Audio Engineering Society

Convention Paper

Presented at the 124th Convention
2008 May 17–20 Amsterdam, The Netherlands

The SoundScape Renderer: A Unified Spatial Audio Reproduction Framework for Arbitrary Rendering Methods

Matthias Geier, Jens Ahrens and Sascha Spors

Deutsche Telekom Laboratories, Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany

Correspondence should be addressed to Matthias Geier (Matthias.Geier@telekom.de)

ABSTRACT

The *SoundScape Renderer* is a versatile software framework for real-time spatial audio rendering. The modular system architecture allows the use of arbitrary rendering methods. Three rendering modules are currently implemented: Wave Field Synthesis, Vector Base Amplitude Panning and Binaural Rendering. After a description of the software architecture, the implementation of the available rendering methods is explained and the graphical user interface is shown as well as the network interface for the remote control of the virtual audio scene. Finally, the Audio Scene Description Format, a system-independent storage file format, is briefly presented.

1. INTRODUCTION

We present a versatile software framework for spatial audio reproduction called *SoundScape Renderer* (SSR), which was developed at Deutsche Telekom Laboratories. Virtual audio scenes are rendered in real-time and can be manipulated interactively using a graphical user interface and a network interface.

Contrary to most existing systems (e.g. IKA-SIM [1], VirKopf/RAVEN [2], sWONDER [3]), which employ only one rendering algorithm, the design goal of the SSR is to support arbitrary reproduction methods. Until now, we implemented a Wave Field Synthesis (WFS) renderer, a binaural renderer and a Vector Base Amplitude Panning (VBAP) renderer. Future plans include adding a module for Higher Order Ambisonics.

2. SOFTWARE ARCHITECTURE

The SSR is written in C++ under massive use of

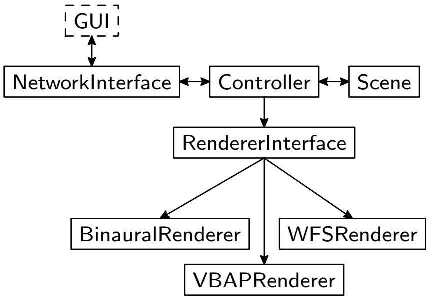


Fig. 1: Software architecture

the Standard Template Library (STL). It is compiled with g++ (the GNU C++ compiler) and runs under Linux. The JACK Audio Connection Kit (JACK) [4] is used to handle audio data which makes it very easy to connect several audio processing programs to each other and to the hardware. This way any program that produces audio data (and supports the JACK) and any live input from the audio hardware can be connected to the SSR and can serve as source input.

Audio scene descriptions (see section 6) and the reproduction setup are stored in XML (eXtensible Markup Language) files. These files can be saved and loaded by means of the Libxml2 library [5]. Both the JACK client library and Libxml2 are written in C, therefore simple C++ wrapper classes have been created.

Audio files used as virtual source signals are played back by means of the Ecasound library [6]. Ecasound supports the JACK, so soundfiles can easily be connected to the JACK ports of the renderer. Virtual source signals can be stored in mono or in multichannel files. If many sources are used, however, audio data can be read more efficiently from one multichannel file than from many mono files.

The rendered loudspeaker or headphone signals are normally played back in real-time. If needed, they can also be written to a multichannel soundfile. This way very complex scenes can be rendered in non-real-time and played back afterwards. The synchronization of playback, rendering and recording is realized with the JACK transport protocol.

The class structure of the SSR is designed in a way that functional units can be exchanged or redesigned easily without changes to the rest of the code. Figure 1 show the basic modules. Several rendering modules can be implemented and one of them will be selected when the SSR is started. The graphical user interface and even the network interface can be switched off if not needed.

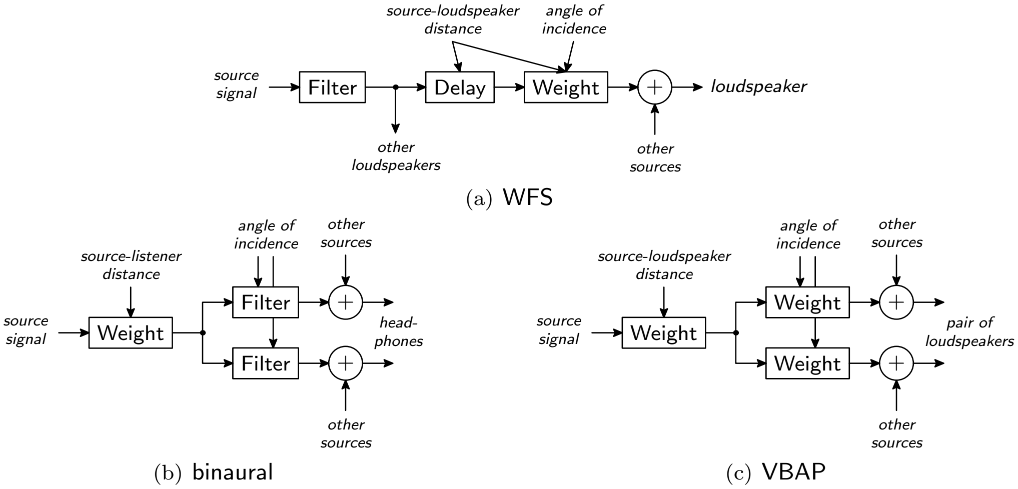
The centerpiece of the SSR is the *Controller* class. From here, all other modules are instantiated as needed: a rendering module for the audio signal processing, optionally one or more graphical user interface(s), a network interface, a class to store all scene information and several other optional modules (e.g. for head tracking and for playing and recording audio files)

When starting the SSR, first, the loudspeaker geometry or the headphone setup is loaded from an XML file. A loudspeaker setup can consist of any number and combination of single loudspeakers, linear arrays and circular array segments. After that, the rendering class is loaded. As mentioned earlier, different types of rendering modules can be used. This is realized by having an abstract interface class from which all concrete renderers are derived. For now, we can choose between the *WFSRenderer*, *BinauralRenderer* and *VBAPRenderer* classes. The *Controller* class does not need to know which kind of renderer is used, it only communicates via the abstract interface. The selected renderer creates the necessary JACK output ports depending on the reproduction setup and discloses them to the *Controller*. Once the renderer module is running, a scene can be loaded from an ASDF file (see section 6). The source data of this file (source name, position, volume, file name, point source/plane wave, . . .) are stored in the *Scene* object. Whenever a source is created, moved, deleted or changed in any other way, the *Scene* object is updated accordingly. Both the *Renderer* and any display module read the current state from this *Scene* object (via the *Controller*) when needed.

3. RENDERING MODULES

Due to the class architecture of the SSR, any two- or three-dimensional reproduction method using loudspeakers or headphones can be easily incorporated.

The signal processing of the different rendering modules uses basically the same building blocks as shown



(a) WFS

(b) binaural

(c) VBAP

Fig. 2: Signal flow in the three rendering modules

in figures 2(a) to (c). With a combination of these three functional units (convolution/filter, delay and weight), most spatialization algorithms can be realized. A convolution engine was implemented to realize the filters used in both the WFS and the binaural renderer. It will also be heavily used for Higher Order Ambisonics.

3.1. Wave Field Synthesis

Wave field synthesis is a spatial sound reproduction technique that utilizes a high number of loudspeakers to create a virtual auditory scene for a large listening area. It overcomes some of the limitations of stereophonic reproduction techniques, e.g. the sweet-spot. The theory of WFS is essentially based on the Kirchhoff-Helmholtz integral [7]. After applying some reasonable approximations to the Kirchhoff-Helmholtz formulation, the loudspeaker signals for WFS can be generated by pre-filtering the source signal, and applying individual weights and delays to the pre-filtered source signal for each loudspeaker as shown in the signal flow graph in figure 2(a). The weights and delays can be derived from the source parameters and the loudspeaker positions. For a review of the technical backgrounds of WFS see [8].

The WFS Renderer calculates the appropriate signal for every loudspeaker depending on the position and other features of the virtual sources. Up to now, both virtual point sources an plane waves can be generated. Before actually computing the contribution of a given source, the SSR determines if the source is focused or non-focused. If a source is inside the loudspeaker array it is focused, otherwise it is non-focused. A source is considered outside of the array if there is at least one array loudspeaker facing away from the source, i. e. the source is located in the half-space opposite of the loudspeaker's main direction of radiation. This criterion is valid for any open or closed array as long as it has no concave parts (which is also a requirement for WFS itself [8]). Depending on whether a virtual source is focused or not, a delay value and a weighting factor is calculated for each source-loudspeaker pair.

In addition to the computation of the loudspeaker signals the WFS Renderer also stores information for each source-loudspeaker pair if it is active or not in the current audio block. This information can be visualized in the graphical user interface (see figure 3 and section 4).

3.2. Binaural Rendering

Binaural rendering uses Head Related Transfer Functions (HRTFs) to reproduce the soundfield at the listeners ears. HRTFs are measured e.g. with a dummy head at a certain angular resolution. Linear interpolation is used to increase this resolution. A pair of HRTFs is chosen depending on the posi-

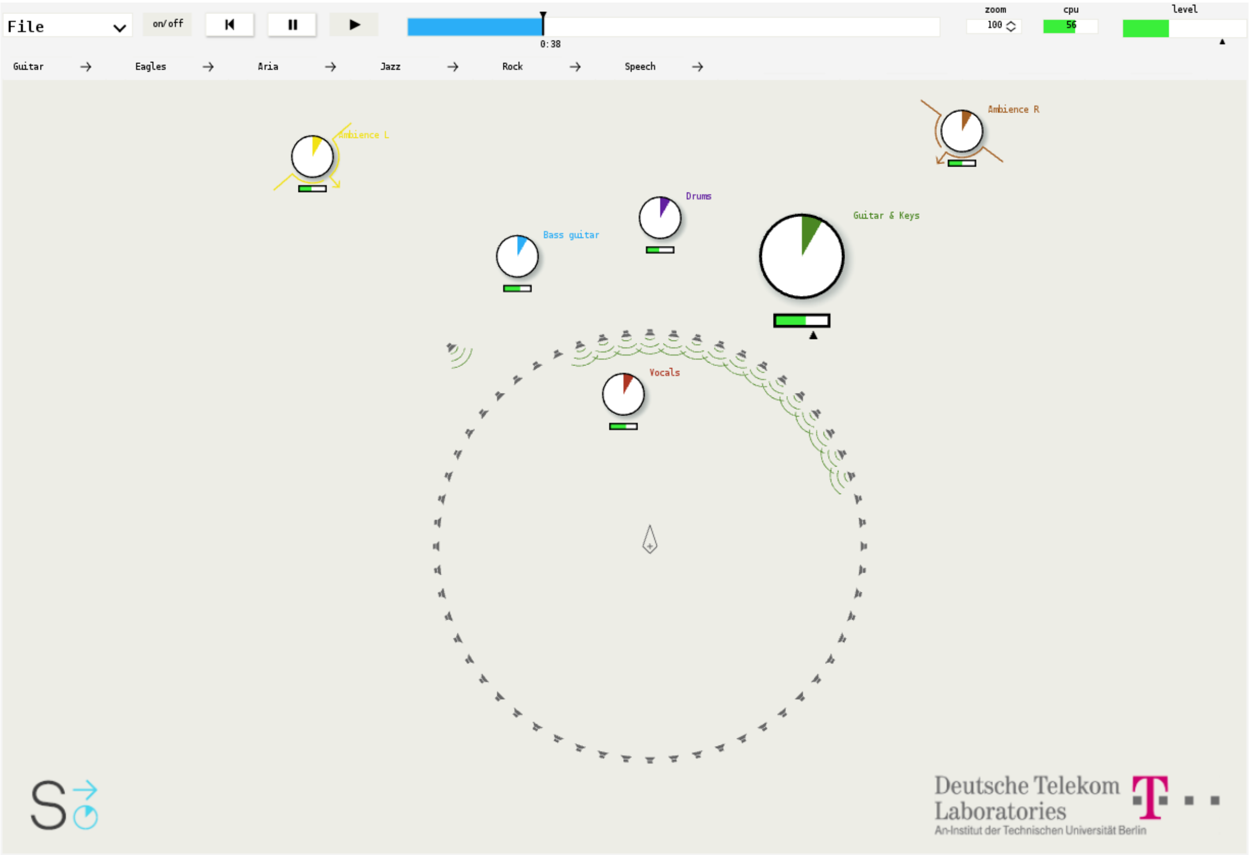


Fig. 3: Screenshot of the SoundScape Renderer's graphical user interface in action using the Wave Field Synthesis renderer. The loaded scene consists of two plane waves, one focused and three non-focused point sources. One of the latter is selected and the loudspeakers which are contributing to its wavefront are marked.

tion of the virtual sound source. These HRTFs are applied to the input signal by convolution. Optionally, the users head movement can be obtained by a head-tracking device. This head orientation is taken into account when calculating the headphone signals resulting in a more realistic experience of the virtual scene. The head tracking module can be compiled into the SSR or it can be connected via the network interface described in section 5.

Figure 2(b) shows the signal flow graph for one virtual source. Each source signal is first attenuated depending on its distance from the listener, then it is filtered using the selected pair of HRTFs to obtain the two output signals for the headphones.

3.3. Vector Base Amplitude Panning

Vector Base Amplitude Panning (VBAP) [9] is an extension of two channel stereo panning techniques. Depending on the position of the virtual sound source a pair of loudspeakers is selected to reproduce the sound from this source and the levels of the two loudspeakers are calculated by amplitude panning laws. In case of a three-dimensional setup a triple of loudspeakers is selected for each source position.

Given the architecture of the SSR, a VBAP renderer is straightforward to implement. Figure 2(c) shows its signal flow graph. The source signals are weighted and played back by two adjacent loudspeakers which are selected based on the angle of incidence of the virtual source.

4. GRAPHICAL USER INTERFACE

The graphical user interface (GUI) plays an important role in the SSR development. It is not intended as a mere tool for the programmers to change parameters of the system, but as an intuitive interface for a broader clientele. It is designed to enable the user to change the virtual scene intuitively and to instantly visualize changes to the scene which are made from outside of the GUI (e.g. via the network interface). The user interface is clear and straightforward, so that even an unexperienced user can easily operate the software.

As shown in figure 3, the sources are displayed as round objects which can be selected and moved around using the mouse or a touchscreen. All user actions can be done using only single left mouse clicks, so the full functionality is available when using a touchscreen. So far, point sources and plane waves are supported. Plane waves are distinguished by an additional arrow showing the propagation direction of the wave front. The symbol in the center of the loudspeaker array is the reference point of the array. Using this reference point, the whole loudspeaker array can be rotated and translated. When a source is selected, the loudspeakers which get a contribution from this source are marked (like for the virtual source named *Guitar & Keys* in the screenshot).

If the binaural renderer is used, the loudspeaker array is replaced by the depiction of a head (the listener's head) on the reference point, as shown in figure 4. The display of the sources, however, is unchanged.

The scene is freely zoomable and the displayed section can be moved by the mouse/touchscreen. On top of the screen there are transport controls to play and pause the source soundfiles and to change the master volume. The time-line shows the progress within the source soundfiles and it can be used to jump to certain file positions. In the top right part, the zoom level and the master volume of the audio scene can be changed. The CPU usage of the rendering engine and the current audio signal level is also shown there.

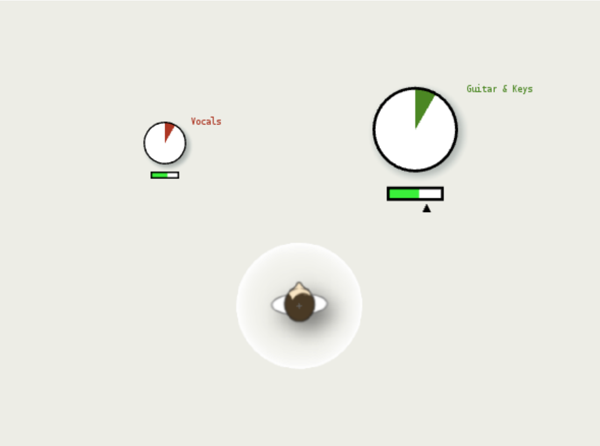


Fig. 4: When using the binaural renderer, a listener is displayed on the graphical interface

The GUI is implemented using version 4 of the Qt toolkit [10]. The display of the virtual scene is realized using OpenGL, so that hardware acceleration can be used. However, if the GUI is not needed, the SSR can be compiled without any Qt or OpenGL dependencies. In this case it can either reproduce a given audio scene or it can be run as a network server and clients (potentially running on other computers) can connect to it and manipulate the scene as described in the following section.

5. NETWORK INTERFACE

The SSR can not only be run as a single entity, but its major components can be distributed over different computers. A network interface was developed to allow the communication between different parts. One of the main applications for this feature is that the audio processing can run on one dedicated computer and the graphical user interface on another. Furthermore, any type of interface or tracking system can be connected and control the SSR via the network interface. Also several connections at a time are possible.

The SSR and its clients communicate using XML messages which are exchanged over a TCP/IP connection. In comparison to a binary format, this makes it easier to add new features. Parsing of the XML messages is done with the Libxml2 library.

The network interface was recently used to connect a multi-touch interface [11] to the SSR.

6. AUDIO SCENE DESCRIPTION FORMAT

Virtual audio scenes are stored in an XML based file format called Audio Scene Description Format (ASDF) [12] which contains geometric information for all virtual sound sources as well as general scene properties.

As the SSR, the ASDF is independent of the spatialization algorithm. Moreover, it is even independent of the SSR itself. It includes no implementation-specific information whatsoever and can therefore be used for any spatial reproduction system out there.

For now, only static scenes can be stored, but a new version of this format is currently developed which will allow moving sources along trajectories, adding and removing sources during the runtime of the scene and other dynamic features.

7. FUTURE WORK

The SoundScape Renderer is work in progress and there are many possibilities to improve and to extend it. We are working on creating dynamic scenes with moving virtual sources and on saving these movements and other dynamic changes to ASDF files.

A Higher Order Ambisonics renderer will be implemented which will help us to evaluate Wave Field Synthesis, Vector Base Amplitude Panning and Ambisonics on the same loudspeaker array. In addition to plane waves and point sources we want to implement directional sound sources in both WFS [13] and Ambisonics [14].

8. REFERENCES

[1] A. Silzle, H. Strauss and P. Novo. IKA-SIM: A system to generate auditory virtual environments. In *116th AES Convention.* Berlin, Germany, May 2004.

[2] T. Lentz et al. Virtual reality system with integrated sound field simulation and reproduction. *EURASIP Journal on Advances in Signal Processing*, Article ID 70540, 2007.

[3] M. A. Baalman et al. Renewed architecture of the sWONDER software for Wave Field Synthesis on large scale systems. In *Linux Audio Conference.* Berlin, Germany, March 2007.

[4] P. Davis et al. *JACK Audio Connection Kit.* `http://jackaudio.org`.

[5] D. Veillard et al. *Libxml2.* `http://xmlsoft.org`.

[6] K. Vehmanen et al. *Ecasound.* `http://www.eca.cx/ecasound`.

[7] A. J. Berkhout. A holographic approach to acoustic control. *Journal of the AES*, 36(12):977–995, December 1988.

[8] S. Spors, R. Rabenstein and J. Ahrens. The theory of Wave Field Synthesis revisited. In *124th AES Convention.* Amsterdam, The Netherlands, May 2008.

[9] V. Pulkki. Virtual sound source positioning using Vector Base Amplitude Panning. *Journal of the AES*, 45(6):456–466, June 1997.

[10] Trolltech ASA. *Qt.* `http://trolltech.com/products/qt`.

[11] K. Bredies et al. The Multi-Touch SoundScape Renderer. In *9th International Working Conference on Advanced Visual Interfaces (AVI).* Napoli, Italy, May 2008.

[12] M. Geier, J. Ahrens and S. Spors. ASDF: Ein XML Format zur Beschreibung von virtuellen 3D-Audioszenen. In *34. Jahrestagung für Akustik (DAGA).* Dresden, Germany, March 2008.

[13] J. Ahrens and S. Spors. Implementation of directional sources in Wave Field Synthesis. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA).* New Paltz, NY, USA, October 2007.

[14] J. Ahrens and S. Spors. Rendering of virtual sound sources with arbitrary directivity in Higher Order Ambisonics. In *123rd AES Convention.* New York, NY, USA, October 2007.