SoundScape Renderer

Release 0.6.1-24-ga5e05f7

SSR Team

2024-03-29

CONTENTS

The SoundScape Renderer (SSR) is a tool for real-time spatial audio reproduction providing a variety of rendering algorithms, e.g. Wave Field Synthesis, Higher-Order Ambisonics and binaural techniques.

website
http://spatialaudio.net/ssr/

downloads
http://spatialaudio.net/ssr/download/

user manual
https://ssr.readthedocs.io/

mailing list
https://groups.google.com/forum/#!forum/SoundScapeRenderer

source code and issue tracker
https://github.com/SoundScapeRenderer/ssr

CHAPTER

ONE

GENERAL

1.1 Introduction

The SoundScape Renderer (SSR) is a software framework for real-time spatial audio reproduction running under GNU/Linux, macOS and possibly some other UNIX variants. The MS Windows standalone version is experimental. The SSR renderers are available as externals for Pure Data for all operating systems. Refer to *SSR in Pure Data*.

The current implementation provides:

- *Binaural (HRTF-based) reproduction*
- *Binaural room (re-)synthesis (BRTF-based reproduction)*
- *Vector Base Amplitude Panning (VBAP)*
- *Wave Field Synthesis (WFS)*
- *Ambisonics Amplitude Panning (AAP)*

There is also the slightly exotic *Generic Renderer*, which is essentially a MIMO convolution engine. For each rendering algorithm, there is a separate executable file.

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 [Geier2008a] or [Geier2012].

Note that so far, the SSR only supports two-dimensional reproduction for most renderers (the binaural renderer with SOFA files being the laudable exception that already supports 3D). 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 from http://spatialaudio.net/ssr/download/, open a shell and use following commands:

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

You have to replace `x.y.z` with the current version number, e.g. `0.5.0`. 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.
- Build the SSR.
- Install the SSR.
- 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 *Running the SSR* 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. Please use headphones to listen to the output generated by the binaural renderer! If you want to use a different renderer, use `ssr-brs`, `ssr-wfs`, `ssr-vbap`, `ssr-aap`, `ssr-dca` or `ssr-generic` instead of `ssr-binaural`.

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

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

For further options, see the section *Running the SSR* and `ssr-binaural --help`.

1.3 Audio Scenes

1.3.1 Format

The SSR can open `.asd` files – refer to the section *Audio Scene Description Format (ASDF)* – 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`, `.aiff`, `.flac` and `.ogg` files.

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.

[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.y.z/src/` where "x.y.z" stands for the version number.

1.3.2 Coordinate System

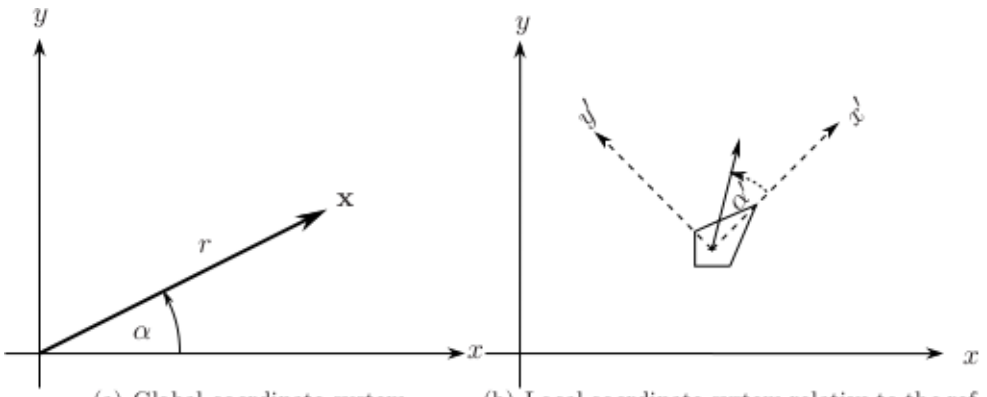


Fig. 1: The coordinate system used in the SSR. In ASDF α and α' are referred to as azimuth – refer to the section *Audio Scene Description Format (ASDF)*.

Fig. *1.1 (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. *1.1 (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.

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. *Head-Tracking*.

1.4 Audio Scene Description Format (ASDF)

Besides pure audio files, SSR can also read the *Audio Scene Description Format (ASDF)* Currently, two versions of the ASDF are supported.

There is a legacy version of the ASDF [Geier2008b], which has been supported since the beginning, but which can only describe static scenes. Recently, support for a new version 0.4 of the format was added, which allows creating scenes with moving sound sources. Detailed documentation is available at https://AudioSceneDescriptionFormat.readthedocs.io/.

The XML syntax between the two versions is not compatible. The new version is only used when version `0.4` is specified like this:

```
<asdf version="0.4">
```

The rest of this section describes the legacy version of the ASDF.

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 significantly more efficient to read data from an interleaved multichannel file compared to reading all channels from individual files**.

1.4.1 Legacy 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 http://spatialaudio.net/ssr/.

1.5 IP Interface

One of the key features of the SSR is the ability to remotely control it via a network interface. This enables you to straightforwardly connect any type of interaction tool from any type of operating system. There are multiple network interfaces available, see *Network Interfaces*.

1.6 Bug Reports, Feature Requests and Comments

Please report any bugs, feature requests and comments at https://github.com/SoundScapeRenderer/ssr/issues or send an e-mail to `ssr@spatialaudio.net`. 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 Contributors

Written by:
Matthias Geier, Jens Ahrens

Scientific supervision:
Sascha Spors

Contributions by:
Peter Bartz, Hannes Helmholz, Florian Hinterleitner, Torben Hohn, Christoph Hohnerlein, Christoph Hold, Lukas Kaser, André Möhl, Till Rettberg, David Runge, Fiete Winter, IOhannes m zmölnig

GUI design:
Katharina Bredies, Jonas Loh, Jens Ahrens

Logo design:
Fabian Hemmert

See also the Github repository https://github.com/soundscaperenderer/ssr for more contributors.

1.8 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 `ssr@spatialaudio.net`). 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.

CHAPTER

TWO

INSTALLATION

The SSR can be installed from packages that are available for GNU/Linux, macOS and (experimentally) MS Windows.

If none of those packages work for you, or if you want to use the latest development version, see *Building the SSR from Source* for instructions.

2.1 Arch Linux

The SSR is available in the official Arch Linux repositories and can be installed using:

```
sudo pacman -S ssr
```

A package script (PKGBUILD) for the development version of the SSR, tracking the master branch, is available in the Arch User Repository (AUR). This requires to be built manually, as is explained in the following subsection.

2.1.1 Building and installing with makepkg

The recommended way of building package scripts from the AUR and installing the resulting packages is to use makepkg in a manual build process.

```
# make sure to have the base-devel group installed
pacman -Sy base-devel
# clone the sources of the package script
git clone https://aur.archlinux.org/ssr-git
# go to the directory with the package script
cd ssr-git
# make and install the package and all of its dependencies
makepkg -csi
```

2.2 Debian/Ubuntu

The SSR is available as Debian package and Ubuntu package. It is called `soundscaperenderer` and you can install it using your favorite package manager/ package manager frontend (*apt-get*, *aptitude*, *synaptic*, ...), all dependencies should be installed automatically.

```
sudo aptitude update
sudo aptitude install soundscaperenderer
```

If you don't need the *Graphical User Interface*, there is also the GUI-less package `soundscaperenderer-nox` with fewer dependencies.

2.3 macOS

The SSR can be installed with Homebrew, all dependencies including the JACK audio server will be installed automatically:

```
brew install SoundScapeRenderer/ssr/ssr
```

We also recommed QJackCtl for convenient usage of JACK:

```
brew install qjackctl
```

Note: Binary packages (so-called "bottles") of the SSR are only available for a few macOS versions. The list of available bottles can be found on the package release page. If no bottle is available for your system, the installation should still work, but the SSR will be compiled from source. This means that more dependencies will be downloaded and the installation process may take significantly longer.

2.4 MS Windows

The MS Windows version of the standalone SSR is experimental at this stage. Find the pre-release of the executables at https://github.com/chris-hld/ssr/releases. Note that this SSR version only works with live inputs currently (it cannot play audio files). It has no limitation otherwise.

The *Pd externals* of the SSR renderers are confirmed to work.

CHAPTER

THREE

BUILDING THE SSR FROM SOURCE

3.1 Tarball

The so-called *tarball* – a file named like `ssr-x.y.z.tar.gz` – can be downloaded from http://spatialaudio.net/ssr/download/.

After downloading the tarball, unpack it and change to the newly created directory:

```
tar xvzf ssr-*.tar.gz
cd ssr-*
```

The tarball has the advantage that it already contains a few auto-generated files and therefore needs fewer dependencies. However, if you want to use the latest development version, you should get the code from the git repository:

3.2 Git Repository

The source code repository of the SSR is located at https://github.com/SoundScapeRenderer/ssr/.

You can *clone* it and change to the newly created directory like this:

```
git clone https://github.com/SoundScapeRenderer/ssr.git --recursive
cd ssr
```

Note: The SSR repository uses multiple Git submodules. If you use the `--recursive` flag as shown above, they will be automatically downloaded. If you didn't use the `--recursive` flag, you can get them by running the command:

```
git submodule update --init
```

When switching branches or when pulling changes from the server, the local submodules might get outdated. In such a case they can be updated with:

```
git submodule update
```

To generate the `configure` script (which is already contained in the tarball), run the command:

```
./autogen.sh
```

For this step, you'll need those extra dependencies (see below for how to install them for both Linux and macOS):

- `libtool`

- automake
- autoconf

If you want to generate the man pages (which are also contained in the tarball), you'll need:

- help2man

3.3 Dependencies

To enable support for the newest ASDF version 0.4, the C-bindings of the asdf-rust library have to be installed before compiling the SSR.

3.3.1 Linux

The following list of packages needs to be installed on your system to be able to build the SSR. The recommended way of installing those packages is to use your distribution's package manager. On Debian/Ubuntu you can use apt-get, aptitude, synaptic etc. However, if you prefer, you can of course also download everything as source code and compile each program yourself.

Note: The package names may vary slightly depending on your distribution or might not be shipped in separate lib or dev packages!

- make
- g++ or clang
- pkg-config
- libxml2-dev
- libfftw3-dev
- libsndfile1-dev
- libjack-jackd2-dev
- jackd2

For playing/recording audio files:

- ecasound
- libecasoundc-dev

For the GUI:

- libqt5-opengl-dev

For all network interfaces:

- libasio-dev

For the WebSocket interface:

- libwebsocketpp-dev

For the FUDI network interface:

- libfmt-dev

For SOFA support:

- `libmysofa-dev`

For VRPN tracker support:

- `vrpn` on Homebrew (has to be compiled from source on Linux)

For support of the *InterSense IntertiaCube3* head tracker:

- See the CI configuration file `.github/workflows/main.yml` for instructions.

For a concrete list of Ubuntu and Homebrew packages, see the CI configuration file `.github/workflows/main.yml`.

If the Qt5 library cannot be found during configuration, try using

```
export QT_SELECT=qt5
```

If there are problems with Qt5's `moc` during the build, you might need to add the corresponding folder (like `/usr/local/opt/qt/bin`) to your `PATH`. It might also help to install the package `qt5-default` to select Qt5 as default Qt version.

On Linux, 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 this after any changes:

```
sudo ldconfig
```

3.3.2 macOS

We recommend installing all dependencies from Homebrew:

```
brew install make automake libtool pkg-config help2man fftw asio fmt vrpn freeglut yarn␣
↪node ecasound jack libsndfile websocketpp qt@5 SoundScapeRenderer/ssr/libmysofa llvm
```

You might be able to skip installing llvm if you have Xcode installed.

And then temporarily link the keg-only Qt5 according to homebrew info (this is neccesary if you already have a newer version of Qt installed, for example if you installed the very useful package `qjackctl`):

```
export PATH="/usr/local/opt/qt@5/bin:$PATH"
export LDFLAGS="-L/usr/local/opt/qt@5/lib"
export CPPFLAGS="-I/usr/local/opt/qt@5/include"
export PKG_CONFIG_PATH="/usr/local/opt/qt@5/lib/pkgconfig"
```

If you want to use `help2man` on macOS, you have to install a Perl package:

```
cpan Locale::gettext
```

3.4 Configuring

Once all dependencies are installed, the SSR can be configured by running:

```
./configure
```

This script will check your system for dependencies and prepare the `Makefile` required for compilation. If any of the required software, mentioned in section *Dependencies* is missing, the `configure` script will signal that.

At successful termination of the `configure` script a summary will show up and you are ready to compile.

The `configure` script accepts many parameters and options, all of which can be listed with:

```
./configure --help
```

For example, certain feature can be disabled like this:

```
./configure --disable-ip-interface
./configure --disable-websocket-interface --disable-gui
```

The `configure` script also recognizes many environment variables. For example, to use a different compiler, you can specify it with `CXX`:

```
./configure CXX=clang++
```

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
```

If you are using an `arm64` CPU (i.e. M1, M2 or newer) on macOS (without Rosetta emulation), you might have to explicitly add some paths to be able to find the libraries installed with `brew`:

```
./configure CPPFLAGS=-I/opt/homebrew/include LDFLAGS=-L/opt/homebrew/lib
```

3.5 Building

If everything went smoothly so far, you can continue with the next step:

```
make
```

This will take some time (maybe a few minutes). If you have a multi-core or multi-processor computer you can speed things up by specifying the number of processes you want to use with `make -j8` (or any other number that you choose).

If there are errors, double-check whether all *Dependencies* are installed and whether the *configuration options* are correct.

3.6 Installing

Until now, everything was happening in the source directory. To be able to use the SSR system-wide, it has to be installed like this:

```
make install
```

Note: To execute this step, you might need superuser privileges. Depending on your system setup, these might be acquired with the help of `sudo`.

Alternatively, you can give your own user account the right to install programs. For example, on Debian and Ubuntu this can be done by adding your user to the `staff` group like this (assuming your user name is `myuser`):

```
sudo adduser myuser staff
```

For the change to take effect, you might have to log out and log in again.

3.7 Uninstalling

If the SSR didn't meet your expectations, we are very sorry, but of course you can easily remove it from your system again using:

```
make uninstall
```

CHAPTER

FOUR

RUNNING THE SSR

4.1 Starting the JACK Audio Server

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

4.2 SSR binaries

After installing the SSR, each renderer (see *The Renderers*) is available as a separate binary: `ssr-binaural`, `ssr-brs`, `ssr-vbap`, `ssr-wfs`, `ssr-aap`, `ssr-dca` (the renderer formerly known as `ssr-nfc-hoa`) and `ssr-generic`.

The following examples use `ssr-binaural`, but you can of course you any renderer you want!

Note that the SSR renderers are also available as externals for Pure Data. Refer to *SSR in Pure Data*.

4.3 Loading a Single Audio File

The easiest way to get a signal out of the SSR is by passing a sound-file directly:

```
ssr-binaural YOUR_AUDIO_FILE
```

Make sure that your amplifiers are not turned too loud…

To stop the SSR use either the options provided by the GUI (Section *Graphical User Interface*) or type `Crtl+c` in the shell in which you started the SSR.

4.4 Loading an Audio Scene File

You can also load *Audio Scenes*:

```
ssr-binaural YOUR_AUDIO_SCENE_FILE.asd
```

4.5 Command Line Options

There are a lot of options that are available for all renderers and only a few that are only available for certain renderers.

Type `ssr-binaural --help` to get an overview of the command line options (the help text is the same for all renderers):

```
Usage: ssr-binaural [OPTIONS] <scene-file>

The SoundScape Renderer (SSR) is a tool for real-time spatial audio reproduction
providing a variety of rendering algorithms.

Options:

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

JACK options:
  -n, --name=NAME     Set JACK client name to NAME
      --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
      --threads=N     Number of audio threads (default: auto)
  -r, --record=FILE   Record the audio output of the renderer to FILE
      --decay-exponent=VALUE
                      Exponent that determines the amplitude decay (default: 1)
      --loop          Loop all audio files
      --master-volume-correction=VALUE
                      Correction of the master volume in dB (default: 0 dB)
      --auto-rotation Auto-rotate sound sources' orientation toward the
                      reference
      --no-auto-rotation
                      Don't auto-rotate sound sources' orientation toward the
                      reference
  -i, --ip-server[=PORT]
                      Start IP server (default off),
                      a port number can be specified (default 4711)
  -I, --no-ip-server  Don't start IP server (default)
      --end-of-message-character=VALUE
                      ASCII code for character to end messages with
```

(continues on next page)

(continued from previous page)

```
                    (default 0 = binary zero)
    --websocket-server[=PORT]
                    Start WebSocket server (default on),
                    a port number can be specified (default 9422)
    --no-websocket-server
                    Don't start WebSocket server
    --fudi-server[=PORT]
                    Start FUDI server (default off),
                    a port number can be specified (default 1174)
    --no-fudi-server
                    Don't start FUDI server (default)
    --follow        Wait for another SSR instance to connect
    --no-follow     Don't follow another SSR instance (default)
-g, --gui           Start GUI (default)
-G, --no-gui        Don't start GUI
-t, --tracker=TYPE  Select head tracker, possible value(s):
                    fastrak patriot vrpn intersense razor
    --tracker-port=PORT
                    Port name/number of head tracker, e.g. /dev/ttyS1
-T, --no-tracker    Don't use a head tracker (default)

-h, --help          Show help and exit
-v, --verbose       Increase verbosity level (up to -vvv)
-V, --version       Show version information and exit
```

Use `$HOME` to refer to your home directory in the case that SSR does not resolve the tilde ~.

4.6 Configuration Files

The general configuration of the SSR (whether GUI is enabled, which tracker to use, and most other command line arguments) can be specified in a configuration file (e.g. `ssr.conf`). By specifying your settings 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.

Configuration files are loaded in the following order, if certain options are specified more than once, the last occurrence counts. This means that it is not the last file that is loaded that counts but rather the last occurrence at which a given setting is specified.

1. `/Library/SoundScapeRenderer/ssr.conf`
2. `/etc/ssr.conf`
3. `$HOME/Library/SoundScapeRenderer/ssr.conf`
4. `$HOME/.ssr/ssr.conf`
5. the path(s) specified with the `--config`/`-c` option(s) (e.g., `ssr-binaural -c my_config.file`)

We explicitly mention one parameter here that might be of immediate interest for you: `MASTER_VOLUME_CORRECTION`. This a correction in dB (!) that 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`).

4.7 Keyboard Actions in Non-GUI Mode

If you start SSR without GUI (option `--no-gui`), it starts automatically replaying the scene that 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.

4.8 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 *Reproduction Setups* and *ASDF*). The recording can then be used to analyze the SSR output or to replay it without the SSR using a software player like `ecaplay` (http://nosignal.fi/ecasound/).

4.9 Head Tracking

We provide integration of the *InterSense InertiaCube3* tracking sensor, the *Polhemus Fastrak* and the *Polhemus Patriot* as well as all trackers supported by *VRPN*. The *Supperware* head tracker is supported indirectly via the Pd patch `pd/supperware_head_tracker_to_ssr.pd`. The head trackers are used to update the orientation of the reference (in binaural reproduction this is the listener) in real-time.

See *Dependencies* for how to compile the SSR with head tracking support.

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, i.e. upwards on the screen ($\alpha = 90°$).

SSR is progressively (and silently) moving from 2D scenes to 3D scenes. The *binaural renderer* can handle head tracking about all three axes of rotation if the HRIRs are provided in SOFA. We recommend using the *browser-based GUI* to monitor the tracking as the built-in GUI only visualizes tracking along the azimuth.

4.9.1 Preparing InterSense InertiaCube3

Make sure that you have the required access rights to the tracker before starting SSR. For you are using the USB connection type

```
sudo chmod a+rw /dev/ttyUSBX
```

whereby `X` can be any digit or number. If you are not sure which port is the tracker then unplug the tracker, type

```
ls /dev/ttyUSB*
```

replug the tracker, execute above command again and see which port was added. That one is the tracker. It's likely that it is the one whose name contains the highest number.

4.9.2 Preparing Polhemus Fastrak/Patriot

Make sure that you have the required access rights to the tracker before starting SSR by typing something like

```
sudo chmod a+rw /dev/ttyS0
```

or

```
sudo chmod a+rw /dev/ttyS1
```

or so.

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

If you are using head tracking about all three axes of rotation, make sure that the tracking sensor is mounted on the headphones such that the cable leaves the sensor towards the left relative to the look direction of the user.

4.9.3 Preparing VRPN

In order to use *Virtual Reality Peripheral Network* (VRPN) compatible trackers create a config file `vrpn.cfg` with one of the following lines (or similar)

```
vrpn_Tracker_Fastrak MyFastrak /dev/ttyUSB0 115200
vrpn_Tracker_Fastrak MyOtherFastrak COM1 115200
vrpn_Tracker_Liberty MyPatriot /dev/ttyUSB1 115200
```

… and start `vrpn_server`. You can choose the name of the Tracker arbitrarily. Then, start the SSR with the given Tracker name, e.g.:

```
ssr-binaural --tracker=vrpn --tracker-port=MyFastrak@localhost
```

If the tracker runs on a different computer, use its hostname (or IP address) instead of localhost. You can of course select your head tracker settings by means of *Configuration Files*.

4.10 Using the SSR with DAWs

As stated before, the SSR is currently not able to dynamically replay audio files (refer to Section *ASDF*). 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 on SSR website http://spatialaudio.net/ssr/ you'll find an scene description that does exactly this.

DAWs like Ardour (https://ardour.org) 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.

4.11 Using the SSR with different audio clients

This page contains some short description how to connect your own audio files with the SSR using different audio players.

4.11.1 VLC Media Player

How to connect the SSR in binaural playback mode with the own audio library using Jack and VLC Media Player:

After installing Jack and the SSR (with all needed components: see *Configuring*) it is necessary to install the VLC Media Player with its Jack plugin (for example UBUNTU):

1. `sudo apt-get install vlc vlc-plugin-jack`

   (or use the packet manager of your choice instead of the command line and install: vlc and vlc-plugin-jack)

2. After installing open VLC Media Player and navigate to Tools->Preferences Select "All" on the bottom left corner In the appearing menu on the left navigate to "Audio"->"Output Module" and extend it by using "+"

3. In the submenu of "Output Module" select "JACK" and replace "system" by " Binaural-Renderer" in the "Connect to clients matching"-box. Do not forget to enable "Automatically connect to writable clients" above. (Otherwise you have to connect the audio output of vlc with the SSR input after every played audio file using jack.)

   (*Note*: If you want to use another Renderer, e.g. for WFS, you have to enter "WFS-Renderer" in the box)

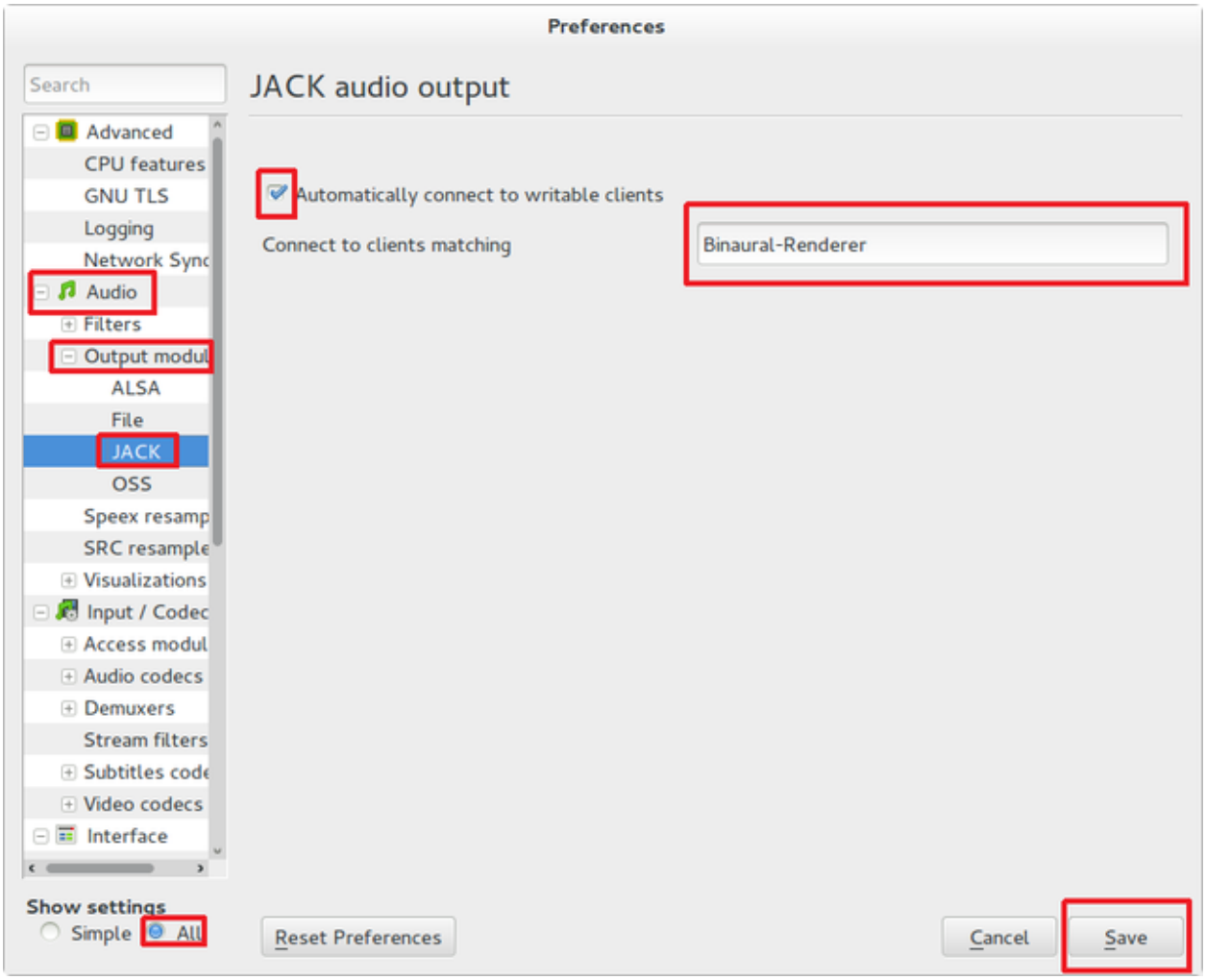
4. Save your changes.

5. Start everything together using the command line:

```
qjackctl -s & vlc & ssr-binaural --gui /"path_of_your_scene_file(s)"/stereo.asd &

This will start jack, vlc and the ssr with the GUI and a provided stereo
scene (TODO: LINK) (stereo.asd)
```

6. Open an audio file in vlc and press play


Preferences
Search
JACK audio output
Advanced
CPU features
GNU TLS
Logging
Network Sync
Audio
Filters
Output modul
ALSA
File
JACK
OSS
Speex resamp
SRC resample
Visualizations
Input / Codec
Access modul
Audio codecs
Demuxers
Stream filters
Subtitles code
Video codecs
Interface
Automatically connect to writable clients
Connect to clients matching
Binaural-Renderer
Show settings
Simple
All
Reset Preferences
Cancel
Save

4.11.2 Using a Head-Tracker

Running with InterSense tracker support

Due to copyright reasons, the SSR does not come with a built-in InterSense tracker support. So first you have to build the SSR with InterSense tracker support yourself (see the CI configuration file `.github/workflows/main.yml` for instructions).

If you are using a USB-to-Serial interface with your tracker, you need to install drivers for that. This seems to work fine for the interface made by InterSense: https://ftdichip.com/drivers/vcp-drivers/

To check if the system sees the tracker do:

```
ls -l /dev/tty.usb*
```

On the MacBooks tested, the serial ports were called `/dev/tty.usbserial- 00001004` or `/dev/tty.usbserial-00002006` depending on which USB port was used.

To make the SSR use the InterSense tracker with these ports, you have two options:

Using the command line (only one port can be specified):

```
open -a SoundScapeRenderer --args --binaural "--tracker=intersense
--tracker-port=/dev/tty.usbserial-XXXXXXXX"
```

... or using config files:

Add these lines to a config file (multiple ports can be specified):

```
TRACKER = intersense
TRACKER_PORTS = /dev/tty.usbserial-XXXXXXXX /dev/tty.usbserial-YYYYYYYY
```

It's recommended to use the config file approach - best use a global :ref:` config file<ssr_configuration_file>`.

Running with Razor AHRS tracker support

If you happen not to own a Polhemus or InterSense tracker to do your head- tracking, an alternative would be to use our DIY low-cost Razor AHRS tracker.

If you have Arduino installed on you machine, FTDI drivers will be there too. Otherwise get the driver from https://ftdichip.com/drivers/vcp-drivers/.

To check if the system sees the tracker do:

```
ls -l /dev/tty.usb*
```

This should give you something like `/dev/tty.usbserial-A700eEhN`.

To make the SSR use this Razor AHRS tracker, you have two options:

Using the command line:

```
open -a SoundScapeRenderer --args --binaural "--tracker=razor
--tracker-port=/dev/tty.usbserial-XXXXXXXX"
```

... or using config files:

Add these lines to a config file:

```
TRACKER = intersense
TRACKER_PORTS = /dev/tty.usbserial-XXXXXXXX
```

It's recommended to use the config file approach - best use a global *config file*.

CHAPTER

FIVE

THE RENDERERS

5.1 General

5.1.1 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.

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.

5.1.2 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 that 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 taken into account. This 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 that is specified in ASDF. This origin point also specifies the location of the symbol that 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.

5.1.3 Subwoofers

All loudspeaker-based renderers support the use of subwoofers. Outputs of the SSR that are assigned to subwoofers receive a signal having full bandwidth. So, you will have to make sure yourself that your system lowpasses these signals appropriately before they are emitted by the subwoofers.

You might need to adjust the level of your subwoofer(s) depending on the renderers that you are using as the overall radiated power of the normal speakers cannot be predicted easily so that we cannot adjust for it automatically. For example, no matter of how many loudspeakers your setup is composed of the VBAP renderer will only use two loudspeakers at a time to present a given virtual sound source. The WFS renderer on the other hand might use 10 or 20 loudspeakers, which can clearly lead to a different sound pressure level at a given receiver location.

For convenience, ASDF allows for specifying permantent weight for loudspeakers and subwoofers using the `weight` attribute:

```
<loudspeaker model="subwoofer" weight="0.5">
  <position x="0" y="0"/>
  <orientation azimuth="0"/>
</loudspeaker>
```

`weight` is a linear factor that is always applied to the signal of this speaker. Above example will obviously attenuate the signal by approx. 6 dB. You can use two ASDF description for the same reproduction setup that differ only with respect to the subwoofer weights if you're using different renderers on the same loudspeaker system.

5.1.4 Distance Attenuation

Note that in all renderers – except for the BRS and generic renderers –, the distance attenuation in the virtual space is $\frac{1}{r}$ with respect to the distance r of the respective virtual point source to the reference position. Point sources closer than 0.5 m to the reference position do not experience any increase of amplitude. Virtual plane waves do not experience any algorithmic distance attenuation in any renderer.

You can specify your own preferred distance attenuation exponent exp (in $\frac{1}{r^{exp}}$) either via the command line argument `--decay-exponent=VALUE` or the configuration option `DECAY_EXPONENT` (see the file `data/ssr.conf.example`). The higher the exponent, the faster is the amplitude decay over distance. The default exponent is $exp = 1$[1]. Fig. *3.1* illustrates the effect of different choices of the exponent. In simple words, the smaller the exponent the slower is the amplitude decay over distance. Note that the default decay of $\frac{1}{r}$ is theoretically correct only for infinitessimally small sound sources. Spatially extended sources, like most real world sources, exhibit a slower decay. So you might want to choose the exponent to be somewhere between 0.5 and 1. You can completely suppress any sort of distance attenuation by setting the decay exponent to 0.

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 *Configuration File*). The desired amplitude reference distance for a given sound scene can be specified in the scene description (Section *ASDF*). The default value is 3 m.

The overall amplitude normalization is such that plane waves always exhibit the same amplitude independent of what amplitude reference distance and what decay exponent have been chosen. Consequently, also virtual point source always exhibit the same amplitude at amplitude reference distance, whatever it has been set to.

5.1.5 Doppler Effect

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

5.1.6 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. *3.2*.

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·*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 [Ahrens2008a] and [Ahrens2008b] requires a different implementation approach which is computationally significantly more costly.

[1] A note regarding previous versions of the WFS renderer: In the present SSR version, the amplitude decay is handled centrally and equally for all renderers that take distance attenuation into account (see Table *2*). Previously, the WFS renderer relied on the distance attenuation that was inherent to the WFS driving function. This amplitude decay is very similar to an exponent of 0.5 (instead of the current default exponent of 1.0). So you might want to set the decay exponent to 0.5 in WFS to make your scenes sound like they used to do previously.

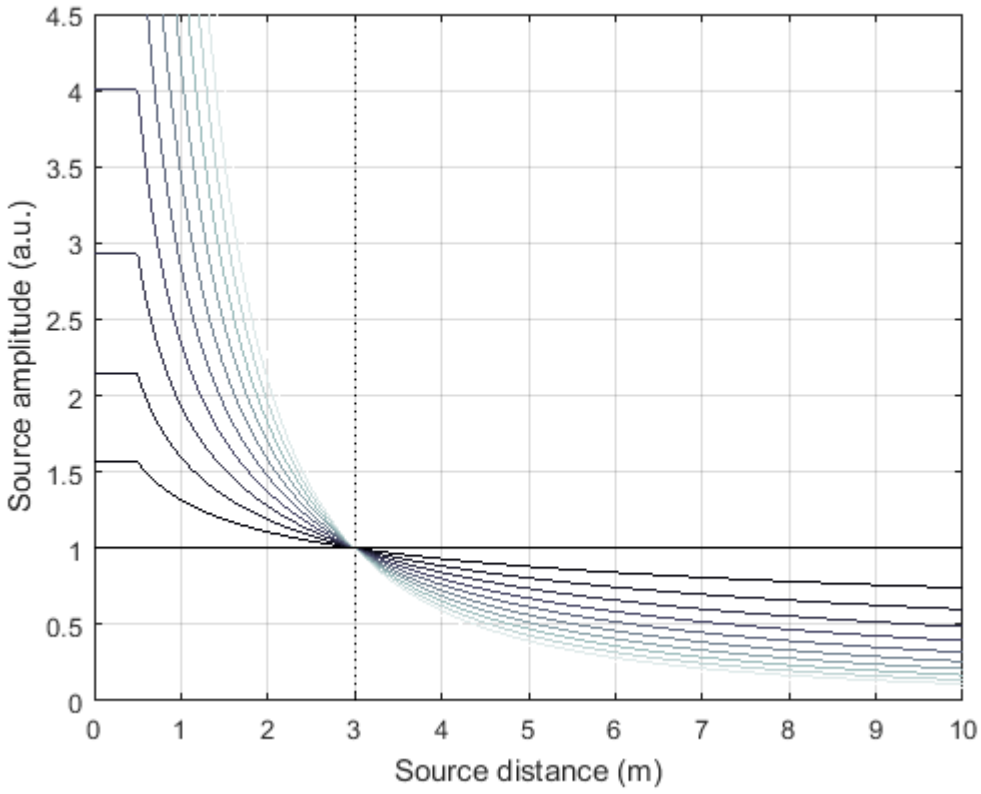


Fig. 1: Illustration of the amplitude of virtual point sources as a function of source distance from the reference point for different exponents exp. The exponents range from 0 to 2 (black color to gray color). The amplitude reference distance is set to 3 m. Recall that sources closer than 0.5 m to the reference position do not experience any further increase of amplitude.

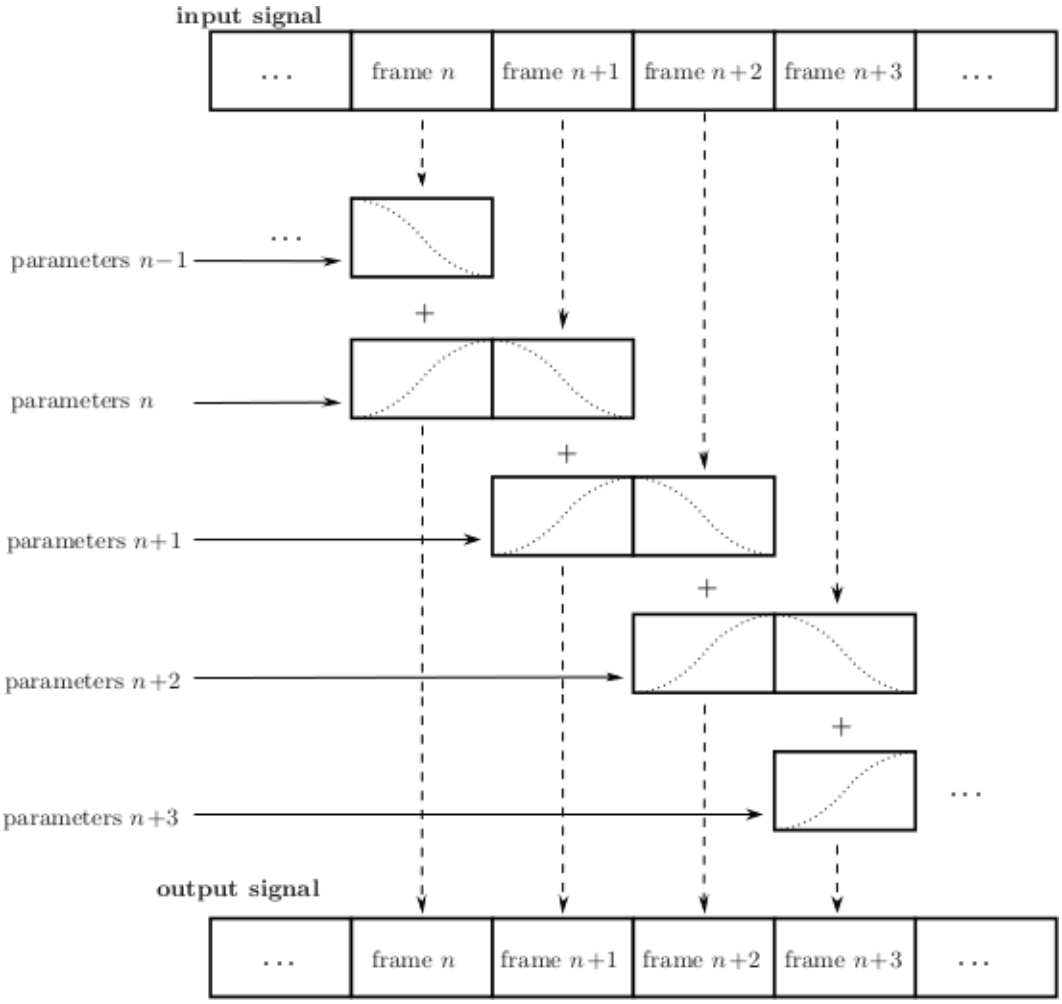


Fig. 2: Illustration of the frame-wise signal processing as implemented in the SSR renderers (see text)

5.2 Binaural Renderer

Executable: `ssr-binaural`

Binaural rendering is an approach where the acoustical influence of the human head is electronically simulated to position virtual sound sources in space. **Be sure that you are using headphones to listen.**

The acoustical influence of the human head is coded in so-called head-related impulse responses (HRIRs) or equivalently by head-related transfer functions. 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 *Configuration File*) to specify your custom location. The SSR connects its outputs automatically to outputs 1 and 2 of your sound card.

For virtual sound sources that are closer to the reference position (= the listener position) than 0.5 m, 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[3]:

- 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 $1°$
- ...
- 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.

[3] Note that the binaural renderer has the currently hidden and undocumented feature of being able to read HRIRs in SOFA. It supports head tracking about all three axes of rotation in this case.

5.2.1 The HRIR sets shipped with SSR

SSR comes with two different HRIR sets: FABIAN and KEMAR (QU). The differ with respect to the manikin that was used in the measurement (FABIAN vs. KEMAR). The reference for the FABIAN measurement is [Lindau2007], and the reference for the KEMAR (QU) is [Wierstorf2011]. The low-frequency extension from [SpatialAudio] has been applied to the KEMAR (QU) HRTFs.

You will find all sets in the folder `data/impulse_responses/hrirs/`. The suffix `_eq` in the file name indicates the equalized data. The unequalized data is of course also there. See the file `data/impulse_responses/hrirs/hrirs_fabian_documentation.pdf` for a few more details on the FABIAN measurement.

Starting with SSR release 0.5.0, the default HRIR set that is loaded is headphone compensated, i.e., we equalized the HRIRs a bit in order to compensate for the alterations that a typical pair of headphones would apply to the ear signals. Note that by design, headphones do not have a flat transfer function. However, when performing binaural rendering, we need the headphones to be transparent. Our equalization may not be perfect for all headphones or earbuds as these can exhibit very different properties between different models.

We chose a frequency sampling-based minimum-phase filter design. The transfer functions and impulse responses of the two compensation filters are depicted in Fig. *3.3*. The impulse responses themselves can be found in the same folder like the HRIRs (see above). The length is 513 taps so that the unequalized HRIRs are 512 taps long, the equalized ones are 1024 taps long.

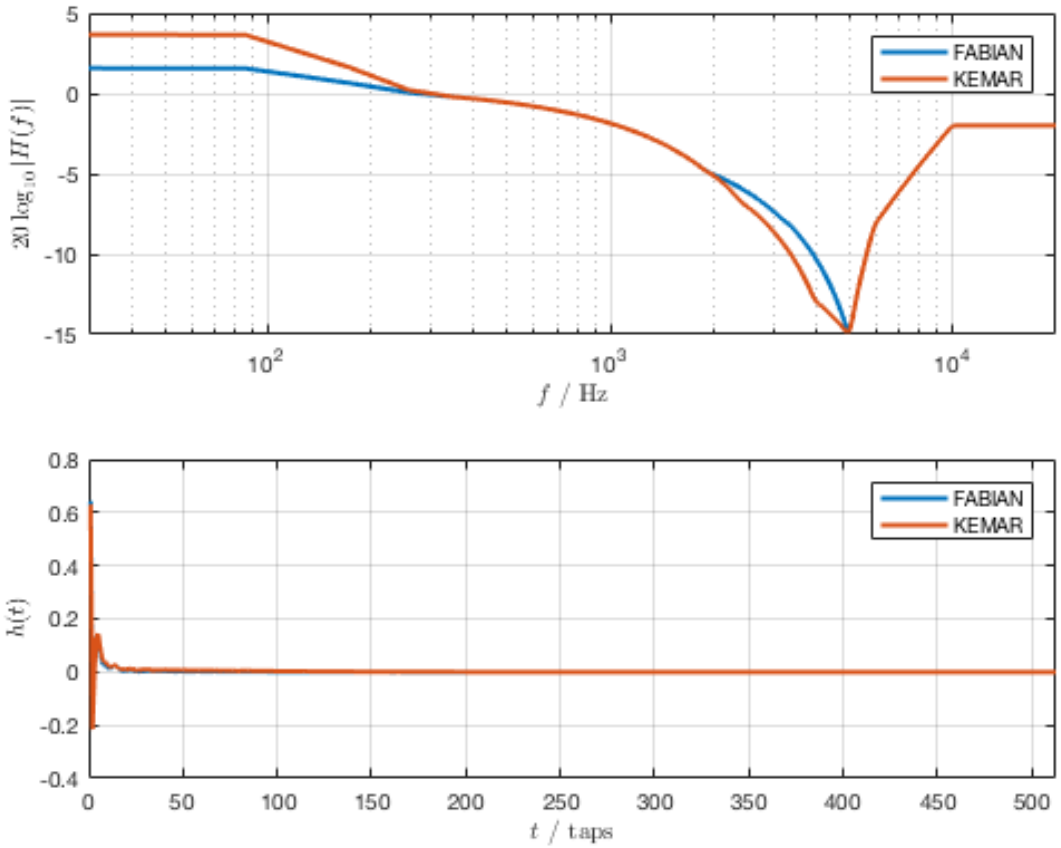


Fig. 3: Magnitude transfer functions and impulse responses of the headphone compensation / equalization filters

Recall that there are several ways of defining which HRIR set is loaded, for example the `HRIR_FILE_NAME` in the *SSR*

configuration files property, or the command line option `--hrirs=FILE`.

5.2.2 Preparing HRIR sets

You can easily prepare your own HRIR sets for use with the SSR by adopting the MATLAB script `data/matlab_scripts/prepare_hrirs_cipic.m` to your needs. This script converts the HRIRs of the KEMAR manikin included in the CIPIC database [AlgaziCIPIC] to the format that the SSR expects. See the script for further information and how to obtain the raw HRIRs. Note that the KEMAR (CIPIC) HRIRs are not identical to the KEMAR (QU) ones.

5.3 Binaural Room Synthesis Renderer

Executable: `ssr-brs`

The Binaural Room Synthesis (BRS) renderer is a binaural renderer (refer to Section *Binaural Renderer*) 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 *Binaural Renderer*). 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 *Audio Scenes*). 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 BRIRs of the FABIAN manikin 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. You can obtain the data from [BRIRs].

5.4 Vector Base Amplitude Panning Renderer

Executable: `ssr-vbap`

The Vector Base Amplitude Panning (VBAP) renderer uses the algorithm described in [Pulkki1997]. 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 *ASDF*), 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 *ASDF*), 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.

5.5 Wave Field Synthesis Renderer

Executable: `ssr-wfs`

The Wave Field Synthesis (WFS) renderer is the only renderer so far that discriminates between virtual point sources and plane waves. It implements the simple (far-field) driving function given in [Spors2008]. 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 *Configuration File*) 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.

5.5.1 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 *Running SSR*) or in the SSR configuration file (Section *Configuration File*). 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 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 [Spors2006] 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 [Spors2008].

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.

5.5.2 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 [sec:asdf]). The `weight` determines the linear (!) weight to be applied to the respective loudspeaker. It defaults to 1 if it is not specified.

5.6 Ambisonics Amplitude Panning Renderer

Executable: `ssr-aap`

The Ambisonics Amplitude Panning (AAP) renderer does very simple Ambisonics rendering. It does amplitude panning by simultaneously using all loudspeakers that 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 that 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 *Running SSR*) or the SSR configuration file (Section *Configuration File*). 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.

5.6.1 Conventional driving function

By default we use the standard Ambisonics panning function presented, for example, in [Neukom2007]. It reads

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

whereby α_0 is the azimuth angle of the position of the considered secondary source, α_s is the azimuth angle of the position of the virtual source, both in radians, and M is the Ambisonics order.

5.6.2 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 [Neukom2007] 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 *Configuration File*) to be `TRUE` or `true`.

5.7 Distance-coded Ambisonics Renderer

Executable: `ssr-dca`

Distance-coded Ambisonics (DCA) is sometimes also termed "Nearfield Compensated Higher-Order Ambisonics". This renderer implements the driving functions from [Spors2011]. The difference to the AAP renderer is a long story, which we will elaborate on at a later point.

Note that the DCA renderer is experimental at this stage. It currently supports orders of up to 28. There are some complications regarding how the user specifies the locations of the loudspeakers and how the renderer handles them. The rendered scene might appear mirrored or rotated. If you are experiencing this, you might want to play around with the assignment of the outputs and the loudspeakers to fix it temporarily. Or contact us.

Please bear with us. We are going to take care of this soon.

5.8 Generic Renderer

Executable: `ssr-generic`

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 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 https://github.com/SoundScapeRenderer/example-scenes and check out the file `generic_renderer_example.asd` which comes with all required data.

Look also *here* for more general signal processing examples using SSR.

	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[2]	distance attenuation	model
binaural renderer	+	+	+	+	+	only w.r.t. ampl.
BRS renderer	+	+	-	-	-	-
VBAP renderer	+	+	+	+	+	only w.r.t. ampl.
WFS renderer	+	+	+	+	+	+
AAP renderer	+	+	+	-	+	only w.r.t. ampl.
generic renderer	+	+	-	-	-	-

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

5.9 Summary

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

[2] So far, only planar sources have a defined orientation. By default, their orientation is always pointing from their nominal position to the reference point no matter where you move them. Any other information or updates on the orientation are ignored. You can changes this behavior by using either the command line option `--no-auto-rotation`, using the `AUTO_ROTATION` configuration parameter, or hitting `r` in the GUI.

CHAPTER

SIX

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. *4.1* 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.

Fig. 1: No anti-aliasing on the left image.

6.1 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. *4.2*.

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. *Mouse Actions* for information on how to operate on the time line.

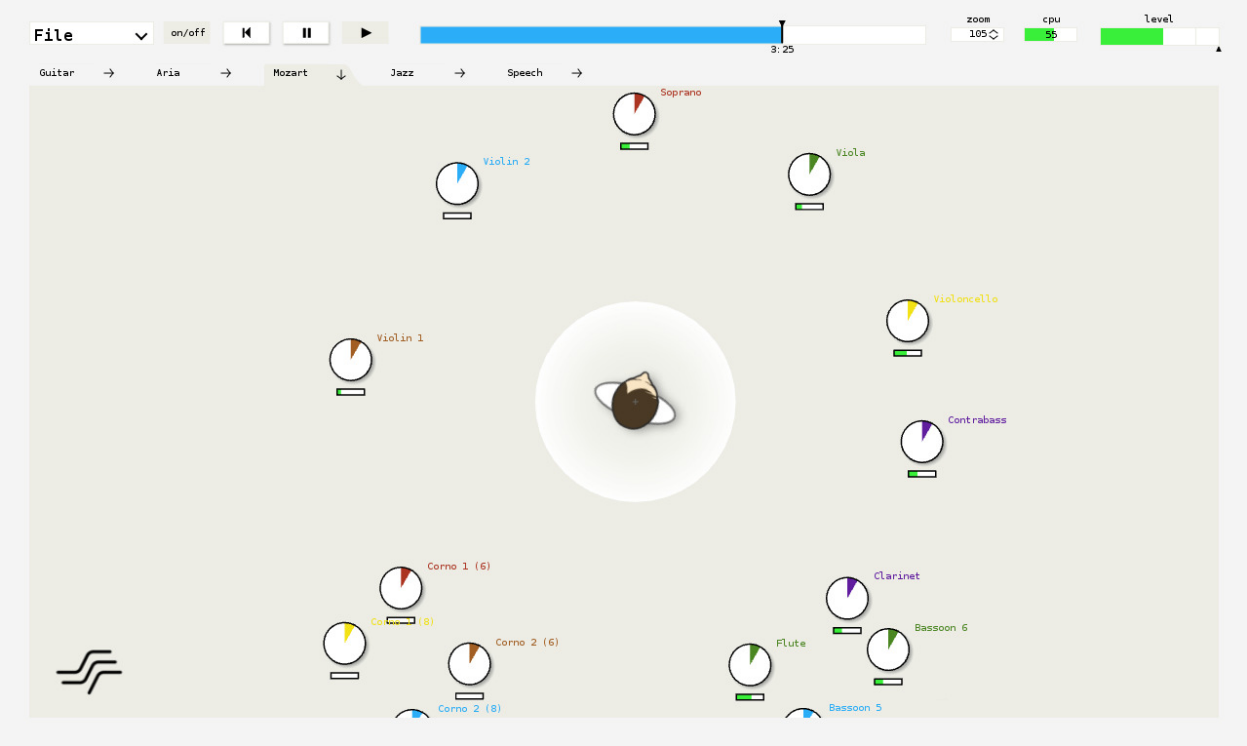


Fig. 2: Screen shot of the SSR GUI.

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. *Configuration-File*. 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` files. 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. *1.1*. 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. *4.2* 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).

6.2 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 Sec. *Keyboard Actions*).

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[1]. 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[2] on a sound source opens a window that lists the properties of the source such as position, volume, etc. Refer to Fig. *4.3* and Sec. *Source Properties Dialog*.

A right mouse click on the scene background[3] 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.

[1] On a touchpad on macOS, swipe up and down with two finger to zoom in and out (don't click!).
[2] On a touchpad on macOS, click with two fingers simultaneously.
[3] On a touchpad on macOS, click with two fingers simultaneously, hold the click and move one finger, or both fingers simultaneously, or use a third finger to operate the rubber band.

6.2.1 Source Properties Dialog

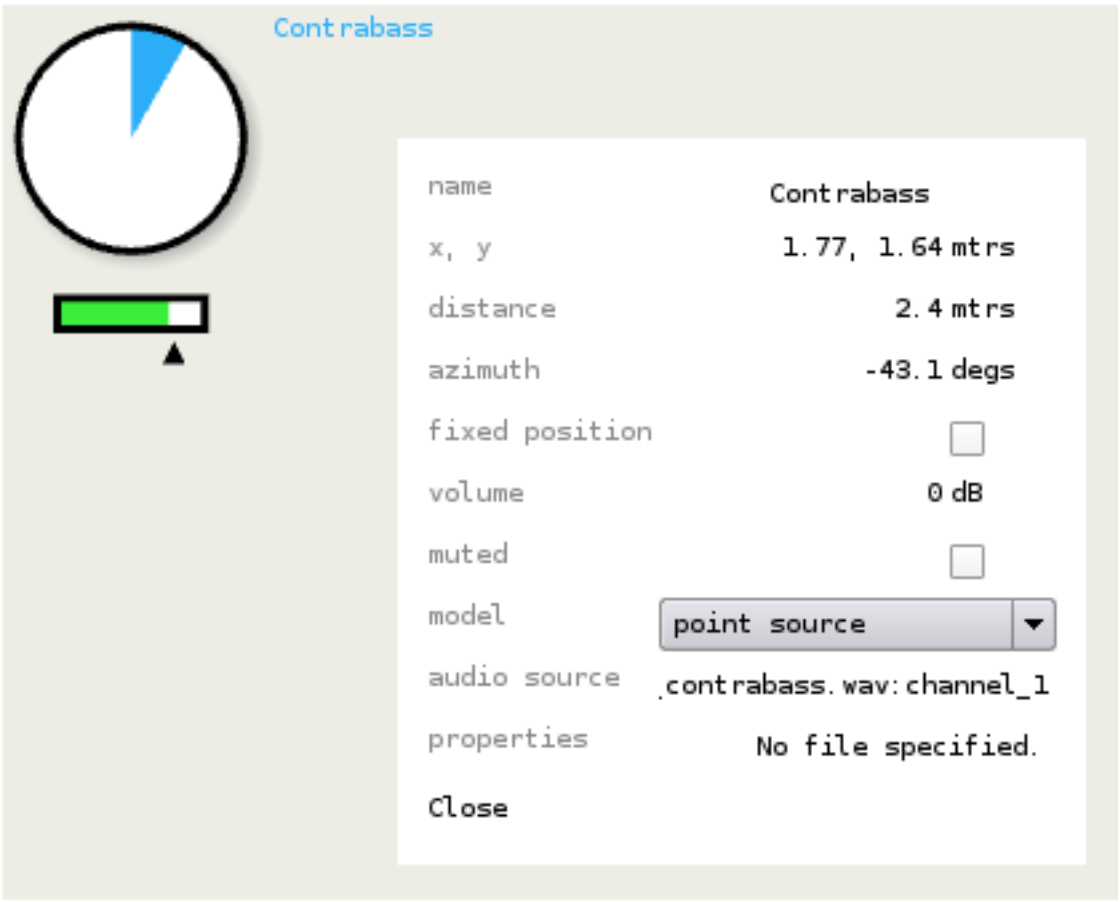


Fig. 3: Source properties dialog

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. Refer to Fig. *4.3* to see the complete list of properties this dialog shows.

6.2.2 Drag & Drop of Scenes

You can also drag & drop scenes (or audio files) into the GUI to open them. Currently, you can only drag & drop one single file.

6.3 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. *Running SSR*).

- `+/-`: 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:math:^*circ* 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*
- `r`: toggles whether or not all sources are always automatically oriented toward the reference.
- `s`: if no source selected: unsolos all potentially soloed sources, otherwise: solos selected sound sources.
- `Shift+s`: solo in place, unsolos all other 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

CHAPTER

SEVEN

BROWSER-BASED GUI

Warning: Currently, the browser GUI is only a highly experimental prototype. It should be usable, but there are many missing features and other possibilities for improvements.

As always, contributions are very welcome!

Using the SSR's *WebSocket-based Network Interface*, it is possible to control the SSR from an off-the-shelf web browser. Just make sure the SSR has been started with the `--websocket-server` option and visit the URL http://localhost:9422.

If you just want to test the WebSocket connection, you can visit http://localhost:9422/test.

You can of course also access the SSR if it is running on a different computer in the network, by simply using the appropriate host name or IP address instead of `localhost`.

CHAPTER

EIGHT

NETWORK INTERFACES

Warning: We did not evaluate any of the network interfaces in terms of security. So please be sure that you are in a safe network when using them.

The *SoundScape Renderer* has no less than three network interfaces using different protocols:

- The *WebSocket protocol* has built-in support in all modern web browsers and can be used to control the SSR from a web browser.
- The *FUDI protocol* is the native protocol of the visual programming language Pure Data.
- The *legacy XML-based protocol* is still supported for backwards-compatibility but should not be used for new projects.

8.1 WebSocket-based Network Interface

Warning: We did not evaluate any of the network interfaces in terms of security. So please be sure that you are in a safe network when using them.

The WebSocket network interface can be activated with:

```
ssr-binaural --websocket-server
```

This of course works for all renderers, not only `ssr-binaural`.

Once the SSR is running, you can use your browser to connect to http://localhost:9422/test. This will show a simple test client which you can use to check whether everything is working.

If you want to try out the *Browser-based GUI*, use the address http://localhost:9422.

By default, the port number 9422 is used. You can choose a different port with:

```
ssr-binaural --websocket-server=4321
```

The relevant settings in the *Configuration Files* are:

```
WEBSOCKET_INTERFACE = on
WEBSOCKET_PORT = 4321
```

8.2 FUDI-based Network Interface (for Pure Data)

Warning: We did not evaluate any of the network interfaces in terms of security. So please be sure that you are in a safe network when using them.

The FUDI network interface can be activated with:

```
ssr-binaural --fudi-server
```

This of course works for all renderers, not only `ssr-binaural`.

The default port is 1174; you can choose a different port with:

```
ssr-binaural --fudi-server=4321
```

The relevant settings in the *Configuration Files* are:

```
FUDI_INTERFACE = on
FUDI_PORT = 4321
```

8.2.1 Controlling the SSR from Pure Data

Have a look at the Pd patch `ssrclient.pd` in the `pd/` directory.

Note: If you are using Pure Data to control the stand-alone SSR application, you might want to consider using the SSR externals for Pure Data instead, see *SSR as a Library*.

8.2.2 Controlling the SSR from a Terminal

The FUDI interface is very simple and thanks to that any tool that can send TCP/IP messages can be used to control the SSR. One such tool is netcat.

While the SSR is running (with the FUDI interface activated), launch this command in a separate terminal window:

```
nc localhost 1174
```

Now you can type FUDI messages (don't forget the semicolon at the end!) which will be sent to the SSR when you press `Return`.

To then, for example, move source 1 to the position (x, y) = (1, 1), type:

```
src 1 pos 1 1;
```

If you want to receive messages from the SSR, use something like this:

```
nc localhost 1174 <<< "subscribe scene; subscribe renderer;"
```

8.3 Legacy XML-based Network Interface

Warning: We did not evaluate any of the network interfaces in terms of security. So please be sure that you are in a safe network when using them.

The legacy network interface can be activated with:

```
ssr-binaural --ip-server
```

This of course works for all renderers, not only `ssr-binaural`.

The default port is 4711; you can choose a different port with:

```
ssr-binaural --ip-server=4321
```

The relevant settings in the *Configuration Files* are:

```
NETWORK_INTERFACE = on
SERVER_PORT = 4321
```

8.3.1 Messages

This is just a short overview about the XML messages which can be sent to the SSR via TCP/IP. By default, messages have to be terminated with a binary zero (`\0`). This can be changed to, for example, a newline / line feed (`\n`) or to a carriage return (`\r`) using one of the following ways:

- Command line: Use the command line option `--end-of-message-character=VALUE` VALUE is the ASCII code for the desired character (binary zero: 0; line feed: 10; carriage return: 13).
- Configuration file: Here, there is the option END_OF_MESSAGE_CHARACTER. See the example `data/ssr.conf.example`. Use ASCII codes here as well.

The choice of delimiter applies to, of course, both sent and received messages.

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>`

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>`

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="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>`

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>`

CHAPTER

NINE

SSR AS A LIBRARY

Until now, we were talking about the stand-alone SSR application that uses the JACK audio backend. But all renderers can also be used as a library in any C++ project, using any audio backend (or none).

Only the bare renderers are available; no GUI, no network interface, no scene files, no head trackers. Multi-threading is still available!

The easiest way to use an SSR renderer as a library is to include the SSR repository as a Git submodule into your own project. For more details, have a look at the examples in the SSR repository:

- Pure Data externals, see flext directory
- MEX files for Octave/Matlab, see mex directory

CHAPTER

TEN

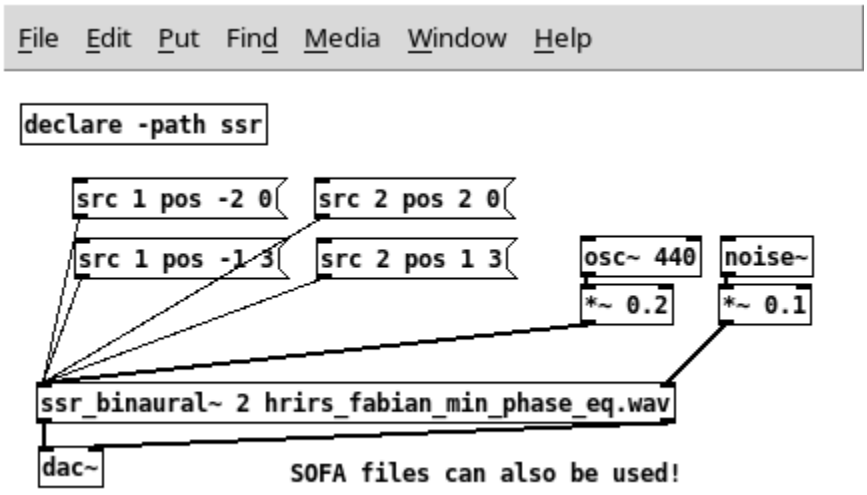
USE CASES

10.1 SSR in Pure Data

The SSR renderers are available as externals in Pure Data (Pd). Install them in Pd via *Help* → *Find externals*, then search for 'ssr'. This is available on both Linux and macOS, whereby the macOS version for Macs with Apple silicon is experimental. If this one does not work for you, use Pd for Intel processors with Rosetta.

Each renderer is available as a separate external, namely `ssr_binaural~`, `ssr_brs~`, `ssr_dca~`, `ssr_aap~`, `ssr_wfs~` and `ssr_vbap~`. The externals have to be added to Pure Data's path, for example by creating an object `[declare -path ssr]` in your patch.

Here is a screenshot of what it looks like when using SSR's binaural renderer with the minimum-phase EQ'd HRIRs of the FABIAN manikin and 2 virtual sound sources in Pd, whereby the signal that feeds source 1 is a sine of 440 Hz, and the signal that feeds source 2 is noise:



The source positions, orientations and many other things can be changed by sending messages to the external. See the help patch for all available messages.

The computational performance of SSR in Pd is somewhat lower than that of the standalone version because we have not got it to work in Pd with internal block sizes that are larger than 64. Such short blocks cause filters to be cut in very large numbers of partitions, which produces computational overhead.

10.2 Using SSR for Listening Experiments

This is one of the main applications for which we use SSR. Most of the times, we perform the experiment over headphones. We use the BRS renderer in which each condition of the experiment is represented by a virtual source. The impulse responses that are associated to that source implement the actual experimental condition. Lastly, we use external software like Python or MATLAB to have a GUI for the subjects and to play audio that we route to SSR via JACK. Switching from one condition to another one is implemented by muting an unmuting the corresponding virtual sources in SSR through the network interface.

Watch this 1m:35s video to see this in action: https://youtu.be/aaAc7cDlacU

All resources for the subsequent subsections are found here: https://github.com/SoundScapeRenderer/listening-test

10.3 Using SSR for General Realtime Multichannel Signal Processing

SSR is designed for providing the spatial audio rendering functionality as discussed throughout this documentation. Looking at the entire concept from a different angle enables a large set of applications that are not obvious at first sight. One application that we would like to discuss here with some amount of comprehensiveness is dynamic or "realtime" multichannel signal processing. More precisely, we will demonstrate how SSR allows for performing multichannel realtime convolution whereby we can exchange the filters that are being applied.

10.3.1 1-in-1-out

Let's assume that we want to filter a single-channel signal in realtime with a filter that we would like to be able to replace as the convolution is going. We use the Binaural Renderer for this with one virtual source.

SSR accepts one input channel, which it convolves with two different dedicated impulse responses (one for the left and one for right ear) to produce two output signals. We would like to process only one channel so that we simply ignore the second output. The file `data_linked_in_ssr_manual/impulse_responses/irs_1-in-1-out.wav` that contains the pre-computed impulse responses that we would like to filter with looks as follows:

- 1st channel: impulse response of the filter that will be invoked by calling the index `0`
- 2nd channel: only zeros (we don't want it to produce output)
- 3rd channel: impulse response of the filter that will be invoked by calling the index `359`
- 4th channel: only zeros (we don't want it to produce output)
- …
- 719th channel: impulse response of the filter that will be invoked by calling the index `1`
- 720th channel: only zeros (we don't want it to produce output)

(The indexing is a bit funny because we're using the binaural render for convenience. In real-life, you would most likely use the BRS renderer with which the indexing is 0, 1, 2, …, 359 instead of the 0, 359, 358, …, 1 that it is here.)

You have probably gotten the idea already. We can use this setup for 1-in-1-out dynamic filtering by not interpreting the loaded filter as ear impulse responses that are called via given orientation angles of the user. Rather, we think of filter indices that we can call using the head tracking interface. We only need to replace the head tracking with another interface that allows us to select the desired filter indices. Here is a Pd patch that does exactly this: `data_linked_in_ssr_manual/select_filter_by_index.pd`

Start the SSR using:

```
ssr-binaural --fudi-server=1147 --hrirs=data/scenes/impulse_responses/irs_1-in-1-out.wav
```

and drag and drop an audio file into the GUI.

Start Pd with the control patch and play around with the filter index. You will find that, depending on the chosen filter index, different amounts of lowpass filtering will be applied in this example.

10.3.2 Multiple Input Multiple Output

Let's increase the complexity by a considerable amount and perform 2-in-8-out as an example for multiple input multiple output (MIMO) dynamic convolution. We used this in the experiment in [Ma2019] to create a user-tracked 8-channel loudspeaker array (8 outs) that delivers binaural content (2 ins) by means of crosstalk cancelation. We are actually dealing with two 1-in-8-out systems (one for the left ear content and one for the right ear content) the outputs of which are added.

The user was able to be located at any possible position along a straight line of length 80 cm that was parallel to the loudspeaker array. We used a tracking system to monitor the instantaneous position.

Here is how we implemented it:

- We used the BRS renderer for the implementation. It has two outputs, which means that we need 4 SSR running in parallel.
- Each BRS renderer employed a scene with 2 virtual sound sources.
- SSR 1: Virtual source 1 produced the output signal for the left ear content for loudspeakers 1 and 2; virtual source 2 produced the output signal for the right ear content for loudspeakers 1 and 2
- SSR 2: Same like SSR 1 but for loudspeaker 3 and 4
- SSR 3 and 4 drive loudspeakers 5 and 6 as well as 7 and 8, respectively.

Each combination of input and output allows for applying 360 different impulse responses the indices of which we can select using the head tracking interface. This means that we were required to quantize the user's position to 360 different positions along that 80-cm-long-line which effectively reduced the head tracking accuracy to 0.8/359 m = 2 mm. We precomputed all impulse responses for all combinations of input and output channel and user position in MATLAB.

The last component that remains to be implemented is a patch that transforms user position to filter index and distributes that to all SSR synchronously. We did this with this Pd patch: `data_linked_in_ssr_manual/tracker_to_4_ssr.pd`. You will see that there is no mechanism for guaranteeing that all filter indices arrive synchronously. We rather send updates as soon as they come in from the tracker. The last index that an SSR instance receives just before the processing of a new signal block is the index that SSR uses. We did not notice a single occasion when this led to audible consequences because of a lack of synchronicity.

When running several SSR at a time, we need to make sure that they all use different JACK client names as well as that all SSR instances receive TCP/IP messages on different ports. SSR will otherwise refuse to start.

Here is a shell script that work on both Linux and macOS is SSR is installed: `data_linked_in_ssr_manual/start_ssr_4_times.sh` (and here one for the macOS app: `data_linked_in_ssr_manual/start_ssr_4_times_macos_app.sh`, make them executable using `chmod a+x SCRIPT_NAME.SH`, in the macOS script, you need to adapt the global paths to the asdf files) that starts the 4 SSR instances for the 8-channel crosstalk-canceling array. It then waits 5 s to make sure that all SSR instances have started up and then performs the required JACK connections. Note the `--input-prefix=XXX:XXX` and `--output-prefix=YYY:YYY` arguments. These make sure that SSR does not automatically connect to existing JACK ports. We did this for convenience to have manual control over which connections are established. All SSR instances would otherwise connect to output channels 1 and 2 automatically.

Afterwards, start Pd with the patch referenced above.

The audio signal was played from a GUI via JACK like we did it with other *listening experiments*.

Note that you will need an audio interface with at least 8 output channels for all of the above to work. You will otherwise receive error messages about failure to establish some of the JACK connections.

CHAPTER

ELEVEN

KNOWN ISSUES

This page describes a few open issues and – if available – possible work-arounds. Resolved issues are also listed here, in case you want to use an old version of the SSR for some reason.

Please also visit https://github.com/SoundScapeRenderer/ssr/issues to check for even more issues. If you want to report a problem, please open a new issue there.

11.1 Open Issues

11.1.1 `make dmg` on macOS chokes on symbolic links

On some file system (e.g. network shares with ACLs) you might get an error like this:

```
... copy helper ... copy error (canceling): /Volumes/SoundScape Renderer ...
... Operation not supported ... could not access /Volumes/...
hdiutil: create failed - Operation not supported
make[1]: *** [dmg] Error 1
make: *** [dmg] Error 2
```

This has something to do with symbolic links and the way how `hdiutil` handles them. If you get this error, just try to compile the SSR from a different location. You can do this by either moving all the source files somewhere else, or by doing something like this:

```
cd /tmp
mkdir ssr-bundle
cd ssr-bundle
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
$PATH_TO_SSR_SOURCE/configure --enable-app-bundle
make
make dmg
```

In this example, `$PATH_TO_SSR_SOURCE` is the directory where you put the SSR source files. Instead of `/tmp` you can of course use something else, but with `/tmp` it should work on most systems out there.

If you don't like this work-around, you may also play around with `fsaclctl`.

11.1.2 Only `WAVE_FORMAT_PCM` and `WAVE_FORMAT_IEEE_FLOAT` are supported.

Multi-channel WAV files would normally use the format `WAVE_FORMAT_EXTENSIBLE`, see https://web.archive.org/web/20220911060540/http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html.

However, Ecasound doesn't know this format, that's why we have to use one of the above mentioned formats, although for files with more than 2 channels this is not compliant to the WAV standard.

To check the exact format of your WAV files, you can use sndfile-info (Debian package sndfile-programs), and to convert your files, you can use, for example, sox (Debian package sox) with the wavpcm option:

```
sox old.wav new.wavpcm
mv new.wavpcm new.wav
```

11.1.3 SSR crashes with a segmentation fault on Max OS X

If this happens whenever you are opening an audio file or loading a scene that involves opening an audio file, then this might be due to Ecasound. We've seen this with the app bundle. Try the following:

Download the Ecasound source code from http://nosignal.fi/ecasound/download.php. `Cd` into the folder and compile Ecasound with:

```
./configure
make
```

Finally, replace the Ecasound executable in the SSR bundle with something like this:

```
sudo cp ecasound/ecasound /Applications/SoundScapeRenderer-0.4.2-74-gb99f8b2/
↪SoundScapeRenderer.app/Contents/MacOS/
```

You might have to modify the name of the SSR folder in the above command as you're likely to use a different version.

11.1.4 A file that can't be loaded results in a connection to a live input

If there is an error in loading a specified audio file, the corresponding source is still created and (unexpectedly) connected to the first soundcard input channel.

We believe this is a bug in the JACK system related to the API function jack_connect()_: If the `destination_port` argument is an empty string, the function returns (correctly) with an error. However, if the `source_port` argument is an empty string, the port is connected to the first "system" port (or the first port at all, or who knows . . .). And this is the case if the SSR cannot open a specified audio file.

If you also think that's a bug, feel free to report it to the JACK developers.

11.1.5 Conflicting JACK and Ecasound versions

There is a problem due to a special combination of ecasound and JACK versions on 64 bit systems leading to an error (terminating the SSR) similar to this:

```
 (ecasoundc_sa) Error='read() error', cmd='cs-connect' last_error='' cmd_cnt=6
last_cnt=5.
```

We experienced this error on 64 bit systems with ecasound version 2.4.6.1 and 2 .7.0 in combination with JACK version 0.118.0. A similar error occured on macOS with ecasound version 2.7.2 and JackOSX version 0.89 (with Jackdmp 1.9.8).

Please try to update to the newest ecasound and JACK versions.

11.1.6 Annoying Ecasound message

You may have seen this message:

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

You can totally ignore this, but if it bothers you, you can disable it easily by specifying the following line in `/etc/bash.bashrc` (system-wide setting) or, if you prefer, you can put it into your `$HOME/.bashrc` (just for your user account):

```
export ECASOUND=ecasound
```

11.1.7 Ecasound cannot open a JACK port

Sometimes, when Ecasound is installed via Homebrew, it can have trouble finding JACK. As a result SSR displays the sound source symbols in the GUI, but they don't play audio, and an according error message is posted in the SSR terminal.

Type `ecasound -c` in a terminal to start Ecasound in interactive mode. Then type `aio-register` to list all available outputs that Ecasound has recognized. If JACK is not listed, then download the Ecasound source code from http://nosignal.fi/ecasound/download.php, and

```
./configure --enable-jack
make
make install
```

The last line might have to be

```
sudo make install
```

11.1.8 Long paths to audio files on macOS

It can happen that SSR displays this error message when loading audio files directly:

```
Error: AudioPlayer::Soundfile: ERROR:  Connecting chainsetup failed: "Enabling␣
↪chainsetup: AUDIOIO-JACK: Unable to open JACK-client" (audioplayer.cpp:310)
Warning: AudioPlayer: Initialization of soundfile '/Users/YOUR_USERNAME/Documents/audio/
↪YOUR_AUDIO_FILE.wav' failed! (audioplayer.cpp:87)
```

Opening such a file would result in a JACK port name that is too long. You can resolve this limitation by moving the audio file to a location that produces a shorter (full) path name or by wrapping the audio file in an asd-file.

11.1.9 Segmentation Fault when Opening a Scene

This problem occured on some old SuSE systems.

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.

11.1.10 Choppy Sound on Cheap (On-Board) Sound Cards

Sometimes JACK doesn't play well with those on-board sound cards. One possibility to improve this, is to increase the frames/period setting from the default 2 to a more generous 3. This can be done in the Settings dialog of qjackctl or with the command line option `-n`:

```
jackd -n 3
```

11.1.11 `dylibbundler` doesn't grok Qt Frameworks

If `make dmg` doesn't copy the Qt `.dylib` files into the application bundle (to `Contents/Libraries`), you might try the following commands (or similar, depending on the exact Qt installation).

Go to the online manual to copy and paste them.

```
install_name_tool -id /opt/local/lib/libQtCore.dylib /opt/local/Library/Frameworks/
↪QtCore.framework/QtCore
install_name_tool -id /opt/local/lib/libQtGui.dylib /opt/local/Library/Frameworks/QtGui.
↪framework/QtGui
install_name_tool -change /opt/local/Library/Frameworks/QtCore.framework/Versions/5/
↪QtCore /opt/local/lib/libQtCore.dylib /opt/local/Library/Frameworks/QtGui.framework/
↪QtGui
install_name_tool -id /opt/local/lib/libQtOpenGL.dylib /opt/local/Library/Frameworks/
↪QtOpenGL.framework/QtOpenGL
install_name_tool -change /opt/local/Library/Frameworks/QtCore.framework/Versions/5/
↪QtCore /opt/local/lib/libQtCore.dylib /opt/local/Library/Frameworks/QtOpenGL.framework/
↪QtOpenGL
install_name_tool -change /opt/local/Library/Frameworks/QtGui.framework/Versions/5/QtGui␣
↪/opt/local/lib/libQtGui.dylib /opt/local/Library/Frameworks/QtOpenGL.framework/QtOpenGL
```

You need the appropriate rights to change the library files, so you probably need to use `sudo` before the commands.

WARNING: You can totally ruin your Qt installation with this stuff!

To get some information about a library, you can try something like those:

```
otool -L /opt/local/Library/Frameworks/QtOpenGL.framework/QtOpenGL
otool -l /opt/local/Library/Frameworks/QtOpenGL.framework/QtOpenGL
otool -D /opt/local/Library/Frameworks/QtOpenGL.framework/QtOpenGL
```

11.1.12 Second instance of SSR crashes

This happens when two or more instances of the SSR are started with the IP server enabled. Start all (or at least all instances higher than 1) with the `-I` flag to disable the IP interface.

11.1.13 Error `ValueError: unknown locale: UTF-8` when building the manual

This can happen on non-US Macs. Go to your home folder `/Users/YOUR_USER_NAME`, open (or create) the file `.bash_profile` and add the following to this file:

```
export LC_ALL=en_US.UFT-8
export LANG=en_US.UTF-8
export LANGUAGE=en_US.UTF-8
export LC_CTYPE=en_US.UTF-8
```

You might have to re-open the terminal or log out and in again to see the effect.

11.2 Resolved Issues

11.2.1 SSR for macOS: qt_menu.nib not found

This was fixed in MacPorts, see https://trac.macports.org/ticket/37662. Thanks to Chris Pike! Since version 0.5 (switching to qt5), qt_menu.nib is not needed any more.

11.2.2 Compilation Error on Ubuntu and Archlinux

This issue was resolved in version 0.3.4. Some newer distributions got more picky about the necessary `#include` commands. If the SSR refuses to compile, add this to the file `src/gui/qopenglplotter.h` (somewhere at the beginning):

```
#include <GL/glu.h>
```

On macOS you'll need this instead:

```
#include <OpenGL/glu.h>
```

11.2.3 Polhemus tracker does not work with SSR

This issue was resolved in version 0.3.3, where we changed the tracker selection. Use `--tracker=fastrak`, `--tracker=patriot` and `--tracker=intersense`, respectively. The serial port can be specified with `--tracker-port=/dev/ ttyUSB0` (or similar).

This can happen when both the Intersense tracker as well as the Polhemus tracker are compiled and the file `isports.ini` is present. The latter tells the Intersense tracker which port to use instead of the standard serial port `/dev/ttyS0`. If the `isports.ini` file lists the port to which the Polhemus tracker is connected, it can happen that something that we have not fully understood goes wrong and the Pohlemus data can not be read. In this case you can either rename the file isports.ini or change its content.

It might be necessary to execute `echo C > /dev/ttyS0` several times in order to make Polhemus Fastrak operational again. Use `echo -e "C\r" > /dev/ttyS0` for Polhemus Patriot. You can check with `cat /dev/ttyS0` if it delivers data.

11.2.4 Missing GUI Buttons and Timeline

This issue was resolved in version 0.3.2, the default setting for `--enable-floating-control-panel` is chosen depending on the installed Qt version. As of version 0.5 (switching to qt5), the floating control panel is always enabled.

Different versions of Qt show different behaviour regarding OpenGL Overlays and as a result, the GUI buttons are not shown in newer Qt versions.

To overcome this limitation, we provide two GUI variants:

- Traditional GUI, can be used up to Qt 4.6.x
- Floating control panel, which is used with Qt 4.7 and above

The floating control panel is the default setting on macOS, for Linux it can be activated with:

```
./configure --enable-floating-control-panel
```

11.2.5 OpenGL Linker Error

This issue was resolved in version 0.3.2.

On some systems, after running make, you'll get an error mentioning " glSelectBuffer".

For now, this is the solution (see also the issue below):

```
./configure LIBS=-lGL
```

11.2.6 IP interface isn't selected although boost libraries are installed

This issue was resolved with dropping boost::asio for asio in version 0.5.0.

For older builds, you might need to add the `-lpthread` flag:

```
./configure LIBS=-lpthread
```

11.2.7 Audio files with spaces

This issue was resolved in version 0.3.2.

Please do not use audio files with spaces for scenes. Neither the filename nor the directory referenced in the scene (asd-file) should contain spaces.

BIBLIOGRAPHY

[Geier2008a] Matthias Geier, Jens Ahrens, and Sascha 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).

[Geier2012] Matthias Geier and Sascha Spors. Spatial audio reproduction with the SoundScape Renderer. In 27th Tonmeistertagung – VDT International Convention, 2012.

[Geier2008b] Matthias Geier, Jens Ahrens, and Sascha Spors. ASDF: Ein XML Format zur Beschreibung von virtuellen 3D-Audioszenen. In 34rd German Annual Conference on Acoustics (DAGA), Dresden, Germany, March 2008.

[Ahrens2008a] Jens Ahrens and Sascha 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.

[Ahrens2008b] Jens Ahrens and Sascha 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.

[Lindau2007] Alexander Lindau and Stefan Weinzierl. FABIAN - Schnelle Erfassung binauraler Raumimpulsantworten in mehreren Freiheitsgraden. In Fortschritte der Akustik, DAGA Stuttgart, 2007.

[Wierstorf2011] Hagen Wierstorf, Matthias Geier, Alexander Raake, and Sascha Spors. A Free Database of Head-Related Impulse Response Measurements in the Horizontal Plane with Multiple Distances. In 130th Convention of the Audio Engineering Society (AES), May 2011.

[SpatialAudio] https://github.com/spatialaudio/lf-corrected-kemar-hrtfs (commit 5b5ec8)

[AlgaziCIPIC] V. Ralph Algazi. The CIPIC HRTF database. https://web.archive.org/web/20170916053150/interface.cipic.ucdavis.edu/sound/hrtf.html.

[BRIRs] The Sputnik BRIRs can be obtained from here: https://github.com/ssr-scenes/tu-berlin/tree/master/sputnik. More BRIR repositories are compiled here: http://www.soundfieldsynthesis.org/other-resources/#impulse-responses.

[Pulkki1997] Ville Pulkki. Virtual sound source positioning using Vector Base Amplitude Panning. In Journal of the Audio Engineering Society (JAES), Vol.45(6), June 1997.

[Spors2008] Sascha Spors, Rudolf Rabenstein, and Jens Ahrens. The theory of Wave Field Synthesis revisited. In 124th Convention of the AES, Amsterdam, The Netherlands, May 17–20, 2008.

[Spors2006] Sascha Spors and Rudolf 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.

[Neukom2007] Martin Neukom. Ambisonic panning. In 123th Convention of the AES, New York, NY, USA, Oct. 5–8, 2007.

[Spors2011] Sascha Spors, Vincent Kuscher, and Jens Ahrens. Efficient Realization of Model-Based Rendering for 2.5-dimensional Near-Field Compensated Higher Order Ambisonics. In IEEE WASPAA, New Paltz, NY, USA, 2011.

[Ma2019] X. Ma, C. Hohnerlein, J. Ahrens. Concept and Perceptual Validation of Listener-Position Adaptive Superdirective Crosstalk Cancelation Using a Linear Loudspeaker Array. JAES 67(11), p. 871-881, 2019, DOI: 10.17743/jaes.2019.0037