AUDIENCE for Pd, a scene-oriented library for spatial audio

Regis Rossi A. FARIA
Audio Engineering and Coding Center at LSI-EPUSP and Music Department at FFCLRP-USP
University of São Paulo
Av. Prof. Luciano Gualberto, tv.3, n.158
São Paulo, Brazil, 05508-010
regis@usp.br

Abstract

AUDIENCE for Pd is a library for audio immersion, sound scene production and auralization. The software enables to work independently the sound scene composition, its acoustic rendering, the encoding of its sound field in a suitable multichannel format, and finally its audition using 2D/3D loudspeaker modes. The system architecture is scalable, many scenes and sound sources can be concurrently processed, and it is possible to interactively place and move objects in the scenes in real time. This paper presents the system architecture, design premisses, and the software implementation. The library structure is explained, as well as how to use it, giving application examples. Ongoing developments and future perspectives conclude the article.

Keywords

spatial audio, sound scene orientation, sound encoding, auralization.

1 Introduction

The AUDIENCE for Pd (Audience4Pd) is a software library made of Pure Data (Pd) objects (externals) and abstractions (patches) conceived to build general spatial audio applications. A major feature is in the sound scene conception independence of the listening mode, once the user can work out a desired sound scene spatial experience regardless of the loudspeaker configuration used to listen. The final sound field can be exported and played back in stereo and surround output formats.

Its development started back in 2005 and, together with some customized hardware for multichannel audio distribution and amplifier designs, it constitutes the main result of the namesake project "AUDIENCE – AUDio Immersion ExperieNce by Computer Emulation"[1].

Since then, many contributors have been involved in software development, testing concepts, designs and applications, reaching the current version 2.0.2 released in two different distributions: a full and restricted version (AUDIENCE), and a basic and free version (OpenAUDIENCE).

1 www.lsi.usp.br/interativos/neac/audience/

This paper introduces the system architecture, and the design and implementation concepts behind its development, explaining how it works, implemented features, and giving examples of applications built with it. The main components of the current library version are presented, and it concludes with a roadmap of new developments towards the next generation.

2 System architecture

The AUDIENCE system architecture is based on the concept of *decoupled functional layers*. *Simplicity*, *layers* and *interfacing* are keywords to understand this architecture.

The first proposal of the AUDIENCE architecture was published in 2005 [1], and since then it has been detailed and further consolidated in later published papers [2][3][4] [5]. Many practical applications were valuable for validating the concepts and distinguishing features.

This section is an overview of the architecture, explaining its core and key concepts, such as the functional layer and interfaces, showing how the system architecture is devised.

2.1 Functional layers

The core of the architecture in on the functional layers, which can be viewed as clusters of related functions. There are four main audio processing layers in the architecture, that are mandatory for any spatial production-reproduction chain. They are:

- L1: scene composing and description layer, resolving the auditory scene
- L2: scene acoustic model rendering or simulation, using some acoustics method
- L3: spatial audio coding, packaging audio data (with metadata or not) into a specific format for distribution or storage
- L4: spatial audio reproduction, decoding and generating multichannel output for a

given final audition configuration or surround loudspeaker mode.

These layers constitute *the core* of the architecture. Besides the functions within them, there are also other functions that fit in what can be called *auxiliary* (or service) layers. These implement functionalities that, while not directly connected to the spatial audio processing tasks, are required to link the core to running applications and devices. Usually they are platform-dependent and shall not interfere in the subject dealt by the main audio processing chain. They are:

- *communication interfaces,* implementing communication mechanisms and input/output (I/O) ports for the transmission and reception of audio data and metadata to/from the main layers.
- *user interface/control*, implementing the *GUI* blocks that command the main core. It is responsible for admitting control and interactive actions from the user/listener and sending commands to the main layers. However, if GUI blocks embed L1 functionalities they will get a L1 status.

Figure 1 shows the architecture hierarchy of layers.

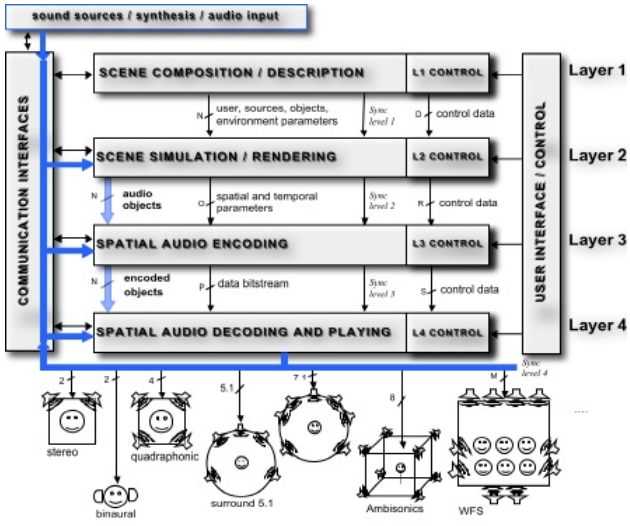

Figure 1 – AUDIENCE system reference architecture

Notice that the architecture proposes a hierarchy of layers, where L1 is the highest semantic level and L4 is the lowest one. Synchronization and other data rate increase from L1 to L4. Observe also that there are different classes of information in the interfaces between layers, among metadata, audio data, sync signals and control data. Figure 1 suggests a top-down data flow in the chain, from layer *i* to layer *i+1*. In some circumstances one processing chain may feed input to another chain and then it happens that a layer *i* will feed a layer *j*, where $i, j \in \{1, 2, 3, 4\}$. A practical sequence encompasses functions in different stages, and these data naturally include audio payload, metadata, control parameters and sync signals.

The layers must be independent from each other, and then shall have a self-contained and sufficient set of working rules and algorithms to control its behavior, e.g. algorithms to act in case of audio clipping, invalid data range, etc. Inputs coming from graphical user interfaces (GUIs) provide control to all layers. GUI's can deliver specific commands to control functions and behaviors in all layers.

Functions in a layer can deliver messages, data and instructions to other layers using some sort of communication interface, which is shown in a generalized form in the figure 1. Audio data (shown in blue color) can flow directly from L2 to L3 and from L3 to L4. Audio data flowing from L4 to other layers is indicated by the feedback of audio from L4 into the audio interfaces.

The next section presents the design and implementation of the library according to this architecture.

3 System design and implementation

This sections presents the software design premisses, the development history, the library organization, its features and objects, and how it was conceived to work.

3.1 Design premisses

A practical realization of a software for spatial audio based on the AUDIENCE architecture was advantageous to be developed over Pd due to the several reasons: (a) flexible programming platform for prototyping audio processing functions in the form of blocks that can be connected through cords, (b) real time operation, (c) portability, and (d) free access.

An auralization software will execute several functions that appropriately fit into one of the architecture layers. The main requirement was to map all necessary functions in every layer to individual objects in Pd, so as the data flow could be established by connecting them.

To build an AUDIENCE functional object in Pd some requirements shall be observed. First, the definition of its target layer. Each object implements one functionality in one and just one of the layers, i.e. every object must *belong to one valid layer* among the main 1, 2, 3 or 4 layer, or to the auxiliary layer. To permit an immediate identification of the layer it belongs to all blocks are named according to the

following syntax:

audce+_+layer+_+functionName

Second, the design of its interface and core process: the function(s) that it execute will use input data/audio to produce output data/audio. Data can be supplied in several ways but mainly through argument creation or passed in messages to the block. Output data can be piped to outlets. A general rule is that a layer *i* block shall accept layer *i-1* outputs and produce inputs signals to the layer *i+1*.

The third requirement is its compliance to the layer message set: an object may use defined parameters, attributes of scene objects, which are used or addressed by other blocks in the scene patch. For example, L1 blocks may change the user position and need to propagate this new position using a defined syntax. AUDIENCE has a mechanism to update the data values using *message passing* that can be propagated using a communication send/receive mechanism, the simplest one being a data connection from one block outlet to a block inlet. An object in layer *i* must be able to receive messages with data to this layer, and to interpret them. Usually messages have the name of the parameters itself and are followed by its new values.

The accompanying software documentation carries a list of implemented messages than can be used to build inter-object communication mechanisms (e.g. using direct connections with inlets/outlets, send/receive Pd messages, or UTP messages).

Finally, the implementation form must be defined. It can be built as a *patch* block (*abstraction*) using the Pd built-in objects, or built as an *external*, linking a function written in C/C++.

3.2 Development history

The AUDIENCE software has been developed since 2005 at the Audio Engineering and Coding Center (NEAC) at the Polytechnic School of the University of São Paulo, and been used in several scientific and artistic projects since then.

The first objects were designed and built during the AUDIENCE project. Aiming at flexible and scalable immersive audio tools (hardware and software) for applications in complete virtual reality (sound and visual immersion) this project was concluded in 2005 delivering an octophonic speaker system for a CAVE with an auralization engine capable of receiving scene data from immersive visual applications and rendering the sound scenes using Ambisonics to encode 2D/3D sound fields and decode them to a cubic rig. Several experiments with virtual auditory scenes with musical instruments and planar rigs for audition were also implemented which led to extending the number of software objects and application-specific patches.

This development naturally opened room for a second round or investigations, as the architecture and auralization engine had shown potential to solve problems in a more general scope related to the design of sound systems, and culminated with a second research project: the OpenAUDIENCE. Starting in 2006 this project led to the establishment of guidelines to extend the library by creating new components open to the community, and led to the consolidation of a free version of the software, bringing also resources for programming.

The current software library is a direct result of these development projects. In the first versions it worked with 2, 4, 6 and 8 loudspeakers setups running Ambisonics and incorporated a simple GUI for L1 scene creation and control. The perceptual spatial impressions were quite good for open-field virtual worlds and got improved as layer-2 processes were implemented, such as an image-source L2-algorithm [3]. Progressively new methods and techniques were incorporated and presently the full version works with Ambisonics (up to 3rd order), multichannel PCM, MPEG-4 AAC, MPEG Surround, and further blocks are on the way.

The OpenAUDIENCE (OA) is a free distribution of the software library, based on the full version. The main difference between them is the existence of non-free (proprietary) objects, libraries and source codes included only in the full version.

While the full-version library AUDIENCE is available to the team of developers and researchers involved in development projects, the OA version can be downloaded freely from its project website[2]. This project is open to any contributors from both technical and artistic backgrounds willing to create new applications or develop new objects (functions).

3.3 Library organization

The AUDIENCE for Pd is a collection of functional components in the form of Pd objects and patches. A functional component in this

2 Formerly hosted at recently deactivated FAPESP incubator at http://openaudience.incubadora.fapesp.br, a website migration is planned soon.

library is simply a processing block built as a self-contained Pd object or a patch.

The blocks are sorted in 4 functional layers plus an auxiliary one, running on top of Pd. The figure 2 shows the software stack.

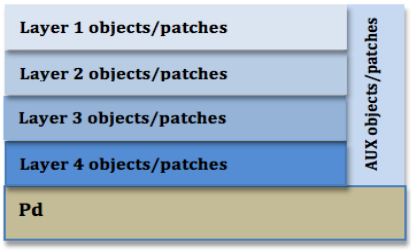

Figure 2 – Software stack

The current full library structure is organized in 12 directories:

- *app*: applications and demos directories
- *aux*: auxiliary objects[3]
- *bin*: extra binaries necessary in some applications
- *doc*: documentation
- *img*: images used in interfaces
- *include*: include files for compiling/building
- *L1*: layer 1 objects and abstractions
- *L2*: layer 2 objects and abstractions
- *L3*: layer 3 objects and abstractions
- L4: layer 4 objects and abstractions
- *lib*: third-party libraries
- *snd*: sounds
- *src*: source files
- *tst*: test patches

The full distribution is aimed to developers and licensees, as it includes all components. The OA distribution comes with basic applications in */app* (examples) and, while it may include {*src,lib,bin}*, it excludes all restricted components. A minimum executable core needs only {*L1,L2,L3,L4,aux*} to run.

In the subdirectory */tst* one will find useful test patches to verify the status of many functionalities. For instance, the patch *test_audience_volume_patches* summarizes all volume abstractions available in the current version.

An included *readme.txt* file summarizes few words on how the software works and information about the directory structure, development guidelines, installation and operational notes, a documentation guide, and a version reference list (release notes). All external objects are compiled to Win32 (.dll), MAC OSX (.pd_darwin) and Linux (.pd_linux).

3 For the Win32 environment this directory is named *auxiliary*, as this OS does not permit the creation of any directory *aux* in its file system.

3.4 Features and library components

Table 1 lists the main functional objects and abstractions implemented in AUDIENCE for Pd version 2.0.2. For each block it is presented (a) its layer group (L1, L2, L3, L4 or Auxiliary), (b) the name of the object or abstraction in Pd, and (c) a short description.

The full distribution version 2.0.2 supports a scalable interactive sound scene graphical user interface (GUI), Ambisonics (up to the 3rd order) and MPEG-4 AAC codecs, image-source-based acoustic simulation, multichannel recording and playing (including 5.1) and it can decode to 11 different speaker layouts.

The basic objects described in table 1 will process audio and information for one sound source in scene. To easy the processing of several sources at once there are abstractions in the distribution combining several basic objects and extending the functionality to a set of *N* sound objects (e.g 3, 5 or 8 sources). For example, to encode and mix at once 3 sources in 1st order Ambisonics one can use the abstraction *audce_L3_3obj_amb1st~*. Extension abstractions are listed and available inside their layer's subdirectory in the distribution.

A number of objects and abstractions are included in the auxiliary layer with resources for essential functions not belonging to the 4 main layers, such as data formatting, graphical user interfaces, arithmetic and DSP operations, mixing operations, data I/O, timing and scheduling, synchronization, transport and signal switching. They are helpful to build new abstractions and applications patches, and to set and control parameters.

Auxiliary objects include multichannel audio file recorders, data format converters (e.g. time-to-samples), audio format converters (e.g. 6 audio-signal to 5.1-file), mixers, convolvers, and abstractions for building control interfaces for transport, volume (loudness) and time flow.

Some alternative abstractions implement useful variations in slider scale excursion style (e.g. logarithmic or linear) and in working ranges (e.g. min/max linear and dB gain values). For example the *audce_aux_objvol1~* is a volume control with VU meter with gain in the range [-100, +12] dB. The volume abstractions have inlets to receive on/off commands and input level setup from outside.

Layer	Object	Description
L1	audce_L1_gui	Scene GUI for positioning sound sources (sound objects)
L2	audce_L2_allen	Image source acoustic simulator for one source, outputs an Ambisonics Impulse Response (B-IR)
L2	audce_L2_RMgen	Rendering Matrix generator for n sound objects w/ dedicated output per each one
L3	audce_L3_amb_3rd~	Ambisonics encoder up to 3rd order
L3	audce_L3_aacenc	MPEG-4 AAC encoder
L3	audce_L3_5.1render	5.1 mode audio program generator using the metadata produced by L2 Rendering Matrix
L3	audce_L3_ambirconv~	Ambisonic Impulse Response (B-IR) set convolver for one sound source, outputs B-Format
L4	audce_L4_amb_3rd~	Ambisonics decoder up to 3rd order
L4	audce_L4_aacdec	MPEG-4 AAC decoder
L4	audce_L4_amb_rot~	Object for rotating ambisonics scene (B-Format field) around reference axes
L4	audce_L4_objplay~	One source sound file player, abstraction interface w/ file_open + transport
L4	audce_L4_objplay1~	One source sound file player, interface w/ file_open + transport + time counter
L4	audce_L4_objplay2~	One source sound file player, interface w/ file_open + transport + time counter + volume setup
Aux	audce_aux_convolver~	External to stream-like convolve one audio signal with a Impulse Response in real time
Aux	audce_aux_counter	Abstraction providing a time meter in minutes and partials
Aux	audce_aux_dbvol~	Tiny patch for volume control in dB
Aux	audce_aux_mixer~	Mixer for n-signals (channels) with individual level setup via message
Aux	audce_aux_recordmc~	Abstract GUI to record a multichannel sound file w/ setup for no. of channels (up to 6) and SR
Aux	audce_aux_playmcfile~	Abstract GUI to play a multichannel sound file w/ setup for no. of channels (up to 6) and SR
Aux	audce_aux_move	Experimental abstraction to provide automation for sound object motion in sound scene
Aux	audce_aux_objtime	Patch to control time of a sound object in the sound scene in the main patch
Aux	audce_aux_vol~	Patch to control volume of a sound file or audio line
Aux	audce_aux_objvol~	Patch to control volume of a sound file or audio line with VU meter

Table 1: Main objects/abstractions available in the AUDIENCE for Pd version 2.0.2

Also, there are a number of default messages used to convey commands and to set parameters in blocks in a patch. Simplicity is a rule for defining syntax style, such as in this example: "*src_pos n x y z*" is a message for positioning sound object *n* at the cartesian coordinates {*x,y,z*} in the scene, and similarly "*rcv_pos x y z*" positions the listener in the scene.

Most objects have *help patches (files)* containing information about its creation arguments, the data it holds (e.g. sound object position in space, order of Ambisonics, etc.), usage examples, the I/O interface signals (i.e. the valid signals that are accepted in inlets and signals available in the outlets), known bugs and limitations. They also include a version history (release notice) and links to affine objects ("see also").

3.4.1 Licenses

AUDIENCE and OA are released under a series of different licenses that apply on a *object-usage basis*. Most objects/abstractions .pd files and/or help files include information about version release and licenses that may apply, as they individually use different resources available in the Pd extended version, in proprietary and/or specific third-party objects.

Native AUDIENCE objects are BSD licensed, and where third-party libraries or GPL software are used, their terms are applicable and acknowledged. Third-parties' objects may be GPL-licensed or adopt other scheme. Individual files must be consulted for individual licensing conditions as well as the accompanying manual, with detailed information on this matter. Some objects, such as the convolvers, for example, use the GPL FFTW3 library from Matteo Frigo, and L3/L4 AAC codecs use aacPlus© licensed from Coding Technologies (later on Dolby).

3.4.2 Creating new objects

Creating new AUDIENCE externals will require a C/C++ development environment and access to the source files, headers and the *makefile*. The externals' source files and the *makefile* lie into the /*src* directory. Header files are inside the /*include* directory. The *makefile* script holds compilation and building instructions for the compiler and it is meant for use with all supported OS's.

The accompanying documentation has a section instructing interested developers on how to build new AUDIENCE objects and testing them.

3.5 How it works

A practical spatial audio application is built creating data flows by instantiating and connecting objects. Figure 3 shows an example of a typical top-down data flow to illustrate the processing rationale.

It shows a patch using "generic" objects per layer generating the combined sound field of 3 sound sources irradiating in a sound scene, and generating an audition in 5.1 surround mode. In the top layer 1 there is a scene data holder, which parses data to the subsequent processing objects in layer 2. Acoustic simulators in layer 2 can produce, for instance, impulse responses in B-Format and pass them to the spatial-temporal encoders in layer 3. These spatial encoders take streaming audio data from each sound source (src1, src2 and src3) and generate their individual sound fields as listened in a specific position in the scene (the receiver position).

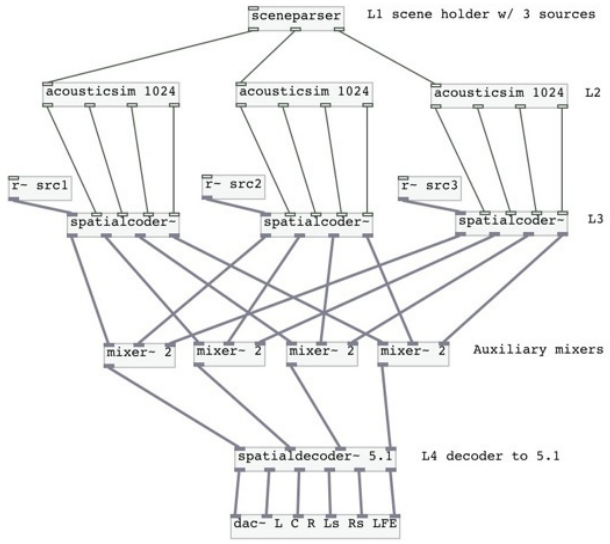

Figure 3 – Generic top-down data flow

As we are interested in the complete scene sound, the 3 individual sound-field data sets are mixed (one mixer for each sound-field channel) and the result is the distribution output program (e.g. in B-Format).

The last component, the layer 4 decoder, could be local or remote, and will be able to decode the distribution media to an specific output loudspeaker mode. This operation reconstructs the sound field of the scene, agnostically from what has been done in the other layers.

Many pipelines of similar processes can be created in parallel and real-time operation may be established. As long as a data path exists, live processes can work around the clock, continuously executing, exactly as acoustic processes occur in the nature. For example, as long as a reverberant chamber exists it will always process all incoming sounds. Similarly, processes can be roughly terminated by simply dismounting the chain, disconnecting the objects and destroying them. This intuitive process of creating and destroying applications is a powerful and typical feature of the Pd and other similar object-oriented graphical programming platforms.

4 Usage

This section covers installation, how to use the library and application examples.

4.1 Installation and licenses

From the installation source, replicate the */audience* directory structure in you file system, and have the Pd search path updated pointing to *L1, L2, L3, L4, aux* (auxiliary) and *bin* sub-directories. Individual help files and the manual troubleshooting should be consulted in case of problems. Most externals will display error messages in console if not created.

4.2 Building applications or how to use it

AUDIENCE for Pd offers resources to build a wide range of spatial audio applications, such as sound-field recorders and players, acoustic simulators, multichannel sound creation/ manipulation and mixing, 2D/3D sound scene encoding (e.g. using Ambisonics up to the 3rd order), spatial sound reproduction (auralization) to 11 output loudspeaker setups, convolution of impulse responses with dry sound sources, and to establish audio and metadata streams from/to other applications (e.g. computer graphics or virtual reality).

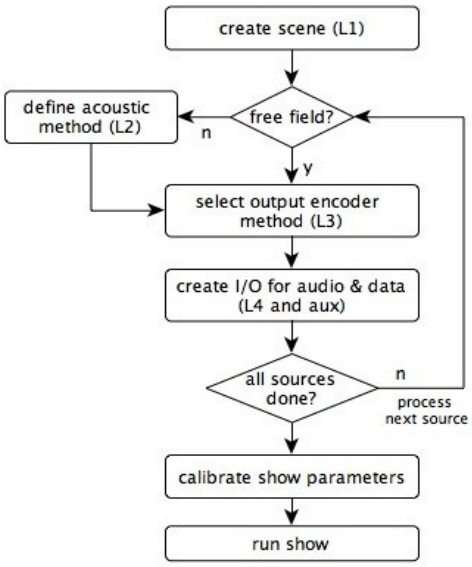

Figure 4 – Basic usage flowchart

The flowchart in figure 4 summarizes the basic tasks to create applications using AUDIENCE. A typical project flow involves creating a sound scene (L1 object), configuring an acoustic rendering scheme (L2 object), encoding the spatial sound of the rendered scene (layer L3 object) and then decoding and playing the sound scene through a loudspeaker rig (layer 4 object). Auxiliary functions, such as transport, mixing and volume controls are available from objects in the auxiliary layer.

As soon as a new Pd patch is created, the first step is to i*nstantiate a L1 scene object* (e.g. a L1 GUI) with the desired geometry and number of sound sources. Alternatively, one can have an external application send scene data to a L1 abstraction that will parse and distribute the L1 messages thereafter [2].

If an acoustic simulation is desired, *select an appropriate L2 processor* (e.g. an image-source simulator) that will render the scene for one irradiating source and produce a set of parameters (e.g. an impulse response) to drive a L3 encoding method. For a free-field spatialization instead, go directly to the *L3 encoder selection*. In AUDIENCE the default spatial-temporal encoder selection is Ambisonics, and for a *n*-source scene you will need *n* pairs of L2-L3 objects, one per source.

The L3 processes consume audio and metadata to generate the final encoded multichannel media, so the next phase is to *create the input/output links* for audio source and data (e.g. a L4 sound file player or a adc~ object) and *L4 monitors*, with any needed auxiliary objects to configure them (e.g. volume controllers, time counter and transport commands).

The above process should be repeated for all sound sources in scene. After that, the encoded media for every source shall be mixed together, producing a compound signal set that corresponds to the whole rendered scene (e.g. linearly summing the B-Format 4-channels of source 1 with those of source 2).

The final basic task is to *calibrate the show parameters,* i.e. to configure all sound sources parameters and their behavior (e.g. source position and its volume calibrated at a given distance from the listener). If show parameters change during a performance, it might be necessary to use an automation abstraction to drive them (e.g. the *audce_aux_move,* that automates simple trajectories for moving sound sources in time).

As soon as all the links are connected, the whole patch will behave like a virtual acoustic environment containing the irradiating sound sources. The difference is that the user can define the format and mode of the output sound, i.e. the output signal set that will drive a loudspeaker rig for listening.

4.3 Application examples

AUDIENCE was used in 2007 to spatialize the piece "The Unanswered Question" by Charles Ives (1906) with 9 instruments (a string quartet, a wind quartet and a trumpet) [6]. In the same year its architecture was proposed as a framework for the new MPEG SAOC (Spatial Audio Object Coding) codec [7].

In 2009 the electro-acoustic piece "Kitchen" by Fernando-Lopez Lezcano, with 8 sound sources, was spatialized during the SBCM 2009 symposium in Recife[4]. AUDIENCE is presently employed as auralization engine in a CAVE system at the University of São Paulo, and has been further used as reference audio engine for reconfigurable digital music applications, soundscape design applications and scene-oriented mixing.

The figure 5 shows an example of application patch with 1 scene containing 2 sound objects of different audio sources: a flute (from a sound file) and a voice (from a real time *adc~* port). The figure illustrates the flexibility in rendering the sound field independently for each source, as the voice uses the L2 *audce_L2_allen* image-source acoustic simulator and the flute is processed in open-field. Based on the image-source method proposed by Allen (1979) the *audce_L2_allen* calculates the ray propagation from a sound source to the receiver and all the reflections obtained in a box-like room in a given time range, outputting a set of impulse responses in Ambisonics B-Format [3].

In L3 both sources are encoded into Ambisonics (1st order) and mixed to derive an single sound-field. However, two different output listening modes are used: a stereo (e.g. for simple monitoring) and a quadraphonic output.

5 Next phases and conclusions

This paper presented an example of a general spatial audio application build-up framework for Pd using the AUDIENCE architecture, its basic functions and usage. The system is going to its 3rd generation, with design improvements in usability and sound quality, expanding the number of spatial processing functions, and looking forward to wide-spreading its availability for new applications and new user groups, considering for instance its inclusion in the *Pd Extended* distribution.

4 http://compmus.ime.usp.br/sbcm/2009/

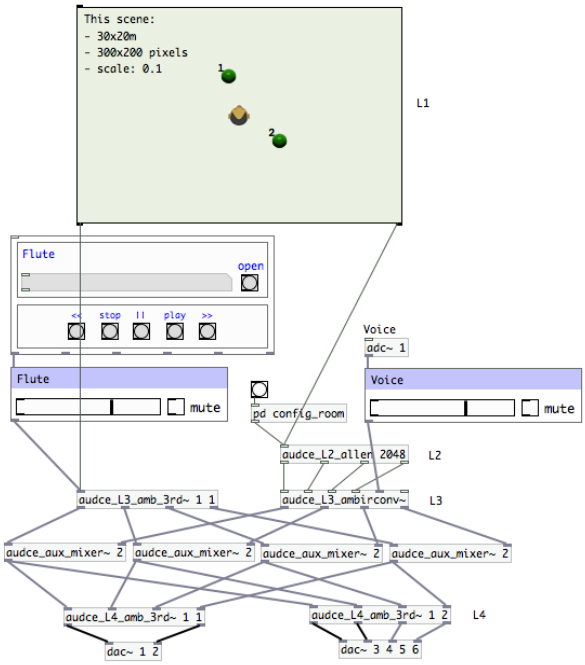

Figure 5 – Patch processing a scene with 2 sources (one with L2 processing), encoding in Ambisonics and listened through 2 different modes (2.0 and 4.0)

A new development started within the scope of the MÓBILE project (www.eca.usp.br/mobile/) towards supporting *Wave Field Synthesis* encoding/decoding. Also, a L2 acoustic simulator with *ray-tracing* and *radiosity* support, a loudness manager and an object-oriented timeline are currently under development.

An *universal object template* is a future wish to be developed, so that creation of new blocks get easier and the incorporation of an external algorithm or integration to an external plug-in or other software is done faster, more in a *plug-and-play* fashion.

There are also a number of experimental objects and abstractions under development. On spatial audio codecs the L3 MPSenc and L4 MPSdec are an experimental pair of MPEG Surround encoder and decoder objects based on the reference software issued in 2007. These developments, however, are not stable yet. Another experimental abstraction under development is the *audce_L4_sceneplay*, which aims at extending the L4 object players to all sources in a given scene.

More detailed information on advanced features of the architecture and specific applications are planned for future papers.

Acknowledgements

This software has had contributions of many researchers along the way. We would like to thank specially the researchers Leandro F. Thomaz and Renan Vital, the Laboratory of Integrable System (LSI) of the University of São Paulo for hosting the projects, and the company Coding Technologies (later on Dolby) for providing support to MPEG-4 AAC codecs. We also thank the support of Brazilian agency FAPESP through the process 2008/08632-8.

References

[1] R. R. A. Faria: *Auralização em ambientes audiovisuais imersivos.* PhD Thesis, Electronic Systems Engineering, Polytechnic School, University of São Paulo, 2005.

[2] R. R. A. Faria et al: "AUDIENCE – Audio Immersion Experiences in the Caverna Digital", SBCM 2005 - 10th Brazilian Symposium on Computer Music, 2005, Belo Horizonte, pp.106-117.

[3] R. R. A. Faria and J. A. Zuffo: "An auralization engine adapting a 3D image source acoustic model to an Ambisonics coder for immersive virtual reality", AES 28th International Conference, 2006, Piteå, Sweden, pp.157-166.

[4] L. F. Thomaz, R. R. A. Faria, M. K. Zuffo and J. A. Zuffo: "Orchestra spatialization using the AUDIENCE engine", ICMC 2006 – International Computer Music Conference, 2006, New Orleans.

[5] R. R. A. Faria et al: "Improving spatial perception through sound field simulation in VR", VECIMS 2005 – IEEE Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2005, Giardini Naxos, Italy.

[6] L. F. Thomaz: *Aplicação à música de um sistema de espacialização sonora baseado em Ambisonics*. Master Dissertation, Electronic Systems Engineering, Polytechnic School, University of São Paulo, 2007.

[7] R. R. A. Faria: ISO/IEC JTC1/SC29/WG11 (MPEG). Document m14753. *New reference architecture for broad-scope spatial audio object coding and spatial sound frameworks*. Lausanne, Switzerland, Jul. 2007. /MPEG, 81st MPEG & 42nd JPEG Meeting/