A VERSATILE SOFTWARE ARCHITECTURE FOR VIRTUAL AUDIO SIMULATIONS

Nicolas Tsingos

Bell Laboratories

tsingos@research.bell-labs.com

ABSTRACT

Existing real-time audio rendering architectures provide rigid development frameworks which are not adapted to a wide range of applications. In particular, experimenting with new rendering techniques is virtually impossible.

In this paper, we present a novel, platform-independent software architecture that is well suited for experimenting with multi-channel audio mixing, geometrical acoustics and 3D audio processing in a single framework. Our architecture is divided into two layers. A low level DSP layer is responsible for streaming and processing audio buffers using a general *filter*-based formalism. Built on top is an audio rendering layer responsible for general geometry-based audio rendering and configuration of the rendering setup. In particular, we introduce *sequence* objects which we use to define and control arbitrary sound propagation paths in a geometrical 3D virtual environment. We discuss implementation details and present a variety of prototype applications which can be efficiently designed within the proposed framework.

1. INTRODUCTION

Virtual acoustics simulations are essential applications of today's audio processing techniques, especially in the context of interactive virtual environments (gaming, simulators), acoustics simulations (concert hall design, environmental noise studies) and realistic multi-channel mixing.

Several digital signal processing hardware and software architectures have been proposed to achieve real-time auralization in virtual acoustic environments. However, most of them are targeted towards very specific applications, e.g. 3D positional audio and reverberation effects for video games. As a result, they tend to provide control at a very high level which greatly impacts their flexibility and extensibility. Moreover, they cannot be used to auralize sound in arbitrary environments based on geometrical sound propagation paths (see Figure 1).

In this paper, we propose a new, platform-independent, software architecture and Application Programmer Interface (API), well suited to a wide variety of applications. Our audio rendering pipeline offers to the programmer several entry points resulting in improved flexibility. As a result, our software architecture allows for incorporating multi-channel audio mixing, geometrical acoustics, 3D positional audio rendering and lower level DSP operations into the same framework.

We propose a two-layer architecture. A low level layer is responsible for common DSP operation on multi-channel sound buffers through the definition of *filters* which can be used to perform any kind of operation on a source buffer. Our prototype implementation supports a multi-processing kernel for the filters.

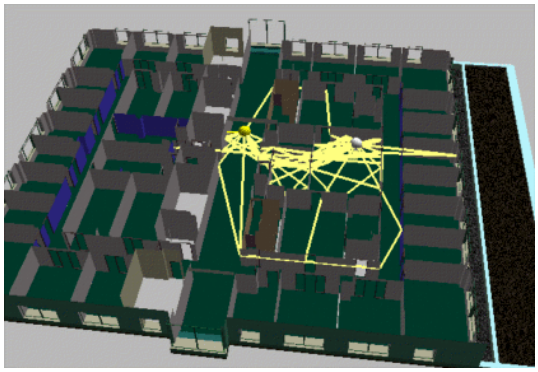

Figure 1: A real-time geometrical acoustics simulation based on beam tracing which computes early diffracted and reflected paths in large, densely occluded, virtual environments (2000 polygons in the case of the pictured building floor). Currently available audio rendering APIs do not offer enough flexibility to achieve interactive rendering of computed sound propagation paths.

Built on top is an audio rendering layer responsible for general geometry-based rendering, including 3D positional audio processing, and rendering configuration control. Unlike previous approaches, our entry point to the geometrical acoustics rendering pipeline is based on individual sound propagation paths. Control of the sound paths is achieved either by specifying the position and attributes of equivalent virtual *image-sources* or simply by specifying *sequences* of corresponding scattering events (reflections, diffractions, etc.) and associated geometrical primitives (surfaces, edges, etc.). We separate the construction of the propagation paths from the generation of the corresponding potential propagation sequences, the latter being left to the main application. It is thus possible to use any geometrical technique (ray tracing, beam tracing or image sources) to find the relevant propagation sequences. Moreover, we propose adaptive techniques to tune the rendering of each sound path individually, using less resources for perceptually less meaningful paths. Our architecture is designed to support time-critical multiprocessing applications, where the geometrical acoustics simulation is not necessarily carried out synchronously with the actual audio processing. In this context, we discuss tools for coherent update of the geometrical parameters over time, resulting in artifact-free sound generation.

Our paper is organized as follows: in Section 2, we discuss prior related work. Then, we present an overview of our software architecture in Section 3. In Sections 4 and 5 respectively, we present the low level DSP layer and the higher level geometrical acoustics/audio rendering layer. Finally, we give further detail on our prototype implementation and present sample applications, which were efficiently produced using our API, before concluding.

2. RELATED WORK

For the past 20 years, a tremendous effort has been devoted in the audio community to the problem of simulating and auralizing virtual sound sources [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. Although most aspects of the involved signal processing operations have been and are still subject to intense studies [13], no sufficiently flexible audio rendering software architecture has been proposed to date.

Several hardware solutions and associated software APIs have been proposed in the past few years but they fail to provide the necessary level of control for a variety of applications. First, most of them are targeted towards very specific applications such as video games and try to provide only high level controls to the game programmer: they are dedicated to 3D positional audio with additional reverberation modeling and do not allow for more accurate audio rendering relative to the geometry of the considered virtual environment [14, 15, 16, 17, 18, 19]. While this is not a major concern for video games or audio production applications, it might become a severe limitation for accurate simulation systems.

Some systems implement geometry-based rendering techniques [20, 21] but are limited by an inefficient simulation technique (recursive image-sources generation) and a high level interface (setting the position of sources, listener and surfaces) which prevents the use of alternate methods to generate the sound propagation paths. As a result, only a very small number of paths can be computed, updated and rendered. Moreover, geometry-based rendering in a complex, fully dynamic, environment is still difficult to achieve without artifacts (sound paths popping in and out due to occlusions, need for diffraction modeling, need for resampling to accommodate delay variation along moving paths and Doppler shifting effects, *etc.*).

Other hardware systems such as *convolution engines* [20, 22] provide efficient tools to perform large convolutions which are well suited to late reverberation effects. However, they can hardly be used to auralize the output of a typical interactive geometry-based simulation which usually consists of a moderate number of direct, early reflected and possibly diffracted sound paths (e.g., 1 to 50 per sound source).

No existing hardware-supported system provides a satisfying solution to efficient rendering of independent sound paths mostly because they fail to provide an interface to the rendering pipeline at the sound-path level.

Recently, several efforts have been made in order to bridge geometry-based and perceptual-based techniques. Savioja *et al* [23] presented the description of a software system for audio rendering in virtual environments. However, here again, although many details of the signal processing are explored, it is not clear whether the described system can easily accommodate a wide range of applications. In particular, the presented signal processing architecture is fixed and no API is described for development needs. The same comment applies to MPEG4 [24, 25] spatial audio extensions which allow for describing a 3D audio scene with both geometry-based and perceptual-based information.

Recent work on the *VAS* toolkit [12] is probably the closest in spirit to ours. The authors aim at introducing a general software toolkit for creating virtual sonic environments. In particular, they introduce very interesting tools to deal with procedurally generated sound sources and perceptually-based scheduling and processing of sound generation. However, the toolkit seems to be mostly dedicated to spatializing direct sound from the sources and its interface seems to be lacking low level tools for audio buffer handling and DSP operations, which might impact its portability and openness.

In contrast to the previous systems, whose frameworks tend to be fixed, low level development systems such as IRCAM's *MAX* [26] have been presented in the context of musical applications and encountered a wide success in the computer music community. They provide a variety of signal processing tools which can be combined together to create a more complex application. Although, extensions of the *MAX* system to virtual acoustics applications (IRCAM's *Spat* [4]) have been proposed, they are still lacking proper higher level tools to accommodate efficient geometry-based acoustic rendering in complex dynamic virtual environments.

In the remainder of this paper, we describe a software architecture which intends to balance simplicity of use and desire for control and openness by allowing a programmer to perform low level DSP operations over the audio data while providing higher level entry points well suited to geometry-based virtual audio applications. We also discuss new tools for more flexible and efficient integration of geometry-based audio rendering in a wide range of virtual acoustics applications.

3. OVERVIEW OF PROPOSED ARCHITECTURE

Our architecture is decomposed into two main layers both accessible from the application level (see Figure 2).

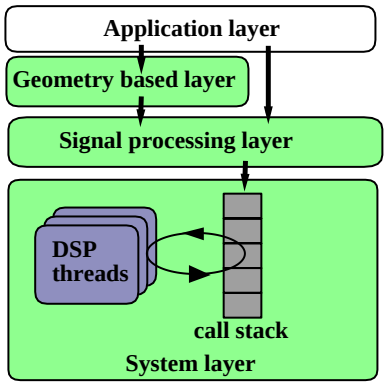

Figure 2: Overview of the proposed architecture which combines a high level geometry-based interface with low level signal processing calls. All necessary calculations are handled by a parallel DSP kernel (if multiple processors are available on the host computer).

The lower level layer is a DSP layer that manages sample buffers which are the key objects for sound manipulation. In addition to providing multi-channel audio I/O and buffer management capabilities, this layer provides the definition of *filter* objects which can be used to process the audio data. We further describe the DSP layer and filter objects in Section 4.

On top of the DSP layer, we introduce a higher level audio rendering layer which uses geometrical information about the environment. In our case, geometrical properties, such as sound source position, are not attached to a particular sound buffer as in [14]. In the same way, we do not attach environmental reverberation effects to a buffer or to a particular sound source or listener as in [14, 12]. We introduce a new class of general *sequence objects* which are used to represent a 3D propagation path derived from geometrical

information. Such objects are rendered to a destination buffer using a specific audio processing filter (see Section 6.1). The virtual audio scene can include any number of such paths which are used to represent direct or indirect contributions of multiple sources to one or several listeners. Full control of the number of paths per sound source and how they are generated is thus left to the main application. We also describe a mechanism to update the geometrical information asynchronously from the actual audio rendering stage. We further detail these points in Section 5.

Since the application can access both geometry-based routines and lower level DSP calls, the environment is fully open. In particular, it is easy to combine geometry-based rendering with statistical late reverberation usually implemented with IIR filters. This can be achieved directly by introducing new *late reverberation* filters objects in the DSP layer.

4. THE DSP LAYER

The basic object provided by our architecture is a (multi-channel) *audio buffer*. The DSP layer provides an interface to creating, handling and processing audio buffers.

4.1. Audio buffers and buffer handling

We use a classic technique to represent multi-channel audio data as an array of interleaved integer or floating point values. Audio buffers are characterized by several properties such as their number of channels, sampling rate of the audio data and quantization. They are also given a *chunk size* which is the atomic number of audio frames to be processed by calls to filter objects or to be streamed in or out. In our current implementation, the chunk size is specified when creating the buffer and may be different for every buffer. However, once fixed it cannot be changed. We did not found this to be a limitation but, if necessary, a variable processing size could easily be used. The typical chunk size we used for audio rendering is 1024 samples (about 1/50th of a second at CD quality). The total length of an audio buffer can then be defined either as a fixed number of such chunks or by a total number of audio frames.

Our buffer objects do not carry higher level properties (such as source location, etc.) as DirectSound 3D buffers. Instead, they act mainly as storage space for audio data. Buffers can contain any number of channels or tracks. These channels can be used in the usual multi-channel "surround" sense but can also be used as completely independent tracks. In particular, multiple channels could be used to store a frequency band decomposition of a single mono signal and perform operations on these different bands.

Finally, our buffers are all *circular* buffers, *i.e.* data access outside the buffer range is looped over the beginning or the end. Creating buffers "longer than needed" is necessary to apply IIR (need for past samples) or FIR (need for feed forward access) filters. Current processing or streaming point is defined by a current chunk index which needs to be shifted over time.

4.2. Buffer I/Os

A convenient way to design a platform-independent audio buffer I/O and streaming both from disk (audio file loading), audio hardware (audio playback) or network, is through a *stream object*. Our architecture supports I/O stream objects which are used to direct buffer data to a specific output (file, audio hardware, network, etc.) or to load a buffer with data from equivalent inputs. In our current implementation, no data conversion is performed at this stage, so the I/O object configuration (channels, sample rate, etc.) must match the configuration of the sound buffer. A particularity of the I/O streaming objects is that they can be used either directly or through an attached callback responsible for handling the buffer data. This allows for using *polling* capabilities already existing at the operating system level on certain platforms (e.g., SGI Irix) [1].

4.3. DSP operations on buffers

DSP operations on sound buffers are defined through *filter objects* which provide a basic command to process a source buffer. A filter here is not limited to finite or infinite impulse response filtering but is simply a procedure applying any kind of processing to an audio buffer. Each filter may possess its own user defined parameters and derived methods in order to provide control at the required level (for instance, a *late reverberation filter* might have a *reverberation time* control). Filters can be used inside other filters which makes it easy to extend an existing set of operations. Filter objects can be applied directly to buffer objects or called via a multi-threaded processing kernel for parallel execution. The processing kernel is currently implemented as a stack of processing commands which can be accessed by a configurable number of processing threads. Scheduling controls are available to select the desired processing order for each filter call. The DSP kernel together with the I/O stream object described in the previous section are the only system dependent components in our architecture.

Our current implementation supports three basic filters on audio buffers: *mix*, *convolve* and *resample*. All operations take a selected source track in a source buffer, apply the operation (which can take a few additional parameters) and copy or add the resulting data to the selected destination track of the destination buffer. The *mix* operation allows for copying a chunk (as defined by the buffer chunk size) of data from a source buffer to a destination buffer. It also supports a constant integer delay and a linear ramp between two specified gains which can be applied while the data is copied. The *convolve* operation applies a IIR or FIR filter to the current chunk of the source buffer and copies the resulting data to the current chunk of the destination buffer. If recursive IIR filtering is used, samples inside the previous active chunk will be accessed. If FIR feed forward filtering is used, samples inside future active chunks will be accessed. Our circular shifting mechanism allows to accommodate both. Our final operation, *resample* differs from the previous ones, since it allows for accessing audio data at any (non-integer) time index [27, 28]. It is similar to the mix operation but allows to access a chunk of audio data with floating point delay. Two delays can be specified which are linearly interpolated as the chunk of source data is copied to the destination buffer. Several schemes can be used for the waveform interpolation depending on the desired quality and computing resources (for example, see Section 6.2).

5. THE GEOMETRICAL ACOUSTICS AND AUDIO RENDERING LAYER

Our second, higher level layer, implements interface to geometry-based audio rendering and 3D positional audio. We introduce

[1] the system will automatically call the audio callback to fill the buffer as often as required to guarantee continuous audio output, possibly slowing down other processes.

novel *sequence objects* carrying geometrical information which are used to render a source buffer to a destination buffer. These objects correspond to individual sound propagation paths and are the entry point to our geometrical acoustics rendering pipeline.

A sequence object, corresponding to a unique sound propagation path, is introduced to store a list of pointers to geometrical objects including two endpoints (source, listener) and possibly several surfaces or edges (for reflection and diffraction). Accordingly, basic classes are provided to handle geometrical primitives such as points, polygonal surfaces and wedges. These basic primitives can be used by the main application to derive source, listener and surface objects including acoustic properties such as directivity or filtering functions.

Sequence objects are typically obtained as the output of a geometrical acoustics simulation (see Figure 3). We do not assume any *a priori* method to generate the propagation sequences, such as brute-force recursive enumeration (e.g., image-sources), since far better techniques have been recently introduced [29, 30, 31].

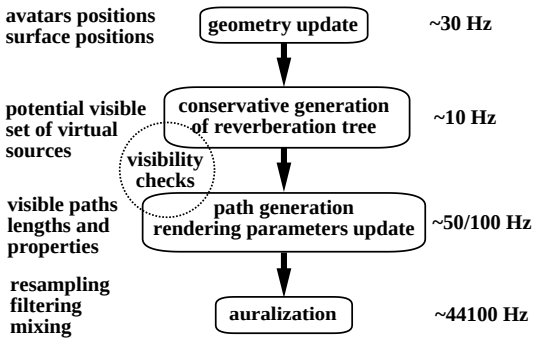

Figure 3: Typical organigram of a geometry-based virtual acoustics application. Every stage may executed concurrently and asynchronously. We indicate for each stage of the process the corresponding operations and typical refresh rates.

In a typical application, sound path creation/update and audio rendering will be executed concurrently. The main application will be responsible for running a geometrical simulation to generate a list of potential propagation sequences, while our API will be used to compute associated path attributes and validity. In the case of dynamic environments, the paths attributes (length, hit points on surfaces, visibility, *etc.*) must be updated continuously over time.

A sequence object supports two update modes. First, an automatic mode in which the main application only updates the geometrical objects properties (e.g., location of source, listener and surfaces) while all attributes of the corresponding path are updated automatically including length, hit points, incident and outgoing direction on surfaces. This is quite easy since each path is the shortest path from the source to the listener stabbing a given sequence of surfaces and (possibly) edges [32]. [2] This limits our current implementation to specular reflections, transmission and edge diffraction. Indeed, a sequence of such events can be represented, under the assumptions of geometrical acoustics, by a single propagation path or, in an equivalent manner, a virtual point source.

A second possibility to update a sequence is to let the main application directly provide the location of the corresponding virtual source, which might be more convenient or efficient in some cases. The same applies to other geometrical properties of the paths, such as hit points and incident/outgoing directions on surfaces. This gives complete control on path generation to the main application but still provides a possible entry point to our framework for subsequent processing.

Visibility updates must be provided by the main application, since we do not provide any access to the global spatial data structure in our architecture. However, validity checks along the path are provided and can be enabled on a per surface (or edge) basis. This is useful if the paths are generated using the image-sources technique or conservative beam-tracing [30] (path intersection with *portals* must be verified) but is useless if ray-tracing with an extended spherical listener is used [33]. Visibility problems in the case of sound waves are also closely linked to the treatment of diffraction by obstacles. The main application is also responsible for providing visibility checks and possible diffraction treatment (for instance by introducing new diffracted propagation paths from edges) [1, 32].

In the context of dynamic environments, we recently introduced original techniques allowing the main application to update sequences and path attributes asynchronously from the subsequent audio rendering [34]. Accordingly, our sequence objects also include a *prediction* mechanism which can be used to extrapolate necessary geometrical attributes (path length, incident direction on listener, etc.). When continuous changes occur in the environment, continuously varying attributes for every sound path can be computed by our sequence object. Artifact-free sound generation can then be maintained, even if the main application provides slow or discontinuous updates of the possible propagation sequences when sources and listeners are moving through the virtual environment. Moreover, when a significant number of sequences are considered, the cost of constructing the corresponding propagation paths might also impact the update rate significantly (especially when edge diffraction is present). Such a way of uncoupling geometrical calculation from signal processing is thus mandatory.

6. PROTOTYPE SOUND PATH RENDERER

In this section we discuss different aspects of an extension to our framework which implements a prototype sound path renderer. This extension defines additional objects to perform all signal processing operations involved in the auralization of a sequence object: a *geometrical rendering filter* and a *mapping* filter.

6.1. Geometrical rendering filter

To render a given sequence of geometrical events (i.e. reflections, transmissions, diffractions) we introduce a specific filter object as defined in Section 4.3. Given a pointer to a propagation sequence, the filter uses the current attributes of the corresponding propagation path to process a given source buffer and write the resulting data to a destination buffer. This usually involves resampling of the source audio data, filtering and possibly panning for various configurations of speakers [13, 12, 23]. Our current implementation of such a filter models frequency-dependence as a simple re-equalization over a set of predefined frequency bands but more complex models could be used [23]. All signal processing is implemented using basic filters from our lower level API.

[2] an application of the generalized Fermat principle.

6.2. Adaptive rendering of sound paths

The originality of our geometrical rendering filter is to provide adaptive rendering of individual sequence objects. First, since we offer direct manipulation of propagation sequences, the main application is totally free to generate any number of sequences per sound source or to select the sequences to render based on perceptual factors [12]. Moreover, the complexity of the DSP operations involved in rendering a sequence can be tuned to provide more processing power and resolution to most significant sound paths. In particular, we propose to perform adaptive resampling when computing variable delay effects (Doppler shifts). Depending on the amplitude for the path contribution and the compression/expansion ratio of the signal, it is possible to choose between nearest neighbor, linear or higher level interpolation to achieve the resampling of the audio data. More complex oracles involving predicted signal-to-noise ratios could also be derived. The same adaptive processing can be applied to the filtering stage, when rendering a sound path: the number of frequency bands or the type of filters to use, can be selected depending on the attenuation spectrum along the path. Finally, the resolution level can be set globally which allows for rapid switching from a high quality/slower implementation to a low quality/faster implementation.

6.3. Audio rendering configuration for sound paths

Virtual audio simulations are likely to be used with a variety of audio output formats depending on the application (stereophony, quadriphony, 5.1 and more discrete channels, matrixed formats, stereo, stereo binaural/transaural, etc.). Completely retargeting an application to a particular format, though probably more efficient in terms of signal processing cost, might be a hassle. Since our sequence objects keep track of all the geometrical information up to the actual rendering stage it is easy to accommodate a large variety of rendering options without any modification to the main application code. It also makes it possible to perform the geometrical simulation once and render the audio simultaneously for different listeners over different setups.

Several *mapping filters* can be defined to render a single sound propagation path from a point source to a point listener to a given number of output channels. They can implement a variety of techniques such as virtual microphone setups simulating various recording devices, direction based amplitude panning, binaural processing, *etc.* The mapping filter is called to produce the desired output, given a source buffer, a sequence object and a compatible destination buffer (*i.e.*, with the proper number of channels). Such mappings allow a single sound path to be spatialized over multiple channels. If several sound paths must be computed for a single sequence (e.g., one for each ear) multiple sequence objects with different listening locations must be created and independently rendered to the proper channels. The combinations of our geometrical rendering and mapping filters is close in spirit to the *localizer* object introduced in [12]. However, separating the "spatialization" phase allows for improved flexibility and minimizes the changes to the application when experimenting with a new 3D audio rendering technique or setup.

7. APPLICATIONS

Our API is object oriented and implemented in C++, which makes it easy to derive and create new processing objects or controls. Our current implementation runs on SGI and PC Windows systems. It provides classes for multi-channel sound buffer handling and streaming, DSP operations, geometrical sound paths rendering, and networking/distributed processing capabilities. The current implementation supports a multi-threaded DSP kernel for use on multi-processor machines. We used our software architecture to design several applications including: a basic sound file playback/acquisition from or to disk, a client/server application for audio streaming and rendering, a real-time multi-channel mixing and panning application and a geometry based auralization application for fully-dynamic virtual environments including real-time simulation of sound reflection and diffraction. In the first two cases, only the low level part of the API was necessary. For the multi-channel panning application, we used our sequence objects to model direct sound paths from several sources. Different mapping filters were used to render the sequences over various speaker setups but no complex geometrical simulation was involved. In the final case, our API was used to directly construct and render the propagation paths from potential propagation sequences generated by the main application. A ray-tracing process was used to generate the sequences corresponding to every source-listener channel [23, 34]. We also derived several late reverberation filters which were directly used to complement our geometry-based rendering. An arbitrary number of such filters can be used in a single scene (as far as enough computing power is available) and thus reverberations depending on every source-listener channel can be introduced.

Our API can be used together with standard graphics rendering APIs such as *OpenGL/GLUT* or *OpenInventor* [35, 36]. In particular, we used them as visualization/interface tools for our panning and virtual environment applications.

8. CONCLUSION

In this paper, we presented a new software architecture to achieve real-time audio rendering for a broad variety of applications. Contrary to prior spatial audio systems, we provide a low level control over audio data. Hence, the system is fully extensible since new DSP operations can be directly implemented. However, we maintain a higher level interface for geometry-based audio rendering and 3D positional audio. This high level interface is built on top of the lower layer and thus both can coexist inside the same application. Our interface to geometrical acoustics is provided at the sound path level which allows for using any existing geometrical simulation technique to generate the sound paths. In that context we described new tools to achieve asynchronous and artifact-free sound generation and automatic tuning of the required DSP workload on a per path basis. We implemented a prototype object-oriented API following the described architecture which runs on SGI Irix and PC Windows platforms. Our first experiments demonstrated that it is a very efficient and versatile tool to build a variety of applications involving audio DSP and auralization in virtual environments. We are planning to release a β-version of our API for research purposes.[3]

[3]please, check at: `http://www.multimedia.bell-labs.com/Research/VirtualAcoustics/` for updates.

9. REFERENCES

[1] Nicolas Tsingos and Jean-Dominique Gascuel, "Soundtracks for computer animation: sound rendering in dynamic environments with occlusions," *Proceedings of Graphics Interface'97*, pp. 9–16, May 1997.

[2] F.R. Moore, "A general model for spatial processing of sounds," *Computer Music Journal*, vol. 7, no. 3, pp. 6–15, Fall 1983.

[3] J.M. Jot, "An analysis/synthesis approach to real-time artificial reverberation," *Proc. of ICASSP*, 1992.

[4] Jean-Marc Jot, "Real-time spatial processing of sounds for music, multimedia and interactive human-computer interfaces," *Multimedia Systems*, vol. 7, no. 1, pp. 55–69, 1999.

[5] Henrik Møller, "Interfacing room simulation programs and auralisation systems," *Applied Acoustics*, vol. 38, pp. 333–347, 1992.

[6] J. Hahn, H. Fouad, L. Gritz, and J. Wong Lee, "Integrating sounds and motions in virtual environments," *Sound for Animation and Virtual Reality, Siggraph'95 Course #10 Notes*, Aug. 1995.

[7] M.R. Schröder, "Digital simulation of sound transmission in reverberant spaces," *J. of the Acoustical Society of America*, vol. 47, no. 2 (part 1), pp. 424–431, 1970.

[8] Stephen Travis Pope and Lennart E. Fahlén, "The use of 3D audio in a synthetic environment: An aural renderer for a distributed virtual reality," *ICMC Proceedings*, pp. 146–149, 1993.

[9] Irwin Zucker, "Reproducing architectural acoustical effects using digital soundfield processing," *Proc. AES 7th International Conference*, pp. 227–232, 1989.

[10] M.R. Schroeder, "Natural sounding artificial reverberation," *J. of the Audio Engineering Society*, vol. 10, no. 3, pp. 219–223, 1962.

[11] Holger Strauss and Jens Blauert, "Virtual auditory environments," *Proc. FIVE Conference'95*, pp. 123–131, 1995.

[12] H. Fouad, J.A. Ballas, and D. Brock, "An extensible toolkit for creating virtual sonic environments," *ICAD'2000*, may 2000.

[13] E.M. Wenzel, J.D. Miller, and J.S. Abel, "A software-based system for interactive spatial sound synthesis," *ICAD'2000*, may 2000.

[14] B. Bargen and P. Donelly, *Inside Direct X*, Microsoft Press, 1998.

[15] "Intel© Realistic Sound Experience (3D RSX)," 1998, http://developer.intel.com/ial/rsx/index.htm.

[16] "Environmental audio extensions: EAX 2.0 Creative©," 1999, http://www.soundblaster.com/eaudio.

[17] "ZoomFX, MacroFX, Sensaura©," 1999, http://www.sensaura.co.uk.

[18] "OpenAL: an open source 3D sound library," 2000, http://www.openal.org.

[19] W.F. Dale, "A machine-independent 3D positional sound application programmer interface to spatial audio engines," *Proceedings of the AES 16th international conference, Spatial sound reproduction, Rovaniemi, Finland*, pp. 160–171, april 1999.

[20] S.H. Foster, E.M. Wenzel, and R.M. Taylor, "Real-time synthesis of complex environments," *Proc. of the ASSP (IEEE) Workshop on Application of Signal Processing to Audio and Acoustics*, 1991.

[21] "A3D 2.0 software development kit, user's guide Aureal©," 1999.

[22] A. Reilly and D. McGrath, "Convolution processing for realistic reverberation," *Proc. 98th Audio Engineering Society Convention*, Feb. 1995.

[23] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen, "Creating interactive virtual acoustic environments," *J. of the Audio Engineering Society*, vol. 47, no. 9, pp. 675–705, Sept. 1999.

[24] J. Huopaniemi and R. Väänänen, "Advanced audio rendering capabilities for MPEG4 v.2 BIFS," 1998.

[25] J.M. Jot, L. Ray, and L. Dahl, "Extension of audio BIFS: Interfaces and models integrating geometrical and perceptual paradygms for the environmental spatialization of audio," 1998.

[26] M. Puckette, "Combining event and signal processing in the max graphical programming environment.," *Computer Music Journal*, vol. 15, no. 1, 1991.

[27] Holger Strauss, "Implementing doppler shifts for virtual auditory environments," *Proc. 104th Audio Engineering Society Convention*, May 1998.

[28] Udo Zölzer and Thomas Bolze, "Interpolation algorithms: theory and applications," *Proc. 97th Audio Engineering Society Convention, preprint 3898*, Nov. 1994.

[29] M. Monks, B.M. Oh, and J. Dorsey, "Acoustic simulation and visualisation using a new unified beam tracing and image source approach," *Proc. Audio Engineering Society Convention*, 1996.

[30] T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West, "A beam tracing approach to acoustic modeling for interactive virtual environments," *ACM Computer Graphics, SIGGRAPH'98 Proceedings*, pp. 21–32, July 1998.

[31] T. Funkhouser, P. Min, and I. Carlbom, "Real-time acoustic modeling for distributed virtual environments," *ACM Computer Graphics, SIGGRAPH'99 Proceedings*, pp. 365–374, Aug. 1999.

[32] Nicolas Tsingos, Thomas Funkhouser, Addy Ngan, and Ingrid Carlbom, "Modeling acoustics in virtual environments using the uniform theory of diffraction," *to appear in ACM Computer Graphics, SIGGRAPH 2001 proceedings, Los Angeles,*, Aug. 2001.

[33] H. Lehnert, "Systematic errors of the ray-tracing algorithm," *Applied Acoustics*, vol. 38, 1993.

[34] N. Tsingos, "Artifact-free asynchronous geometry-based audio rendering," *ICASSP'2001, Salt Lake City, USA*, may 2001.

[35] Josie Wernecke, *The Inventor Mentor*, Addison Wesley, 1994.

[36] J. Neider, T. Davis, and W. Mason, *OpenGL Programming Guide*, Addison-Wesley, 1993.