THE SOUNDSCAPE RENDERER: A VERSATILE SOFTWARE FRAMEWORK FOR SPATIAL AUDIO REPRODUCTION

Matthias Geier, Jens Ahrens, André Möhl, Sascha Spors, Jonas Loh, Katharina Bredies

Deutsche Telekom Laboratories
Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin, Germany
e-mail: Matthias.Geier@telekom.de

ABSTRACT

We present a new software system for spatial audio reproduction called SoundScape Renderer. *It is currently used for rendering 2-dimensional virtual acoustic scenes using either Wave Field Synthesis, binaural rendering or Vector Based Amplitude Panning. A further extension to higher order Ambisonics is planned. However, in this paper we will focus on the WFS functionality.*

We also describe the physical setup of the WFS system installed at the Usability Laboratory of Deutsche Telekom Laboratories, the natural habitat of the SoundScape Renderer.

Finally, we present a working draft for an XML file format enabling the exchange of acoustic scenes between different software systems. This Audio Scene Description Format *(ASDF) is entirely open and we encourage everyone to participate in its further definition.*

1. THE SOUNDSCAPE RENDERER

Beginning in late 2006, the *SoundScape Renderer* (SSR) was written to drive the Wave Field Synthesis (WFS) system in the new Usability Laboratory space inaugurated in spring 2007 at Deutsche Telekom Laboratories in Berlin. However, its functionality is not limited to this system, nor to WFS in general. The SSR is a versatile software framework for any kind of spatial audio reproduction. It was conceived as a modular and transparent research tool for different spatial reproduction methods, for multi-party communication scenarios, for interaction with spatial audio and related research areas.

The first rendering module of the SSR was for WFS, which is also the main emphasis of this paper. A binaural demonstrator used earlier at Telekom Laboratories was incorporated into the SSR as another rendering module. Most recently, support for Vector Based Amplitude Panning [1] was added.

1.1 Software Architecture

The SSR is written in C++ under massive use of the standard template library (STL). It is compiled with g++ (the GNU C++ compiler) and of course runs under Linux. The JACK audio connection kit (JACK)[1] 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 sound (and supports the JACK) and any live input from the audio hardware can be manually connected to the SSR and can serve as source input.

Audio scene descriptions and the reproduction setup are stored in XML files. These files can be saved and loaded by means of the libxml2 library[2]. Both the JACK client library and libxml2 are written in C, therefore simple C++ wrapper classes have been created.

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. The centerpiece of the SSR is the *Controller* class. From here, all other modules are instantiated as needed. First, the loudspeaker geometry is loaded from an ASDF file (see section 3). A loudspeaker setup can consist of any number and combination of single loudspeakers, linear arrays and circular array segments. After that, the *Renderer* class is loaded. As mentioned earlier, different types of rendering modules can be used. This is realized by having an abstract interface class named *Renderer* 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 3). 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.

1.2 Audio File Handling

Audio files used as virtual source signals are played back by means of the Ecasound library[3]. 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 faster from one multichannel file than from many mono files.

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

[1] `http://jackaudio.org`

[2] `http://xmlsoft.org`

[3] `http://www.eca.cx/ecasound`

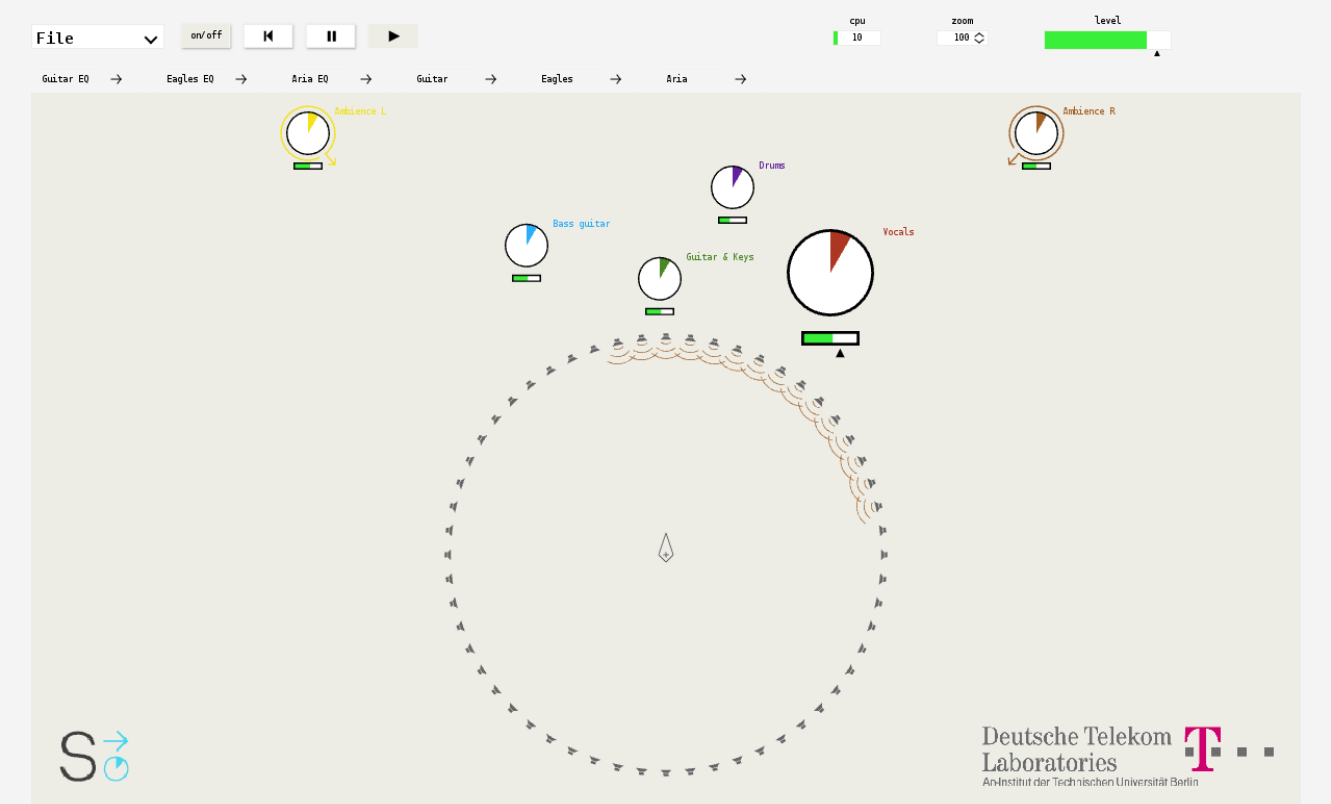

Figure 1: *Screenshot of the SoundScape Renderer's graphical user interface in action. The loaded scene consists of two plane waves and four non-focused point sources. One of the latter is selected and the loudspeakers which are contributing to its wavefront are marked.*

1.3 WFS Renderer

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 point sources an plane waves are supported. Before actually computing the contribution of a given source, it 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.

Depending on the focusedness of the sources a delay value and a weighting factor is calculated for each source-loudspeaker pair. Each audio block is rendered two times, once with the delay and weighting factor of the previous block and once with the current values. These two blocks are cross-faded with a raised cosine window. This way we avoid discontinuities in the signal on fast source movements. A positive side effect of this method is that if a source changes its status from inactive to active or vice versa, it automatically gets a fade-in or a fade-out, respectively. However, if there are no fast source movements the crossfade can be deactivated to save CPU load and to avoid a spectral coloration of the sound. An alternative algorithm based on interpolation will be implemented in the future and the two methods will be compared.

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 1 and section 1.5).

1.4 Other Renderers

So far, the capabilities of the SSR also include binaural rendering and Vector Based Amplitude Panning, although its main application has been WFS. The binaural renderer uses BruteFIR[4] as powerful convolution engine. Any form of listener tracking can be realized via the network interface described in section 1.6.

Due to the class architecture of the SSR, any 2-dimensional reproduction method using loudspeakers or headphones can be easily incorporated. An extension to higher order Ambisonics [2] is planned.

1.5 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

[4] `http://www.ludd.luth.se/~torger/brutefir.html`

an attractive interface for a broader clientele who will use it in in-house demos as well as on expositions or suchlike. 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 1, the sources are displayed as round objects which can be selected and moved around using the mouse or 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. When a source is selected, the loudspeakers which get a contribution from this source are marked (like for the virtual source named *Vocals* 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. The display of the sources 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 controls to play and pause the source soundfiles and to change the master volume.

The GUI is implemented using version 4 of the Qt toolkit[5]. However, if the GUI is not needed, the SSR can be compiled without any Qt dependencies. The display of the virtual scene is realized using OpenGL, so that hardware acceleration is possible.

1.6 Network Interface

The SSR can not only be run as a single application, but its major components can run on 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 network interface uses the TCP internet protocol and the control messages are sent as XML tags. The parsing of the messages is done with the libxml2 library.

2. PHYSICAL SETUP @ TELEKOM LABS

The WFS system in our new Usability Laboratory space features a circular loudspeaker array suspended from the ceiling with its height adjustable by an electric winch. The array has a diameter of three meters and comprises 56 loudspeakers (ELAC 301). It is left as an exercise for the reader to calculate the loudspeaker distance.

The computer running the SSR is placed in the next room to avoid fan noise. We use a dual-core dual AMD Opteron PC running Gentoo Linux. The audio signals are output via an RME HDSP MADI Soundcard.

The 56-channel MADI signal is split up into seven ADAT signals with an RME ADI-648 converter. The seven multichannel amplifiers with ADAT input are custom-made by the Chair of Multimedia Communications and Signal Processing at University of Erlangen-Nuremberg.

[5] `http://trolltech.com/products/qt`

3. AUDIO SCENE DESCRIPTION FORMAT

The SSR can load both virtual scenes and reproduction setups from XML (eXtensible Markup Language) files. For this purpose we defined a file format called *Audio Scene Description Format* (ASDF). This format should not be limited to the SSR, we want it rather to become an open exchange format which will make it easy to reproduce the same scene on different locations and systems. This way, for example an electroacoustic spatial composition could be performed on many different locations without the need to rearrange it anew for each venue.

In the following paragraphs, we present the format in its current state in which it is in use within the SSR. An ASDF file consists of four optional parts: `header`, `scene_setup`, `reproduction_setup` and `score`. The last part is not yet defined because for now, only static scenes are supported in the SSR. For some details about the ASDF consider the following small example scene:

```
<?xml version="1.0"?>
<asdf>
  <header>
    <name>Simple Example Scene</name>
  </header>
  <scene_setup>
    <source name="Vocals" model="point">
      <file channel="1">demo.wav</file>
      <position x="-2" y="2"/>
    </source>
    <source name="Ambience" model="plane">
      <file channel="2">demo.wav</file>
      <position x="2" y="2"/>
    </source>
  </scene_setup>
</asdf>
```

The `header` element holds general information about the ASDF file, like name, description and creation date. In the `scene_setup` part, all sources are defined. The example shows both a point source and a plane wave. Positions are specified in meters in a right handed cartesian coordinate system. As shown, one sound file can be used for several virtual sources.

As mentioned before, the ASDF can also describe loudspeaker setups. The loudspeaker setup at Telekom Laboratories can be defined as follows:

```
<?xml version="1.0"?>
<asdf>
  <header>
    <name>Circular Loudspeaker Array</name>
  </header>
  <reproduction_setup>
    <circular_array number="56">
      <first>
        <position x="1.5" y="0"/>
        <orientation azimuth="-180"/>
      </first>
    </circular_array>
  </reproduction_setup>
</asdf>
```

For the definition of an arbitrary loudspeaker array, the `reproduction_setup` section can contain any number

of the elements `loudspeaker`, `circular_array` and `linear_array`. Arrays can be specified by the number of speakers and by the `first` and `second`, or by the `first` and `last` loudspeaker. If only a `first` loudspeaker is given in a `circular_array`, a full circle is assumed.

The definition of the ASDF is still in an early stage. We greatly appreciate any feedback on the topic and would be glad if many people and institutions could participate in creating a commonly accepted exchange format.

4. OUTLOOK

The SoundScape Renderer is still quite young and there are many possibilities to improve and to extend it. We will be 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 compare Wave Fiel Synthesis, Vector Based 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 [3] and Ambisonics [4].

REFERENCES

[1] Ville Pullki, "Virtual sound source positioning using Vector Based Amplitude Panning", *Journal of the Audio Engineering Society (JAES)*, vol. 45(6), June 1997.

[2] Jérôme Daniel, "Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia," PhD thesis, Université Paris 6, 2001.

[3] Jens Ahrens and Sascha Spors, "Implementation of Directional Sources in Wave Field Synthesis," *2007 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, October 21-24, 2007, New Paltz, NY.

[4] Jens Ahrens and Sascha Spors, "Rendering of virtual sound sources with arbitrary directivity in higher order Ambisonics", *123rd Convention of the Audio Engineering Society (AES)*, 2007 October 5-8 New York, NY.