RECENT DEVELOPMENTS IN SLAB: A SOFTWARE-BASED SYSTEM FOR INTERACTIVE SPATIAL SOUND SYNTHESIS

Joel D. Miller

Raytheon Technical Services Co.
NASA Ames Research Center
Mail Stop 262-6
Moffett Field, CA 94035-1000 USA
jdmiller@mail.arc.nasa.gov

Elizabeth M. Wenzel

Spatial Auditory Displays Lab
NASA Ames Research Center
Mail Stop 262-2
Moffett Field, CA 94035-1000 USA
bwenzel@mail.arc.nasa.gov

ABSTRACT

This paper provides an update on the features of SLAB, a software-based real-time virtual acoustic environment (VAE) rendering system designed for use in the personal computer environment. SLAB is being developed as a tool for the study of spatial hearing.

The SLAB software is being released to the research community under a free-public license for non-commercial use. It is our hope that researchers will find it useful in conducting research in advanced auditory displays and will also add their own extensions to the software to provide additional functionality. Further information about the software can be found at: http://human-factors.arc.nasa.gov/SLAB.

1. INTRODUCTION

Interest in the simulation of acoustic environments has prompted a number of technology development efforts over the years for applications such as auralization of concert halls and listening rooms, virtual reality, spatial information displays in aviation, and better sound effects for video games. Each of these applications implies different task requirements that require different approaches in the development of rendering software and hardware. For example, the auralization of a concert hall or listening room requires accurate synthesis of the room response in order to create what may be perceived as an authentic experience. Information displays that rely on spatial hearing, on the other hand, are more often concerned with localization accuracy than the subjective authenticity of the experience. Virtual reality applications such as astronaut training environments, where both good directional information and a sense of presence in the environment are desired, may have requirements for both accuracy and realism.

All applications could benefit from further research that specifies the perceptual fidelity required for adequate synthesis [e.g., 1, 2]. For example, it is commonly assumed that only the direct-path head-related transfer functions (HRTFs) need to be rendered at the highest possible fidelity while early reflections may be rendered with less fidelity, i.e., fewer filter coefficients [3]. However, the number of coefficients actually used is often based on a designer's best guess and the limitations of a particular system, rather than the outcome of perceptual studies. Such studies can give system designers guidance about where to devote computational resources without sacrificing perceptual validity.

The goal of SLAB is to provide an experimental platform with low-level control of a variety of signal-processing parameters for conducting such studies. For example, some of the parameters that can be manipulated include the number, fidelity (number of filter taps), and positioning (correct vs. incorrect) of reflections, system latency, and update rate. The project is also an attempt to provide a low-cost system for dynamic synthesis of virtual audio over headphones that does not require special purpose signal processing hardware. Because it is a software-only solution designed for the Windows/Intel platform, it can take advantage of improvements in hardware performance without extensive software revision.

2. SLAB ACOUSTIC SCENARIO

To enable a wide variety of psychoacoustic studies, SLAB provides extensive control over the VAE rendering process. It provides an API (Application Programming Interface) for specifying the acoustic scene and setting the low-level DSP parameters as well as an extensible architecture for exploring multiple rendering strategies.

The acoustic scenario of a sound source radiating into an environment and heard by a listener can be specified by the parameters shown in Table 1. Currently, the SLAB Renderer supports all but the following parameters: radiation pattern, air absorption, surface transmission, and late reverberation.

SOURCE	ENVIRONMENT	LISTENER
Location	Speed of Sound	Location
(Implied Velocity)	Spreading Loss	(Implied Velocity)
Orientation	Air Absorption	Orientation
Sound Pressure Level	Surface Locations	HRTF
Waveform	Surface Boundaries	ITD
Radiation Pattern	Surface Reflection	
Source Radius	Surface Transmission	
	Late Reverberation	

Table 1. *Acoustic Scenario Parameters*.

In addition to the scenario parameters, SLAB provides hooks into the DSP parameters, such as the FIR update smoothing time constant or the number of FIR filter taps used for rendering. Also,

various features of the renderer can be modified, such as exaggerating spreading loss or disabling a surface reflection.

Recently implemented features include source trajectories, API scripting, user callback routines, reflection offsets, the Scene layer, and internal plug-ins. An external renderer plug-in interface is currently under development that will allow users to implement and insert their own custom renderers. This paper focuses on software architectural issues and provides an update to the work demonstrated and discussed in [4-7].

3. THE SLAB USER RELEASE

SLAB is being released via the web at http://human-factors.arc.nasa.gov/SLAB. The SLAB User Release consists of a set of Windows applications and libraries for writing spatial audio applications. The primary components are the SLABScape demonstration application, the SLABServer server application, and the SLAB Host and SLAB Client libraries.

3.1. SLABScape

SLABScape allows the user to experiment with the SLAB Renderer API. This API provides access to the acoustic scenario parameters listed in Table 1. The user can also specify sound source trajectories, enable Fastrak head tracking, edit and play SLAB Scripts, A/B different rendering strategies, and visualize the environment via a Direct3D display.

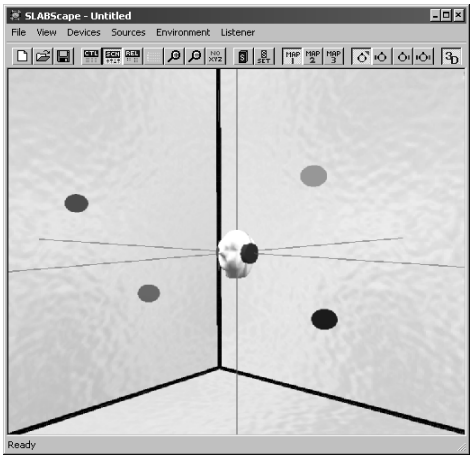

Figure 1. *SLABScape Screenshot.*

3.2. SLABServer

The SLABServer application allows a workstation to be dedicated as a stand-alone SLAB Server. In this configuration, the entire computational load is transferred to the server. This allows for more robust rendering and frees user workstation resources.

3.3. SLAB Libraries

The SLAB Host and SLAB Client libraries encapsulate the SLAB Renderer API and allow the user to develop SLAB-based applications. The APIs of the two libraries are essentially identical. Once the IP address of the server has been specified, the client library mimics the host library. Both libraries can be linked into the user's application simultaneously, allowing the user to decide at run-time whether to use host mode or client/server mode.

4. DESIGN OVERVIEW

In the following sections, an overview is provided of SLAB's software architecture, rendering model, and latency.

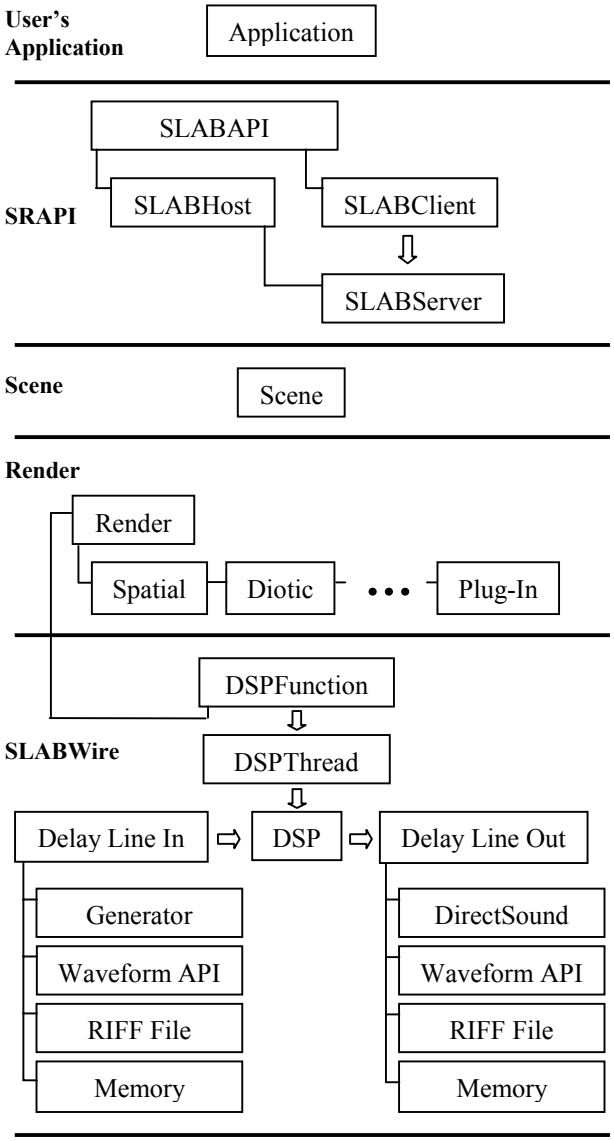

Figure 2. *SLAB Software Architecture - Layers and Classes (thick solid lines = layers, boxes = classes, thin solid lines = class hierarchies, vertical arrows = control flow, horizontal arrows = signal flow).*

4.1. Software Architecture

SLAB is a software-only solution written entirely in C++ using the Win32 SDK (Software Development Kit) and the Microsoft Foundation Classes. C++ was chosen as the development language for its speed and its object-oriented nature. An object-oriented approach was taken for its emphasis on modularity, extensibility, and maintainability. Microsoft Windows was selected as the operating system for its developer resources, persistent APIs, and the price/performance ratio of the Windows/Intel platform. The following SLAB layers discussion refers to the SLAB software architecture shown in Figure 2.

4.1.1. The SRAPI Layer

The user of SLAB interacts with the SRAPI (SLAB Renderer API) layer. This layer passes acoustic scenario parameters to the Scene and provides high-level control of the Scene, Render, and SLABWire layers via a host or client/server interface. The SRAPI layer is also responsible for processing SLAB Scripts. SLAB Scripts allow several API commands to be combined in a macro and sequenced over time. An example SRAPI code fragment appears in Figure 3.

```
CSLABAPI* pSLAB;
IDSRC     idSrc;

// allocate SLAB
if( bAllocateInHostMode )
  pSLAB = SLABAPIHost();
else  // allocate in client/server mode
  pSLAB = SLABAPIClient( "10.0.0.1" );
// render a spatial display
pSLAB->Render( RENDER_SPATIAL );
// allocate a wave file sound source
idSrc = pSLAB->SrcFile( "test.wav" );
// locate source 1m forward, 1m right
pSLAB->SrcLocate( idSrc, 1.0, -1.0, 0.0 );
Sleep( 5000 );  // wait 5s
// render a diotic display
pSLAB->Render( RENDER_DIOTIC );
Sleep( 5000 );  // wait 5s
delete pSLAB;
```

Figure 3. *SRAPI Example - an A/B comparison of a spatial and diotic display.*

4.1.2. The Scene Layer

The Scene layer contains all scenario state information. It performs source trajectory updating and room image model and listener-relative geometry calculations, providing a list of sound image incident angles and distances to the Render layer. All renderers use the same Scene object. Currently, the image model is limited to a rectangular room with six first-order reflections per sound source.

4.1.3. The Render Layer

The Render layer performs acoustic scenario rendering. It is constructed such that any rendering algorithm adhering to the internal SLAB Plug-In format (i.e. a subclass of Render) can be inserted into the rendering framework. This can occur in real-time allowing for the A/B-ing of different rendering techniques. A general-purpose plug-in strategy is currently being developed to allow users to import their own renderers. The SLAB Renderer is encapsulated in the Spatial class; it will be discussed in more detail later in the next section. Diotic is an example of an included alternate renderer that simply renders a diotic display.

4.1.4. The SLABWire Layer

The SLABWire layer manages sound input and output and routes sound samples through SLAB's signal processing chain. It is encapsulated in its own library and operates in its own thread of execution. SLABWire also provides interpolated delay line and DSP parameter tracking (a.k.a. smoothing) functionality. Not all features of the SLABWire layer are available via the SLAB Renderer API. For example, the user cannot currently select Waveform API input or output.

4.1.5. Everything You Need to Know About Object-Oriented Programming in Two Paragraphs

For those unfamiliar with object-oriented programming, a subclass extends the functionality of its base class and/or provides a different implementation for the same class interface. An object (the *thing* one uses) is an instantiation of a class (a description of the *thing*) (e.g. this is analogous to the relationship of a circuit to a schematic). An object of a subclass can be used in place of an object of its base class. In other words, a Spatial object *is a* Render object; a Render object *is a* DSPFunction object. This is termed "polymorphism" and is responsible for the inherent extensibility and flexibility of object-oriented programming.

Polymorphism allows multiple renderers to "plug into" the Render layer and the Render layer to plug into the SLABWire layer. It also allows the DSP object to operate on different types of sound input and output without knowledge of implementation details. Further, the SLAB user can take advantage of polymorphism by manipulating SLAB through the SLABAPI object for both host and client/server modes, allowing the user's code to be identical for both modes (see Figure 3).

4.2. The SLAB Renderer

The SLAB Renderer is based on HRTF filtering and is encapsulated in the Spatial object in Figure 2. The listener HRTF database 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 DSPFunction object (Figure 2) receives a frame of samples (remember, the Spatial object *is a* DSPFunction object). For a sample rate of 44100 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.

Scenario Update - converts scenario parameters to DSP parameters. The Spatial object performs a scenario update each time the user updates a scenario 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 scenario update rate is 120Hz.

- *Performed in Scene layer:*
  - *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 scenario 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 (a.k.a. "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.
  - *Provided 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.

4.3. SLAB Latency and Sound Buffer Management

The internal latency of SLAB is defined to be the time it takes for a scenario parameter modification to be rendered at the sound output of the system. Since the frame size is small, the internal latency is largely determined by the DSP parameter-tracking time-constant (a.k.a. smooth-time) and the size of the DirectSound output buffer. The latency of each is added to calculate the total internal latency. Since the smooth-time is adjustable by the user, a smooth-time of 0ms is assumed in the discussion below. In practice, the smooth-time is adjusted to be as low as possible without causing audible artifacts. A typical smooth-time value is 15ms.

When a scenario update occurs, the DSP parameters are updated within two milliseconds (two frames, 64 samples), ignoring smooth-time. The next frame of input samples is then filtered with the updated parameter values with the result transferred to the DirectSound write buffer. Within three milliseconds the write buffer (128 samples) data is transferred to the DirectSound output buffer (1024 samples). Assuming a full output buffer (worst case latency), the samples are available at sound output 23ms later.

Since the output buffer is somewhat costly to manage, the write buffer helps to minimize computational load. Ideally, the output buffer is kept fairly full in order to protect against SLABWire thread starvation. Thread starvation can result in output buffer underflow causing an audible click.

To measure internal latency under Windows98, an interval counter was placed between the parallel port and the sound output. A byte was written to the parallel port immediately prior to updating the listener orientation with an API function. The result of this update was a transition from an all zero HRIR to a single impulse HRIR. With a 128 sample write buffer and a 1024 sample output buffer, the measured internal latency of the system was 24ms. Preliminary measurements indicate that the latency under Windows98 and Windows2000 is comparable.

5. COMPARISON TO OTHER VAE SYSTEMS

Different VAE applications emphasize different aspects of the listening experience that require different approaches to rendering software/hardware. Auralization requires computationally intensive synthesis of the entire binaural room response that typically must be done off-line and/or with specialized hardware. A simpler simulation that emphasizes accurate control of the direct path, and perhaps a limited number of early reflections, may be better suited to information display. The fact that such a simulation does not sound "real" may have little to do with the quality of directional information provided. Achieving both directional accuracy and presence in virtual reality applications requires that head tracking be enabled with special attention devoted to the dynamic response of the system. A relatively high update rate (~60 Hz) and low latency (less than ~100 ms) may be required to optimize localization cues from head motion and provide a smooth and responsive simulation of a moving listener or sound source [8-11]. Implementing a perceptually adequate dynamic response for a complex room is computationally intensive and may require multiple CPUs or DSPs.

One solution for synthesizing interactive virtual audio has been the development of hybrid systems [e.g., 3, 12]. These systems attempt to reconcile the goals of directional accuracy and realism by implementing real-time processing of the direct path and early reflections using a model (e.g., the image model) combined with measured or modeled representations of late reflections and reverberation. During dynamic, real-time synthesis, only the direct path and early reflections can be readily updated in response to changes in listener or source position. A densely measured or interpolated HRTF database is needed to avoid artifacts during updates. Late portions of the room response typically remain static in response to head motion, or given enough computational power, could be updated using a database

VAE System / Primary Target Application	Audio Display	User Interface	OS	Implementation	Rendering Domain / Room Model
SLAB / research	headphone	C++	Windows 98/2k	software / Intel	time (HRIR) / image model
DIVA / research	headphone, speakers	C++	UNIX, Linux	software / SGI	time (HRIR) / image model
AuSIM / research	headphone	C	client-server model (client: Win98/2k, DOS, Mac, etc.)	software / Intel	time (HRIR) / direct path
Spat (IRCAM) / research	headphone, speakers	Graphical (Max, jMax)	Mac, Linux, IRIX	software / Mac, Intel, SGI	time (HRIR) / reverb engine
AM3D / research, games	headphone, speakers	C++	Windows 98/2k	software / Intel (MMX)	? / direct path
Tucker-Davis / research	headphone	Graphical / ActiveX	Windows 98/2k	special purpose DSP hardware (RP2.1)	time (HRIR) / direct path, reverb engine
Lake / research, entertainment	headphone, speakers	C++	Windows NT	special purpose DSP hardware (CP4, Huron)	frequency (HRTF) / precomputed BRIR
Creative Audigy / games	headphone, speakers	C++	Windows 98/2k	consumer sound card	proprietary / reverb engine
Sensaura / entertainment	headphone, speakers	3D sound engine	N/A	software / hardware	proprietary / reverb engine
QSound / games	headphone, speakers	3D sound engine	N/A	software / hardware	proprietary / reverb engine
Crystal River Convolvotron / research	headphone	C	DOS	special purpose DSP hardware	time (HRIR) / direct path

Table 2. *Summary table describing system characteristics for various VAE systems.*

VAE System	# Sources	Filter Order	Room Effect	Scenario Update Rate	Internal Latency	Sampling Rate
SLAB	arbitrary, CPU-limited (4 typical)	arbitrary (max. direct: 128, reflections: 32)	image model 6 1st order reflections	arbitrary (120 Hz typical, 690 Hz max.)	24 ms default (adjustable output buffer size)	44.1 kHz
DIVA	arbitrary, CPU-limited	arbitrary, modeled HRIRs (typical direct: 30, reflections: 10)	image model 2nd order reflections, late reverb	20 Hz	~110-160 ms	arbitrary (32 kHz typical)
AuSIM	32 per CPU GHz	arbitrary (128 typical, 256 max.)	N/A	arbitrary (375 Hz default max.)	8 ms default (adjustable output buffer size)	44.1 kHz 48 kHz (default) 96 kHz
AM3D	32-140, CPU-limited	?	N/A	~22 Hz max.	45 ms min.	22 kHz (current) 44.1 kHz (future)
Lake	1 (HeadScape, 4 DSPs)	2058 to 27988	precomputed response	?	0.02 ms min.	48 kHz
Convolvotron	4	256	N/A	33 Hz	32 ms	50 kHz

Table 3. *Summary table describing system specifications for various VAE systems.*

of impulse responses pre-computed for a limited set of listener-source positions. Model-based synthesis is computationally more expensive but requires less memory than data-based rendering [12]. The Lake Huron/HeadScape system relies entirely on long, densely pre-computed binaural room impulse responses (BRIRs) rendered with a fast frequency-domain algorithm. The early portion of the BRIR (4000 samples) is updated in response to head motion and the late reverberation remains static.

Tables 2 and 3 summarize system characteristics and specifications for some of the currently available virtual audio systems targeting different applications. (The Crystal River Convolvotron is listed for "historical" comparison purposes.) These systems tend to fall into two categories. Those aimed at high-end simulations for research purposes (e.g., auralization, psychoacoustics, information displays, virtual reality) tend to emphasize high-fidelity rendering of direct path and/or early

reflections, accurate models of late reverberation, and good system dynamics (high update rate, low latency). Other systems are directed toward entertainment and game applications. The rendering algorithms in such systems are proprietary and appear to emphasize efficient reverberation modeling; it is often not clear whether the direct path and/or early reflections are independently spatialized. The information in the tables is based on published papers in a few cases [e.g., 3, 6, 10] but more often on product literature and websites [13]. It is often difficult to determine details about a particular system's rendering algorithm and performance specifications. For example, critical dynamic parameters like scenario update rate and internal rendering latency are not readily available or not enough information about the measurement scenario is provided to evaluate the quoted values. Some systems listed in Table 2 are not present in Table 3 because not enough information was found regarding system performance specifications.

Informal listening tests of the SLAB system indicate that its dynamic behavior is both smooth and responsive. The smoothness is enhanced by the 120-Hz scenario update rate, as well as the parameter tracking method, which smooths at rather high parameter update rates; i.e., time delays are updated at 44.1 kHz and the FIR filter coefficients are updated at 690 Hz. The responsiveness of the system is enhanced by the relatively low latency of 24 ms. The scenario update rate, parameter update rates, and latency compare favorably to other virtual audio systems.

6. CONCLUSIONS AND FUTURE DIRECTIONS

The goal of SLAB is to provide a software-based experimental platform with low-level control of a variety of signal-processing parameters for conducting psychoacoustic studies. To meet this goal, a modular, object-oriented design approach was taken.

Recent additions to SLAB include source trajectories, API scripting, user callback routines, and reflection offsets. These features are included in the SLAB v4.3 release. A refined version of these features will soon be available in SLAB v5.0. Other v5.0 additions include the Scene layer and internal plug-ins. Presently in development for the v5.0 release are an external plug-in format, an HRTF per source feature, and a "sound event" architecture where sources are allocated and freed while rendering.

Future development includes enhancing the acoustic scenario with the addition of source radiation pattern, air absorption, surface transmission, and late reverberation models. To enable complex room geometries and higher order reflections, multiple processor systems and distributed architectures will be explored.

7. REFERENCES

[1] D. R. Begault, "Audible and inaudible early reflections: Thresholds for auralization system design." *100th Conv. Aud. Eng. Soc,* Copenhagen, preprint 4244, 1996.

[2] D. R. Begault, E. M. Wenzel & M. R. Anderson, "Direct comparison of the impact of head tracking, reverberation, and individualized head-related transfer functions on the spatial perception of a virtual speech source." *J. Aud. Eng. Soc.,* vol. 49, pp. 904-916, 2001.

[3] L. Savioja, J. Huopaniemi, T. Lokki, & R. Väänänen, "Creating interactive virtual acoustic environments." *J. Aud. Eng. Soc.,* vol. 47, pp. 675-705, 1999.

[4] J. D. Miller, J.S. Abel, and E.M. Wenzel, "Implementation issues in the development of a real-time, Windows-based system to study spatial hearing," *J. Acoust. Soc. Am.,* vol. 105, p. 1193, 1999.

[5] E. M. Wenzel, J. D. Miller, and J. S. Abel, "Sound Lab: A real-time, software-based system for the study of spatial hearing," *108th Conv. Aud. Eng. Soc,* Paris, preprint 5140, 2000.

[6] E. M. Wenzel, J. D. Miller, and J. S. Abel, "A software-based system for interactive spatial sound synthesis," *ICAD 2000, 6th Intl. Conf. on Aud. Disp.*, Atlanta, Georgia, 2000.

[7] J. D. Miller, "SLAB: A software-based real-time virtual acoustic environment rendering system." [Demonstration], *ICAD 2001, 9th Intl. Conf. on Aud. Disp.*, Espoo, Finland, 2001.

[8] J. Sandvad, "Dynamic aspects of auditory virtual environments." *100th Conv. Aud. Eng. Soc,* Copenhagen, preprint 4226, 1996.

[9] E. M. Wenzel, "Analysis of the role of update rate and system latency in interactive virtual acoustic environments." *103rd Conv. Aud. Eng. Soc,* New York, preprint 4633, 1997.

[10] E. M. Wenzel, "The impact of system latency on dynamic performance in virtual acoustic environments." *Proc. 15th Int. Cong. Acoust. & 135th Acoust. Soc. Amer.* Seattle, pp. 2405-2406, 1998.

[11] E. M. Wenzel "Effect of increasing system latency on localization of virtual sounds." *Proc. Aud. Eng. Soc. 16th Int. Conf. Spat. Sound Repro.* Rovaniemi, Finland. April 10-12, New York: Audio Engineering Society, pp. 42-50, 1999.

[12] R. S. Pelligrini, R. S. "Comparison of data- and model-based simulation algorithms for auditory virtual environments." *107th Conv. Aud. Eng. Soc,* Munich, 1999.

[13] Websites: www.3dsoundsurge.com www.ausim3d.com www.ircam.fr www.am3d.com www.tdt.com www.lake.com.au www.creative.com www.sensaura.com www.qsound.com

Acknowledgements: Work supported by the NASA Aerospace Operations Systems Program and by the United States Navy (SPAWARSYSCEN, San Diego).