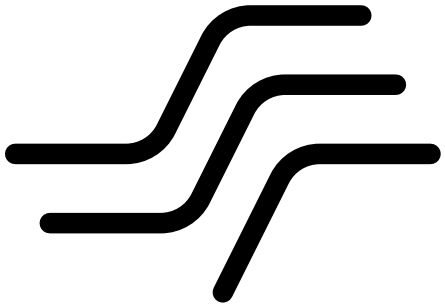
Introduction to the SoundScape Renderer (SSR)

Jens Ahrens, Matthias Geier and Sascha Spors

SoundScapeRenderer@telekom.de

May 21, 2010



This document gives a general overview of the architecture and functionality of the SoundScape Renderer (SSR). Please note that both this document as well as the SSR are work in progress.

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

Contents

1 General stuff

1.1 Disclaimer

1.2 Introduction

The SoundScape Renderer (SSR) is a software framework for realtime spatial audio reproduction running exclusively under Linux. Mac OS support is in preparation. The current implementation provides Wave Field Synthesis (WFS), binaural (HRTF-based) reproduction, binaural room (re-)synthesis (HRIR-based reproduction), Ambisonics amplitude panning (AAP), and Vector Base Amplitude Panning (VBAP). The rendering algorithm is chosen upon execution of the software.

The SSR is thought 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. 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.

1.3 Quick start

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

```
tar xvfz ssr-x.x.x.tar.gz
cd ssr-x.x.x
./configure
make
make install
qjackctl &
ssr YOUR_AUDIO_FILE
```

With above commands you are performing the following steps:

- Unpack the distributed tar-ball containing the source-code

- Go to the extracted directory [1].
- Configure the SSR
- Install the SSR
- Open the graphical user interface for JACK. Please click "Start" to start the server. As alternative you can start JACK with `jackd -d alsa -r 44100 -p 512`
- Open the SSR with a audio-file of your choice.

If compiled appropriately, SSR will start with the binaural renderer. Please use headphones to listen to the generated output. It will load the audio file YOUR_AUDIO_FILE and create a virtual sound source for each channel that the audio file contains.

If your graphics card is not so powerful or you want to dedicate all your resources to the audio processing, try

```
./ssr -G YOUR_AUDIO_FILE
```

that disables the GUI and starts reproducing

1.4 Audio scenes

1.4.1 Format

The SSR can open `.asd`-files (refer to section 1.6) as well as pure audio files. If a pure 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 arbitrary channels, SSR randomly arranges the resulting virtual sound sources.

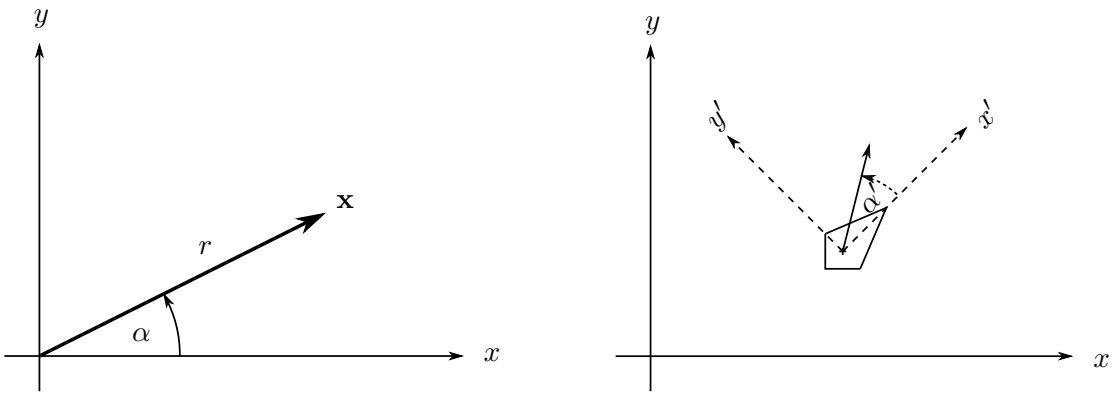
Audio file types which can be opened principally include all types that ecasound and libsndfile can open. In particular this includes .wav and .aif. Even mp3 could work, depending on the codecs installed.

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.

1.4.2 Coordinate system

Figure 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 figure 1(b). The loudspeakers are positioned with respect to the primed coordinates (x', y', etc.).

[1] Note that all relative paths which are mentioned in this document are relative to this folder, which is the folder where the SSR resides. Therefore, e.g. `src` could be something like `/home/YOUR_ACCOUNT/ssr-0.3.0/src`



(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 1.6).

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.

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 determines the way loudspeakers are driven. It is advisable to put the reference point to your preferred listening position. In the binaural methods the reference point reflects the listener indicating its position and orientation and has therefore even more fundamental impact on the reproduction.

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 extend. However, for the binaural methods it is essential that both the reference position and orientation (i.e. the listeners position and orientation) are tracked and updated in real-time. Refer also to section 2.7.

1.5 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 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 (`-m`).

1.6 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]. The ASDF is used to describe both audio scenes as well as the setup of the reproduction system (this might change in future versions). Since these two setups are described independently, mutual interference is avoided. 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. Furthermore, it is not possible to dynamically control the audio replay within a scene. 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**.

1.6.1 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>
```

And 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>
```

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>`.

1.6.2 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.

Examples for the most common loudspeaker setups like stereo and 5.1 can be found in `data/reproduction_setups` or `/usr/local/share/ssr/reproduction_setups` if you installed the SSR. use the `-s` command line option to load them. Open the files in a text editor to see what they contain. `data/loudspeaker_setups/loudspeaker_setup_with_nearly_all_features.asd` contains (nearly) all possible options.

More examples for audio scenes can be downloaded from [3].

Besides simple scene examples you will find the file `example_scenes/scene_with_nearly_all_features.asd` there which illustrates the possible features which have been implemented. Note that all examples can be directly loaded into the SSR when JACK runs at 44.1 kHz sampling rate. Each file contains a header with comments on what it contains. Open the files in a text editor and check.

1.7 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. However, the implementation is currently not fully tested. Therefore, we do not give explicit documentation about its use here. 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/`.

1.8 Tools

We have implemented some tools to use the remote control features of the SSR. They are currently in an early stage and therefore we have not included them in the release version. If you are interested in those for you own work please contact us.
Currently we have implemented:

- a PD patch that allows to control most of the SSR features
- basic tests with Ruby, ActionScript and a few other programming languages

1.9 Known problems

This is a brief outline of problems occuring in the use of the SSR. Note that this listing is far from being exhaustive.

If you discover other problems or bugs, please report them to *SoundScapeRenderer@telekom.de* and we will take care of them.

- Sometimes the SSR freezes when starting. In that case you have to restart the JACK audio server and try again.
- When you start the SSR with GUI everything is alright at first. As soon as you open a scene a segmentation fault arises. This is a problem in the interaction between Qt and OpenGL. As a workaround, comment the line
  `renderText(0.18f * scale, 0.13f * scale, 0.0f, source->name.c_str(), f);`
  in the file `src/gui/qopenglrenderer.cpp` and recompile the code. The consequence is that the names of the sound sources will not be displayed anymore.
- If you discover the following terminal output

```
**********************************************************************
* Message from libecasoundc:
*
* 'ECASOUND' environment variable not set. Using the default value
* value 'ECASOUND=ecasound'.
**********************************************************************
```

don't worry, that's harmless. But if it bothers you, you may want to add the following line to the configuration file of your terminal:
`export ECASOUND=ecasound`

If you don't know what's the configuration file, you could try the file `~/.bashrc`.

- If you discover the error

  ```
  Error: Coultn't connect 'VBAP-Renderer:out_1' with 'system:playback_1'!
  ```

  or similar then add the command line argument `--output-prefix=alsa_pcm:playback_`. This error occurs when the JACK ports of your soundcard have different names than the SSR expects (the SSR expects the output prefix `system:playback_`). By applying the option `--output-prefix` you change the naming that the SSR expects. If you are not sure about the naming on your machine, start the application `jack_lsp`. It will list the names of all active JACK ports.

1.10 Bug reports, feature requests and comments

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. ☺

1.11 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 release of the SSR:

- Possibility to create audio scenes using the GUI.
- Saving audio scenes.
- Possibility to apply a headphone compensation filter.
- Possibility to apply filter on subwoofers.
- Near-field compensated Ambisonics.
- Support of Mac OS X.

1.12 Contributors

Besides the authors of this document the following people have contributed to the development of the SSR (in alphabetical order):

- Katharina Bredies
- Florian Hinterleitner
- Torben Hohn
- Lukas Kaser
- Jonas Loh
- André Möhl

1.13 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 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 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 [3]. After downloading you can unpack the tar-ball with the command `tar xvzf filename.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.

At termination of the configure-script a summary will show up. Note that libraries indicated with "Found..." are required for compilation, but software in section "Build with..." is optional. For example the library "fftw" is required for running the SSR, but GUI or head-tracker libraries are not.

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 and OpenGL (QtCore, QtGui and QtOpenGL) [8]
- GLUT [9] or freeglut [10]
- libxml2 [11]
- Boost [12]: Boost.Asio library is needed, which has been included since version 1.35.

We provide a simple integration of two tracking systems mentioned in the following subsections. This sections will describe how to prepare your system to enable the tracker support of the SSR. 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` eventually.
- 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`. All headers in `yourpath` should be found and be available for a successful compilation.

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 will show a report. Please ensure that all libraries in section "Found..." are commented with "yes" before you install the SSR. The commands

```
make
make install
```

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/`.

If you want to compile, but not install the SSR, you can omit `make install`. Section 2.5 will show how to run it in this case.

Exact informations about configuring, compiling and running a standard GNU-package can be found in the provided INSTALL-file.

2.4 Cleaning up

This section shows different ways of cleaning our package:

- `make clean`: Files generated during compilation will be cleaned up.
- `make uninstall`: This will remove the SSR from your system.
- `make distclean`: Files generated with `./configure` get removed here.

2.5 Running the SSR

Before you start the SSR, start JACK [4], e.g. by typing
`jackd -d alsa -r 44100 -p 512` in a shell or using the graphical user interface "qjackctl" [13].

If you installed our package with `make install` the SSR can be started by typing `ssr` to a shell. 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 an enabled binaural-renderer; please use headphones for listening with this renderer.

If you only compiled the SSR without installing, you can start it with a shell script contained in our tar-ball. Go to the directory `data/` and type `./ssr.sh`. This will call the binary file and start the SSR. Note that all options that apply for the installed binary `ssr` apply also for this script, e.g. `./ssr.sh --help`

Type `ssr --help` to get an overview of the command line options and various renderers:

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

OPTIONS:
 -a, --vbap                Start with stereophonic (VBAP) renderer.
 -b, --binaural            Start with binaural renderer
 -c FILE, --config=FILE    Read configuration from FILE.
 -d, --aap                 Start with Ambisonics amplitude panning renderer.
 -e FILE, --prefilter=FILE Load WFS prefilter from FILE.
 -g, --gui                 Start GUI (default).
 -G, --no-gui              Don't start GUI.
 -i, --ip-server           Start IP server (default).
 -I, --no-ip-server        Don't start IP server.
 -j, --in-phase-rendering  Use in-phase rendering for Ambisonics.
 -k, --brs                 Start with BRS renderer.
 -l, --generic             Start with generic renderer
 -m VALUE, --mast-vol-corr Correction in dB applied to the master volume (default: 0).
 -n VALUE, --hrir-size     Max. length of HRIRs to be used by binaural and BRS renderer.
 -o VALUE, --ambi-order    Ambisonics order to use.
 -p FILE, --hrirs=FILE     Load the HRIRs for binaural rendering from FILE.
 -r FILE, --record=FILE    Record the audio output of the renderer to FILE.
 -s FILE, --setup=FILE     Load reproduction setup from FILE.
 -t, --tracker             Start tracker (default).
 -T, --no-tracker          Don't start tracker.
 -v, --verbose             Increase verbosity level.
```

```
-V, --version             Show version information and exit.
-w, --wfs                 Start with WFS renderer.
--input-prefix=PREFIX     Prefix of JACK  input ports. (def: "system:capture_")
--output-prefix=PREFIX    Prefix of JACK output ports. (def: "system:playback_")

--               All arguments after "--" are passed on to the GUI.
```

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.

Recording of the SSR output You can record the audio output of the SSR using the `-r 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 1.6). The recording can then be used to analyze the SSR output or to replay it without the SSR using a software player like "ecasound" [7].

2.6 Warnings and errors

Independent from the current verbosity level, the SSR gives some information about what happens by reporting warnings and occasionally also errors.

Warnings are not serious. They are rather status reports. Don't worry if you don't understand them.

When errors are reported, then something is really going wrong. Error reports are much less cryptic than some of our warnings, so you should understand what the score is. If not → *SoundScapeRenderer@telekom.de*.

2.7 Head tracking

We provide integration of the InterSense InertiaCube3 tracking sensor [14] and the Polhemus Fastrak [15]. 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 [14], please download the "InterSense Software Development Kit (SDK)" from

[14].

Unpack the archive and place the following files:

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

Our 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. If you want to disable this tracker, use `./configure --disable-intersense` and recompile.

2.7.2 Preparing Polhemus Fastrack

For incorporation of the Polhemus Fastrack [15] 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 1.6). 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 [16] support JACK and their use is therefore straight forward. 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`. 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 exactly in the same order as they are specified in the `.asd` file. The first specified loudspeaker appears at output channel 1, the second loudspeaker at output channel 2 etc ...
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 circles and linear arrays in a useful manner.
- 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 object `<skip number="5"/>` instead of a loudspeaker, then the channel numbers of all loudspeakers following the `skip` object will be higher by the value specified in the attribute `number` than the order of specification suggests.

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 Distance attenuation

Note that in all renderers - except for 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.
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 1.5). The desired amplitude reference distance for a given sound scene can be specified in the scene description (section 1.6). The default value is 3 m.

3.1.3 Doppler Effect

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

3.1.4 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 figure 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 [17, 18] 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 1.5) 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) 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°

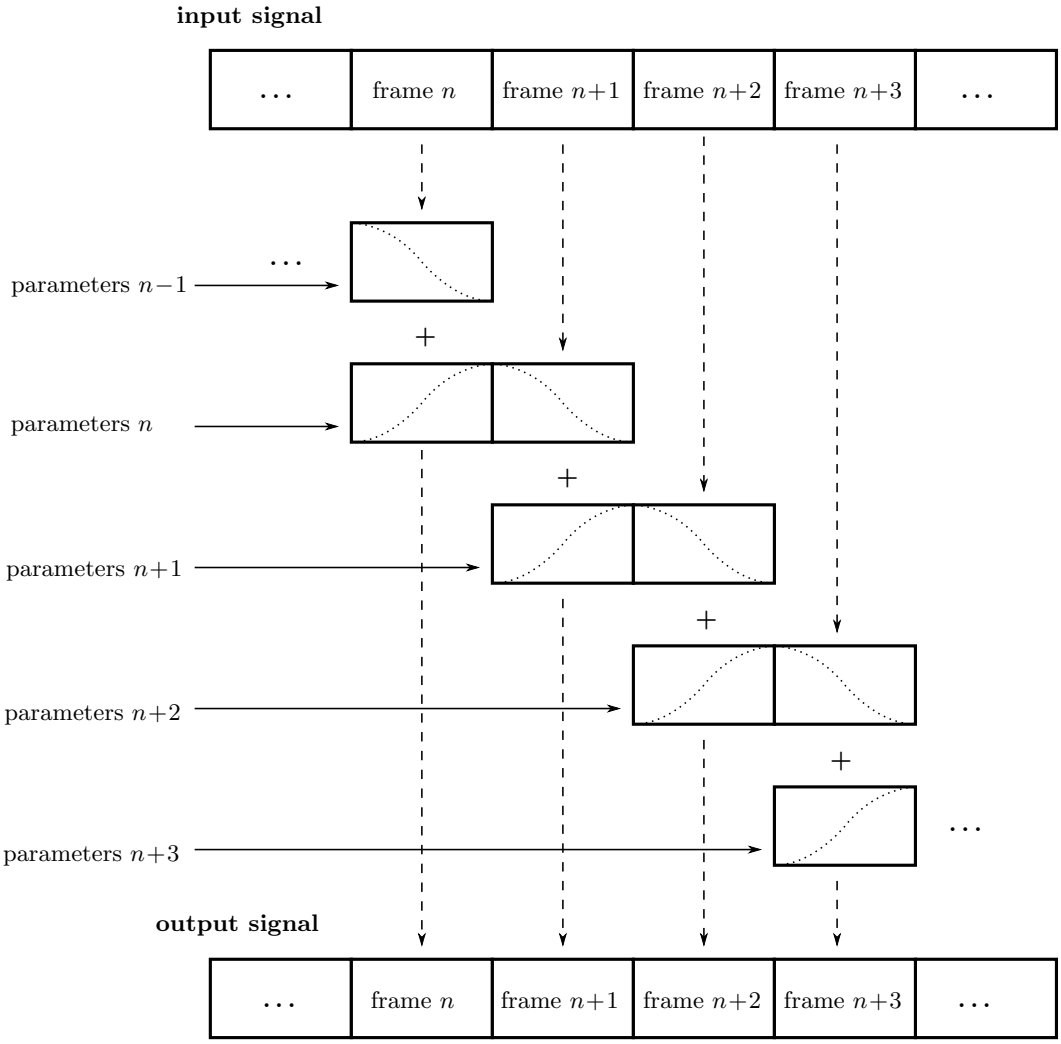


Figure 2: Illustration of the frame-wise signal processing as implemented in the SSR renderers (see text).

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

  ...

- 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 `-n 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). 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 [19] 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 [20] 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 [21] to the format which the SSR expects. See the script for further information and how to obtain the raw HRIRs.

3.3 Binaural Room Scanning renderer

The Binaural Room Scanning (BRS) renderer is a binaural renderer (confer 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 (confer also to section 1.4). 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 [19] 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 Vector Base Amplitude Panning renderer

The Vector Base Amplitude Panning (VBAP) renderer uses the algorithm described in [22]. 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 (confer to section 1.6), 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 (confer to section 1.6), 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.

3.5 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 [23]. 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 1.5) 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 1.5). 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 [20] 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 [24] 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 [23].
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 (confer to section 1.6). The `weight` determines the linear (!) weight to be applied to the respective loudspeaker. It defaults to 1 if it is not specified.

3.6 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 1.5). 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 [25] 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 [25] 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 `ssr.conf` to be "TRUE" or "true".

3.7 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.

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 realtime 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.

3.8 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 multicore 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.

Note however that the parallel processing renderers are untested and we are not so sure about what they do. The first thing that you will realize is that the overall signal level meter in the GUI will not work. But there might be more surprises.

3.9 Summary

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

	individual delay	weight
binaural renderer	-	-
BRS renderer	-	-
VBAP renderer	+	+
WFS renderer	-	+
APA 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	+	+	+	+	+
APA 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.

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**. You might have some trouble on notebooks. 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 figure 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 cource 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 figure 4.

On the top left you will find the file menu where you can open files and quit the application. Next to it, 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 section 4.2 for information on how to operate on the time line.

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 frame-wise basis.

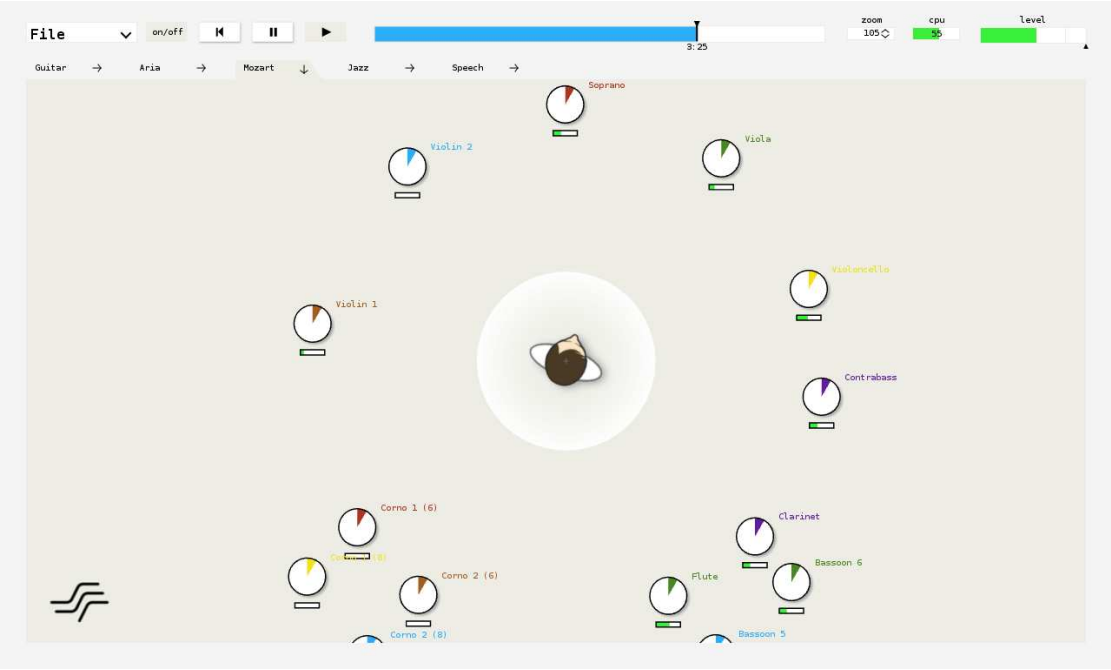


Figure 4: Screen shot of the SSR GUI.

Further right there's the label to indicate the current zoom factor in percent.

An 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 frame-wise. 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 are defined in a file called `scene_menu.conf`. This file might look like this:

```
# 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 dash (#) in a line is ignored.

- A valid entry consists of the path (relative or absolute) to the respective description file (or pure audio file) followed by space and a short keyword that is supposed to 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. 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 figure 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 figure 4 you see a number of sound sources with their individual audio level meters (combined with their individual volume sliders) underneath. 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). In figure 4 you see the system that is installed in our Usability laboratory. It is a ring of a diameter of a bad 3 meters hosting 56 equiangularly spaced loudspeakers plus a subwoofer.

The rhombus inside the loudspeaker ring indicates the location and orientation of the reference point of the current setup.

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

You can change the zoom either by clicking into the zoom label and dragging up respectively down for zooming in respectively out or by using 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 them to the appropriate nominal position. Right clicking on a sound source opens a window which lists the properties of the source such as position, volume, etc. .

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

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).

4.3 Keyboard actions

The following keyboard actions have been implemented:

`+/-`: 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.

`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>`

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. The SoundScape Renderer. http://www.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] OpenGL.org. Glut - the opengl utility toolkit. http://www.opengl.org/resources/libraries/glut.

[10] The free opengl utility toolkit. http://freeglut.sourceforge.net/.

[11] Daniel Veillard. Libxml2. http://xmlsoft.org.

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

[13] Paul Davis et al. JACK Audio Connection Kit - Qt GUI Interface. http://qjackctl.sourceforge.net/.

[14] InterSense Inc. http://www.isense.com.

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

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

[17] 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.

[18] 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.

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

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

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

[22] V. Pulkki. Virtual sound source positioning using vector base amplitude panning. In *Journal of the Audio Engineering Society (JAES)*, Vol.45(6), June 1997.

[23] 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.

[24] 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.

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