AN EFFICIENT HOST/CO-PROCESSOR SOLUTION FOR MPEG-4 AUDIO COMPOSITION

L. Le Bourhis, G. Zoia, M. Mattavelli, D. J. Mlynek

**Swiss Federal Institute of Technology, Integrated Systems Center C3I,
CH-1015 Lausanne Switzerland.**

Abstract

This paper presents an efficient software host/co-processor architecture for the implementation of MPEG-4 Audio composition. The proposed solution is based on a specific partition between general-purpose tasks and DSP-oriented functionality, thus achieving portability, efficient partitioning of the processing resources and memory management.

Introduction

The new MPEG-4 Systems standard [1] provides extended capabilities for audio processing and reproduction. The possibilities offered by the so-called BIFS format for scenes, are much more powerful and efficient than those provided by similar multimedia languages such as VRML [2,3]. The MPEG-4 AudioBIFS provides mechanisms to perform mixing, delay, 3-D spatial processing and other various effects that can greatly enhance the perceived audio experience delivered by high-quality coding tools such as MPEG-2 AAC. However, these new powerful and flexible ways of handling audio rendering require a large amount of processing resources and present new implementation challenges in comparison with the classical audio coding schemes. Moreover, the complexity of audio scenes structures requires appropriate strategies for resource partitioning, memory management and an efficient implementation of the various processing nodes [4].

This paper presents an architecture for the implementation of an MPEG-4 Systems Audio Compositor aiming to achieve:

- portability on different host/co-processor platforms
- frame based processing
- flexibility for the integration of other additional processing nodes.

The results presented in this paper summarize the extensive studies and recent achievements of the authors concerning the implementation of MPEG-4 Systems Audio nodes [4,5].

An approach towards an efficient "MPEG-4 Audio Systems" implementation

The appropriate definition and the optimization of software architectural features become an issue of paramount importance in MPEG-4 Audio Systems. The complexity of actual audio scenes and the amount of processing resources required are not anymore a negligible implementation aspect for a multimedia system. Major problems that need to be solved are: sampling rate conversion, synchronization among nodes, composition buffer structures, memory management and the design of a flexible architecture for the integration of spatial processing features [4].

First of all, scene composition allows the presence of different coded audio objects at different sampling rates. Algorithms typically used for high-quality and precise sampling rate conversion necessitate large amounts of processing power. A virtual node dedicated to this conversion processing has been introduced. This node is based on a three-stage asynchronous sampling rate converter and used for all possible conversions in an MPEG-4 audio scene. In fact, the usage of this converter node, along with a frame-based processing, permits to get rid of memory buffers between nodes, reducing both memory usage and system complexity. Looking at the scene description and analyzing the different AudioSource sampling rates, we can find the optimal placements for these conversion nodes.

A goal of the described implementation is to develop a "frame-based" compositor model. A fixed frame-based model can work on a constant time-window, thus providing a "deterministic" behavior that greatly eases system synchronization issues, which can be very problematic in audio composition of complex scenes yielded by "MPEG-4 Systems" environments. The first idea would be to send the input samples as far as possible directly into the Audio subtree. Problems arise when two nodes have the same parents but have a different number of available samples: the remaining samples need to be buffered in the nodes until new samples come. The objective of the proposed "frame-based" compositor model is to minimize buffer usage by introducing them only on a temporary basis, thus avoiding buffers between nodes. This means that memory is allocated only in specific conditions and for each single execution cycle. To address this issue, we must ensure that all the samples sent to the audio tree will be consumed in a single cycle: this implies that samples must be framed at the input according to their sampling rate.

Another key-point in an efficient implementation is the interaction among the audio decoders and the

composition buffers. In an MPEG-4 audio scene, decoders feed the audio tree with composition buffers, and the associated corresponding time information to perform temporal synchronization. The compositor is in charge of managing these buffers to create the complete scene. Structures associated to buffers are necessary to handle the samples coming from the different decoders and to provide mechanisms to identify buffers associated to the several AudioSource nodes. The basic idea is to associate a bufferId with each output channel of a decoder, and to provide these Ids to the AudioSource nodes so that they can target the correct buffers.

In order to ensure a synchronized "frame-based" processing, composition time stamps are analyzed with additional parameters such as StartTime and StopTime fields. Results of this analysis permit to build valid frames to be used with our fixed length "frame-based" model.

Information storage and recovery for buffers and nodes are also crucial issues for an efficient implementation of an MPEG-4 compositor. Scene updates by user interaction or scripting languages need an open and flexible compositor architecture. Thus, a software node architecture, common to all processing nodes, has been developed. This architecture successfully ease children information's gathering, and facilitate sample progressions through the tree structure that represents the audio scene. A flat memory model is used for every data storage: all data, either static or coming from intermediate processing nodes, are stored in one-dimensional tables and are accessed by offset pointers. This memory structure has been designed to satisfy the constraint of portability of the software architecture to most DSP platforms.

To further reduce memory usage, it is useful to avoid sending empty frames (filled with zeroes) through the tree. For instance, if an AudioSource reaches its stop time, there is no need to create a frame with zeroes. To solve this problem, each channel has an activity flag associated with it. When normal processing occurs, this flag is on, whereas if no valid information is available, it is set to off. Proper handling of this flag in the subsequent node allows saving relevant processing time; there is no need to perform a mixing operation between two channels of zeroes. Some nodes can be reluctant to this, as it is the case for AudioDelay nodes. When its child channel has no activity, it still has to put zeroes in its delay line. At its output a detection mechanism (looking if the output frame is empty) can allow to get the off status back and to avoid propagating again an empty frame.

Once the complete software architecture of the compositor is defined, any specific complex audio rendering algorithm can be implemented by simply inserting the suitable processing algorithm in the compositor architecture. For instance very complex and sophisticated algorithms such as mixing and 3-D spatialisation can be easily integrated in the proposed architecture.

Implementation and results

A platform independent model for MPEG-4 audio composition has been successfully developed. The use of a dedicated virtual sampling rate conversion node enables the simplification of nodes interconnection problems and allows building a "frame-based" compositor. A fixed compositor frame length of 10 ms has been chosen for the actual implementation. An efficient 3-stage asynchronous converter has been used for all typical audio sample rate conversions. Specific mechanisms such as click-free switch and mix for parameters update have also been implemented. The final step of the platform independent implementation was the definition of a host/co-processor communication protocol. Data transfers between these independent devices can be established by command methods, such as sending buffer frames, adding nodes to the scene or updating parameters. Experiments of MPEG-4 audio composition using a GPP/DSP configuration have been successfully conducted using a Pentium/Motorola DSP56301 platform. These results validate the proposed software architecture and the flexible integration of various processing nodes.

Conclusion and further work

We have outlined in this paper an efficient software host/co-processor architecture for the new MPEG-4 Audio composition. This solution is based on a careful partition between general-purpose tasks and DSP-oriented functionality; main goals for this implementation were portability, efficient partitioning of the processing resources and memory management.

Further work to be done is mainly in two directions: a complete integration of Structured Audio for downloadable processing algorithms and the extension towards advenced spatial capabilities proposed by version 2 of MPEG-4.

References

[1] A. Eleftheriadis, C. Herpel, G. Rajan and L.Ward, Editors: ISO/IEC SC29WG11 Document No. 14496-1 (MPEG-4 Systems) Draft International Standard. MPEG 1998.

[2] Official Draft #3, ISO/IEC SC29WG11 Document No. 14772: The Virtual Reality Modeling Language Specification. July 1996.

[3] E. Scheirer, R. Väänänen, J. Huopaniemi: AudioBIFS: the MPEG-4 Standard for Effects Processing. Proceedings of the COST-G6 Workshop on Digital Audio Effects Processing (DAFX '98), Barcelona, November 1998.

[4] G.Zoia, L.LeBourhis, U.Horbach and A.Karamustafaoglu: Proposed revision of Systems and Audio profiles and levels from an analysis of audio composition. MPEG98, Document M3604, Dublin - July 1998.

[5] L.LeBourhis, G.Zoia: About AudioBIFS level definitions. MPEG98, Document M4111, Atlantic City - October 1998.