AudioBIFS: Describing Audio Scenes with the MPEG-4 Multimedia Standard

Eric D. Scheirer, *Student Member, IEEE*, Riitta Väänänen, and Jyri Huopaniemi, *Member, IEEE*

***Abstract*— We present an overview of the AudioBIFS system, part of the Binary Format for Scene Description (BIFS) tool in the MPEG-4 International Standard. AudioBIFS is the tool that integrates the synthetic and natural sound coding functions in MPEG-4. It allows the flexible construction of soundtracks and sound scenes using compressed sound, sound synthesis, streaming audio, interactive and terminal-dependent presentation, three-dimensional (3-D) spatialization, environmental auralization, and dynamic download of custom signal-processing effects algorithms. MPEG-4 sound scenes are based on a model that is a superset of the model in VRML 2.0, and we describe how MPEG-4 is built upon VRML and the new capabilities provided by MPEG-4. We discuss the use of structured audio orchestra language, the MPEG-4 SAOL, for writing downloadable effects, present an example sound scene built with AudioBIFS, and describe the current state of implementations of the standard.**

***Index Terms*— Audio coding, MPEG-4, SAOL, SNHC audio, 3-D audio.**

I. Introduction

THE Moving Pictures Experts Group (MPEG) subcommittee of the International Standardization Organization (ISO) began a new work item in 1995 to standardize low-bit-rate coding tools for the Internet and other bandwidth-restricted delivery channels. This project, now known as MPEG-4 [1], [2], will reach international standard status in mid-1999 as ISO 14496. However, during the period since its inception, the scope of MPEG-4 has expanded. It now includes not only traditional coding methods optimized for low-bit-rate transmission, but also highly novel technology that enables the object-based description of synthetic content, audiovisual scenes, and the synchronization of synthetic and natural content.

Among these new tools is the Binary Format for Scene Description, or *BIFS*. BIFS enables the concise transmission of audiovisual scenes composited from several component pieces of content such as video clips, computer graphics, recorded sound, and parametric sound synthesis. The part of BIFS controlling the compositing of sound scenes is called *AudioBIFS*. AudioBIFS provides a unified framework for sound scenes that use streaming audio, interactive and terminal-adaptive presentation, three-dimensional (3-D) spatialization, and/or dynamic download of custom signal-processing effects. Many of the concepts in BIFS originate from the Virtual Reality Modeling Language (VRML) standard [3], but the audio toolset is built from a different philosophy. AudioBIFS contains significant advances in quality and flexibility compared to VRML audio.

In this paper, we present an in-depth examination of the capabilities of AudioBIFS. We explore the relationship between AudioBIFS and the audio coding techniques in MPEG-4 and the relationship between AudioBIFS and audio in VRML. We present an example AudioBIFS sound scene and conclude with a discussion of current and future implementations of the MPEG-4 standard.

Manuscript received January 25, 1999; revised May 24, 1999. This paper was presented in part at the 1st COST/G6 Workshop on Digital Audio Effects Processing (DAFX-98), Barcelona, Spain, November 1998. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. M. R. Civanlar.

E. D. Scheirer is with the Machine Listening Group, Media Laboratory, Massachusetts Institute of Technology, Cambridge MA 02139-4307 USA.

R. Väänänen is with the Laboratory of Acoustics and Audio Signal Processing, Helsinki University of Technology, Helsinki, Finland.

J. Huopaniemi is with the Speech and Audio Systems Laboratory, Nokia Research Center, Helsinki, Finland.

Publisher Item Identifier S 1520-9210(99)06731-0.

II. MPEG-4 Audio and AudioBIFS

MPEG-4 is an *object-based* standard for multimedia. That is, a particular movie, radio program, or interactive multimedia application is transmitted as a number of *media objects*. These media objects may be streaming video segments, streaming video "sprites," still images, streaming audio tracks, synthetic visual graphics, or sound-synthesis instructions, among other types. The coding methods for each type of media object are specified in the MPEG-4 Audio and MPEG-4 Video standards. In a compliant MPEG-4 application, only MPEG-specified media objects may be contained in the bitstream.

As these elements are received by the client, or *decoding terminal*, they are composited together into an *audiovisual scene*. It is the scene, not the primitive media objects, that is presented to the person viewing the content. The instructions for composition are conveyed in a special format called BIFS. They may specify that certain media objects should be transformed before scene compositing—for example, a streaming video might be turned sideways or a soundtrack attenuated—or that certain objects should not be used at all in particular circumstances. BIFS and AudioBIFS are specified in the MPEG-4 Systems standard (ISO 14496-1).

In the present paper, we focus mainly on the sound-compositing capabilities of MPEG-4. The sound coding tools are described in detail elsewhere, both in the technical literature [4]–[7] and in the MPEG-4 Audio standard itself (ISO 14496-3), which is the official reference. There is an equivalent body of work on visual aspects of the standard that is outside the scope of our presentation.

This section will present an brief overview of the sound coding tools, discuss the sound-compositing philosophy in MPEG-4, and compare this philosophy with that of the popular VRML standard.

A. Sound Coding in MPEG-4

There are two groups of sound coding tools in MPEG-4: the *natural* tools [4], [5] that allow digital audio to be compressed and transmitted, and the *synthetic* tools [6], [7] that allow parametric descriptions of sounds to be transmitted and used to drive synthesis upon receipt.

The natural audio tools enable the compressed transmission of speech and wideband audio at ranges from 6 kb/s for low-bitrate speech coding to 64 kb/s per channel for high-quality multichannel sound. At the upper end of this range, the MPEG-4 tools have been demonstrated in psychoacoustic evaluation [8] to be nearly *perceptually transparent*; that is, even the most skilled listeners can barely distinguish the coded signal from the original in rigorous testing conditions.

There are three main audio coding tools in MPEG-4. The *general audio* (GA) coder allows the transmission of high-quality broadband multichannel signals such as music at bitrates from 16 to 64 kb/s/channel. This coder is a state-of-the-art, scalable version of well-known *perceptual compression* techniques [9]; it is based on the MPEG-2 Advanced Audio Coding standard [10] with additional improvements in quality and functionality for MPEG-4. The *CELP* coder uses codebook-excitation-linear-prediction techniques [11], [12] to enable highly compressed speech coding between 16 and 24 kb/s. The *parametric speech* coder is based on the harmonic vector excitation coding method [13] and provides toll-quality speech down to 6 kb/s.

There are two synthetic audio coders in MPEG-4. One provides an interface to text-to-speech systems: the so-called *text-to-speech-interface* (TTSI) receives a bitstream that contains phonemic and prosodic data and controls an external speech synthesizer [7]. No particular method of speech synthesis is specified in the standard. Only the interface and bitstream format are standardized in MPEG-4 TTSI.

The second is a very general music-and-sound-effects synthesis toolset called *structured audio* (SA). The structured audio coder allows transmission of sound-synthesis algorithms in a new "Music V" language called SAOL, for Structured Audio Orchestra Language [14] (SAOL is pronounced like the English word "sail"). An MPEG-4 terminal that supports structured audio has the ability to understand SAOL code and execute real-time synthesis of the algorithms transmitted. Transmitting sound as synthesis algorithms is a recent development [15], and MPEG-4 is the first standard to make use of this capability. In addition, a wavetable synthesis format called Structured Audio Sample Bank Format (SASBF) was developed in collaboration with the MIDI Manufacturers Association and is standardized in MPEG-4. The algorithmic and wavetable synthesis capabilities may be used at the same time in a synthetic soundtrack [16].

The music language SAOL is also important to the audio compositing tools. As we will describe in Section III-B, SAOL is used in MPEG-4 for downloading user-definable effects-processing algorithms. The convergence between the coding techniques for structured audio and effects processing in MPEG-4 [17] is one of the elegant and important aspects of the standard.

The sounds transmitted and decoded using the MPEG-4 audio tools are not immediately played back for the listener. Rather, they are *composited* together into a soundtrack; it is the soundtrack, not the component parts, that is presented. The composition process may be very simple, as in direct linear mixing, or very complex, with arbitrary effects-processing code downloaded and multiple sound objects presented spatially using 3-D audio. The description of the composition capabilities in MPEG-4 makes up Section III of the present paper.

B. Scene Graph Concepts

Both VRML and MPEG-4 BIFS rely on the *scene graph* to describe the organization of audiovisual material. We briefly outline the important concepts of scene-graph organization here to provide context for the material that follows.

A scene graph represents content as a set of hierarchically related *nodes*. Each node in the visual scene graph represents a visual *object* (like a cube or image), a *property* of an object (like the textural appearance of a face of a cube), or a *transformation* of a part of the scene (like a rotation or scaling operation). By connecting multiple nodes together, object-based hierarchies are formed. For example, one node might correspond to the location of a virtual character (an "avatar"). The *subgraphs*, or sets of connected nodes subsidiary to the avatar node, would represent the head and limbs of the character. By transforming the positions of the limbs, they may be made to move. By transforming the position of the character, all of the subgraphs ("local coordinate spaces") are automatically transformed as well, and so the character moves but the limbs stay in the same *relative* positions. An example scene graph is presented in Fig. 1.

Each node has several *fields* that detail the properties of the object. For an object node like a cube, the fields give the size and shape of the object. For a property node, the fields specify particular properties such as the color of the cube and the image to be texture-mapped to the cube. For a transform node, the fields specify the set of subsidiary nodes that are affected by the transformation, as well as the details of the transform.

Interactive media is created with scene graphs using an *event-routing* model. As the user moves the mouse or other input device around the scene and selects objects, they may be programmed to transmit events. The events are routed from one object to another, where it triggers some useful function. For example, as shown in Fig. 1, a button object can be attached to a **TouchSensor** node. When this button is clicked, the **TouchSensor** sends a event, which can be routed to the **startTime** field of a sound-playing node to trigger the playback of a sound. The content author specifies the particular event mechanisms used in a scene as part of the scene graph.

C. Sound Scenes in VRML

In order to compare AudioBIFS with a previous standard for interactive sound, we provide a brief outline of the audio capabilities of the well-known VRML standard [3]. VRML is primarily a language for the description of computer-graphics objects and their interaction properties, but it also has limited

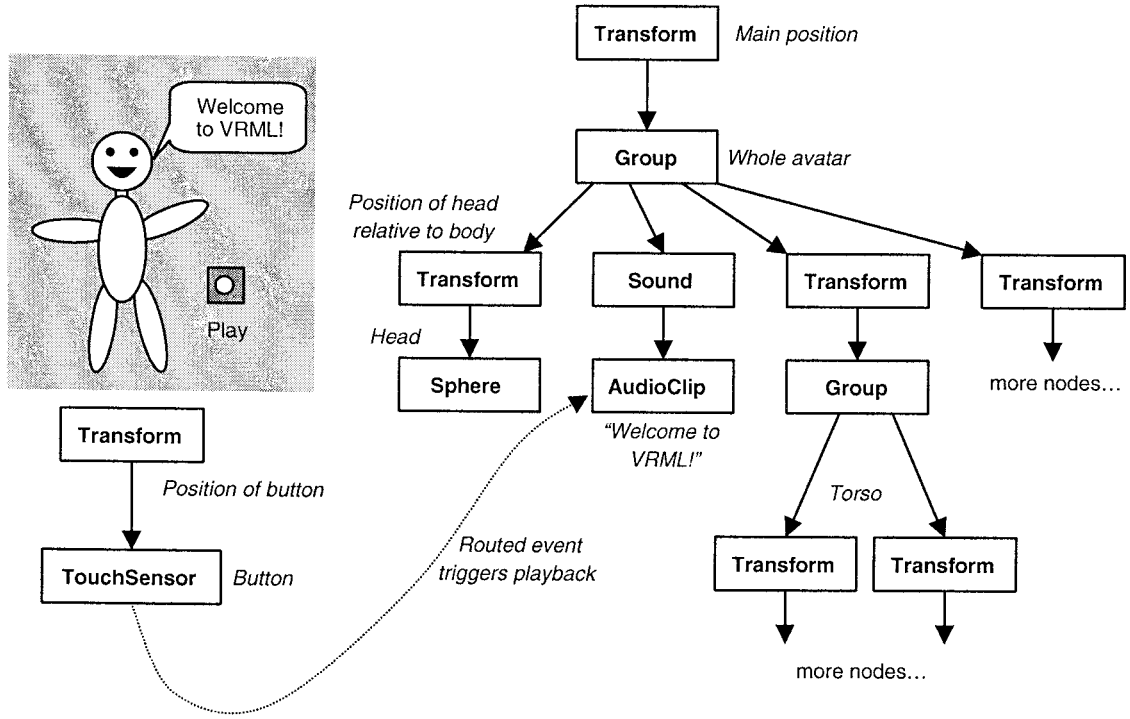

Fig. 1. An example scene in VRML, demonstrating the scene-graph concepts. An avatar is built from a head (modeled here by a sphere) and a number of other nodes, linked together hierarchically in a scene graph. Since the positions and rotations of the objects in the scene are hierarchically defined, changing the top-level transform (labeled *main position*) changes the positions of all the objects beneath. A button, when pressed, routes an event to the **AudioClip** node that starts the sound playback.

TABLE I
AUDIO NODES IN VRML

Name	Function
Sound	Position sound in virtual environment.
AudioClip	Include waveform audio for use in scene.

capabilities for the creation of interactive sound scenes. The VRML standard defines two nodes, **AudioClip** and **Sound**, that are used to incorporate sound objects into a virtual three-dimensional scene (Table I).

The **AudioClip** node provides audio data that can be referenced by **Sound** nodes; **AudioClip** can be thought of as a property node of the **Sound** node. The VRML standard specifies that **AudioClip** points to the location of an externally available sound file in a field called **url**. The location pointed to by this field contains a sound clip encoded in the WAVE format. The standard also recommends that MIDI playback be supported, but a VRML implementation is not required to do so.

AudioClip is a *time-dependent* VRML node, which means that it activates and deactivates itself at specified times. Fields called **startTime** and **stopTime** are provided for this purpose. The sound may also be looped for continuous presentation by setting a flag named **loop**. The **pitch** field specifies the rate at which the sampled sound is played. Changing the **pitch** field affects both the pitch and playback speed of a sound. By interactively controlling these fields through event routing, the sound playback can be controlled by a user or by a script. **AudioClip** does not *itself* play sound; it only *provides* sound material for use by one or more **Sound** nodes.

The **Sound** node specifies the location (spatial position) of a sound object in a VRML scene. The sound object is attached through a field called **source** and can be provided as either an **AudioClip** node (for audio only) or a **MovieTexture** node (for video with audio). The sound that results is located at a point, in the local coordinate system, specified by the **location** field. It emits sound in a frequency-independent ellipsoidal pattern, with the orientation of the ellipsoid defined by the **direction** field.

The audible sound field produced in a scene by the **Sound** node is shown Fig. 2. It consists of two nested ellipsoids whose shapes are defined by fields **maxBack, maxFront, minBack**, and **minFront**. Within the inner ellipsoid, the sound is scaled by the **intensity** field and there is no attenuation, i.e., the sound level is independent of the location of the virtual listener.[1] Between the inner and outer ellipsoid, the sound level decreases linearly on a decibel scale from 0 dB (the level inside the inner ellipsoid) to −20 dB. Outside the outer ellipsoid, no sound is rendered.

The **spatialize** field specifies whether or not the audio object will be spatialized when presented. If the **spatialize** field contains the value TRUE, the virtual listener's direction and the relative location of the **Sound** node is taken into account

[1]Throughout the article, we distinguish "virtual listener," the location of the avatar in the 3-D environment, from "listener," the real person who is viewing the content on a computer, set-top box, or mobile terminal.

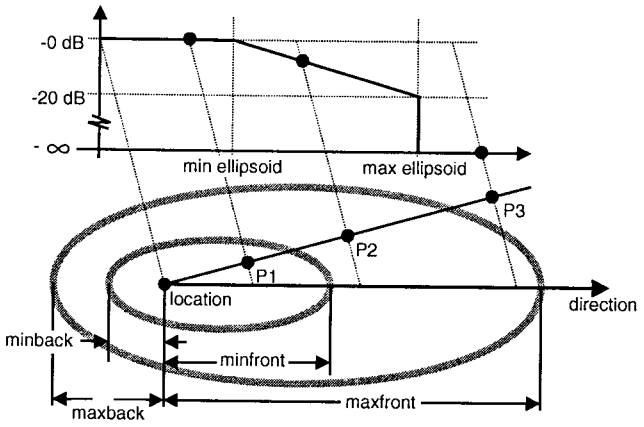

Fig. 2. The VRML "ellipsoidal" sound-attenuation model, adapted from [3]. The ellipsoids are specified with the parameters **location, minfront, maxfront, minback, maxback,** and **direction,** and are used to control the attenuation applied to a sound at location **location** in the local coordinate system. The graph above the ellipsoids shows the attenuation at various listening positions. The attenuation is calculated at three different positions, **P1, P2,** and **P3.** Within the inner ellipse (**P1**), there is no attenuation. Between the inner and outer ellipses (**P2**), the gain drops off linearly from 0 (at the inner ellipse) to −20 dB (at the outer ellipse). Outside the outer ellipse (**P3**), no sound is produced.

during playback. However, the method of spatialization is not *normative* (defined in the standard); it is assumed that the renderer uses the maximum sophistication available—typically amplitude panning in simple implementations and HRTF-based processing in more complex ones.

When multiple **Sound** nodes are contained in a single scene, a VRML browser typically adds together the (potentially spatial) sound from each to create the overall audio scene that is presented to the (real) listener, although the VRML standard is silent regarding the proper actions in this case.

Although BIFS inherits many functions from VRML, it also contains many improvements, particularly regarding sound quality and functionality. BIFS is specified as a compressed binary format, and thus equivalent scenes are smaller and quicker to transmit in BIFS than in VRML. VRML does not directly address issues relating to multichannel sounds (how to mix or spatialize them), and does not provide any direct control over mixing beyond intensity control. VRML does not specify a behavior if sounds are provided at different sampling rates, nor does it provide capability for streaming audio into a scene continuously—only clips of prerecorded sound may be used in VRML. AudioBIFS specifies actions and behaviors for all of these cases.

VRML implementations have become widely available in the last year. There are now several major companies providing VRML plugins for popular WWW browsers on a variety of platforms, and numerous authoring tools available. Major content providers such as CNN (www.cnn.com) are augmenting their sites with VRML content.

D. Sound Scenes in MPEG-4

There are two main modes of operation that are supported by AudioBIFS, the MPEG-4 audio compositing toolset. We term them *virtual-reality* compositing and *abstract-effects* compositing.

In virtual-reality compositing, the goal is to recreate a particular acoustic environment as accurately as possible. Sound should be presented spatially according to its location relative to the virtual listener in a realistic manner; moving sounds should have a Doppler shift; distant sounds should be attenuated and low-pass filtered to simulate the absorptive properties of air; and sound sources should radiate sound unevenly, with sonic directivity that is frequency-dependent as a function of angle of radiation. This type of scene composition is useful for "virtual worlds" applications and video-games, where the goal is to immerse the user as fully as possible in a synthetic environment. The VRML sound model described in the preceding section embraces this philosophy, albeit with fairly lenient requirements on how various sound properties must be realized in an implementation. The VRML sound nodes offer no functionality for such acoustical phenomena as sound reflections, reverberation time, the Doppler effect, frequency-dependent distance attenuation, or more sophisticated modeling of sound-source directivity.

In abstract-effects compositing, the goal is to provide content authors with a rich suite of tools from which they can choose the right effect for a given situation based on artistic considerations. As Scheirer [17] discusses in depth, the goal of sound designers for traditional media such as films, radio, and television is not to recreate a virtual acoustic environment (although this would be well within the capability of today's film studios), but to apply a body of artistic knowledge regarding "what a film should sound like." Spatial effects are sometimes used, but often in a non-physically–realistic way; the same is true for the variety of filters, reverberations, and other sound-processing techniques used to create various artistic effects.

MPEG realized in the early development of the MPEG-4 sound compositing toolset that if the tools were to be useful to the traditional content community—always the primary audience of MPEG technology—then the abstract-effects composition model would need to be embraced in the final MPEG-4 standard. However, new content paradigms, game developers, and virtual-world designers demand tools for the physical simulation of sound propagation as well.

MPEG-4 AudioBIFS therefore integrates these two components into a single standard. Sound in MPEG-4 may be postprocessed with arbitrary, downloaded filters, reverberators, and other digital–audio effects. It may also be spatialized and physically modeled according to the parameters of a simulated virtual world. These two types of postproduction may be freely interchanged and combined in MPEG-4 audio scenes.

The overall integration of synthetic sound, natural sound, virtual-reality postproduction, and abstract-effects postproduction is termed *synthetic/natural hybrid coding* of audio, or SNHC audio. MPEG-4 is the first audio standard to support significant SNHC functionality.

E. MPEG-4 Versions

MPEG-4 is being standardized in two versions. Version 1 was completed in March 1999 and will be published in

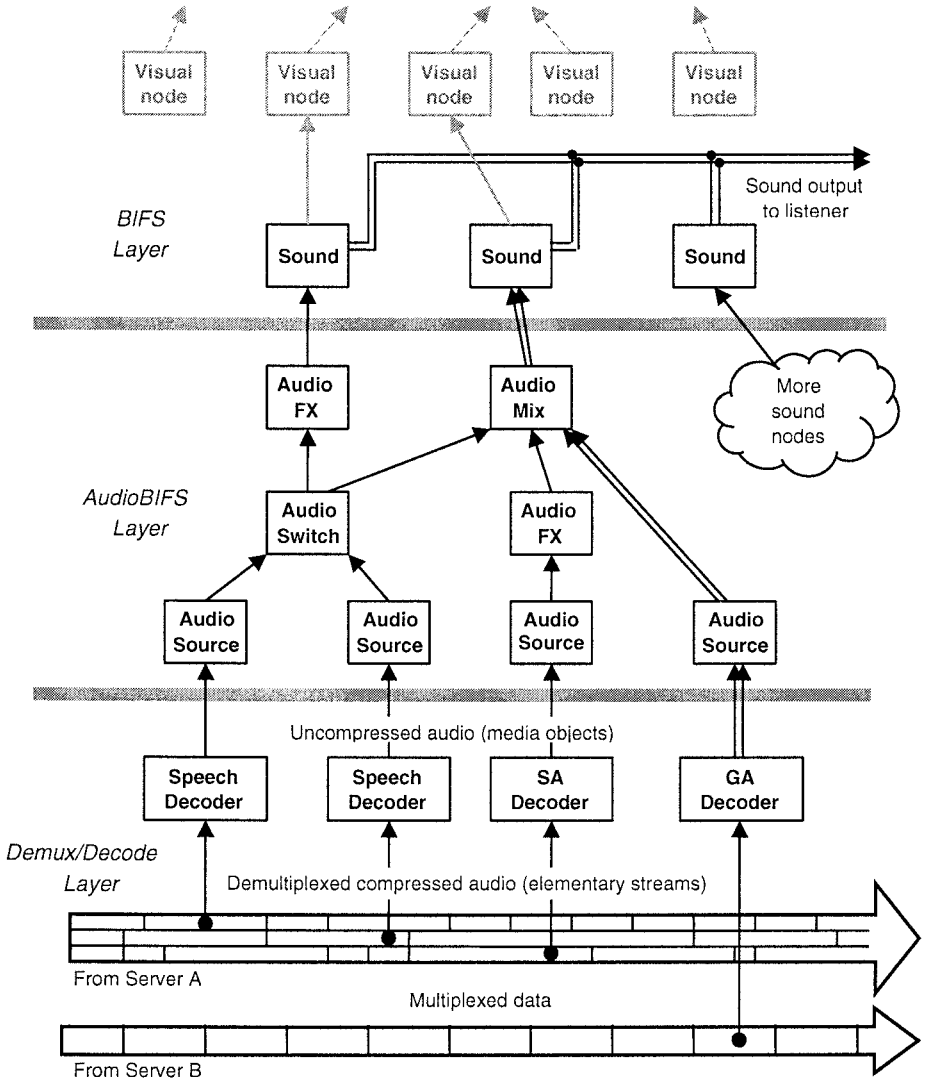

Fig. 3. The MPEG-4 audio system, showing the interaction between decoding, scene description, and audiovisual synchronization. The conceptual flow is from the bottom of the figure to the top. At the bottom, two multiplexed MPEG-4 bitstreams, each from a different server, convey several elementary streams containing compressed data. Each bitstream is demultiplexed; a total of four elementary streams are produced. The elementary streams are decoded using various MPEG-4 decoders into four primitive media objects containing uncompressed PCM audio data. The audio data is manipulated by the AudioBIFS scene graph and presented to the listener as though it emanates from the **Sound** nodes. © 1999 Marcel Dekker [7], used with permission.

mid-1999; Version 2 will follow a year later. Version 2 (which is technically an Amendment to MPEG-4) will be completely backward-compatible with Version 1 and will provide extensions in certain directions, such as advanced environmental auralization, Java capability, and a file format allowing MPEG-4 audio and video streams to be efficiently stored on fixed media such as CD-ROM's.

The present paper focuses mainly on the description of AudioBIFS capabilities in Version 1 (and thus is applicable to both versions). The discussion of Version 2 capabilities is confined to Section IV. Unless specifically mentioned otherwise, any general discussion of MPEG-4 applies to both Versions 1 and 2.

III. AudioBIFS Version 1

In this section, we describe the technical operation of the audio scene capabilities of MPEG-4. We begin with a high-level introduction to the overall audio system and then proceed to list each of the nodes that collectively comprise AudioBIFS and to explain the purpose and functioning of each.

A. The MPEG-4 Audio System

A schematic diagram of the overall audio system in MPEG-4 is shown in Fig. 3 and may be a useful reference during the discussion to follow.

Sound is conveyed in the MPEG-4 bitstream as several *elementary streams* that contain coded audio in the formats described in Section II-A. There are four elementary streams in the sound scene in Fig. 3. Each of these elementary streams contains a *primitive media object*, which in the case of audio is a single-channel or multichannel sound that will be composited into the overall scene. In Fig. 3, the GA-coded stream decodes into a stereo sound and the other streams into monophonic sounds.

The different primitive audio objects may each make use of a different audio decoder. For example, an MPEG-4 bitstream could contain a background music track coded using GA coding, two dialogue tracks (in different languages) coded using CELP coding, and a sound-effects track coded using structured audio. Multiple instances of each decoder may be used. For example, three different speech tracks, each in its own CELP stream, may be transmitted in a scene.

The multiple elementary streams are conveyed together in a multiplexed representation. Multiple multiplexed streams may be transmitted from multiple servers to a single MPEG-4 receiver, or *terminal*. There are two multiplexed MPEG-4 bitstreams, each originating from a different server, shown in Fig. 3. Encoded video content can also be multiplexed into the same MPEG-4 bitstreams. As they are received in the MPEG-4 terminal, the MPEG-4 bitstreams are demultiplexed, and each primitive media object is decoded. The resulting sounds are not played directly, but rather made available for scene compositing using AudioBIFS.

Also transmitted in the multiplexed MPEG-4 bitstream is the BIFS scene graph itself (the part of the bitstream that conveys the BIFS data is not shown in Fig. 2). BIFS—and AudioBIFS—are simply parts of the content like the media objects themselves; there is nothing "hardwired" about the scene graph in MPEG-4. The scene graph is transmitted at the beginning of the content session and may be dynamically updated as the content plays with a special stream of "BIFS Update" commands. Content developers have wide flexibility to use BIFS in a variety of ways. In Fig. 3, the BIFS and AudioBIFS parts of the scene graph are separated for clarity, but there is no technical distinction between AudioBIFS and the rest of BIFS.

AudioBIFS, like the rest of BIFS, is comprised of a number of nodes that can be interlinked to form a scene graph. However, the concept of the AudioBIFS scene graph is somewhat different; it is termed an *audio subgraph*. Whereas the main (visual) scene graph represents the position and orientation of visual objects in presentation space and their properties such as color, texture, and layering, an audio subgraph represents a signal-flow graph describing digital-signal-processing manipulations. Sounds flow in from MPEG-4 audio decoders at the bottom of the scene graph. Each "child" node presents its output (result from processing) to one or more "parent" nodes. Through this chain of processing, sound streams eventually arrive at the top of the audio subgraph. The "intermediate results" in the middle of the manipulation process are not sounds to be played to the user. Only the result at the top of each audio subgraph is presented, after the chain of audio nodes has processed the sound. We term a finished sound at the top of an audio subgraph a *sound object*.

Audio processing using the scene graph and AudioBIFS is tightly coupled with real-time audio decoding using the MPEG-4 audio tools as described above. The **AudioSource** node (see Section III-B1) connects primitive audio material, produced by the audio decoders, to the scene graph. Sound begins flowing into the scene at each of these nodes. At the top, each audio subgraph is rooted in a **Sound** node (see Section III-B7), which allows sounds to be attached to visual objects in the world and dynamically moved in response to user interaction. Many audio subgraphs may be present in any audiovisual scene, and not every sound object has to be attached to a visual object. In Fig. 3, there are three sound objects, with the audio subgraph fully expanded for two of them. These same two **Sound** nodes are associated with visual objects—each of them has a parent in the main scene graph. The third (the right-most, for which the subgraph is not fully expanded) does not have any visual correlate in the scene.

TABLE II
AUDIO NODES IN MPEG-4 VERSION 1 AudioBIFS

Name	Function
AudioSource	Attach sound decoder to scene graph
AudioMix	Mix M channels of sound into N channels
AudioSwitch	Select subset of M input channels of sound
AudioDelay	Delay sounds for synchronization
AudioFX	Apply algorithmic signal-processing effects
AudioBuffer	Cache sound for use in interactive playback
Sound	Position sound in 3-D virtual environment
Sound2D	Position sound in 2-D scene
Group	Group multiple nodes together for hierarchical transformation
ListeningPoint	Specify location of virtual listener in scene
TermCap	Query terminal for available playback resources

The MPEG-4 Systems standard contains a specification for the resampling, buffering, and synchronization of sound in AudioBIFS. Although we will not discuss these aspects in detail, the MPEG-4 standard precisely specifies the resampling and buffering requirements associated with each of the nodes described in Section III-B. These aspects of MPEG-4 are *normative*; that is, every MPEG-4 terminal must implement them the same way. This makes the sound-processing behavior of an MPEG-4 terminal highly predictable to content developers and able to produce sound of consistently high quality.

B. AudioBIFS Nodes

There are eight BIFS nodes that comprise the AudioBIFS toolset. In addition, a few of the general-purpose BIFS nodes have associated sound behavior. This section discusses each of the AudioBIFS nodes, giving their syntax and semantics and describing their function in an audio scene (Table II).

As described in Section II-B, each node has several *fields* that specify the parameters of operation of the node. In MPEG-4 BIFS, these fields and their operating range are carefully quantized and transmitted in a binary data format for maximum compression of the scene graph. Here, we give a more conceptual description using the nonnormative textual names of the fields.

1) AudioSource: The **AudioSource** node is the point of connection between real-time streaming audio and the AudioBIFS scene. The **AudioSource** node attaches an audio decoder, of one of the types specified in the MPEG-4 audio standard, to the scene graph, and allows audio to flow out of it.

The **AudioSource** node has *time-sensitive* fields (**startTime** and **stopTime**) that allow the playback of sound data to be

started, stopped, paused, and rewound, when the transmission scenario allows such function (in a one-way satellite broadcast paradigm, "fast-forward" is not possible and arbitrarily long "rewinds" require arbitrarily much storage). Fields named **pitch** and **speed** allow the playback pitch and speed to be controlled for decoders which allow this functionality (only the Structured Audio and HVXC decoders in MPEG-4 Version 1). A field named **numChan** specifies how many channels of audio, from those produced by the decoder, should be used. A field called **phaseGroup** allows the content developer to specify that there are *phase relationships* among the several channels of audio produced by the decoder—that is to say, to declare that, from a seven-channel decoded stream, the first two channels (for example) are a stereo pair, the next four are a quadraphonic set unrelated to the first two, and the final channel is not related to any of the first six. This information is important for executing effects on multichannel sets and producing spatial audio.

Finally, there is a field called **children** that is only used in a special case pertaining to the structured audio decoder. See the discussion under **AudioBuffer** for more details.

2) AudioMix: The **AudioMix** node allows M channels of input sound to be mixed into N channels of output sound through the use of a mixing matrix. The M channels of input may be all from the same child source, all from different children, or any desired combination. If the child sound sources are at different sampling rates, all of the input data is resampled to the highest of the sampling rates of the children before mixing. The resampling always goes to the highest rate for maximum sound quality; there is no option to downsample sounds or use another sampling rate in the scene graph.

The fields of the **AudioMix** node are **children**, which attaches the child AudioBIFS nodes; **matrix**, which contains the mixing matrix; **numInputs**, containing the number of input channels (needed so that the shape of **matrix** is known) and **numChan** and **phaseGroup**, which are as in **AudioSource**—they identify these characteristics for the sound output from the node.

3) AudioSwitch: The **AudioSwitch** node allows N channels of output to be taken as a subset of M channels of input, where $N \leq M$. It is equivalent to, but easier to compute than, an **AudioMix** node in which $N \leq M$ and all matrix values are zero or one. This node allows efficient selection of certain channels, perhaps on a language-dependent basis. As with **AudioMix**, input sounds are resampled to a single rate before selection occurs.

The fields of the **AudioSwitch** node are **children**, which attaches the child nodes; **whichChoice**, which specifies the particular subset of channels to pass through; and **numChan** and **phaseGroup**, which are as in **AudioSource**.

4) AudioDelay: The **AudioDelay** node allows several channels of audio to be delayed by a specified amount of time, to enable small shifts in stream timing for media synchronization. As with **AudioMix** and **AudioSwitch**, if the input channels are not all at the same sampling rate, they are resampled before the delay is computed.

The fields of **AudioDelay** are **children**, which attaches the child nodes; **delay**, which specifies the amount of time delay; and **numChan** and **phaseGroup**, which are as in **AudioSource**.

5) AudioFX: The **AudioFX** node allows the dynamic download of custom signal-processing effects to apply to several channels of input sound. A special sound-processing language called SAOL [14], as discussed in Section II-A, allows arbitrary effects-processing algorithms to be transmitted in the scene graph.

The use of SAOL to transmit audio effects means that MPEG does not have to standardize the "best" artificial reverberation algorithm (for example), but also that content developers do not have to rely on terminal implementors and trust in the quality of the algorithms present in an unknown playback device. Since the execution method of SAOL algorithms is precisely specified, the content developer has precise control over exactly which reverberation algorithm (for example) is used in a scene. If a reverb with particular properties is desired, the content author transmits it as part of the bitstream and its use is guaranteed. An example reverberator written in SAOL is shown in Section V.

SAOL has many useful algorithms built into it, such as comb and allpass filters, multitap fractional delay lines, digital FIR and IIR filters, a flexible parametric compressor, and chorus and flanging operations. It is arbitrarily extensible to include new algorithms in that SAOL is not a "suite of digital effects" but a *language* for describing synthesis and digital-effects algorithms. Any algorithm for digital sound manipulation can be written in SAOL.[2]

Time-varying parametric effects can be controlled using the scripting language SASL (Structured Audio Score Language), also standardized in the MPEG-4 Audio standard. SASL is a simple but flexible protocol for specifying time-varying parameters to synthesis and digital-effects algorithms. For example, the shape of a resonant filter used to process a voice track in an interactive music composition might change over time. The sequence of parameter changes required to encode this behavior can be represented in SASL.

As with other AudioBIFS nodes, multiple child nodes may be attached to the **AudioFX** node. If these children are running at different sampling rates, the input data is resampled before it is presented to the SAOL signal-processing algorithms. The **phaseGroup** fields of the children are made available to the SAOL orchestra, and the algorithms in SAOL may thereby depend on the particular phase-relationships of the inputs. For example, a digital reverb may be written to behave differently on a stereo pair than on two uncorrelated input signals. The position of the **Sound** node in the overall scene,

[2]This statement is proved by making a connection between the SAOL language and a Turing machine (TM). It is straightforward to construct an "effects-processor" that implements a TM in SAOL and provides a program (an effects-processing algorithm) to run in the TM as a parameter stream. As proved in standard references ([18], for example), the demonstration that a computational system can simulate a TM is sufficient to conclude that the system is capable of computing any computable function. Under the reasonable assumption that any desired audio effect is computable, this construction thus proves that every effects algorithm may be delivered as a SAOL orchestra (although, of course, this statement says nothing about the computational cost). This does not imply that the practice of simulating a TM in the decoder is the preferred manner of transmitting effects-processing algorithms—most algorithms have much more direct implementations using the standard capabilities of SAOL [14].

as well as the position of the virtual listener in the 3-D environment, are also made available to the **AudioFX** node, so that the effects-processing may also depend on the spatial locations (relative or absolute) of the virtual listener and virtual source.

The fields of the **AudioFX** node are **children**, which attaches the child nodes; **orch**, which specifies the SAOL orchestra; **score**, which specifies the SASL script, if needed; **params**, which allows scene-graph-level interaction control of effects (see Section III-C) and **numChan** and **phaseGroup**, which are as in the other nodes.

6) AudioBuffer: The **AudioBuffer** node allows a segment of audio to be excerpted from a stream, and then triggered and played back interactively. It is similar in concept to the VRML node **AudioClip**, but contains additional semantics to enable its use in one-way streaming media applications (where random-access and dynamic retrieval is not possible). The **AudioBuffer** node does not itself contain any sound data; instead, it records the first n seconds of sound produced by its children. It captures this sound into an internal buffer. Then, it may later be triggered interactively (see the section on interaction below) to play that sound back.

This function is most useful for "auditory icons" such as feedback to button-presses. It is impossible to make streaming audio provide this sort of audio feedback, since the stream is (at least from moment to moment) independent of user interaction. The limited backchannel capabilities of MPEG-4 are not intended to allow the rapid response required for audio feedback. To use the **AudioBuffer** node to create an audio feedback event, the sound desired is streamed into the **AudioBuffer** node, either directly from a decoder, or from an audio subgraph that creates the sound from component objects. Rather than immediately pass this sound through, as the other audio nodes do, the **AudioBuffer** node holds the sound in a buffer for later use. At some later time, mouse-click events (for example) are routed to the **startTime** field of the **AudioBuffer** node, which plays the buffered sound at that time. Each time the **startTime** field is changed, the sound plays again.

As with other AudioBIFS nodes, multiple child nodes may be attached to the **AudioBuffer** node. If these children are running at different sampling rates, the input data is resampled before it is presented to the SAOL signal-processing algorithms.

The fields of **AudioBuffer** are **children**, which attaches the child nodes; **length**, which specifies how much sound to record; **startTime** and **stopTime**, which control interactive playback of the sound; and **numChan** and **phaseGroup**, which are as in the other nodes.

There is a special function of **AudioBuffer** that allows it to cache samples for use in sampling synthesis in the Structured Audio decoder. The **children** field of the **AudioSource** node may only be used when the **AudioSource** node is attached to a structured audio decoder. In this case, the **children** must all be **AudioBuffer** nodes. When this construction is present, the sounds recorded in the **AudioBuffer** nodes are made available to the structured audio decoder attached to the **AudioSource** for use in the synthesis process. This allows MPEG-4 compression techniques to be applied to sound samples, which can greatly reduce the size of bitstreams that use sampling synthesis.

7) Sound: The semantics of the **Sound** node in MPEG-4 are similar to that of the VRML standard, i.e., the sound attenuation region (fields **direction, minBack, maxBack, minFront, maxFront**) and spatialization (fields **location, spatialize**) are defined in the same way as in Section II-C. This node is used in MPEG-4 to attach sound to 3-D audio scenes.

In contrast with VRML, where the **Sound** node accepts raw sound samples directly and no intermediate processing is done, in MPEG-4 any of the AudioBIFS nodes may be attached to the **Sound** node. Thus, if an **AudioSource** node is the child node of the **Sound** node, the sound as transmitted in the bitstream is added to the sound scene; however, if a more complex audio scene graph is beneath the **Sound** node, the mixed or effects-processed sound is presented. The spatialization effects may be added to sound whether or not complex processing has taken place. However, spatialization is not applied to multiple channels of sound that have phase interactions among them (as specified using the **phaseGroup** fields of the children), as to do so can produce unpleasant "phasing" effects. If the content author truly wishes the individual channels of a stereo or multichannel set to be spatialized, he or she may split them up with **AudioMix** nodes and then apply spatialization separately.

The particular spatial effects applied to a sound depend on the location of the sound and that of the virtual listener in the virtual world. The content author may also specify that no spatial effect applies to a certain sound. All of the spatial and nonspatial sounds produced by the **Sound** node(s) in the scene are summed and presented to the user. The methods of spatialization and presentation are not normative in MPEG-4.

8) Sound2D: The **Sound2D** node is used to attach sound to two-dimensional (2-D) BIFS Scenes. The source of audio is the same as in the **Sound** node, with the similar possibility to route the audio through an audio subtree.

The spatialization in this node is carried out in a 2-D plane, allowing the spatialization to happen in a restricted manner. The assumed field of view in a 2-D scene is a 2 m $\times$ 1.5 m area viewed from a 1 m distance, and the 3-D spatialization is done according to the sound location in the corresponding azimuth and elevation angles, with the maximum sophistication possible.

9) Group (and Other Grouping Nodes): Several general BIFS nodes allow multiple nodes to be grouped together. These *grouping nodes* include **Group, Group2D, Transform,** and **Transform2D.** Grouping nodes allow higher levels of the scene to spatially transform multiple low-level elements. For example, the sound of an automobile as heard from the street could be modeled with several objects: an "engine sound" that is located under the hood, an "exhaust sound" that is located in the tailpipe, and a "radio sound" that is located inside the passenger area. These three sounds are grouped together under a **Group** node; then, when the **Group** node is moved in the scene, the three subsounds each move, but maintain the same relative positions.

When grouping nodes are used in the scene to group together multiple **Sound** nodes, the sounds represented in each

are summed together. When nodes such as **Transform** are used, they modify the location and direction of the (spatially presented) sounds grouped under them relative to the local coordinate system. Thus, the **Transform** node can be used conveniently to move or rotate a group of sound objects in a scene; it is more useful in a virtual-reality scene than in a purely abstract-effects scene, since its only effect is on the virtual locations of sounds.

10) ListeningPoint: This node controls the position of the listening point in a scene. The listening point at any time is a 3-D location and a "facing direction" in the 3-D coordinate space making up the virtual world. The listening point thus has six degrees of freedom and may be moved and rotated freely about the space.

The spatial positions of sources are calculated relative to the listening point. The listening point is the location in the virtual scene at which the virtual listener's ears are located. By default, if no **ListeningPoint** node is used, the viewpoint (the position of the virtual viewer's "eyes") and the listening point are the same. The **ListeningPoint** node only directly affects sounds produced by the **Sound** node, when spatialization is used there. The listening-point location is also provided to the **AudioFX** node so that the SAOL code may provide virtual-listener-location-dependent processing.

11) TermCap: The **TermCap** node is not an AudioBIFS node specifically, but provides capabilities that are useful in creating terminal-adaptive scenes. The **TermCap** node allows the scene graph to query the terminal on which it is running, to discover various properties of its hardware and performance. For example, **TermCap** may be used to determine the ambient noise floor of the environment, measured in a nonnormative way. Based on the result, different parts of the scene graph may be switched in and out. This applies not only to the audio sources (primitive media objects) themselves, but also to the manner in which they are postprocessed. For example, a scene could specify that a compressor is applied in a noisy environment such as an automobile, but not in a quiet environment such as a listening room.

Like other capabilities in MPEG-4, the particular action that is taken based on **TermCap** are not "built in" to the terminal, but downloaded in the bitstream. The content developer, not the terminal manufacturer, decides what should happen in the case of (for example) a noisy environment, and this can differ from application to application and from one piece of content to another. Other audio-pertinent resources that may be queried with the **TermCap** node include: the number and configuration of loudspeakers, the maximum output sampling rate of the terminal, and the level of sophistication of 3-D audio functionality available.

C. Interactive Audio Scenes

The AudioBIFS nodes described in the previous section may be used in static presentations, in which all of the parameters are downloaded in a fixed scene graph and a single piece of content is played back. Facilities in MPEG-4 also allow the construction of sophisticated *interactive* content.

Most of the fields in the AudioBIFS nodes are termed *exposed* fields. That is, their values may change during the content playback. The exposed fields may be changed by the content server, using a special "BIFS Animation" syntax in the BIFS data stream. They may also be changed by an interactive event-routing model identical to the one in VRML as described in Section II-B. These changes may be driven by user interaction with an interface or other external commands. Thus, if the content contains a user interface that allows the user to manipulate the values in the **matrix** field of an **AudioMix** node, the result is to give the listener control over the "fader levels" in postproduction.

Each of the important control parameters is exposed for each node. The **params** field of the **AudioFX** node allows further user interaction with the scene, by allowing event routing to control some of the parameters of downloaded effects-processing algorithms. The semantics of the **params** field change from application to application, depending on how the values are used by a particular SAOL effect. 128 user-definable parameters are provided.

These interaction capabilities are not provided "by default." There is no way for a user to manipulate MPEG-4 content unless the content developer specifically provides the interaction mechanism. Thus, both fixed content and manipulated content may be created in MPEG-4.

The scene graph itself may be modified through a special stream called the "BIFS Update" stream. The BIFS Animation and BIFS Update streams are multiplexed into the overall MPEG-4 bitstream as described in Section III-A; the effects of the BIFS Update may be as simple as adding one node to the scene graph, or as complex as replacing the entire scene graph with a new scene graph.

D. Profiles and Levels of AudioBIFS

The MPEG-4 standard is very complex and implementing all of it is a somewhat daunting task. The standards development process has identified several profiles, which are subsets of functionality that may be implemented in a conforming system. Only a system that conforms to one of the specific profiles may be termed "MPEG-4 compliant." The profiles of MPEG-4 are application-driven, so it is expected that in the future, new profiles will give rise to new applications. Currently, the "Complete" profile demands implementation of all AudioBIFS nodes, and the "Complete 2D," and "Audio" profiles each demand implementation of all AudioBIFS nodes except **Sound** (**Sound2D** is required in these profiles). The Complete Profile includes all 2-D and 3-D visual and audio capabilities of the standard, the Complete 2D Profile only the 2-D capabilities, and the Audio Profile is targeted at radios and other audio-only devices. This profile does not require implementation of any of the visual capabilities of the standard. Finally, there is a "Simple 2D" profile that includes only the **Sound2D** and **AudioSource** nodes as well as simple visual capabilities. This profile provides functionality similar to that of the MPEG-2 standard.

Within each profile, levels are defined to restrict the amount of computational complexity required by the scene. Since the syntactic scene graph can become arbitrarily large, it is always possible to deliver a scene that is too complex

for a given decoder to render in real-time. The level of a decoder describes the amount of computation that the decoder is capable of providing, so that content authors may be aware of the capabilities of a target decoder. The levels for audio capabilities are not yet set, although the measurement paradigm is well understood: the total number of sample-rate conversions and mixing operations in the scene are counted, and a simulation tool is provided for computing the complexity of **AudioFX** nodes. Based on feedback from implementators and content developers, corrigenda to the standard will set levels suitable for the marketplace.

IV. MPEG-4 Version 2

In this section we describe the features proposed for AudioBIFS in MPEG-4 Version 2, which will become an official amendment to MPEG-4 in January 2000. The AudioBIFS extensions to the first version of the MPEG-4 standard concern audio environment modeling in a manner more natural than is possible in the current BIFS and VRML standards. As MPEG-4 Version 1 augments the virtual-reality model of sound in VRML with a versatile abstract-effects model, MPEG-4 Version 2 extends the simple virtual-reality model to include two rich and robust techniques for creating virtual audio environments. The first technique is *physical*: modeling of the acoustic environment is bound to the physical reality defined by the visual scene. The second is *perceptual*: creation and modification of environmental sound characteristic is based upon perceptual parameterization.

In this section, we briefly discuss the concepts behind virtual audio environments. We then explain the physical and perceptual approaches to environmental modeling in MPEG-4 Version 2. We discuss the different application areas targeted by the two approaches.

A. Physical Modeling of Acoustic Environments

By *physical modeling of acoustic environments* we mean processing sound so that the acoustic effects processing corresponds to the visual scene. This involves modeling individual sound reflections off the walls, modeling sound propagation through objects, simulating air absorption, and rendering late diffuse reverberation, in addition to the 3-D positional rendering of source locations. This type of environmental spatialization is sometimes referred to as *auralization* or *virtual acoustics* [19], [20].

Virtual acoustics is a relatively new field of research that combines traditional acoustic-modeling techniques for sources, rooms, and listeners with the modeling of virtual environments. Audiovisual interaction is one of the important features of virtual acoustics—the aim is a virtual environment where auditory and visual events are related. Audiovisual objects change both their auditory and visual characteristics according to their position, orientation, materials, and visibility in a scene.

The audio approach to virtual environments [21] can be divided into three tasks: defining the virtual environment, modeling (real-time or non-real-time) the virtual sound process, and generating audio for presentation to the (real) listener. The sound-modeling process itself may be separated into source, environment, and virtual-listener models. This separation is intuitively well understood, since it is the basis of a normal communication chain (source-medium-receiver).

The *source model* in a virtual acoustic environment includes the sound content and the directivity properties of the emitter, which can be modeled efficiently using digital filters. The *environment model* aims at reproducing the effect of the surrounding space (listening room, concert hall, metro station, etc.). There are multiple approaches to this part. The most efficient are time-domain hybrid methods combining ray-tracing and image source method for direct sound and early reflections with late reverberation modeling based on statistical parameters [20], [22]. The *listener model* is closely related to the method of reproducing the auditory sensation. Different 3-D processing is needed for different types of reproduction such as headphone, stereophonic, and multichannel loudspeaker listening.

B. Physical-Modeling Extensions to AudioBIFS in MPEG-4 Version 2

In Version 1 of the MPEG-4 standard, as in the VRML standard, the virtual-reality sound-source model only provides techniques for placing the sound source in the 3D space and for coarse simulation of sound source directivity by the elliptical sound source patterns (Section II-C).

To improve this model, the spectral content of the sound should change as a function of distance. This occurs in natural environments because of the low-pass filtering effect of air absorption. Another improvement would enable more flexible simulation of the frequency-dependent radiation patterns of real sound sources. For example, a brass instrument has more high-frequency spectral content when listened to from the front as compared with behind. Finally, the sound source model in Version 1 AudioBIFS does not take into account effects of the environment and the medium. Among these are the Doppler effect caused by the coupling of the propagation delay of the sound to the relative movement of the sound source and the virtual listener, and the interaction (reflection, transmission, occlusion) of sound with objects in the medium.

For the second phase of the standard, three new nodes—**AcousticScene**, **AcousticMaterial**, and **DirectiveSound**—have been proposed for advanced auralization of audiovisual scenes [23] (Table III). With these new nodes it is possible to define geometrical regions in the scene where different acoustic responses are applied to sound according to the virtual locations of the sound source and the virtual listener. The **AcousticScene** and **DirectiveSound** nodes together allow the specification of properties of sound propagation and attenuation in the medium. The **AcousticMaterial** node gives visual and acoustic properties to polygonal surfaces. In the following, we illustrate how these nodes can be used to build up a region with an acoustic response.

1) AcousticScene: The **AcousticScene** node is used to govern an entire auralization process. As a child node of a **Group** node, it binds together acoustically relevant surfaces under that

TABLE III
ADDITIONAL AUDIO NODES IN MPEG-4 VERSION 2 AudioBIFS

Name	Function
AcousticScene	Group sounds together in an auralization process.
AcousticalMaterial	Specify reflection and transmission impulse responses for an object in a scene.
DirectiveSound	Specify frequency-dependent directivity modeling for a sound.

Group. **AcousticScene** has fields for defining a 3-D listening area; the viewpoint and the sound source must both lie in this area in order for the sound to be audible. It also has a field for specifying a frequency-dependent reverberation time that is used to add artificial reverberation to sounds.

The parent **Group** node of an **AcousticScene** may contain any BIFS nodes (audio or visual) in its children and children's subtrees. However, only polygonal surfaces that are defined with the **IndexedFaceSet** visual node may be given acoustic properties and taken into account in the auralization process. The **AcousticMaterial** node, below, is used to give the acoustic properties to the surfaces in the **AcousticScene**. The listening volume specified in an **AcousticScene** defines the outermost boundaries for the auralization, so that it encloses all the acoustic surfaces under the same **Group**. This enables the use of several areas with different acoustic responses, or "rooms," in the same BIFS scene. By keeping the rendering areas of different **AcousticScenes** apart, it is possible to restrict the complexity of sound processing to only one auralization process at a time.

2) AcousticMaterial: **AcousticMaterial** is a superset of the **Material** node that is used to give reflectivity and sound passing properties to surfaces that are defined in **IndexedFaceSet** nodes. **IndexedFaceSets** are used in visual BIFS to create polygons and 3-D objects with arbitrary shapes, and are therefore suitable for building up a room with reflecting walls that pass a portion of the sound energy through to the other side. Both the reflectivity and the sound transmission properties of the **AcousticMaterial** are given in a transfer function coefficient form, to enable frequency-dependent gain and efficient and scalable implementation.

When **AcousticMaterial** nodes are present in an AudioBIFS scene, the detailed acoustics can be described even for complex room configurations. The specular reflections at room boundaries are computed dynamically, and each sound reflection can be synthesized with the correct apparent direction and delay, according to the virtual positions of the sound source, the virtual listener, and the reflecting surface. Since the **AcousticScene** binds together the surfaces under the same auralization process, higher order reflections may be computed whenever there are enough computational resources. Implementation aspects of this process have been addressed in previous papers [21]. Fig. 4 shows a wire-frame model of a room built from acoustically reflective and partly transparent surfaces.

3) DirectiveSound: The **DirectiveSound** node enables the flexible definition of frequency-dependent directivity modeling of sound sources. It is an extension of the **Sound** node, and is used in the same manner, but with the addition of direction-dependent filtering. **DirectiveSound** nodes are only rendered when they lie within the same auralization region as the listener, as defined in an **AcousticScene**. As these sound sources and the listener move form one **AcousticScene** region to another, the change in the acoustics of the environment can be perceived. In addition to the **spatialize** field inherited from **Sound** node, **DirectiveSound** has another boolean field called **roomEffect** that enables sound processing according to the acoustic surfaces and the reverberation definitions. With this field set to FALSE, the effect of the acoustic environment is not rendered, and thus it is possible to have sources with low sound processing cost, but still with more advanced directivity and sound propagation properties than with the **Sound** node.

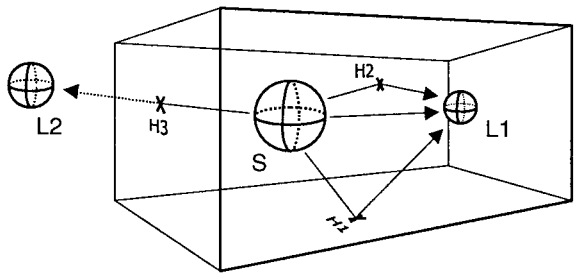

Fig. 4. Sound source **S** in a room with acoustically reflecting and partly transparent walls. At the virtual listener position **L1**, direct sound and reflections are perceived; the reflected sound is defined by the transfer functions **H1** and **H2**. Virtual listener **L2** only receives the direct sound filtered by the sound transmission filter defined for the obstructing wall; the sound is defined by the transfer function **H3**.

The **directivity** field of this node specifies the frequency-dependent gain as a function of angle between the listening point and the main direction axis of the sound source. It is given as a set of transfer functions. The directivity parameter can be given for an arbitrary set of angles, or whenever the virtual listener is between two angles with specified directivity filters. The distance-dependent attenuation is defined by a -60 dB attenuation distance, within which the sound is linearly attenuated in the decibel scale. It is not heard outside this distance. By setting the value of this field to zero, there is no attenuation, i.e., the sound level remains constant in the scene. Additionally, frequency-dependent air absorption (generally, increasing low-pass filtering as a function of distance) can be applied to the sound source by setting the value of a boolean field called **useAirabs** to TRUE. This gives a more natural feeling of the distance between the virtual source and virtual listener.

The **DirectiveSound** node also allows the content author to control the propagation speed of sound between the source

and the virtual listener with the **speedOfSound** field. This has significance when the sound reflects off surfaces, and the delays of the reflections are computed according to the length of the sound path and the speed of sound in the medium. When there is relative speed between the source and the virtual listener, Doppler effect is applied to the sound. The default value of the speed of sound is close to that of sound in air, i.e., 340 m/s, but can be changed for each sound individually if the strength of the Doppler effect or the delay of the reflections are to be exaggerated. By lowering the speed of sound, for example, the effective acoustic room size can be increased.

C. Perceptual Parameters in Audio Environment Modeling

Audio spatialization can also be approached from a nonphysical viewpoint, investigating the perception of spatial audio and room acoustical quality. This process is termed the *perceptual approach* to acoustic environment modeling.

Perceptual parameters have recently been introduced into the draft of MPEG-4 Version 2 as another method of creating environmental acoustic effects in the scene, independent of the visual (and physical) reality. These parameters enable the creation of environmental acoustic effects separately for each sound source, adjusted to characterize the perceptual quality of the source and the environment in a 3-D space. High-level perceptual parameters (such as source presence and brilliance, room reverberance, heaviness, liveness, envelopment) are used to derive low-level energy parameters for the control of direct sound, and the different parts of the room impulse response, i.e., the directional and diffuse early reflections, as well as the late reverberation [20], [22], [24]. The high-level parameters have been derived based on subjective testing of perceived room acoustical quality [22].

Based on these parameters, a real-time spatial sound processing scheme has been derived [24], which enables computationally efficient yet perceptually relevant 3-D audio rendering. The only input required from a geometrical representation of the acoustic space and its objects is the distance and orientation between the source and the virtual listener. The perceptual rendering engine does not need to utilize other geometrical knowledge of the acoustic space (wall positions, their reflection or transmission characteristics), because the static early reflection patterns and late reverberation decay are implicitly characterized by low-level parameters that have a fully specified translation from high-level perceptual parameters.

This approach is meant mainly for applications where the environmental response does not have to correspond to the visual environment, but where high-quality virtual room-acoustic effects are nevertheless desired. The perceptual parameters and processing are therefore also useful for audio-only postproduction in MPEG-4.

V. A Short Example

This section provides a short example to show how various AudioBIFS nodes interact. The sound scene in Fig. 5 synchronizes a synthetic music track with a voice-over that has an artificial reverberation applied to it. The resulting soundtrack

```
Group {                          // main node
  children [
    Sound {                      // top of AudioBIFS subgraph
      spatialize 0
      source [                   // only one source per Sound
        AudioMix {
          numChan 2              // a four-into-two mixdown
          phaseGroup [1 1]
          matrix [0.8 0.4 1.0 0.1 0.4 0.8 0.1 1.0]
          children [             // two children of Mix
            AudioSource {            // first child
              url "music"
              numChan 2
              phaseGroup [1 1]
              },
            AudioFX {                // second child
              orch "..."         // see Figure 6
              numChan 2          // turns mono into stereo
              children [
                AudioSource {
                  url "speech"
                  numChan 1
                  }
] } ] } ] } ] } ] }
```

Fig. 5. An AudioBIFS scene, represented in a textual format similar to VRML. This scene mixes two sound sources into a presentation. The first source is a stereo music sound; the second is a monophonic speech sound. The second sound is passed through a stereo reverberator before it is mixed with the first. A mixing matrix is provided with the **AudioMix** node to specify the relative levels of the stereo mixdown. The sound resulting from this mix is presented to the listener in a nonspatialized manner. Not all fields are shown for each node. In a real scene, this textual format is not used; rather, the BIFS data is conveyed in a compressed binary format.

(as well as other examples) can be downloaded from the MPEG-4 Structured and SNHC Audio web site, which is presently maintained at http://sound.media.mit.edu/mpeg4 by the first author.

There are a few simplifications made to this AudioBIFS scene for presentation. In a real scene, the **url** fields of the **AudioSource** nodes would contain indexes indicating which elementary stream to attach. Not all fields are shown for each node. Additionally, in a real MPEG-4 transmission, this textual format is not used; rather, the equivalent data is transmitted in a compressed binary representation.

The **orch** field of the **AudioFX** node contains the tokenized SAOL code of an effects-processing algorithm. One such algorithm is shown in Fig. 6. It implements the Schroeder reverberator [20], applying it to a mono signal in two decorrelated ways to produce a stereo result. As can be seen in this figure, it is easy to change the properties of the reverb in a wide variety of ways by changing the SAOL code. A full discussion of the capabilities of SAOL may be found elsewhere [14].

VI. Implementations

There are several BIFS and/or AudioBIFS implementation projects underway at the time of writing. “IM-1” is an MPEG-4 demonstration project showing systems capabilities in a real-time framework. A separate audio-only project has been undertaken to verify the audio multiplex and synchronization capabilities. This project is integrated with the SAOL reference software. Finally, several private industrial projects

```
global {
  outchannels 2;
   send(schroeder; 1; input_bus);
}

instr schroeder(rt) {
  // Schroeder reverb: rt = -60 dB ring time in sec
  asig ap1, ap2;
  asig c1, c2, c3, c4;
  asig outL, outR;

  ap1 = allpass(input[0], 0.0017, 0.7);
  ap2 = allpass(ap1, 0.005, 0.7);
  c1 = comb(ap2, 0.030, combgain(0.030, rt));
  c2 = comb(ap2, 0.0343, combgain(0.0343, rt));
  c3 = comb(ap2, 0.0393, combgain(0.0393, rt));
  c4 = comb(ap2, 0.045, combgain(0.045, rt));

  outL = (c1 + c2 + c3 + c4)/4;
  outR = (c1 - c2 + c3 - c4)/4;

  output(input + outL,input + outR);
}

opcode combgain(xsig t, xsig rt) {
  // calculate the feedback coeff to make comb filter
  // have desired ring time
  return(exp(log(10) * -3 * t / rt));
}
```

Fig. 6. SAOL **AudioFX** orchestra, for use with the scene graph in Fig. 5, that processes an input sound with the Schroeder reverberator [20]. The **input** bus, containing the speech sound output from the decoder, is passed on to the **schroeder** instrument. This instrument implements the desired reverberation algorithm, using comb filters and allpass filters as basic building blocks. An expanded description of SAOL can be found in other references [14].

are underway that will soon result in high-quality real-time MPEG-4 systems becoming widely available.

A. IM-1 Demonstration Software

IM-1 is the MPEG-4 Systems demonstration software implementation. It has been developed by an MPEG-4 working group created for this purpose. The aim of this project is to develop, integrate and demonstrate the Systems capabilities of Version 1 of the MPEG-4 standard. Features in Version 2 are currently being integrated into the IM1 software.

The IM-1 software is programmed in C++, and two versions of it exist, a 2-D player that relies on DirectX, and a 2-D/3-D player that is based on OpenGL. The sound capabilities provided in this system are those enabled by the **Sound** and **AudioSource** nodes.

B. Audio/Systems Verification

To examine the detailed interaction between the audio coders and the multiplex and compositing systems, an "audio/systems integration" project was undertaken. This project resulted in the construction of a non-real-time multiple-audio-codec decoder/compositor that does not run fully automatically, but nonetheless was extremely valuable in proving the concepts of the system. It is currently being extended to provide complete and integrated MPEG-4 audio decoding and playback capabilities.

This system implements the **AudioMix, AudioSwitch, AudioDelay, AudioSource,** and **AudioFX** nodes. A SAOL system (see below) is integrated to provide full AudioFX capability. High-quality sample-rate conversion is also included.

In operation, a demultiplexer and the natural audio decoding tools are executed independently to produce "composition buffers" in disk files that contain the PCM output of the decoders for use by the **AudioSource** nodes. An integrated audio compositor/synthesizer executes the structured audio decoding and simultaneously composites the natural and synthetic outputs together according to the AudioBIFS instructions.

C. SAOL Reference Software

The MIT Media Laboratory has implemented the entire SAOL specification in a non-real-time reference software implementation. This source code is freely available (in the public domain) and is suitable for exploring both structured audio techniques and the capabilities of the AudioFX node. This implementation and many sample synthesis and effects-processing algorithms are available from the SA home page at http://sound.media.mit.edu/mpeg4.

VII. Conclusion

We have described AudioBIFS, the MPEG-4 standard for effects processing and audio scene description. AudioBIFS is a powerful, flexible format that serves the needs of virtual-world builders and traditional media developers alike.

By using the capabilities of MPEG-4 AudioBIFS and the other MPEG-4 Audio tools, a great many new types of content become available to the multimedia author. Future research in this area will include the development of efficient implementations for playing back synthetic/natural hybrid audio content, and new types of authoring tools to enable its efficient creation. Finally, there are many intriguing unsolved problems in the area of automatically creating hybrid and object-based soundtracks automatically from digital audio input. The MPEG-4 standard provides a single representation format in which to conduct such experiments in new encoding technologies.

It is important to note that this paper does not itself represent a standard or the views of the standardization body ISO/IEC JTC1/SC29/WG11, but only the opinions of three individuals involved in technical aspects of the standardization process. Certain elements described herein, particularly those pertaining to Version 2 of MPEG-4, may change between the time of this writing and final standardization. The MPEG process uses the open-standards model; suggestions for improvement are welcome from any party at any time.[3]

References

[1] *Coding of Multimedia Objects (MPEG-4)*,ISO/IEC 14496:1999, 1999.
[2] R. Koenen, "MPEG-4: Multimedia for our time," *IEEE Spectrum*, vol. 36, no. 2, pp. 26–33, 1999.
[3] *Virtual Reality Modeling Language (VRML)* ISO 14472-1:1997, 1997.

[3]The tools and techniques in Version 1 of MPEG-4 AudioBIFS have been donated to ISO and the audio community by their developers, who maintain no patent rights or proprietary control over the technical content. Patent-free status for the AudioBIFS tools has been maintained in the hope that acceptance and implementation of the most advanced audio tools in the MPEG-4 standard will help to drive forward the marketplace for advanced digital-audio technology on personal computers.

[4] S. R. Quackenbush, "Coding of natural audio in MPEG-4," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Seattle, WA, 1997, pp. 3797–3800.
[5] J. Johnston, S. R. Quackenbush, and J. Herre, "MPEG-4 natural audio," in *Advances in Multimedia: Systems, Standards, and Networks*, A. Puri and T. Chen, Eds. New York: Marcel Dekker, in press.
[6] E. D. Scheirer, "The MPEG-4 Structured Audio standard," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Seattle, 1998, pp. 3801–3804.
[7] E. D. Scheirer, Y. Lee, and J.-W. Yang, "Synthetic audio and SNHC audio in MPEG-4," in *Advances in Multimedia: Systems, Standards, and Networks*, A. Puri and T. Chen, Eds. New York: Marcel Dekker, 1999.
[8] G. A. Soulodre, T. Grusec, M. Lavoie, and L. Thibault, "Subjective evaluation of state-of-the-art two-channel audio codecs," *J. Audio Eng. Soc.*, vol. 46, no. 3, pp. 164–177, 1998.
[9] N. Jayant, J. Johnston, and R. Safranek, "Signal compression based on models of human perception," *Proc. IEEE*, vol. 81, pp. 1385–1422, Oct. 1993.
[10] M. Bosi, K. Brandenburg, S. Quackenbush, L. Fielder, K. Akagiri, H. Fuchs, M. Dietz, J. Herre, G. Davidson, and Y. Oikawa, "ISO/IEC MPEG-2 advanced audio coding," *J. Audio Eng. Soc.*, vol. 45, no. 10, pp. 789–814, 1997.
[11] B. S. Atal and M. R. Schroeder, "Predictive coding of speech signals and subjective error criteria," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, pp. 247–254, Mar. 1979.
[12] A. Gersho, "Advances in speech and audio compression," *Proc. IEEE*, vol. 82, pp. 900–918, June 1994.
[13] M. Nishiguchi and J. Matsumoto, "Harmonic and noise coding of LPC residuals with classified vector quantization," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Detroit, 1995, pp. 484–487.
[14] E. D. Scheirer and B. L. Vercoe, "SAOL: The MPEG-4 structured audio orchestra language," *Comput. Music J.*, vol. 23, no. 2, pp. 31–51, 1999.
[15] B. L. Vercoe, W. G. Gardner, and E. D. Scheirer, "Structured audio: The creation, transmission, and rendering of parametric sound representations," *Proc. IEEE*, vol. 85, pp. 922–940, May 1998.
[16] E. D. Scheirer and L. Ray, "Algorithmic and wavetable synthesis in the MPEG-4 multimedia standard," in *Proc. 105th Conv. Audio Engineering Society*, (reprint 4811), San Francisco, CA, 1998.
[17] E. D. Scheirer, "Structured audio and effects processing in the MPEG-4 multimedia standard," *Multimedia Syst.*, vol. 7, no. 1, pp. 11–22, 1999.
[18] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley, 1979.
[19] M. Kleiner, B.-I. Dalenbäck, and P. Svensson, "Auralization—An overview," *J. Audio Eng. Soc.*, vol. 41, no. 11, pp. 861–875, 1993.
[20] W. G. Gardner, "Reverberation algorithms," in *Applications of Digital Signal Processing to Audio and Acoustics*, M. Kahrs and K. Brandenburg, Eds. New York: Kluwer, 1998, pp. 85–132.
[21] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen, "Virtual environment simulation: Advances in the DIVA project," in *Proc. Int. Conf. Auditory Display*, Palo Alto, CA, USA, 1997, pp. 43–46.
[22] J.-P. Jullien, "Structured model for the representation and the control of room acoustical quality," in *Proc. 15th Int. Congr. Acoustics*, Trondheim, NO, 1995, pp. 517–520.
[23] V. Swaminathan, R. Väänänen, G. Fernando, D. Singer, and W. Belknap, "MPEG-4 Systems Version 2 Committee Draft," Document W2739. Seoul: ISO/IEC JTC1 SC29 WG11 (MPEG), 1999.
[24] J.-M. Jot, "Efficient models for reverberation and distance rendering in computer music and virtual audio reality," in *Proc. Int. Computer Music Conf.*, Thessoloniki, Greece, 1997, pp. 236–243.

Eric D. Scheirer (S'97) was born in Binghamton, NY, in 1971. He received the M.S. degree from the Media Laboratory, Massachusetts Institute of Technology, Cambridge, in 1995 for his work on constrained automatic musical transcription systems. He is currently pursuing the Ph.D. degree in the Machine Listening Group at the Media Lab, where his research focuses on the construction of musically intelligent agents.

He was the principal author of the MPEG-4 Structured Audio and MPEG-4 AudioBIFS specifications and served as an Editor of the MPEG-4 Audio standard. He has a wide range of interests in the application of computing technology to audio and multimedia systems and has published numerous articles on psychoacoustics, musical signal processing, and structured-audio coding.

Mr. Scheirer is a student member of the AES.

Riitta Väänänen was born in Helsinki, Finland, in 1970. She received the M.Sc. degree in electrical engineering from the Helsinki University of Technology (HUT), Helsinki, Finland, in 1997. She is currently pursuing the Ph.D. degree in acoustics and audio signal processing at HUT.

She has worked as a Research Assistant and a Research Scientist at the Laboratory of Acoustics and Audio Signal Processing, HUT, since 1996. Her research activities include room reverberation modeling and modeling of sound sources and acoustic environments in interactive virtual reality systems.

Jyri Huopaniemi (M'94) was born in Helsinki, Finland, in 1968. He studied acoustics and audio signal processing, multimedia , and information technology at Helsinki University of Technology (HUT), Helsinki, Finland, and received the M.Sc. and Lic. Tech. degrees in electrical engineering in 1995 and 1997, respectively. His doctoral thesis (to be published in 1999) is on the topic of virtual acoustics and 3-D sound.

He worked as a Research Scientist at the Laboratory of Acoustics and Audio Signal Processing, HUT, from 1993 until 1998. In 1998, he was a Visiting Scholar at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University, Stanford, CA. He is currently with Nokia Research Center's Speech and Audio Systems Laboratory, Helsinki. His research activities include 3-D sound and auralization, virtual audiovisual environments, digital audio signal processing, psychoacoustics, room acoustics, and musical acoustics. He is author or coauthor of over 35 technical papers published in international journals and conferences and has been actively involved in the MPEG-4 standardization work.

Mr. Huopaniemi is a member of the AES and was secretary and member of committee of the AES Finnish Section from 1995 to He is also a member of the International Computer Music Association and the Acoustical Society of Finland.