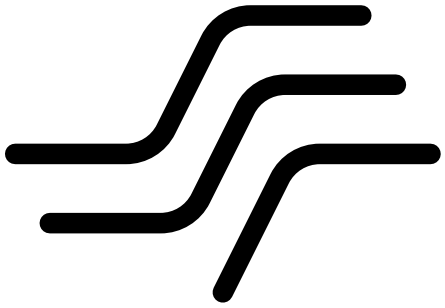
Introduction to the SoundScape Renderer (SSR)

Jens Ahrens, Matthias Geier and Sascha Spors

SoundScapeRenderer@telekom.de

May 3, 2011



 Quality & Usability Lab
Deutsche Telekom Laboratories
Technische Universität Berlin
Ernst-Reuter-Platz 7, 10587 Berlin, Germany

Contents

1 General stuff

1.1 Introduction

The SoundScape Renderer (SSR) is a software framework for real-time spatial audio reproduction running under GNU/Linux, Mac OSX and possibly some other UNIX variants. The current implementation provides Wave Field Synthesis (WFS), binaural (HRTF-based) reproduction, binaural room (re-)synthesis (BRTF-based reproduction), head-tracked binaural playback, Ambisonics Amplitude Panning (AAP), and Vector Base Amplitude Panning (VBAP). There are also some more exotic options like the Binaural Playback Renderer (BPB) and the Generic Renderer. The rendering algorithm is chosen upon execution of the software. For more details see section 3.

The SSR is intended as versatile framework for the state of the art implementation of various spatial audio reproduction techniques. You may use it for your own academic research, teaching or demonstration activities or whatever else you like. However, it would be nice if you would mention the use of the SSR by e.g. referencing [1].

Note that so far, the SSR only supports two-dimensional reproduction for any type of renderer. For WFS principally any convex loudspeaker setup (e.g. circles, rectangles) can be used. The loudspeakers should be densely spaced. For VBAP circular setups are highly recommended. APA does require circular setups. The binaural renderer can handle only one listener at a time.

1.2 Quick Start

After downloading the SSR package, open a shell and use following commands:

```
tar xvzf ssr-x.x.x.tar.gz
cd ssr-x.x.x
./configure
make
make install
qjackctl &
ssr my_audio_file.wav
```

You have to replace `x.x.x` with the current version number, e.g. `0.3.1`. With above commands you are performing the following steps:

- Unpack the downloaded tarball containing the source-code
- Go to the extracted directory[1].
- Configure the SSR
- Install the SSR

[1] Note that most relative paths which are mentioned in this document are relative to this folder, which is the folder where the SSR tarball was extracted. Therefore, e.g. the `src/` directory could be something like `$HOME/ssr-x.x.x/src/` where "x" stands for the version numbers.

- Open the graphical user interface for JACK (`qjackctl`). Please click "Start" to start the server. As alternative you can start JACK with

  ```
  jackd -d alsa -r 44100
  ```

  See section 2.5 and `man jackd` for further options.

- Open the SSR with an audio file of your choice. This can be a multichannel file

This will load the audio file `my_audio_file.wav` and create a virtual sound source for each channel in the audio file. By default, the SSR will start with the binaural renderer. Please use headphones to listen to the generated output!

If you don't need a graphical user interface and you want to dedicate all your resources to audio processing, try

```
ssr --no-gui my_audio_file.wav
```

For further options, see section 2.5 and `ssr --help`.

1.3 Audio Scenes

1.3.1 Format

The SSR can open `.asd` files (refer to section 1.4) as well as normal audio files. If an audio file is opened, SSR creates an individual virtual sound source for each channel which the audio file contains. If a two-channel audio file is opened, the resulting virtual sound sources are positioned like a virtual stereo loudspeaker setup with respect to the location of the reference point. For audio files with more (or less) channels, SSR randomly arranges the resulting virtual sound sources. All types that ecasound and libsndfile can open can be used. In particular this includes `.wav` and `.aiff` files. Even `.mp3` could work, depending on the codecs installed.

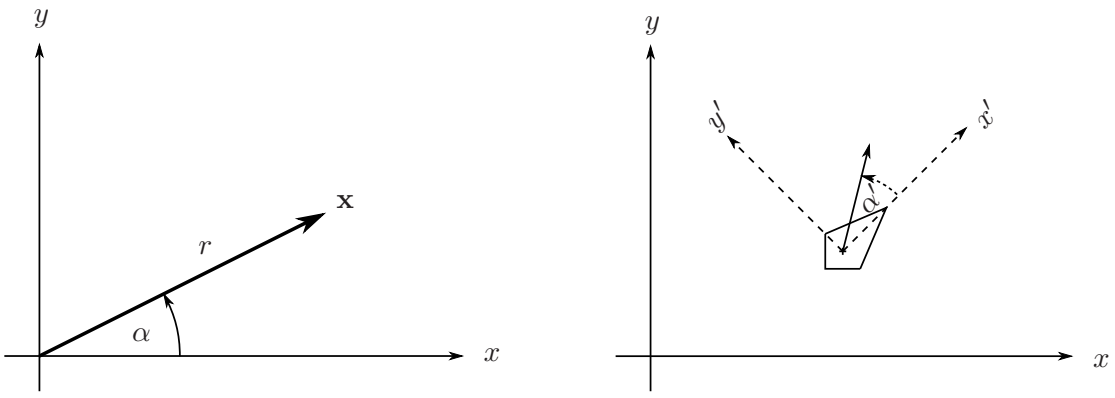
In the case of a scene being loaded from an `.asd` file, all audio files which are associated to virtual sound sources are replayed in parallel and replaying starts at the beginning of the scene. So far, a dynamic handling of audio files has not been implemented.

1.3.2 Coordinate System

Fig. 1(a) depicts the global coordinate system used in the SSR. Virtual sound sources as well as the reference are positioned and orientated with respect to this coordinate system. For loudspeakers, positioning is a bit more tricky since it is done with respect to a local coordinate system determined by the reference. Refer to Fig. 1(b). The loudspeakers are positioned with respect to the primed coordinates (x', y', etc.).

The motivation to do it like this is to have a means to virtually move the entire loudspeaker setup inside a scene by simply moving the reference. This enables arbitrary movement of the listener in a scene independent of the physical setup of the reproduction system.

Please do not confuse the origin of the coordinate system with the reference. The coordinate system is static and specifies absolute positions.



(a) Global coordinate system.

(b) Local coordinate system relative to the reference. The latter is indicated by the rhomb.

Figure 1: The coordinate system used in the SSR. In ASDF α and α' are referred to as azimuth (refer to section 1.4).

The reference is movable and is always taken with respect to the current reproduction setup. The loudspeaker-based methods do not consider the orientation of the reference point but its location influences the way loudspeakers are driven. E.g., the reference location corresponds to the *sweet spot* in VBAP. It is therefore advisable to put the reference point to your preferred listening position. In the binaural methods the reference point represents the listener and indicates the position and orientation of the latter. It is therefore essential to set it properly in this case.

Note that the reference position and orientation can of course be updated in real-time. For the loudspeaker-based methods this is only useful to a limited extent unless you want to move inside the scene. However, for the binaural methods it is essential that both the reference position and orientation (i.e. the listener's position and orientation) are tracked and updated in real-time. Refer also to Sec. 2.7.

1.4 Audio Scene Description Format (ASDF)

Besides pure audio files, SSR can also read the current development version of the *Audio Scene Description Format (ASDF)* [2]. Note however that so far, we have only implemented descriptions of static features. That means in the current state it is not possible to describe e.g. movements of a virtual sound source. As you can see in the example audio scene below, an audio file can be assigned to each virtual sound source. The replay of all involved audio files is synchronized to the replay of the entire scene. That means all audio files start at the beginning of the sound scene. If you fast forward or rewind the scene, all audio files fast forward or rewind. **Note that it is sigificantly more efficient to read data from an interleaved multichannel file compared to reading all channels from individual files**.

1.4.1 Syntax

The format syntax is quite self-explanatory. See the examples below. Note that the paths to the audio files can be either absolute (not recommended) or relative to the directory where the scene file is stored. The exact format description of the ASDF can be found in the XML Schema file `asdf.xsd`.
Find below a sample scene description:

```
<?xml version="1.0"?>
<asdf version="0.1">
  <header>
    <name>Simple Example Scene</name>
  </header>
  <scene_setup>
    <source name="Vocals" model="point">
      <file>audio/demo.wav</file>
      <position x="-2" y="2"/>
    </source>
    <source name="Ambience" model="plane">
      <file channel="2">audio/demo.wav</file>
      <position x="2" y="2"/>
    </source>
  </scene_setup>
</asdf>
```

The input channels of a soundcard can be used by specifying the channel number instead of an audio file, e.g. `<port>3</port>` instead of `<file>my_audio.wav</file>`.

1.4.2 Examples

We provide an audio scene example in ASDF with this release. You find it in `data/scenes/live_input.asd`. If you load this file into the SSR it will create 4 sound sources which will be connected to the first four channels of your sound card. If your sound card happens to have less than four outputs, less sources will be created accordingly. More examples for audio scenes can be downloaded from the SSR website [3].

1.5 IP Interface

One of the key features of the SSR is an interface which lets you remotely control the SSR via a TCP socket using XML messages. This interface enables you to straightforwardly connect any type of interaction tool from any type of operating system. The format of the messages sent over the network is still under development and may very likely change in future versions. Please find some brief information in section 5.

An example how the SSR can be controlled via its network interface is the Python client located in the directory `python_client/` and the provided Pure Data patches.

1.6 Known Problems, Bug Reports, Feature Requests and Comments

For a list of known problems have a look at the SSR development website[2]. Please report any bugs, feature requests and comments to *SoundScapeRenderer@telekom.de*. We will keep track of them and will try to fix them in a reasonable time. The more bugs you report the more we can fix. Of course, you are welcome to provide bug fixes. ☺

1.7 Missing Features

Apropos feature requests: While scanning this document you might realize that there might be certain essential features missing. Actually, we agree. Due to time constraints we postponed the following features – amongst others – to the next (or let's say a later) release of the SSR:

- Possibility to create audio scenes using the GUI.
- Possibility to apply a headphone compensation filter.
- Possibility to apply filter on subwoofers.
- Near-field compensated Ambisonics.

1.8 Contributors

Written by:
Matthias Geier, Jens Ahrens

Scientific supervision:
Sascha Spors

Contributions by:
Peter Bartz, Florian Hinterleitner, Torben Hohn, Lukas Kaser, André Möhl

GUI design:
Katharina Bredies, Jonas Loh, Jens Ahrens

Logo design:
Fabian Hemmert

1.9 Your Own Contributions

The SSR is thought to provide a state of the art implementation of various spatial audio reproduction techniques. We therefore would like to encourage you to contribute to this project since we can not assure to be at the state of the art at all times ourselves. Everybody is welcome to contribute to the development of the SSR. However, if you are

[2] `https://dev.qu.tu-berlin.de/projects/ssr/wiki/Known_Issues`

planning to do so, we kindly ask you to contact us beforehand (e.g. via *SoundScapeRenderer@telekom.de*). The SSR is in a rather temporary state and we might apply some changes to its architecture. We would like to ensure that your own implementations stay compatible with future versions.

2 Compiling and running the SSR

2.1 Getting the Source

If you didn't receive this manual with the source code of the SSR, you can download it from the SSR website [3]. After downloading, you can unpack the tarball with the command `tar xvzf ssr-x.x.x.tar.gz` in a shell. This will extract the source code to a directory of the form `ssr-x.x.x` where "x" stands for the version numbers. `cd` to this directory and proceed with section 2.2 to configure the SSR.

2.2 Configuring

To build the SSR from source you have to configure first. Open a shell and `cd` to the directory containing the source code of the package and type:

```
./configure
```

This script will check your system for dependencies and prepare the `Makefile` required for compilation. Section 2.2.1 lists the dependencies that must be installed on your system. The `configure` script will signal if dependencies are missing. At successful termination of the `configure` script a summary will show up.

Section 2.2.2 is intended to help you troubleshooting.

2.2.1 Dependencies

At least the following software (libraries and headers) including their development packages (*dev* or *devel*), where available, are required for a full installation of the SSR:

- JACK Audio Connection Kit [4]
- FFTW3 compiled for single precision (`fftw3f`) version 3.0 or higher [5]
- libsndfile [6]
- Ecasound [7]
- Trolltech's Qt 4.2.2 or higher with OpenGL (QtCore, QtGui and QtOpenGL) [8]
- libxml2 [9]
- Boost.Asio [10], included since Boost version 1.35.

We provide a simple integration of two tracking systems. Please read section 2.7 for further informations about head tracking.

2.2.2 Hints on Configuration

If you encounter problems configuring the SSR these hints could help:

- Ensure that you really installed all libraries (`lib`) with devel-package (`devel` or `dev`, where available) mentioned in section 2.2.1.
- It may be necessary to run `ldconfig` after installing new libraries.
- Ensure that `/etc/ld.so.conf` or `LD_LIBRARY_PATH` are set properly, and run `ldconfig` after changes.
- If a header is not installed in the standard paths of your system you can pass its location to the configure script using `./configure CPPFLAGS=-Iyourpath`.

Note that with `./configure --help` all configure-options are displayed, e.g. in section "Optional Features" you will find how to disable compilation of the head trackers and many other things. Setting the influential environment variables with `./configure VARNAME=value` can be useful for debugging dependencies.

2.3 Compiling and Installing

If the configure script terminates with success, it creates a file named `Makefile`. You can build the SSR by typing

```
make
make install
```

This will compile the SSR and install it to your system. We recommend the usage of GCC 4 or higher. Note that with `./configure --prefix` you can specify where to install the SSR on your system. The default prefix is `/usr/local`.

2.4 Uninstalling

If the SSR didn't meet your expectations, we are very sorry, and of course you can easily remove it from your system with

```
make uninstall
```

2.5 Running the SSR

Before you start the SSR, start JACK [4], e.g. by typing
`jackd -d alsa -r 44100` in a shell or using the graphical user interface "qjackctl" [11]. Now, the easiest way to get a signal out of the SSR is by passing a sound-file directly:

```
ssr YOUR_AUDIO_FILE
```

By default, the SSR starts with the binaural renderer; please use headphones for listening with this renderer. Type `ssr --help` to get an overview of the command line options and various renderers:

```
USAGE: ssr [OPTIONS] <scene-file>

OPTIONS:

Choose a rendering algorithm:
    --binaural          Binaural (using HRIRs)
    --brs               Binaural Room Synthesis (using BRIRs)
    --bpb               Binaural Playback Renderer
    --wfs               Wave Field Synthesis
    --aap               Ambisonics Amplitude Panning
    --vbap              Stereophonic (Vector Base Amplitude Panning)
    --generic           Generic Renderer

Renderer-specific options:
    --hrirs=FILE        Load the HRIRs for binaural renderer from FILE
    --hrir-size=VALUE   Maximum IR length (binaural and BRS renderer)
    --prefilter=FILE    Load WFS prefilter from FILE
-o, --ambisonics-order=VALUE Ambisonics order to use (default: maximum)
    --in-phase-rendering     Use in-phase rendering for Ambisonics

JACK options:
    --input-prefix=PREFIX    Input  port prefix (default: "system:capture_")
    --output-prefix=PREFIX   Output port prefix (default: "system:playback_")
-f, --freewheel         Use JACK in freewheeling mode

General options:
-c, --config=FILE       Read configuration from FILE
-s, --setup=FILE        Load reproduction setup from FILE
-r, --record=FILE       Record the audio output of the renderer to FILE
    --loop              Loop all audio files
    --master-volume-correction=VALUE
                        Correction of the master volume in dB (default: 0 dB)
-i, --ip-server[=PORT] Start IP server (default on)
                        A port can be specified: --ip-server=5555
-I, --no-ip-server      Don't start IP server
-g, --gui               Start GUI (default)
-G, --no-gui            Don't start GUI
-t, --tracker[=PORT]    Start tracker (default)
                        A serial port can be specified: --tracker=/dev/ttyS1
-T, --no-tracker        Don't start tracker

-h, --help              Show this very help information. You just typed that!
-v, --verbose           Increase verbosity level (up to -vvv)
-V, --version           Show version information and exit
```

Choose the appropriate arguments and make sure that your amplifiers are not turned too loud...

To stop the SSR use either the options provided by the GUI (section 4) or type `Crtl+c` in the shell in which you started the SSR.

Keyboard actions in non-GUI mode If you start SSR without GUI (option `--no-gui`), it starts automatically replaying the scene you have loaded. You can have some interaction via the shell. Currently implemented actions are (all followed by `Return`):

`c`: calibrate tracker (if available)

`p`: start playback

`q`: quit application

`r`: "rewind"; go back to the beginning of the current scene

`s`: stop (pause) playback

Note that in non-GUI mode, audio processing is always taking place. Live inputs are processed even if you pause playback.

Recording the SSR output You can record the audio output of the SSR using the `--record=FILE` command line option. All output signals (i.e. the loudspeaker signals) will be recorded to a multichannel wav-file named `FILE`. The order of channels corresponds to the order of loudspeakers specifed in the reproduction setup (see sections 3.1.1 and 1.4). The recording can then be used to analyze the SSR output or to replay it without the SSR using a software player like `ecaplay` [7].

2.6 Configuration File

The general configuration of the SSR (that means which renderer to use, if GUI is enabled etc.) can be specified in a configuration file (e.g. `ssr.conf`). By specifying your wishes in such a file, you avoid having to give explicit command line options every time you start the SSR. We have added the example `data/ssr.conf.example` which mentions all possible parameters. Take a look inside, it is rather self-explanatory. There are three possibilities to specify a configuration file:

- put it in `/etc/ssr.conf`
- put it in your home directory in `$HOME/.ssr/ssr.conf`
- specify it on the command line with `ssr -c my_config.file`

We explicitly mention one parameter here which might be of immediate interest for you: `MASTER_VOLUME_CORRECTION`. This a correction in dB (!) which is applied – as you might guess – to the master volume. The motivation is to have means to adopt

the general perceived loudness of the reproduction of a given system. Factors like the distance of the loudspeakers to the listener or the typical distance of virtual sound sources influence the resulting loudness which can be adjusted to the desired level by means of the `MASTER_VOLUME_CORRECTION`. Of course, there's also a command line alternative (`--master-volume-correction`).

2.7 Head Tracking

We provide integration of the *InterSense InertiaCube3* tracking sensor [12] and the *Polhemus Fastrak* [13]. They are used to update the orientation of the reference (in binaural reproduction this is the listener) in real-time. Please read sections 2.7.1 and 2.7.2 if you want to compile the SSR with the support for these trackers.

Note that on startup, the SSR tries to find the tracker. If it fails, it continues without it. If you use a tracker, make sure that you have the appropriate rights to read from the respective port.

You can calibrate the tracker while the SSR is running by pressing `Return`. The instantaneous orientation will then be interpreted as straight forward ($\alpha = 90°$).

2.7.1 Preparing InterSense InertiaCube3

If you want to compile the SSR with support for the *InterSense InertiaCube3* tracking sensor [12], please download the *InterSense Software Development Kit* (SDK) from the InterSense website [12]. Unpack the archive and place the files

- `isense.h` and `types.h` to `/usr/local/include`, and
- `libisense.so` (the version appropriate for your processor type) to `usr/local/lib`.

The SSR `configuration` script will automatically detect the presence of the files described above and if they are found, enable the compilation for the support of this tracker. To disable this tracker, use `./configure --disable-intersense` and recompile.

If you encounter an error-message similar to `libisense.so: cannot open shared object file: No such file or directory`, but the file is placed correctly, run `ldconfig`.

2.7.2 Preparing Polhemus Fastrack

For incorporation of the *Polhemus Fastrack* [13] with serial connection, no additional libraries are required. If you want to disable this tracker, use `./configure --disable-polhemus` and recompile.

2.8 Using the SSR with DAWs

As stated before, the SSR is currently not able to dynamically replay audio files (refer to section 1.4). If your audio scenes are complex, you might want to consider using the SSR together with a digital audio work station (DAW). To do so, you simply have to

create as many sources in the SSR as you have audio tracks in your respective DAW project and assign live inputs to the sources. Amongst the ASDF examples we provide at [3] you find an example scene description which does exactly this.

DAWs like Ardour [14] support JACK and their use is therefore straightforward. DAWs which do not run on Linux or do not support JACK can be connected via the input of the sound card.

In the future we will provide a VST plug-in which will allow you to dynamically operate all virtual source's properties (like e.g. a source's position or level etc.). You will then be able to have the full SSR functionality controlled from your DAW.

3 The Renderers

3.1 General

3.1.1 Reproduction Setups

The geometry of the actual reproduction setup is specified in `.asd` files, just like sound scenes. By default, it is loaded from the file `/usr/local/share/ssr/default_setup.asd`. Use the `--setup` command line option to load another reproduction setup file. Note that the loudspeaker setups have to be convex. This is not checked by the SSR. The loudspeakers appear at the outputs of your sound card in the same order as they are specified in the `.asd` file, starting with channel 1.
A sample reproduction setup description:

```
<?xml version="1.0"?>
<asdf version="0.1">
  <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>
```

We provide the following setups in the directory `data/reproduction_setups/`:

- `2.0.asd`: standard stereo setup at 1.5 mtrs distance
- `2.1.asd`: standard stereo setup at 1.5 mtrs distance plus subwoofer
- `5.1.asd`: standard 5.1 setup on circle with a diameter of 3 mtrs
- `rounded_rectangle.asd`: Demonstrates how to combine circular arcs and linear array segments.
- `circle.asd`: This is a circular array of 3 mtrs diameter composed of 56 loudspeakers.
- `loudspeaker_setup_with_nearly_all_features.asd`: This setup describes all supported options, open it with your favorite text editor and have a look inside.

Note that outputs specified as subwoofers receive a signal having full bandwidth. There is some limited freedom in assigning channels to loudspeakers: If you insert the element `<skip number="5"/>`, the specified number of output channels are skipped and the following loudspeakers get higher channel numbers accordingly.

Of course, the binaural and BRS renderers do not load a loudspeaker setup. By default, they assume the listener to reside in the coordinate origin looking straight forward.

3.1.2 A Note on the Timing of the Audio Signals

The WFS renderer is the only renderer in which the timing of the audio signals is somewhat peculiar. None of the other renderers imposes any algorithmic delay on individual source signals. Of course, if you use a renderer which is convolution based such as the BRS renderer, the employed HRIRs do alter the timing of the signals due to their inherent properties.

This is different with the WFS renderer. Here, also the propagation duration of sound from the position of the virtual source to the loudspeaker array is considered. That means that the farther a virtual source is located, the longer is the delay imposed on its input signal. This also holds true for plane waves: Theoretically, plane waves do originate from infinity. Though, the SSR does consider the origin point of the plane wave which is specified in ASDF. This origin point also specifies the location of the symbol which represents the respective plane wave in the GUI.

We are aware that this procedure can cause confusion and reduces the ability of a given scene of translating well between different types of renderers. In the upcoming version 0.4 of the SSR we will implement an option that will allow you specifying for each individual source whether the propagation duration of sound shall be considered by a renderer or not.

3.1.3 Distance Attenuation

Note that in all renderers – except the BRS renderer – distance attenuation is handled as $1/r$ with respect to the distance r of the respective virtual source to the reference position. Sources closer than 0.5 mtrs to the reference position do not experience any increase of amplitude. Virtual plane waves do not experience any algorithmic distance attenuation in any renderer. In future versions of the SSR more freedom in specifying the distance attenuation will be provided.

The amplitude reference distance, i.e. the distance from the reference at which plane waves are as loud as the other source types (like point sources), can be set in the SSR configuration file (Section 2.6). The desired amplitude reference distance for a given sound scene can be specified in the scene description (Section 1.4). The default value is 3 m.

3.1.4 Doppler Effect

In the current version of the SSR the Doppler Effect in moving sources is not supported by any of the renderers.

3.1.5 Signal Processing

All rendering algorithms are implemented on a frame-wise basis with an internal precision of 32 bit floating point. The signal processing is illustrated in Fig. 2.

The input signal is divided into individual frames of size *nframes*, whereby *nframes* is the frame size with which JACK is running. Then e.g. frame number $n+1$ is processed both with previous rendering parameters n as well as with current parameters $n+1$. It is then crossfaded between both processed frames with cosine-shaped slopes. In other words the effective frame size of the signal processing is $2 \cdot nframes$ with 50% overlap. Due to the fade-in of the frame processed with the current parameters $n+1$, the algorithmic latency is slightly higher than for processing done with frames purely of size *nframes* and no crossfade.

The implementation approach described above is one version of the standard way of implementing time-varying audio processing. Note however that this means that with *all* renderers, moving sources are not physically correctly reproduced. The physically correct reproduction of moving virtual sources as in [15, 16] requires a different implementation approach which is computationally significantly more costly.

3.2 Binaural Renderer

Binaural rendering is a technique where the acoustical influence of the human head is electronically simulated to position virtual sound sources in space. **Be sure that you use headphones to listen.** Note that the current binaural renderer reproduces all virtual sources exclusively as point sources.

The acoustical influence of the human head is coded in so-called head-related impulse responses (HRIRs). The HRIRs are loaded from the file `/usr/local/share/ssr/default_hrirs.wav`. If you want to use different HRIRs then use the `--hrirs=FILE` command line option or the SSR configuration file (Section 2.6) to specify your custom location. The SSR connects its outputs automatically to outputs 1 and 2 of your sound card.

For virtual sound sources which are closer to the reference position (= the listener position) than 0.5 mtrs, the HRTFs are interpolated with a Dirac impulse. This ensures a smooth transition of virtual sources from the outside of the listener's head to the inside.

SSR uses HRIRs with an angular resolution of 1°. Thus, the HRIR file contains 720 impulse responses (360 for each ear) stored as a 720-channel .wav-file. The HRIRs all have to be of equal length and have to be arranged in the following order:

- 1st channel: left ear, virtual source position 0°
- 2nd channel: right ear, virtual source position 0°
- 3rd channel: left ear, virtual source position 1°
- 4th channel: right ear, virtual source position 1°

  ...

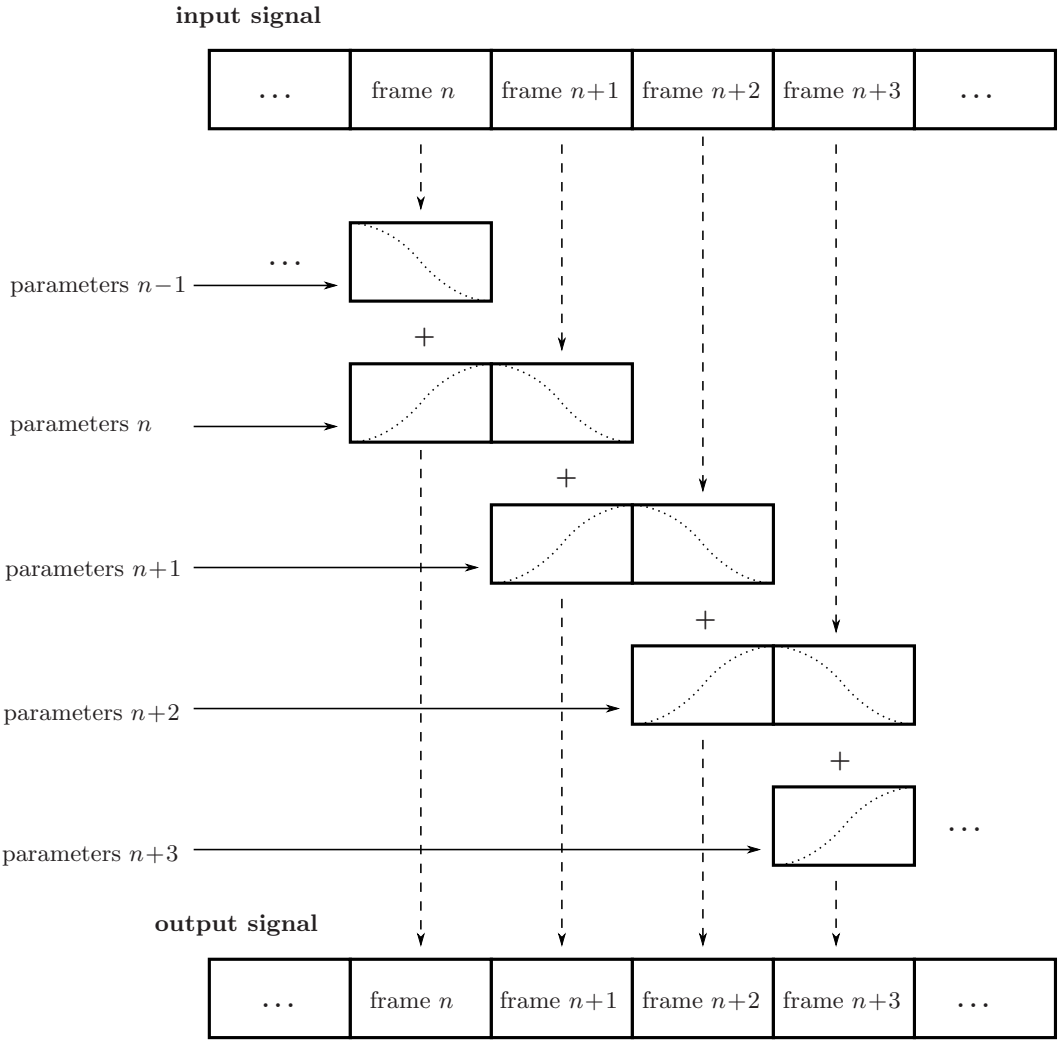


Figure 2: Illustration of the frame-wise signal processing as implemented in the SSR renderers (see text).

- 720th channel: right ear, virtual source position 359°

If your HRIRs have lower angular resolution you have to interpolate them to the target resolution or use the same HRIR for serveral adjacent directions in order to fulfill the format requirements. Higher resolution is not supported. Make sure that the sampling rate of the HRIRs matches that of JACK. So far, we know that both 16bit and 24bit word lengths work.

The SSR automatically loads and uses all HRIR coefficients it finds in the specified file. You can use the `--hrir-size=VALUE` command line option in order to limit the number of HRIR coefficients read and used to `VALUE`. You don't need to worry if your specified HRIR length `VALUE` exceeds the one stored in the file. You will receive a warning telling you what the score is. The SSR will render the audio in any case.

The actual size of the HRIRs is not restricted (apart from processing power). The SSR cuts them into partitions of size equal to the JACK frame buffer size and zero-pads the last partition if necessary.

Note that there's some potential to optimize the performance of the SSR by adjusting the JACK frame size and accordingly the number of partitions when a specific number of HRIR taps are desired. The least computational load arises when the audio frames have the same size like the HRIRs. By choosing shorter frames and thus using partitioned convolution the system latency is reduced but computational load is increased.

The HRIRs `impulse_responses/hrirs/hrirs_fabian.wav` we have included in the SSR are HRIRs of 512 taps of the FABIAN mannequin [17] in an anechoic environment. See the file `hrirs_fabian_documentation.pdf` for details of the measurement.

Preparing HRIR sets You can easily prepare your own HRIR sets for use with the SSR by adopting the MATLAB [18] script `data/matlab_scripts/prepare_hrirs_kemar.m` to your needs. This script converts the HRIRs of the KEMAR mannequin included in the CIPIC database [19] to the format which the SSR expects. See the script for further information and how to obtain the raw HRIRs.

3.3 Binaural Room Synthesis Renderer

The Binaural Room Synthesis (BRS) renderer is a binaural renderer (refer to Section 3.2) which uses one dedicated HRIR set of each individual sound source. The motivation is to have more realistic reproduction than in simple binaural rendering. In this context HRIRs are typically referred to as binaural room impulse responses (BRIRs).

Note that the BRS renderer does not consider any specification of a virtual source's position. The positions of the virtual sources (including their distance) are exclusively coded in the BRIRs. Consequently, the BRS renderer does not apply any distance attenuation. It only applies the respective source's gain and the master volume. No interpolation with a Dirac as in the binaural renderer is performed for very close virtual sources. The only quantity which is explicitely considered is the orientation of the receiver, i.e. the reference. Therefore, specification of meaningful source and receiver positions is only necessary when a correct graphical illustration is desired.

The BRIRs are stored in the a format similar to the one for the HRIRs for the binaural renderer (refer to Section 3.2). However, there is a fundamental difference: In order to be consequent, the different channels do not hold the data for different positions of the virtual sound source but they hold the information for different head orientations. Explicitely,

- 1st channel: left ear, head orientation 0°
- 2nd channel: right ear, head orientation 0°
- 3rd channel: left ear, head orientation 1°
- 4th channel: right ear, head orientation 1°

  ...

- 720th channel: right ear, head orientation 359°

In order to assign a set of BRIRs to a given sound source an appropriate scene description in `.asd`-format has to be prepared (refer also to Section 1.3). As shown in `brs_example.asd` (from the example scenes), a virtual source has the optional property `properties_file` which holds the location of the file containing the desired BRIR set. The location to be specified is relative to the folder of the scene file. Note that – as described above – specification of the virtual source's position does not affect the audio processing. If you do not specify a BRIR set for each virtual source, then the renderer will complain and refuse processing the respective source.

We have measured the binaural room impulse responses of the FABIAN mannequin [17] in one of our mid-size meeting rooms called Sputnik with 8 different source positions. Due to the file size, we have not included them in the release. Please contact *SoundScapeRenderer@telekom.de* to obtain the data.

3.4 Binaural Playback Renderer

The binaural playback (BPB) renderer is actually not a renderer but a playback engine that enables real-time head-tracking in headphone playback. It is similar to BRS with the only difference that it does not employ impulse responses that are applied to the input signal. It is rather such that the entire signals for the two ears for all desired possible head orientations have to be precomputed and are then loaded into the memory. During playback, depending on the instantaneous head orientation of the listener as measured by the tracking system, the corresponding audio data are replayed. If a change in head orientation occurs then a crossfade is applied over the duration of one JACK frame. Playing is automatically looped. To stop replay, mute the source. When the source is unmuted, replay starts at the beginning of the data.

The BPB renderer was designed for the simulation of time-varying systems, which are complicated to implement in real-time. The audio signals can be prepared in any desired software and also costly algorithms that do not run in real-time can be replayed with head-tracking.

As shown in the example `bin/scenes/bpb_example.asd` and similar to the description of a BRS scene, a virtual source has the optional property `properties_file`, which holds the location of the file containing the audio data. By default, it is assumed that the data are stored in a 720-channel audio file the channels of which are arranged similarly to BRS impulse responses.

Loading all 720 channels into memory can result in hundreds of megabytes even for signals of moderate length. In order to avoid restrictions due to the available memory caused by possibly unrequired data it is possible to restrict the interval of head orientations. This restriction has to be applied symmetrically, e.g. ±60°. The resolution between the limits is still 1°. The channel arrangement for the ±60° example would then be

- 1st channel: left ear, head orientation 0°
- 2nd channel: right ear, head orientation 0°
- 3rd channel: left ear, head orientation 1°
- 4th channel: right ear, head orientation 1°

  ...

- 121st channel: left ear, head orientation 60°
- 122nd channel: right ear, head orientation 60°
- 123rd channel: left ear, head orientation 300° (i.e. -60°)
- 124th channel: right ear, head orientation 300° (i.e. -60°)
- 125th channel: left ear, head orientation 301° (i.e. -59°)
- 126th channel: right ear, head orientation 301° (i.e. -59°)

  ...

- 242nd channel: right ear, head orientation 359° (i.e. -1°)

resulting thus in 242 channels. It is not necessary to explicitly specify the desired interval of possible head orientations. The SSR deduces it directly from the number of channels of the `properties_file`. If the listener turns the head to orientations for which no data are available the BPB renderer automatically replays the data for the closest orientation available. We assume that this is less disturbing in practice than a full dropout of the signal.

To fulfill the ASDF syntax, the specification of an input signal is required. In order to avoid the unnecessary opening and replaying of an audio file, we propose to specify an arbitrary input port such as

```
<source name="source" properties_file="../audio/binaural_data.wav">
  <!-- this is arbitrary -->
  <port>0</port>
  <!-- this only influences the GUI -->
  <position x="-2" y="2"/>
</source>
```

3.5 Vector Base Amplitude Panning Renderer

The Vector Base Amplitude Panning (VBAP) renderer uses the algorithm described in [20]. It tries to find a loudspeaker pair between which the phantom source is located (in VBAP you speak of a phantom source rather than a virtual one). If it does find a loudspeaker pair whose angle is smaller than 180° then it calculates the weights g_l and g_r for the left and right loudspeaker as

$$g_{l,r} = \frac{\cos\phi \sin\phi_0 \pm \sin\phi \cos\phi_0}{2 \cos\phi_0 \sin\phi_0} \ .$$

ϕ_0 is half the angle between the two loudspeakers with respect to the listening position, ϕ is the angle between the position of the phantom source and the direction "between the loudspeakers".

If the VBAP renderer can not find a loudspeaker pair whose angle is smaller than 180° then it uses the closest loudspeaker provided that the latter is situated within 30°. If not, then it does not render the source. If you are in verbosity level 2 (i.e. start the SSR with the `-vv` option) you'll see a notification about what's happening.

Note that all virtual source types (i.e. point and plane sources) are rendered as phantom sources.

Contrary to WFS, non-uniform distributions of loudspeakers are ok here. Ideally, the loudspeakers should be placed on a circle around the reference position. You can optionally specify a delay for each loudspeakers in order to compensate some amount of misplacement. In the ASDF (refer to Section 1.4), each loudspeaker has the optional attribute `delay` which determines the delay in seconds to be applied to the respective loudspeaker. Note that the specified delay will be rounded to an integer factor of the temporal sampling period. With 44.1 kHz sampling frequency this corresponds to an accuracy of 22.676 μs, respectively an accuracy of 7.78 mm in terms of loudspeaker placement. Additionally, you can specify a weight for each loudspeaker in order to compensate for irregular setups. In the ASDF format (refer to Section 1.4), each loudspeaker has the optional attribute `weight` which determines the linear (!) weight to be applied to the respective loudspeaker. An example would be

```
<loudspeaker delay="0.005" weight="1.1">
        <position x="1.0" y="-2.0"/>
        <orientation azimuth="-30"/>
</loudspeaker>
```

Delay defaults to 0 if not specified, weight defaults to 1.

Although principally suitable, we do not recommend to use our amplitude panning algorithm for dedicated 5.1 (or comparable) mixdowns. Our VBAP renderer only uses adjacent loudspeaker pairs for panning which does not exploit all potentials of such a loudspeaker setup. For the mentioned formats specialized panning processes have been developed also employing non-adjacent loudspeaker pairs if desired.

The VBAP renderer is rather meant to be used with non-standardized setups.

3.6 Wave Field Synthesis Renderer

The Wave Field Synthesis (WFS) renderer is the only renderer so far which discriminates between virtual point sources and plane waves. It implements the simple driving function given in [21]. Note that we have only implemented a temporary solution to reduce artifacts when virtual sound sources are moved. This topic is subject to ongoing research. We will work on that in the future. In the SSR configuration file (Section 2.6) you can specify an overall predelay (this is necessary to render focused sources) and the overall length of the involved delay lines. Both values are given in samples.

Prefiltering As you might know, WFS requires a spectral correction additionally to the delay and weighting of the input signal. Since this spectral correction is equal for all loudspeakers, it needs to be performed only once on the input. We are working on an automatic generation of the required filter. Until then, we load the impulse response of the desired filter from a .wav-file which is specified via the `--prefilter=FILE` command line option (see Section 2.5) or in the SSR configuration file (Section 2.6). Make sure that the specified audio file contains only one channel. Files with a differing number of channels will not be loaded. Of course, the sampling rate of the file also has to match that of the JACK server.

Note that the filter will be zero-padded to the next highest power of 2. If the resulting filter is then shorter than the current JACK frame size, each incoming audio frame will be divided into subframes for prefiltering. That means, if you load a filter of 100 taps and JACK frame size is 1024, the filter will be padded to 128 taps and prefiltering will be done in 8 cycles. This is done in order to save processing power since typical prefilters are much shorter than typical JACK frame sizes. Zero-padding the prefilter to the JACK frame size usually produces large overhead. If the prefilter is longer than the JACK frame buffer size, the filter will be divided into partitions whose length is equal to the JACK frame buffer size.

If you do not specify a filter, then no prefiltering is performed. This results in a boost of bass frequencies in the reproduced sound field.

In order to assist you in the design of an appropriate prefilter, we have included the MATLAB [18] script `data/matlab_scripts/make_wfs_prefilter.m` which does the job. In the very top of the file, you can specify the sampling frequency, the desired length of the filter as well as the lower and upper frequency limits of the spectral correction. The lower limit should be chosen such that the subwoofer of your system receives a signal which is not spectrally altered. This is due to the fact that only loudspeakers which are part of an array of loudspeakers need to be corrected. The lower limit is typically around

100 Hz. The upper limit is given by the spatial aliasing frequency. The spatial aliasing is dependent on the mutual distance of the loudspeakers, the distance of the considered listening position to the loudspeakers, and the array geometry. See [22] for detailed information on how to determine the spatial aliasing frequency of a given loudspeaker setup. The spatial aliasing frequency is typically between 1000 Hz and 2000 Hz. For a theoretical treatment of WFS in general and also the prefiltering, see [21].

The script `make_wfs_prefilter.m` will save the impulse response of the designed filter in a file like `wfs_prefilter_120_1500_44100.wav`. From the file name you can extract that the spectral correction starts at 120 Hz and goes up to 1500 Hz at a sampling frequency of 44100 Hz. Check the folder `data/impules_responses/wfs_prefilters` for a small selection of prefilters.

Tapering When the listening area is not enclosed by the loudspeaker setup, artifacts arise in the reproduced sound field due to the limited aperture. This problem of spatial truncation can be reduced by so-called tapering. Tapering is essentially an attenuation of the loudspeakers towards the ends of the setup. As a consequence, the boundaries of the aperture become smoother which reduces the artifacts. Of course, no benefit comes without a cost. In this case the cost is amplitude errors for which the human ear fortunately does not seem to be too sensitive.

In order to taper, you can assign the optional attribute `weight` to each loudspeaker in ASDF format (refer to Section 1.4). The `weight` determines the linear (!) weight to be applied to the respective loudspeaker. It defaults to 1 if it is not specified.

3.7 Ambisonics Amplitude Panning Renderer

The Ambisonics Amplitude Panning (AAP) renderer does very simple Ambisonics rendering. It does amplitude panning by simultaneously using all loudspeakers which are not subwoofers to reproduce a virtual source (contrary to the VBAP renderer which uses only two loudspeakers at a time). Note that the loudspeakers should ideally be arranged on a circle and the reference should be the center of the circle. The renderer checks for that and applies delays and amplitude corrections to all loudspeakers which are closer to the reference than the farthest. This also includes subwoofers. If you do not want close loudspeakers to be delayed, then simply specify their location in the same direction like its actual position but at a larger distance from the reference. Then the graphical illustration will not be perfectly aligned with the real setup, but the audio processing will take place as intended. Note that the AAP renderer ignores delays assigned to an individual loudspeaker in ASDF. On the other hand, it does consider weights assigned to the loudspeakers. This allows you to compensate for irregular loudspeaker placement.

Note finally that AAP does not allow to encode the distance of a virtual sound source since it is a simple panning renderer. All sources will appear at the distance of the loudspeakers.

If you do not explicitly specify an Ambisonics order, then the maximum order which makes sense on the given loudspeaker setup will be used. The automatically chosen

order will be one of $^{(L-1)}/_{2}$ for an odd number L of loudspeakers and accordingly for even numbers.

You can manually set the order via a command line option (Section 2.5) or the SSR configuration file (Section 2.6). We therefore do not explicitly discriminate between "higher order" and "lower order" Ambisonics since this is not a fundamental property. And where does "lower order" end and "higher order" start anyway?

Note that the graphical user interface will not indicate the activity of the loudspeakers since theoretically all loudspeakers contribute to the sound field of a virtual source at any time.

Conventional driving function By default we use the standard Ambisonics panning function outlined e.g. in [23] reading

$$d(\alpha_0) = \frac{\sin\left(\frac{2M+1}{2}\ (\alpha_0 - \alpha_\mathrm{s})\right)}{(2M+1)\ \sin\left(\frac{\alpha_0 - \alpha_\mathrm{s}}{2}\right)}\ ,$$

whereby α_0 is the polar angle of the position of the considered secondary source, α_s is the polar angle of the position of the virtual source, and M is the Ambisonics order.

In-phase driving function The conventional driving function leads to both positive and negative weights for individual loudspeakers. An object (e.g. a listener) introduced into the listening area can lead to an imperfect interference of the wave fields of the individual loudspeakers and therefore to an inconsistent perception. Furthermore, conventional Ambisonics panning can lead to audible artifacts for fast source motions since it can happen that the weights of two adjacent audio frames have a different algebraic sign.

These problems can be worked around when only positive weights are applied on the input signal (*in-phase* rendering). This can be accomplished via the in-phase driving function given e.g. in [23] reading

$$d(\alpha_0) = \cos^{2M}\left(\frac{\alpha_0 - \alpha_\mathrm{s}}{2}\right)\ .$$

Note that in-phase rendering leads to a less precise localization of the virtual source and other unwanted perceptions. You can enable in-phase rendering via the according command-line option or you can set the `IN_PHASE_RENDERING` property in the SSR configuration file (see section 2.6) to be "`TRUE`" or "`true`".

3.8 Generic Renderer

The generic renderer turns the SSR into a multiple-input-multiple-output convolution engine. You have to use an ASDF file in which the attribute `properties_file` of the individual sound source has to be set properly. That means that the indicated file has to be a multichannel file with the same number of channels like loudspeakers in the setup. The impulse response in the file at channel 1 represents the driving function for loudspeaker 1 and so on.

	individual delay	weight
binaural renderer	-	-
BRS renderer	-	-
VBAP renderer	+	+
WFS renderer	-	+
AAP renderer	autom.	+
generic renderer	-	-

Table 1: Loudspeaker properties considered by the different renderers.

	gain	mute	position	orientation[a]	model
binaural renderer	+	+	+	-	only ampl.
BRS renderer	+	+	-	-	-
VBAP renderer	+	+	+	-	only ampl.
WFS renderer	+	+	+	+	+
AAP renderer	+	+	+	-	only ampl.
generic renderer	+	+	-	-	-

[a]So far, only plane waves have a defined orientation.

Table 2: Virtual source's properties considered by the different renderers.

Be sure that you load a reproduction setup with the corresponding number of loudspeakers.

It is obviously not possible to move virtual sound sources since the loaded impulse responses are static. We use this renderer in order to test advanced methods before implementing them in real-time or to compare two different rendering methods by having one sound source in one method and another sound source in the other method.

Download the ASDF examples from [3] and check out the file `generic_renderer_example.asd` which comes with all required data.

3.9 Parallel Processing Renderers

The renderers as described above do not support parallel processing. We are currently redesigning the architecture of the SSR in order to support audio processing in multiple threads so that the power of multi-processor/multi-core machines can be properly exploited. The current SSR release contains versions of the WFS, the VBAP, and the binaural renderer which support parallel processing. These versions are disabled at compile time by default. If you want to enable these renderers use the option `./configure --enable-newrenderer` (Section 2.3) when configuring. All renderers other than WFS, VBAP, and binaural will then not be available.

WARNING: The parallel processing renderers are under heavy development. If

you encounter unexpected behaviour or bugs, please report them to *SoundScapeRenderer@telekom.de*. Thank you.

3.10 Summary

Tables 1 and 2 summarize the functionality of the SSR renderers.

4 Graphical User Interface

Our graphical user interface (GUI) is quite costly in terms of computation. So we emphatically recommend that you **properly configure the hardware acceleration of your graphics card**. If you still have performance issues make the window as small as possible. The smaller the window is the less is the processing cost.

The SSR GUI tries to enable samplebuffer support to enable anti-aliasing of the screen output. It will tell you if it didn't work out. Check Fig. 3 to get an idea of the influence of anti-aliasing. One day we will also implement a variable frequency for the screen update so that you can slow it down if CPU load is too high. Of course it won't look as nice then.

Figure 3: No anti-aliasing on the left image.

4.1 General Layout

The graphical user interface (GUI) consists mainly of an illustration of the scene that you are hearing and some interaction tools. The renderer type is indicated in the window title. See a screen shot in Fig. 4.

On the top left you will find the file menu where you can open files, save scenes, and quit the application. So far only the *save scene as...* option is available. That means every time to save the current scene you will be asked to specify the file name. This will be made more convenient in the future.

Next to the file menu, there is a button which lets you activate and deactivate the audio processing. Deactivating the audio processing does not necessarily lower the CPU load. It means rather that the SSR won't give any audio output, neither for involved audio files nor for live inputs.

Next to the processing button, you find the transport section with buttons to skip back to the beginning of a scene, pause replaying, and continue/start playing. Note that pausing a scene does not prevent live inputs from being processed. To prevent audio output switch off processing (see above). You may also replay while processing is switched off to navigate to a certain point in time in the respective scene.

In the top middle section of the GUI there is the audio scene time line. By default, it shows a time interval of two minutes duration. Whenever the progress exceeds the displayed time interval the latter is shifted such that the progress is always properly indicated. Below the handle, there is a numerical indication of the elapsed time with respect to the beginning of the scene. See Sec. 4.2 for information on how to operate on the time line.

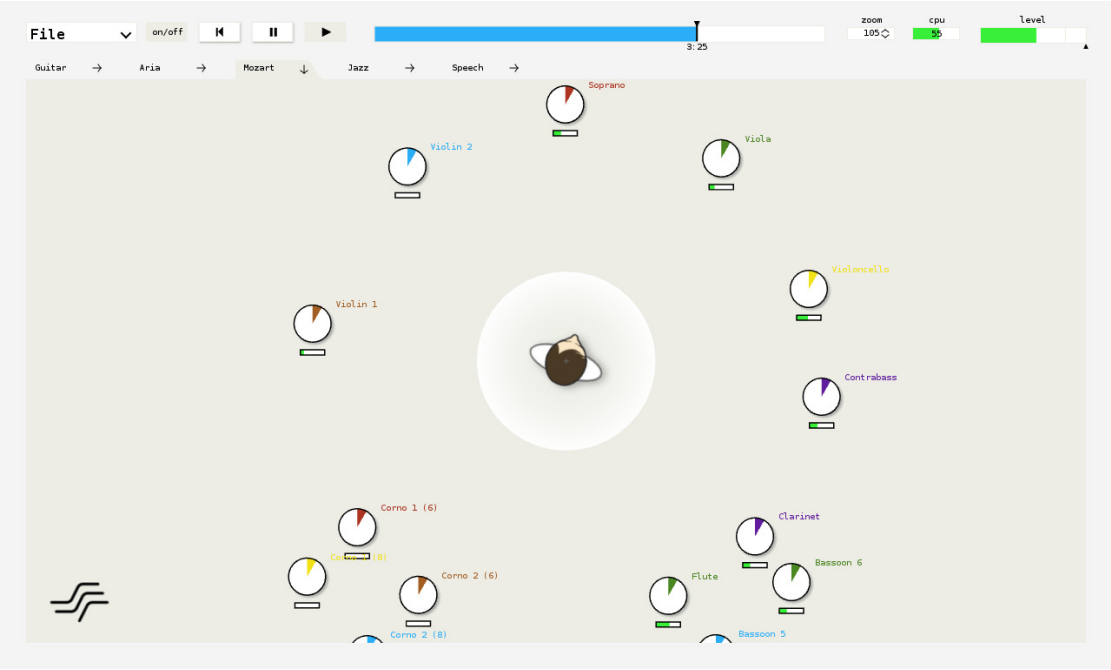


Figure 4: Screen shot of the SSR GUI.

To the right of the time line there's the CPU load gauge. It displays the average CPU load as estimated by the JACK audio server on a block-wise basis. Further right there's the label to indicate the current zoom factor in percent.

And finally, on the top right you find the master level meter combined with the master volume fader. The colored bar indicates an estimation of the relative maximum audio level in dB, also updated block-wise. The left boundary of the meter is at -50 dB; the right boundary is at +12 dB. The black triangle below the colored bar indicates the master volume in dB. Click somewhere into the widget and the master volume gets additionally displayed as a number. Note that this meter displays full scale, i.e. above 0 dB clipping and thus distortion of the output signal occurs! 0 dB is indicated by a thin vertical line.

In the row below the transport section, you occasionally find some tabs giving fast access to a number of scenes. These tabs can be defined in a file. By default, the file `scene_menu.conf` in the current working directory is assumed; there is also an option to specify the file name in the SSR configuration file. Refer to Sec. 2.6. The configuration file for the tabs may contain something like the following:

```
# This file configures the menu for the scene selection.
#
scenes/dual_mono.asd Guitar######### comments are possible
scenes/jazz.asd Jazz
```

```
scenes/rock.asd Rock
#scenes/speech.asd Speech
scenes/live_conference.xml live conference
```

The syntax is as follows:

- Everything after a hash symbol (#) in a line is ignored.
- A valid entry consists of the path (relative or absolute) to ASDF file (or pure audio file) followed by space and a short keyword that will be displayed on the respective tab on the screen.

Of course, also audio files can be specified instead of `.asd`s. Note that so far, no syntax validation is performed, so watch your typing. We furthermore recommend that you keep the keywords short because space on the screen is limited. Note also that only those tabs are displayed which fit on the screen.

The SSR always tries to find the file `scene_menu.conf` in its current working directory (or at the location specified in the SSR configuration file). If is does not find it no tabs will be displayed in the GUI. So you can have several of such files at different locations. We have added an example in folder `data/`.

The main part of the screen is occupied by the graphical illustration of the scene that you are hearing. The orientation of the coordinate system is exactly like depicted in Fig. 1. I.e., the x-axis points to the right of the screen, the y-axis points to the top of the screen. The origin of the coordinate system is marked by a cross, the reference is marked by a rhomb. The direction "straight in front" is typically assumed to be vertically upwards on the screen, especially for binaural techniques. We do so as well. Note that in this case "straight in front" means $\alpha = 90°$ and NOT $\alpha = 0°$.

In Fig. 4 you see a number of sound sources with their individual audio level meters (combined with their individual volume sliders) underneath. The left hand boundary of the level meter is at -50 dB; the right hand boundary is at 0 dB. Spherical sources don't have any additional decoration. The wave front and propagation direction of plane waves are indicated.

You also see icons for the loudspeakers of the current rendering setup (if the currently applied technique employs any).

4.2 Mouse Actions

The GUI is designed such that the most important functionalities can be accessed via a touch screen. Thus, it mostly employs 'left clicks' with the mouse.

The use of the file and transport section is rather intuitive so we won't further explain it here. The time line can be used to jump to a certain position within the sound scene and it also shows the progress of the scene. Click into the white/blue area of the time line in order to jump to a specific point in time, or drag the handle to fast forward or rewind. Left-clicking to the right of the time line skips forward by 5 seconds, left-clicking to the left of the time line skips back by 5 seconds. Double-clicking on the time line skips back to the beginning of the scene. Right-clicking on the time line opens an input

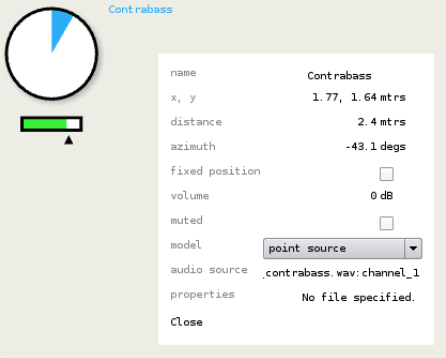


Figure 5: Source properties dialog

window in order that you can numerically specify the time instant to jump to (refer to Sec. 4.3).

You can change the zoom either by clicking into the zoom label and dragging up or down for zooming in or out. Alternatively, you can use the mouse wheel. Clicking and dragging on the background of the screen lets you move inside the scene. A double-click brings you back to the default position and also defaults the zoom.

Clicking and dragging on a sound source lets you select and move it. Note that you cannot directly manipulate the propagation direction of plane waves. It's rather such that plane sources always face the reference point. To change their direction of incidence move the plane wave's origin point to the appropriate position. Right clicking on a sound source opens a window which lists the properties of the source such as position, volume, etc. Refer to Fig. 5 and Sec. 4.2.1.

A right mouse click on the scene background lets you select multiple sound sources via a rubber band.

If you hold the `Ctrl` key pressed during any mouse action then you operate on all selected sound sources at the same time (i.e. mute, move, etc. them).

Click on the SSR logo and you'll see the *About the SSR* information.

4.2.1 Source Properties Dialog

The source properties dialog can be accessed via a right click on a source and shows information about the actual state of the selected source. Its main purpose is to provide the possibility of an exact positioning of sources. The properties `fixed position`, `muted` and `model` can be changed. Please refer to figure 5 to see the complete list of properties this dialog shows.

4.3 Keyboard Actions

A number of keyboard actions have been implemented as listed below. Recall that also some keyboard actions are available when the SSR is run without GUI (refer to Sec. 2.5).

`+/-`: if no sound source is selected: raise/lower master volume by 1dB, otherwise raise/lower the selected sources' volume by 1dB

`Arrow up/down/left/right`: navigate in scene

`Space`: toggles the play/pause state

`Backspace`: skip to beginning of scene

`Return`: calibrate tracker (if present). When pressed, the instantaneous orientation is assumed to be straight forward (i.e. 90° azimuth)

`Ctrl`: when pressed, multiple sound sources can be selected via mouse clicks or operations can be performed on multiple sources simultaniously

`Ctrl+Alt`: individual sound sources can be deselected from a larger selection via a mouse click or the rubber band

`Ctrl+a`: select all sources

`f`: toggles the position-fix-state of all selected sound sources (sources which can not be moved are marked with a little cross)

`m`: toggles the mute state of all selected sound sources (muted sources are displayed with a grey frame instead of a black one)

`p`: toggles the source model between *plane wave* and *point source*

`s`: if no source selected: unsolos all potentially soloed sources, otherwise: solos selected sound sources.

`Ctrl+s`: opens the *save scene as...* dialog

`F11`: toggles window fullscreen state

`1-9`: select source no. 1-9

`0`: deselect all sources

`Ctrl+c`: quit

`Ctrl+t`: open text edit for time line. The format is `hours:mins(2digits):secs(2digits)` whereby `hours:` and `hours:mins(2digits):` can be omitted if desired.

`Esc`: quit

5 Network Interface

This is just a short overview about the XML messages which can be sent to the SSR via TCP/IP. The messages have to be terminated with a binary zero (`\0`).

WARNING: The network interface is under heavy development and the XML messages will very likely change until the next release! Furthermore, we did not evaluate the network interface in terms of security. So please be sure that you are in a safe network when using it.

5.1 Scene

- Load Scene:
  `<request><scene load="path/to/scene.asd"/></request>`
- Clear Scene (remove all sources):
  `<request><scene clear="true"/></request>`
- Set Master Volume (in dB):
  `<request><scene volume="6"/></request>`

5.2 State

- Start processing:
  `<request><state processing="start"/></request>`
- Stop processing:
  `<request><state processing="stop"/></request>`
- Transport Start (Play):
  `<request><state transport="start"/></request>`
- Transport Stop (Pause):
  `<request><state transport="stop"/></request>`
- Transport Rewind:
  `<request><state transport="rewind"/></request>`
- Transport Locate:
  `<request><state seek="4:33"/></request>`
  `<request><state seek="1.5 h"/></request>`
  `<request><state seek="42"/></request>` *(seconds)*
  `<request><state seek="4:23:12.322"/></request>`
- Reset/Calibrate Head-Tracker:
  `<request><state tracker="reset"/></request>`

5.3 Source

- Set Source Position (in meters):
```
<request><source id="42"><position x="1.2" y="-2"/></source></request>
```
- Fixed Position (`true`/`false`):
```
<request><source id="42"><position fixed="true"/></source></request>

<request><source id="42">
  <position x="1.2" y="-2" fixed="true"/>
</source></request>
```
- Set Source Orientation (in degrees, zero in positive x-direction):
```
<request><source id="42"><orientation azimuth="93"/></source></request>
```
- Set Source Gain (Volume in dB):
```
<request><source id="42" volume="-2"/></request>
```
- Set Source Mute (`true`/`false`):
```
<request><source id="42" mute="true"/></request>
```
- Set Source Name:
```
<request><source id="42" name="My first source" /></request>
```
- Set Source Model (`point`/`plane`):
```
<request><source id="42" model="point"/></request>
```
- Set Source Port Name (any JACK port):
```
<request><source id="42" port_name="system:capture_3"/></request>
```
- New Source (some of the parameters are optional):
```
<request>
  <source new="true" name="a new source"
      file="path/to/audio.wav" channel="2">
    <postition x="-0.3" y="1" fixed="true"/>
    <orientation azimuth="99"/>
  </source>
</request>


<request>
  <source new="true" name="a source from pd"
      port="pure_data_0:output0" volume="-6">
    <postition x="0.7" y="2.3"/>
  </source>
```

```
</request>
```

- Delete Source:
  `<request><delete><source id="42"/></delete></request>`

5.4 Reference

- Set Reference Position (in meters):
  `<request><reference><position x="-0.3" y="1.1"/></reference></request>`
- Set Reference Orientation (in degrees, zero in positive x-direction):
  `<request><reference><orientation azimuth="90"/></reference></request>`

References

[1] M. Geier, J. Ahrens, and S. Spors. The SoundScape Renderer: A unified spatial audio reproduction framework for arbitrary rendering methods. In *124th AES Convention*, Amsterdam, The Netherlands, May 2008. Audio Engineering Society (AES).

[2] M. Geier, J. Ahrens, and S. Spors. ASDF: Ein XML Format zur Beschreibung von virtuellen 3D-Audioszenen. In *34rd German Annual Conference on Acoustics (DAGA)*, Dresden, Germany, March 2008.

[3] Quality & Usability Lab, Deutsche Telekom Laboratories, TU Berlin. The SoundScape Renderer. `http://tu-berlin.de/?id=ssr`.

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

[5] Matteo Frigo and Steven G. Johnson. FFTW3. `http://www.fftw.org`.

[6] Erik de Castro Lopo. libsndfile. `http://www.mega-nerd.com/libsndfile`.

[7] Kai Vehmanen. Ecasound. `http://eca.cx/ecasound`.

[8] Trolltech. Qt4. `http://doc.trolltech.com/4.2`.

[9] Daniel Veillard. Libxml2. `http://xmlsoft.org`.

[10] Boost C++ Libraries. `http://www.boost.org`.

[11] Rui Nuno Capela et al. JACK Audio Connection Kit - Qt GUI Interface. `http://qjackctl.sourceforge.net/`.

[12] InterSense Inc. `http://www.isense.com`.

[13] Polhemus Fastrack. `http://www.polhemus.com/?page=Motion_Fastrak`.

[14] Paul Davis. Ardour. `http://www.ardour.org`.

[15] J. Ahrens and S. Spors. Reproduction of moving virtual sound sources with special attention to the doppler effect. In *124th Convention of the AES, Amsterdam, The Netherlands*, May 17–20, 2008.

[16] J. Ahrens and S. Spors. Reproduction of virtual sound sources moving at supersonic speeds in Wave Field Synthesis. In *125th Convention of the AES, San Francisco, CA*, Oct. 2–5, 2008.

[17] Alexander Lindau and Stefan Weinzierl. FABIAN - Schnelle Erfassung binauraler Raumimpulsantworten in mehreren Freiheitsgraden. In *Fortschritte der Akustik, DAGA Stuttgart*, 2007.

[18] The MathWorks, Inc. Matlab. `http://www.mathworks.com`.

[19] R. Algazi. The CIPIC HRTF database. `http://interface.cipic.ucdavis.edu/CIL_html/CIL_HRTF_database.htm`.

[20] V. Pulkki. Virtual sound source positioning using Vector Base Amplitude Panning. In *Journal of the Audio Engineering Society (JAES)*, Vol.45(6), June 1997.

[21] S. Spors, R. Rabenstein, and J. Ahrens. The theory of Wave Field Synthesis revisited. In *124th Convention of the AES, Amsterdam, The Netherlands*, May 17–20, 2008.

[22] S. Spors and R. Rabenstein. Spatial aliasing artifacts produced by linear and circular loudspeaker arrays used for Wave Field Synthesis. In *120th Convention of the AES, Paris, France*, May 20–23, 2006.

[23] M. Neukom. Ambisonic panning. In *123th Convention of the AES, New York, NY, USA*, Oct. 5–8, 2007.