. The release DLL format for the example above is "<slab3d install dir>/bin/r*.dll".
    3. For Generator Plugins, replace the 'r' prefix with a 'g' prefix.

Source Code: The source code for rmyplugin is in <install dir>\src\rmyplugin\. The source code for gslab is in <install dir>\src\gslab\.

slab3d Release Plugins

The following Render Plugins and Generator Plugins are included in the slab3d release. Render Plugins are prefixed with an 'r' and Generator Plugins are prefixed with a 'g'.

- rspatial - contains "Spatial", the slab3d auralization algorithm. The rspatial source code contains the CSpatial class that can be used as a baseclass for special purpose spatial displays (e.g., rsar).
- rdiotic - contains the following Render Plugins: "Diotic", "Left-Monotic", "Right-Monotic", and "Dichotic".
- rmyplugin - contains the "Plugin Example" plugin, an extremely simple diotic display plugin that is useful as a Render Plugin starter project.
- rmixer - contains the "Mixer" plugin for use with the CSRAPI::SrcMix*() functions.
- rsar - was developed for a Search And Rescue communications application. It demonstrates how to extend "Spatial" for special-purpose applications. It also demonstrates the use of the SRAPI extension Plugin Variables. An rsar DLL is not included in the slab3d release and rsar is not supported by SLABScape.
- gslab - contains "noise", "sine", "square", "triangle", "am" (amplitude modulation), and "impulse" generators. In addition to providing slab3d's default signal generators, gslab also serves as the Generator Plugin starter project.
- gstk - wraps a Generator Plugin interface around the STK (Synthesis Toolkit) instruments. For more information regarding STK, see http://ccrma.stanford.edu/software/stk/.

Binaries:

- Built-in Render Plugins: <install dir>\lib\srapimr.lib
- Additional Render Plugins: <install dir>\bin\r*.dll
- Generator Plugins: <install dir>\bin\g*.dll

Source Code:

- Built-in Render Plugins: <install dir>\lib\src\srapi\<plugin name>
- DLLs: <install dir>\src\<plugin name>\

Spatial Render Plugin

Audio rendering in slab3d is divided between the SRAPI library and Render Plugins. The default Render Plugin is the HRTF-based "Spatial" plugin. The listener HRTF database used by Spatial contains minimum-phase head-related impulse response (HRIR) pairs and interaural time delays (ITDs) at fixed azimuth and elevation increments. The azimuth and elevation increments can vary from one database to another. The slabwire frame size is 32 samples, meaning sound samples are routed through the signal processing chain 32 samples at a time. The sample data type is single-precision floating-point and all calculations are performed using single or double-precision floating-point arithmetic. Every frame, the Render Plugin receives a frame of samples. For a sample rate of 44,100 samples/s, the frame rate is 1378 frames/s. Every frame the following processing occurs:

Script, Trajectory, and Callback Update - updates the script, trajectory and callback mechanisms at update rates defined by the user. The callback feature allows user code (e.g., a custom source trajectory) to run inside the slabwire thread.

Acoustic Scene Update - converts scene parameters to DSP parameters. Spatial performs a scene update each time the user updates a scene API parameter (e.g., listener position). The maximum update rate depends on available CPU resources. Since the HRIR FIR filter coefficients are updated every other frame, the absolute maximum update rate is 690Hz. A typical scene update rate is 120Hz.

- *Acoustic Scene Update:*
    - *Tracker Sensor Offset* - compensates for the location of the head tracker sensor.
    - *Image Model* - computes the location of sound source reflection images.
    - *3D Projection* - converts scene information into listener-relative geometric quantities:
        - *Image-Listener Range*
        - *Image Arrival Angle*
- *Signal Flow Translation* - converts listener-relative geometric quantities into FIR coefficients and delay line indices (aka “DSP parameters”) for each sound path and for each of the listener’s ears, modeling:
    - *Propagation Delay*
    - *Spherical Spreading Loss*
    - *HRTF Database Interpolation (FIR Coefficients, ITD)*

Process - processes each sound image, performing the following signal processing tasks:

- *Delay Line* - models propagation delay and ITD.
    - *Delay Line Indices Parameter Tracking* - bumps current delay line indices towards target values every sample.
    - P*rovided by slabwire layer:*
        - *Interpolated Delay Line* - implements a 2x up-sampled, linearly interpolated, fractionally indexed delay line.
- *IIR Filter* - models wall materials with a first-order IIR filter.
- *FIR Filter* - models spherical spreading loss and head related transfer functions.
    - *FIR Coefficient Parameter Tracking* - bumps current FIR coefficients towards target values every other frame.
    - *FIR Filter Operation* - implements an arbitrary length FIR filter. Typically, the direct path is computed with 128 taps and each reflection with 32 taps.

Mix - mixes the direct path and six first-order reflections for an arbitrary number of sound sources.

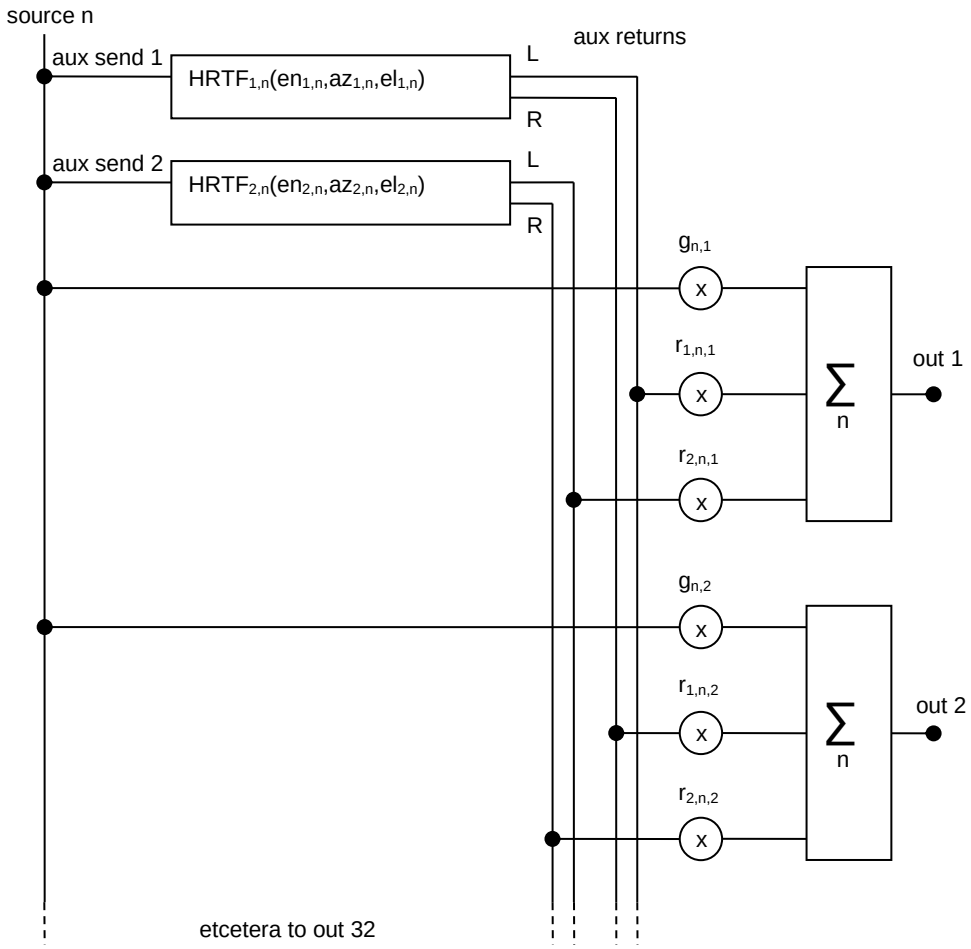
Modeling Notes

- ITD is implemented by lagging the contralateral ear relative to the ipsalateral ear. In a real-world dynamic situation the ITD would be split between a lag and lead, respectively.
- When a sound source is moving relative to the listener, regardless of source-listener distance, the HRTF is updated immediately. In reality, due to propagation delay, there would be a slight delay before the sound wave incident angle changed. This is true of other source attributes as well, e.g., if one considered the source gain as a volume knob on the source.

Mixer Render Plugin

The Mixer Render Plugin block diagram is shown below. Please see the SRAPI Reference Manual and the SrcMix functions for more information.

Mixer Render Plugin Block Diagram



Definitions

n	source number in order allocated
$HRTF_{aux\#,src\#}$	each aux send is routed to an HRTF az,el processing block, there are two independent HRTF processing blocks per source
$en_{aux\#,src\#}$	enable processing block
$az_{aux\#,src\#}$	azimuth
$el_{aux\#,src\#}$	elevation
$g_{src\#,out\#}$	source gain
$r_{aux\#,src\#,out\#}$	aux return gain applied to spatial signal, typically, odd/even out pairs should be identical, $r_{1,n,1} = r_{1,n,2}$, $r_{1,n,3} = r_{1,n,4}$, etc.

The sum blocks sum over all sources.

HRTF Databases

slab3d HRTF (Head-Related Transfer Function) databases contain HRIR (Head-Related Impulse Response) and ITD (Interaural Time Delay) filter data for spatial rendering. Measured HRTFs can be obtained using insert-ear microphones and a spherical array of speakers to construct a spherically-sampled database. The HRIRs are converted to minimum-phase to reduce filter length and to simplify real-time interpolation. Biharmonic spline interpolation is used to resample the data to a uniform grid. The grid is specified by an azimuth increment and an elevation increment. The default slab3d HRTF database is "jdm.slh". The SLH suffix denotes a slab3d HRTF. "HRTF databases" are sometimes referred to as "HRTF maps". See CSRAPI::LstHRTF() in the SRAPI Reference Manual for SRAPI HRTF database selection.

slab3d HRTF database format

HRTF Database Header

```
struct HRTFHeader
{
  short  nVersion;         // database format version (2)
  char   strName[32];      // subject's name
  char   strDate[8];       // date measured
  char   strComment[256];  // text comment
  short  nAzInc;           // az increment, degrees
  short  nElInc;           // el increment, degrees
  short  nNumPts;          // number of HRIR points
  long   lSampleRate;      // sample rate
};
```

The HRIR and ITD data immediately follow the header (below: nAzInc = 30, nElInc = 18, nNumPts = 128).

```
 AZ    EL
 180,  90, left  ear, hrir pt 0        Left ear points
...
                      hrir pt 127
 180,  90, right ear, hrir pt 0        Right ear points
...
                      hrir pt 127
 180,  72, left  ear, hrir pt 0        Elevations (grouped by azimuth)
...
      -90, right ear, hrir pt 127
 150,  90, left  ear, hrir pt 0        Azimuths
...
-180, -90, right ear, hrir pt 127

 180,  90, delay                       Delays (+ left ear lag, - right ear lag)
 180,  72, delay
...
-180, -90, delay
```

jdm.slh parameters

Number of HRIR pairs and ITDs:	143
Azimuth grid:	180 to -180 in -30 degree increments
Elevation grid:	90 to -90 in -18 degree increments
HRIR length:	128 points
Sample Rate:	44100 samples/s

Error Handling

There are two error checking modes corresponding to the two states of SRAPI, static-state and render-time. The two modes are required because of render-time multitasking. When rendering, SRAPI runs in two threads of execution, the API Thread (same as the user's thread) and the Render Thread, a thread created during a call to RenderStart(). Errors can occur inside the Render Thread without an API call (e.g., a state error in a Render Plugin not directly related to an API call). The user cannot check the return value of an API function to catch this error (unless "polling" and then only during the next "error poll", see below).

Static-state Error Catching

Function Return Values: Many SRAPI functions return SLABError. Static-state errors are typically caught with function return values.

Render-time Error Catching

There are two methods for catching render-time errors:

Notification Callback: When using a notification callback, SRAPI notifies the user of an error by calling the user's callback function with SC::NotifyDspDone (i.e., DSP done due to error). Before calling RenderStart(), call SetNotify() specifying a notification callback function. If a render-time error occurs, the notification callback will be called. Use an error query function to verify the notification corresponds to an error (notifications can also be used to indicate render completion using the AutoStop feature).

Error Polling: Whenever an error occurs, SRAPI enters an "error state." In an error state, all error-returning SRAPI functions return the current error. Error Polling refers to catching existing errors via function return values or error status functions. In other words, the user is checking for an error not necessarily caused by the function itself (e.g., an error caused by a previous function where the return value wasn't checked).

Error State

SRAPI enters an "error state" whenever an error occurs. In an error state:

- rendering is stopped
- the sound output stream is stopped
- script processing is stopped
- all functions returning SLABError return the current error
- error information can be queried via ErrorState(), Error(), ErrorString(), and ErrorStack()
- error state maintained until explicitly cleared with ErrorClear()

Error Functions Overview

The following functions exist for querying error information and clearing error state:

- ErrorState(): returns true if in error state, false if not
- Error(): returns the current SLABError
- ErrorString(): returns a string describing the current error
- ErrorStack(): returns a string providing call stack error information for the current error
- ErrorClear(): clears error state
- SetNotify(): sets the notification callback function
- LogName(): sets the name of the log file used for logging error information
- Reset(): resets slab3d

Building slab3d

Pre-built release-mode slab3d libraries, render plugins, generator plugins, and applications are included in the slab3d release. Building derivative works can often be accomplished using the pre-built components (e.g., see the sample applications slabcon and SLABWireDemo).

Note regarding static vs. shared linking: To avoid "DLL hell", slab3d-based apps are typically statically linked. Some technologies, such as the Common Language Runtime, require MFC to be accessed via a shared DLL. Using this configuration with slab3d often results in linking conflicts. The work-around is to use the Visual Studio options Ignore Specific Libraries in concert with Additional Dependencies. Libs ignored by the former can be added using the latter. It might take some permuting, but this should allow a mixed static/shared app to be built.

All slab3d libraries, plugins, and applications can be built using the slab3d release and a few publicly available components. The steps below assume slab3d was installed to the directory \slab3d. This is also the directory used to develop slab3d, so you might see references to this directory. In general, absolute directory paths are avoided, but a few may exist. The slab3d release is built using Microsoft Visual Studio 2015 Professional.

Build Steps

- The June 2010 release of DirectSound is used for low-latency sound device output. The DirectX SDK can be downloaded from the Microsoft Download Center:

  http://www.microsoft.com/en-us/download/default.aspx

  The following Visual C++ settings (or similar) should be used for the headers and libraries. If set using Visual Studio Options (versus Project Settings), these directories will be used for all projects and solutions. They should be place before the Windows SDK directories to avoid header clashes.

  Visual C++ Include Files Directory: C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Include
  Visual C++ Library Files Directory: C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Lib\x86

  Other DirectX 9c and after SDKs should be compatible. Note, though, sometimes a release built on one machine will require a missing DLL when run on another. I tried to avoid this by using static builds, but somewhere along the DirectX 9 release cycle, Microsoft moved code from a static library to a DLL forcing users to update their DirectX End-User Runtime.

- For DIS support, the slabwire library uses the Air Force Research Library's slabDISInterface library. This library is released with slab3d in the usrlib\ directory. Build the slabDISInterface project, debug and release.

- For Polhemus Fastrak support, SLABScape uses the NASA-developed traklib library. Build the traklib project in usrlib\traklib\traklib.sln, debug and release.

- The ASIO sound device interface is used for low-latency sample I/O. slab3d is built using ASIO SDK version 2.2, but 2.0 and 2.1 should build as well. The ASIO SDK files are not included in the slab3d release because the ASIO license does not allow the re-release of ASIO files. The ASIO SDK is available from www.steinberg.net under Support, 3rd Party Developers. Once downloaded, place a copy of the SDK in slab3d's usrlib\ directory, e.g., slab3d\usrlib\asiosdk2.2\ containing directories common\ and host\.

- For XML read/write support, SRAPI uses XmlLite. XmlLite appears to be installed with Visual Studio 2015, though Visual Studio 2005 and the "Microsoft Windows SDK Update for Windows Vista" was originally used. The required files are xmllite.h and xmllite.lib. The version of the SDK I used is titled "Windows SDK for Windows Server 2008 and .NET Framework 3.5" (aka "Windows SDK v6.1"). The ISO file "6.0.6001.18000.367-KRMSDK_EN.iso" is available at the Microsoft Download Center:

  http://www.microsoft.com/downloads

  An ISO mounting program (e.g., MagicDisc) can be used to install directly from the ISO. The default SDK install dir is "C:\Program Files\Microsoft SDKs\Windows\v6.1". XP users might have to download "XMLLite for Windows XP (KB915865)" to obtain the xmllite.dll runtime file.

- SLABScape includes a simple speech recognition demo. This demo was originally coded using Visual Studio 2005 and the Microsoft Speech SDK 5.1 (SpeechSDK51.exe at the Microsoft Download Center). The Speech SDK headers and libs appear to be installed with Visual Studio 2015.

  Note: Even though the Speech SDK headers and libs appear to be installed with Visual Studio 2015, the grammar compiler gc.exe does not appear to be installed. The Windows SDK v6.1 ISO discussed above contains gc.exe in the bin\ directory. That gc.exe was used to build SLABScape's grammar.cfg and grammar.h files. A pre-built version of grammar.cfg is distributed with slab3d, so v6.1 isn't necessarily required. However, if errors are reported building SLABScape under v6.0A, v6.1 might help resolve them. If you are not interested in speech recognition and Windows SDK v6.1, see the define _ENABLE_SAPI in SLABScape file sr.h.

- The slabx assembly and the X-prefixed projects require XNA 4.0. XNA is available from the Microsoft Download Center.

- Build all \slab3d\src\slab3d.sln, debug and release.

Building gstk

gstk is a slab3d generator plugin that contains all of the STK v4.5.1 instruments (STK is "The Synthesis ToolKit in C++").

- Open the solution \slab3d\src\gstk\gstk.sln and build the project gstk, debug and release. This will also build stklib. If warnings occur during the build, they can be safely ignored. stk1 is an STK test application that is not required for gstk.

Cleaning

Batch files are included for cleaning (deleting) files created during the build process:

- cleanrel.bat - used to clean all non-usrlib files except those required for the release package, i.e., some exe's, lib's, and dll's are not deleted. See the cleanrel.bat batch file for details.
- clean.bat - calls cleanrel.bat and then cleans the release exe's, lib's, dll's, and res's.
- cleandev.bat - cleans all usrlib build files except those required to build slab3d components.
- cleanusr.bat - cleans all usrlib build files.

Tips and Known Issues

Tips

To disable source-listener range-dependent gain scaling use SrcRadius(idSrc, dRadius, 0.0).

To place sound sources by specifying azimuth and elevation use SrcLocateRel() or LstPosition(0.0, 0.0, 0.0, 0.0, 0.0, 0.0) with SrcLocatePolar(idSrc, az, el, range).

To specify sound source time delays use LstPosition(0.0, 0.0, 0.0, 0.0, 0.0, 0.0) and SrcLocatePolar(idSrc, az, el, dSoundSpeed * time_delay).

To generate periodic sound sources not implemented in a signal generator, use SiAllocFile() with a wave file editing application. Since SiAllocFile() can loop wave files continuously, it can be used to generate fixed periodic sound sources. For real-time signal generation, one can use the Generator Plugin interface (see CGPlugIn).

There are three methods for muting a sound source:

- Call SrcMute(). Internally, it is similar to the SrcGainScalar() method below. The difference is SrcMute() preserves the gain value.
- Call SrcGainScalar() with a scalar gain of 0.0 or call SrcGain() with a dB gain value less than -97.0 dB (-96.3 dB is the dynamic range of 16-bit integer). This does not affect the processing performed. The computational load remains the same. Since the gain of the reflections is dependent upon the gain of the source, this mutes the reflections as well.
- Call SrcEnable() to disable the source. The source is no longer rendered, eliminating the source entirely. Disabling a source reduces computational load. Note, though, disabling is image-specific. If using reflections, SrcEnable() only impacts the direct path. SrcRefEnable() can be used to disable reflections.

DSP Parameter Caution

The state copy between SRAPI and slabwire is between two different threads of execution. Thus, there can be a slight delay (microseconds to milliseconds) between the setting of a scene parameter, e.g., source location, and the resultant slabwire DSP value, e.g., source azimuth. Source azimuth is calculated by the DSP based on acoustic scene geometry. In general, this delay is not an issue. But, it is not instantaneous, therefore scene queries occurring immediately after a set, e.g., data binding, could report an incorrect value (the old value). Polling is the safest way to ensure the parameters set in one thread of execution have been captured by another. The primary scene parameters of concern are source azimuth, elevation, and range.

The following SRAPI functions can be used to address this issue (see HeadMatch app):
Lock(), UpdateWaitReset(), Unlock(), UpdateWait()

Known Issues

slab3d does not detect all occurrences of DirectSound buffer underflow. Usually, underflow results in an underflow counter increment. But, sometimes, stuttering is heard without an underflow increment. This is most likely to occur when the CPU is at maximum usage or a CPU usage spike occurs (e.g. reading a file). A missed underflow can be caused by a late Windows timer update, allowing the DirectSound pointers to wrap around the buffer to a valid location.

Development has been paused on the SLABScape "hand". It is not documented.

Applications

Several applications are included in the slab3d Release. The ones considered of general use are distributed in exe form. The others can be built using the Microsoft Visual Studio solution slab3d\src\slab3d.sln (slab3d\ refers to the slab3d release installation directory). All application source code is in slab3d\src. In general, the naming convention used is SLAB* for GUI applications and slab* for console applications. All applications are written in C++ or C# using Microsoft Visual Studio.

SLABScape is a virtual environment rendering application. It provides a graphical user interface for several SLAB Render API audio-rendering parameters and Direct3D visual-rendering parameters. SLABScape can be used to create Start Menu links for a slab3d installation (see SLABScape | Help | About...). For more information, see the SLABScape Help Page.

slabcon is a console application demonstrating SRAPI development.

- The minimum API demo shows the minimum SRAPI API use required to spatially render a sound source.
- The general API demo demonstrates multiple sound outputs, signal generator allocation, User sound source sample streaming, and automatic scene updating using Modifiers.
- The Spatial2 render plugin demo demonstrates sound source VU meters, instant replay, and silence skip.
- The HRTF demo demonstrates HRTF database loading and selection. It also demos the use of a sound source azimuth modifier to create a circular trajectory.

SLABWireDemo demonstrates the use of the slabwire library. slabwire sits beneath SRAPI and is used to configure a low-level sound_in-to-DSP-to-sound_out audio rendering chain. SLABWireDemo contains an extremely simple audio effects render plugin that contains passthru, lowpass, and flanger effects and a sample grabber. The sample grabber is used to route samples to a visual waveform display and an FFT. The FFT provides frequency information for a spectrum display and frequency-triggered OpenGL displays.

SLABSound is a sound I/O utility. It provides ASIO, DirectSound, Waveform API, and RIFF file information.

SLABSurface provides a MIDI and/or GUI control surface interface to slab3d. It is useful for controlling and developing slab3d-based synthesis patches and generator plugins.

src\archive contains past projects that are no longer maintained.

XScape Content and slabx Coordinates

This discussion assumes some familiarity with XNA Game Studio 4.0. The default XScape "Content" (models, textures, etc.) is built using the XScape project (see the slab3d\bin\Content\ directory). Additional content can be built using the cbuilder (content builder) project (or another XNA project). cbuilder outputs compiled content to slab3d\bin\ContentBuilder\. A content source item is referred to as an "asset". To be used by XScape, the asset must be compiled into a .xnb file. This limits the asset formats to those supported by the XNA importers (or third-party providers; this discussion is limited to the standard importers). Compiled model and texture assets can be moved to XScape's Content directory for rendering.

MSDN - XNA Game Studio 4.0 - Standard Importers and Processors
http://msdn.microsoft.com/en-us/library/bb447762.aspx

For models (Output Type NodeContent), two importers exist: Autodesk FBX and Microsoft X File.

For textures, the supported types are .bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, and .tga.

An excellent source of model content is Google 3D Warehouse:
http://sketchup.google.com/3dwarehouse/

Models can be downloaded in the Google SketchUp format (.skp) (and sometimes Collada). The free version of SketchUp only exports to Collada (.dae) and Google Earth File (*.kmz). SketchUp Pro includes FBX (http://sketchup.google.com/intl/en/product/whygopro.html).

The free version of SketchUp can be extended via Plugins (tests were originally conducted using SketchUp 7). Three plugins were found that export to the X File format:

1. xExporter.rb by Erwan de Cadoudal
   http://sites.google.com/site/edecadoudal/sketchupwithdirectx

2. ZbylsXExporter.rb by Zbigniew Skowron
   http://www.scriptspot.com/sketchup/scripts/zbylsxexporter

3. 3D Rad Exporter
   http://www.3drad.com/Google-SketchUp-To-DirectX-XNA-Exporter-Plug-in.htm

These will appear in the Google Plugins menu as "DirectX", "Zbyl's Exporter...", and "3D Rad". The first two were used to convert XScape's default content. If the DirectX SDK is installed, DirectX Viewer can be used to evaluate the generated X File. SketchUp Pro's FBX Export can also be used to prepare XNA model content. Different exporters yield different results, so sometimes multiple methods need to be tried before obtaining a usable result.

Once the X File or FBX assets have been created, they can be added to the cbuilder Content (right-click Add | Existing Item...). The model's textures will also need to be added. If it is not obvious which textures are necessary, the Content processor's debug output can be used to add the textures based on error messages. When adding the textures, the Content properties need to be changed from the defaults: Build Action: "Compile" to "None" and Content Processor: "Texture - XNA Framework" to "No Processing Required". The textures are processed with the model so no additional processing is required. These steps place copies of the models and textures in the cbuilder project's Content directory.

When using a SketchUp-generated FBX, the FBX will probably contain path information in the "Filename" and "RelativeFilename" fields. An FBX is a text file so it can be edited in Visual Studio to trim the Filename and RelativeFilename fields to include only the texture filenames.

In general there are three steps one might want to consider before exporting the model:

1. remove any components that aren't desired
2. translate the model so that the origin is in the most meaningful location for virtual world translations and orientations
3. orient the model so that XNA yaw,pitch,roll 0,0,0 is in the desired facing direction

XScape and slabx use the right-handed XNA coordinate system:

- +x right
- +y up
- +z back

XScape default camera facing: -z

SketchUp uses the following right-handed coordinate system (Simple Template - Meters, positive axis solid, negative axis dotted):

- +x right (red)
- +y forward (green)
- +z up (blue)

SketchUp default camera facing: +y

The default camera facing for XScape, DirectX Viewer, and SketchUp are fairly similar with the SketchUp to XNA axes mapped as follows:

- SketchUp +x to XNA +x right
- SketchUp +y to XNA -z forward
- SketchUp +z to XNA +y up

All exporters discussed except xExporter use the mapping above. xExporter maps SketchUp +x to XNA -z.

Orientation angles are positive CCW as viewed from the positive axis towards the origin, e.g., for XNA, +yaw is CCW looking down +y. In XScape and slabx, 0 yaw is defined as facing forward towards -z.

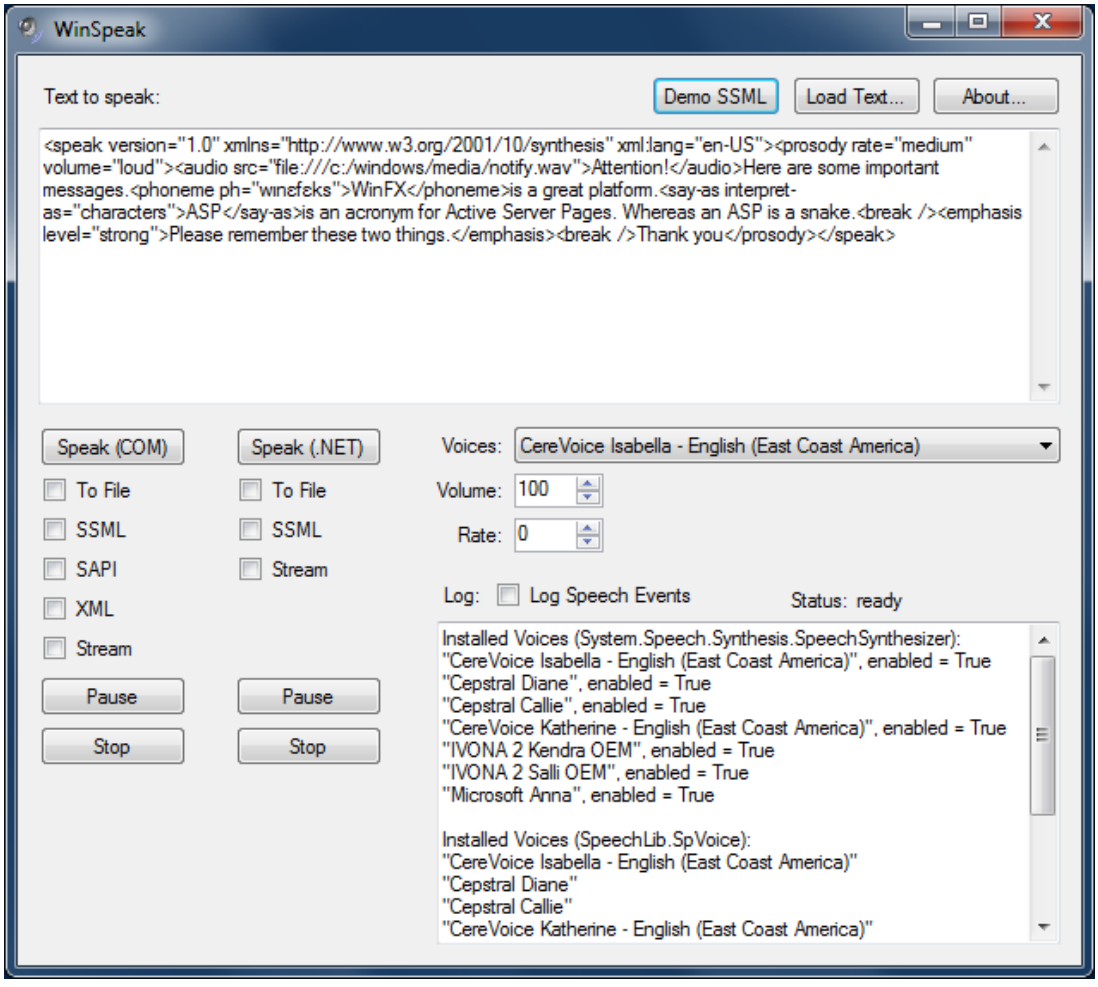
Note, the slabx.dll graphics engine uses the XNA coordinate system defined above. The slab3d.dll (Managed SRAPI) sound engine uses an FLT coordinate system (for historical reasons). To simplify the simultaneous use of both, Managed SRAPI extends SRAPI with XNA coordinate functions.

AvADE (Aviation Auditory Display Engine) Server Software

The AvADE software comprises three applications built into the slab3d virtual acoustic environment rendering system: WinSpeak, AvADE Client, and AvADE Server. Avade was designed with maintainability and extensibility in mind. It can be easily modified to support a wide variety of spatial audio and communications server applications.

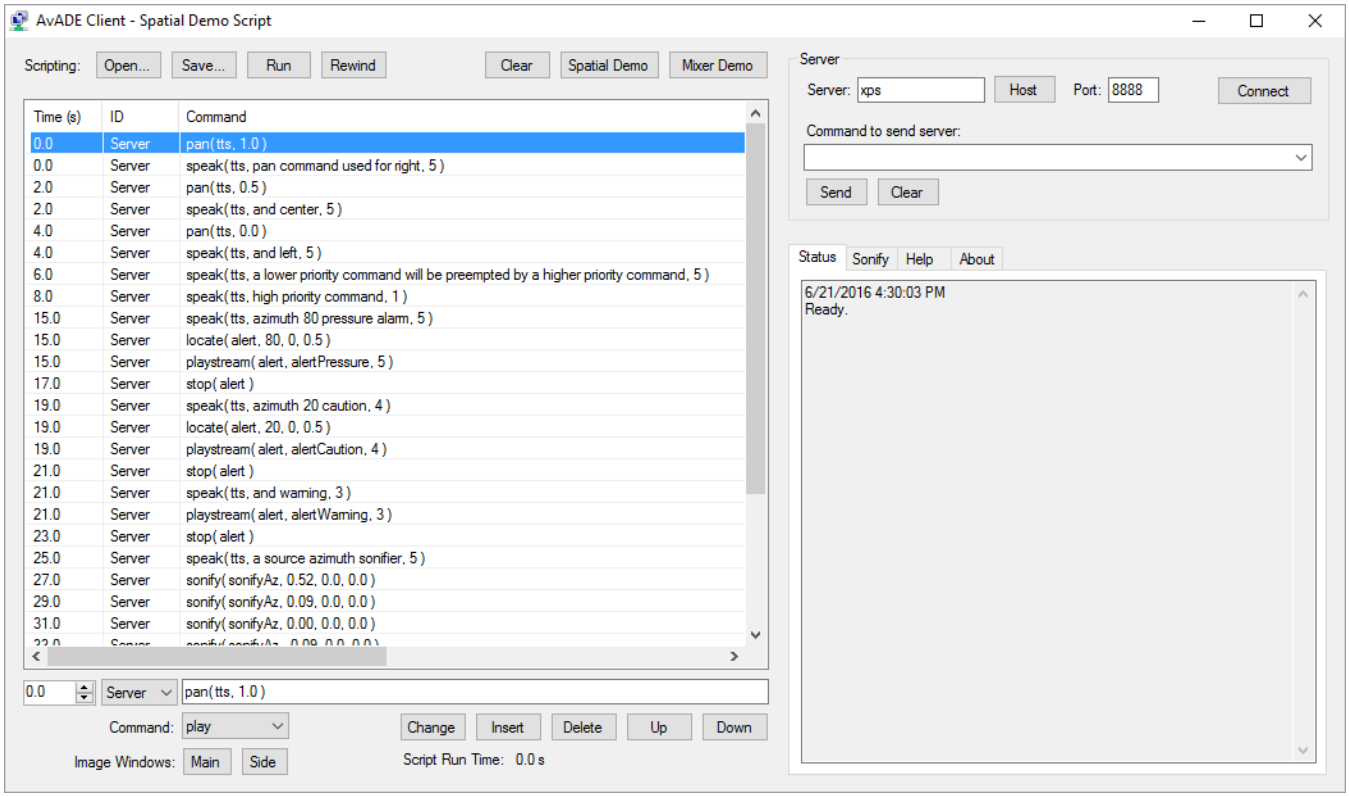
WinSpeak.exe

WinSpeak allows the user to experiment with the SAPI 5.4 TTS (Text-To-Speech) voices installed on the user's Windows 7/8/10 system, including the specification of SSML and SAPI XML pronunciation tags. "Microsoft Anna" is the default SAPI 5.4 voice installed with Windows 7. This voice, or other installed voices, can be used to generate real-time TTS with Avade. There are two Microsoft managed TTS APIs, a COM API, and a .NET API. The .NET API is implemented in AvadeServer. If useful, WinSpeak's COM API can easily be migrated to AvadeServer as an alternate (or preferred) option.



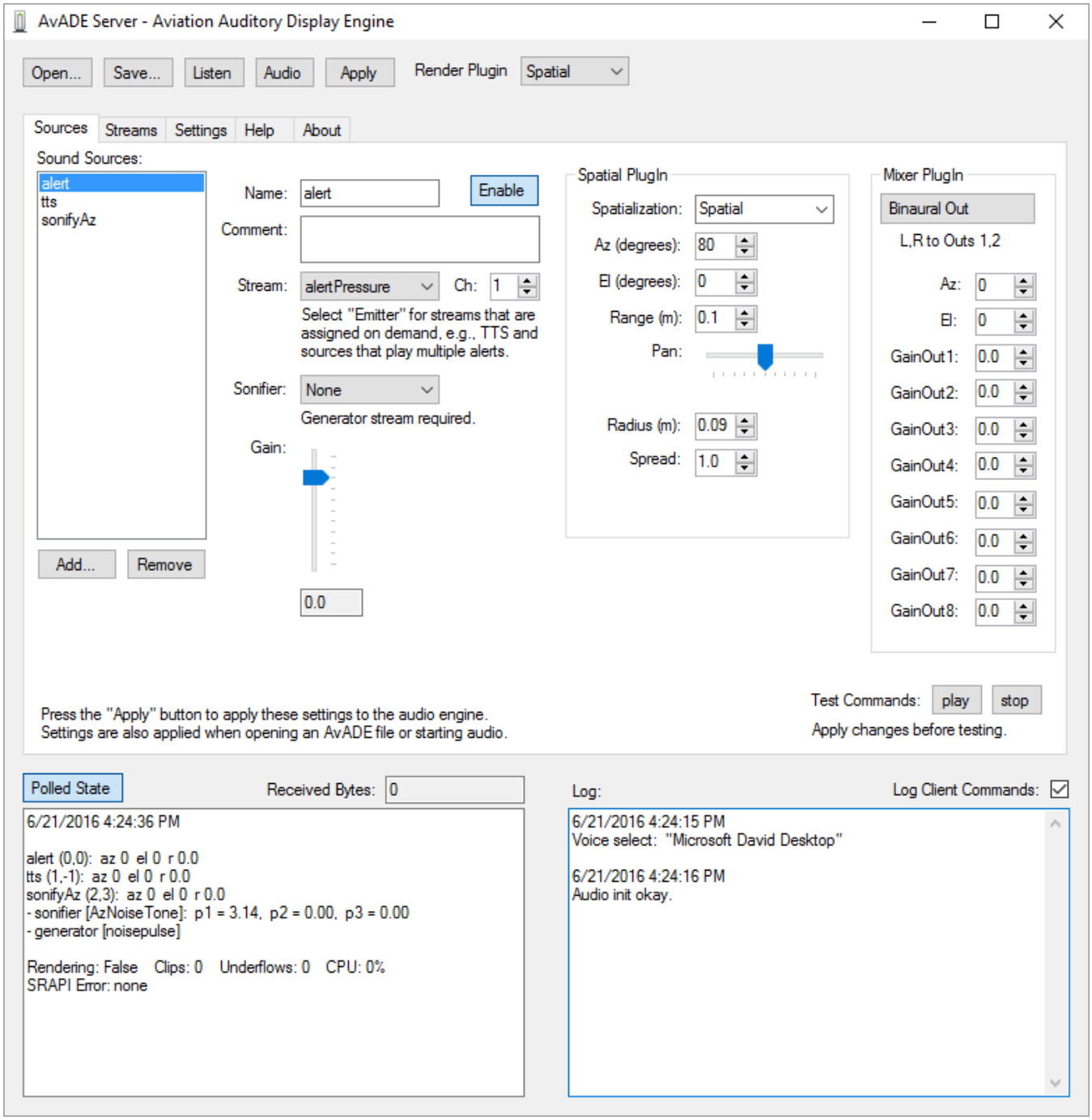
AvADE Client (AvadeClient.exe)

AvadeClient is a test utility that serves as a sample client application. It allows the user to type and send arbitrary text commands to the Avade server while viewing server-to-client responses in a log window. The command entry droplist contains several test commands. The default script works with the default AvadeServer configuration and tests and demos several Avade features.



AvADE Server (AvadeServer.exe)

Avade supports a variety of features organized around the concept of sound "sources" and sample "streams". For the Spatial Render Plugin, a "source" is a virtual environment sound emitter, i.e., a virtual entity with an azimuth, elevation, and range relative to a listener. For the Mixer Render Plugin, a source is a mixer channel strip, i.e., output channel gains and a binaural HRTF effects processor. For both plugins, a stream is the sound sample stream that plays from the sound source, e.g., wave files, TTS speech, DIS Radio communications, signal generators, etc.



Supported Sound Source Parameters

- Both Plugins
  - Enable
  - Gain (dB)
- Spatial Render Plugin
  - Spatialization (Spatial, Relative, Panned)
  - Azimuth, Elevation, and Range (Spatial and Relative sources)
  - Pan (Panned sources)
  - Source Radius and Spread Rolloff (source-listener distance attenuation)
- Mixer Render Plugin
  - Binaural HRTF Processor Enable, Azimuth, and Elevation
  - Output Channel Gain Scalars (ASIO device channels 1-8)

Supported Sample Stream Input Types

- wave files (looping and one-shot)
- real-time TTS speech
- multichannel sound device input (ASIO driver required)
- signal generators (stream type for "sonifiers")
- DIS Radio communications

The underlying audio engine, slab3d, supports several additional source parameters and stream types. Support for these can be added as needed. All source, stream, and Avade parameters can be saved to and read from human-readable XML files. A time-stamped log window provides high-level Avade status information (e.g., user activities, incoming messages). A polled audio engine status window provides low-level information (e.g., auditory scene state, error status). This allows the user to confirm that Avade GUI and client command settings have been instantiated by the audio engine.

A preliminary set of Avade commands have been implemented. Additional commands can be easily added. Wave files can be played and stopped from the client and placed in priority queues for playback. TTS speech can be generated on demand and also placed in priority queues. All higher priority playback plays before lower priority playback. Lower priority playback is interrupted to allow for an incoming higher priority message. The lower priority playback is then played again from the beginning. DIS (Distributed Interactive Simulation) Radio communications (IEEE standard 1278) can be streamed over network connections. Client communications can occur via the included DISTrans and DIS_Radio applications or other DIS radio transmitters. These communications include a frequency that can be used to specify individual communication channels, allowing one to spatialize different channels to different positions in auditory space.

For audio output, Avade supports three device driver interfaces, Waveform-Audio, DirectSound, and ASIO. Waveform-Audio has the highest latency but is supported by all Windows audio hardware. DirectSound is similar to Waveform-Audio but can provide lower latency. ASIO has low latency and flexible configuration options, but is supported by a smaller set of semi-pro and pro audio hardware.

AvADE Commands

AvadeServer responds to the following string commands sent by a UDP or TCP/IP client. The AvadeClient utility can be used to test and demonstrate the use of these commands. These commands are also documented on the AvadeServer and AvadeClient Help tabs.

Note! For commands that have GUI settings, the command value will override the GUI value. The GUI value will remain unchanged. The Apply button applies the GUI values to the renderer. In this context, the GUI values can be thought of as Initial Conditions. The GUI values are also the values saved when saving an AvadeServer configuration.

play(sourceName)
play() rewinds and plays the stream attached to a sound source.

playstream(sourceName, streamName, priority)
playstream() attaches, rewinds, and plays a sound source stream. A stream of higher priority will preempt a stream of lower priority. Priority is specified 1 (highest) to 5 (lowest).

stop(sourceName)
stop() stops the stream attached to a sound source.

enable(sourceName, enableState)
enable() enables (true) or disables (false) sound source rendering.

originLLH(latitude_degrees, longitude_degrees, height_feet)
originLLH() sets the ENU East/North/Up origin for the LLH Latitude/Longitude/Height commands listenerLLH() and obstacleLLH().

listenerLLH(latitude_degrees, longitude_degrees, height_feet, yaw, pitch, roll)
listenerLLH() places the listener using LLH coordinates (see originLLH()).

obstacleLLH(sourceName, latitude_degrees, longitude_degrees, height_feet,
yaw_deg, pitch_deg, roll_deg,
size_height_feet, size_width_feet, size_depth_feet)
obstacleLLH() places a sound source using LLH coordinates (see originLLH()).
Note: The latter six parameters are presently ignored and are provided for future development.

listener(x_meters, y_meters, z_meters, yaw_degrees, pitch_degrees, roll_degrees)
listener() positions the listener using a Cartesian location and Euler orientation.

polar(sourceName, az_degrees, el_degrees, r_meters)
polar() positions a sound source using polar coordinates.

rel(sourceName, az_degrees, el_degrees, r_meters)
rel() positions a sound source relative to the listener. If the listener is moved, the source moves as well.

pan(sourceName, pan_value)
pan() positions a source via left-right gain scalars.
pan_value range: 0.0 = monotic-left, 0.5 = diotic, 1.0 = monotic-right

speak(sourceName, tts_text, priority)
speak() speaks the text specified in tts_text using the source specified by sourceName. tts_text quotation marks are optional. A stream of higher priority will preempt a stream of lower priority. Priority is specified 1 (highest) to 5 (lowest).

sonify(sourceName, sonified_values1,2,3)
sonify() sonifies the float values passed in sonified_values1,2,3. The sonification depends on the Sonifier selected for the source (Sources tab). Not all sonifiers respond to sonify(); some depend on virtual environment state, e.g., ObstacleFM.

mixgain(sourceName, g1, g2, g3, g4)
mixgain() adjusts ASIO output ch 1-4 gain scalars for sourceName.

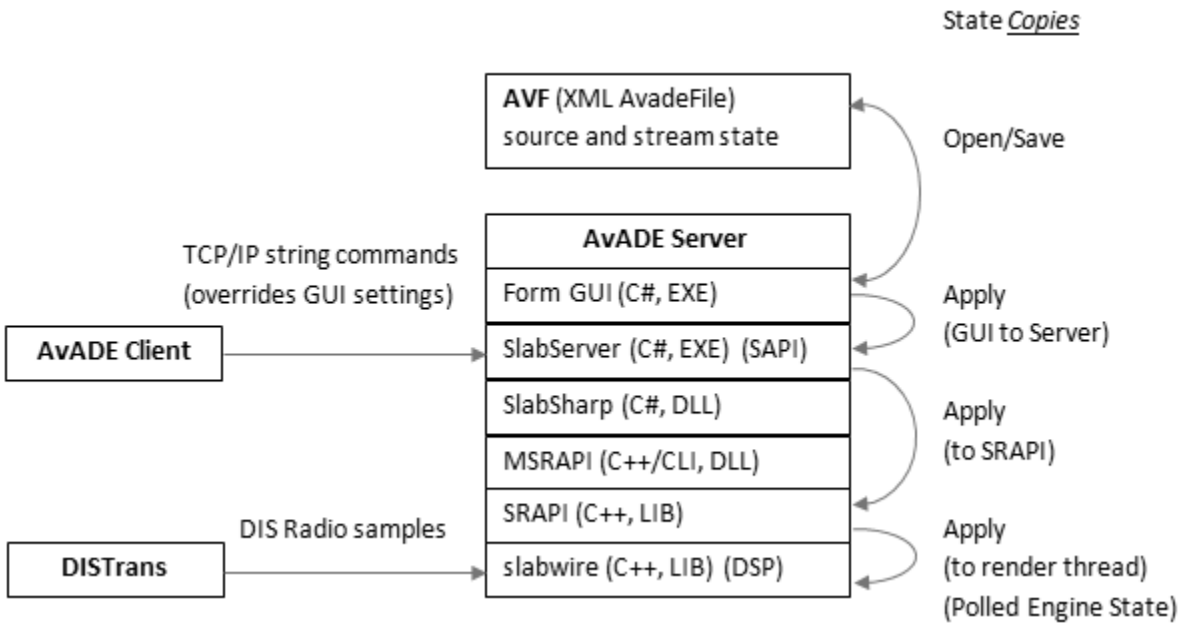
mixhrtf(sourceName, enable, az, el, gain)
mixhrtf() enables/disables HRTF processing using the azimuth, elevation, and gain specified for sourceName. The L/R binaural stream is output on channels 1 and 2 (mixed with mixgain() streams, if any).

Sonifiers

Translation	sonify(transX, transY, transZ)
AzElRange	sonify(az, el, range)
AzNoiseTone	sonify(az, ignored, ignored)
AzImpTone	sonify(az, ignored, ignored)
Drift	sonify(xDisplacement, yDisplacement, zDisplacement)
Generic	sonify(xDisplacement, yDisplacement, zDisplacement)
ObstacleNT	VE source-listener distance thresholds, 90ft Advisory, 60ft Caution, 40ft Warning
ObstacleFM	VE source-listener distance thresholds, 90ft Advisory, 60ft Caution, 40ft Warning
ObstacleFMSin	VE source-listener distance thresholds, 90ft Advisory, 60ft Caution, 40ft Warning

Translation, AzNoiseTone, ObstacleFM, and ObstacleFMSin are fully implemented sonifications. The others were used in development.

AvADE Software Architecture



Appendix

Latency Analysis

Several rounds of slab3d latency analysis have been performed:

2003 J.D. Miller, M.R. Anderson, E.M. Wenzel, and B.U. McClain, “Latency Measurement of a Real-Time Virtual Acoustic Environment Rendering System,” Proc. 2003 ICAD, Boston, MA, July 2003. Paper Poster

2005 The technote "tn20050228_asio_latency_reduction.docx" describes an ASIO-based audio latency analysis and reduction effort.

2012 The technote "tn20120319_av_latency.docx" discusses a Windows Vista XNA 3.1 C# audio-visual latency analysis effort.

2014 Two ASIO audio peripherals were tested for use with the AvADE auditory server. Their latency values are tabulated below.

2014 The technote "tn20120319_av_latency.docx" was revisited for a quick Windows 7 and XNA4 investigation.

2014 The technote "tn20140917_av_latency.docx" discusses a Windows 7 and XNA4 latency analysis performed using the xbox controller. This effort also investigated XNA and Windows timing.

2015 The DIS Radio and VoIP latencies were measured. The results are tabulated below.

2016 The technote "tn20160616_avade_latency.docx" presents an Avade Client/Server latency analysis that compared the MOTU PCIe-424 and the USB RME Fireface UFX audio peripherals.

AvADE ASIO Audio Peripheral Latency Comparison

ASIO driver	**MOTU PCI ASIO**			**Focusrite USB 2.0 Audio Driver**	
buffer size setting*	**64**	**128**	**256**	**89 (2.0ms)**	**133 (3.0ms)**
AvADE Get ASIO Details (samples unless stated)					
min buffer size	64	128	256	89	133
max buffer size	64	128	256	1024	1024
preferred buffer size	64	128	256	89	133
granularity	0	0	0	1	1
selected buffer size	64	128	256	96*	160*
input latency	86	150	278	232	360
output latency	87	151	279	328	520
est API latency (ms)	3-5	6-9	12-18	10-12	15-19
est mean API latency (ms)	4	8	15	11	17
est I/O latency (ms)	5	10	18	15	24

* For the MOTU PCIe/24I/O, buffer sizes are set via the MOTU PCI Audio Console app and the Samples Per Buffer setting. For the Focusrite Scarlett 18i20, buffer sizes are set via the MixControl app and the ASIO Buffer Size setting (in ms). slab3d requires the buffer size to be a multiple of the audio processing frame size (32 samples).

DIS Radio and VoIP Latency

1/20/2015 (revised 6/21/2016, v6.7.5, "Note 2" added)

There are two SRAPI DIS Radio APIs:

1) 2009/JDM = Joel Miller's DIS API developed circa 2009 using the AFRL-provided slabDISInterface library. Stream Types: StreamInDISRadioRT, StreamInDISFreq

2) RT_DIS/JMS = John Stewart's Real-Time DIS API integrated late 2013, built using John's rtdis library. Stream Type: StreamInDISRT

Latency tests were performed with the apps DISTrans (ASIO) and SLABCall (Waveform-Audio, VoIP) transmitting to Netcom (ASIO) on a single PC. Headset mic input was split to left-audio-in on a second PC. Netcom output was sent to right-audio-in on the second PC. All sample rates were 44,100 Hz. The DISTrans/SLABCall/Netcom PC used a Focusrite Scarlett 18i20 with a 2ms ASIO buffer size for an audio I/O device. The measurement PC used a MOTU PCIe 24I/O.

Stream types are shown as defined in slabwire file streams.h and slabsharp file SlabSharp.cs. In SRAPI, these map to corresponding SiAlloc*() functions and settings.

Stream Type	Latency	Settings
StreamInDISRadioRT	33 ms	10ms pre-buffer
StreamInDISFreq	34 ms	10ms pre-buffer
StreamInDISRT	27 ms	10ms pre-buffer
StreamInVoipPort	134 ms	
StreamInVoipID	98 ms	
Scarlett in-to-mix-out	28 samples	

Note 1: The stream type StreamInDISRadio was removed from SRAPI in slab3d v6.7.4 to simplify the 2009 API. Its only advantage was non-real-time output support, e.g.,file alone. Sound device output can be split to a file to provide similar functionality. StreamInDISRadio latency was measured to be 636 ms using a 600 ms pre-buffer (it was a legacy algorithm that required relatively large buffers; StreamInDISRadio was replaced by StreamInDISRadioRT).

Note 2: Given the DIS and VoIP latency results and the fact that the underlying VoIP engine (jvoip) was discontinued years ago, StreamInVoipPort and StreamInVoipID were removed in slab3d v6.7.5. Also, the RT_DIS software contains transmission support (see DISTrans app), so an end-to-end DIS capability now exists making VoIP unnecessary. The Netcom app mentioned above is now DISRec (DIS Receiver).