Getting Started With SART3D

Requirements:

- Matlab R2013b
- Signal Processing Toolbox
- DSP System Toolbox
- Sound Field Synthesis Toolbox (optional, for sound field synthesis methods)
- Audio Hardware with ASIO drivers. If your soundcard does not have specific ASIO drivers, you can install generic ASIO4ALL v2 drivers from http://www.asio4all.com/.

What can I do with SART3D?

SART3D lets you move in real time virtual audio sources from a set of WAV files using multiple spatial audio rendering methods. You can specify locations of virtual sources and loudspeakers and change among the different available rendering methods.

The programmatic structure of the GUI lets the user experience with any loudspeaker setup specified in the configuration structure.

The modular design of the Toolbox lets the user create new rendering methods and experiment with different parameters. As a result, SART3D is a very useful tool for spatial audio research and education.

How do I give it a quick try?

To start playing with SART3D, just be sure to have an available ASIO driver in your system.

1. First, go to Matlab's Preferences and set DSP System Toolbox Audio API to ASIO (Figure 1).
2. Then, change the current Matlab's working folder to your SART3D folder and type SART3D. The toolbox is configured to load automatically a default two-loudspeaker set-up (Figure 2).
3. Check that the bottom pop-up menu contains the ASIO driver that you want to use. Here you can also select the ASIO buffer size and the rendering method (Figure 3).
4. To start listening to a given sound source, select the ones that you want to hear using the checkboxes and push the play button (in this example, we select source 3). Drag and drop the sources to different locations or specify positions using the input edit text boxes (Figure 4). Stop the playback by pressing again the play button.

Note: *When a source location is not reachable with the current set-up/reproduction method the source will not be audible.*

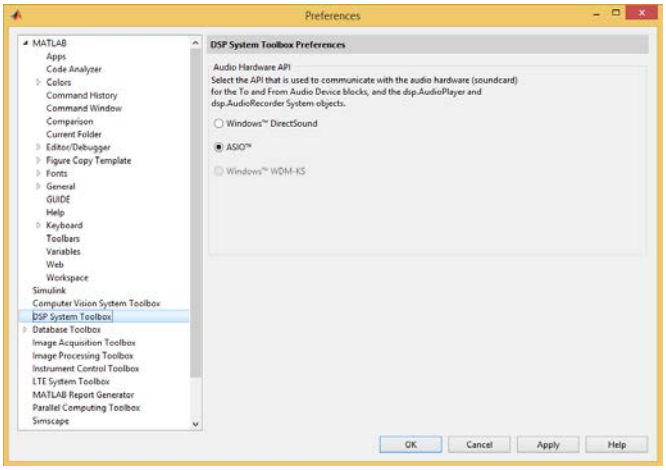


Figure 1. Matlab DSP System Toolbox Audio Hardware API.

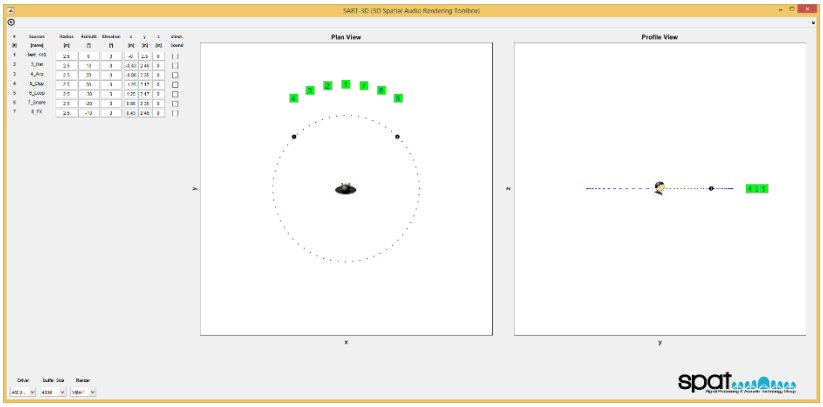


Figure 2. Default reproduction set-up.

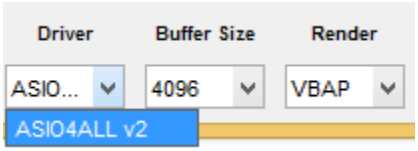
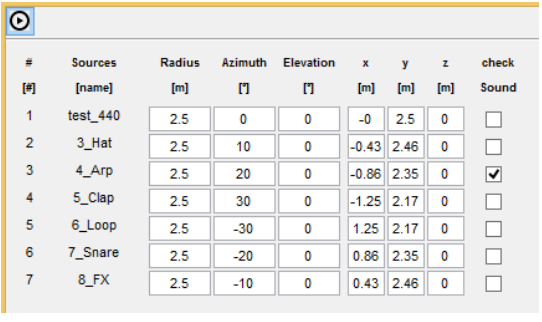


Figure 3. Driver and reproduction method selection.



#	Sources	Radius	Azimuth	Elevation	x	y	z	check
[#]	[name]	[m]	[°]	[°]	[m]	[m]	[m]	Sound
1	test_440	2.5	0	0	-0	2.5	0	☐
2	3_Hat	2.5	10	0	-0.43	2.46	0	☐
3	4_Arp	2.5	20	0	-0.86	2.35	0	☑
4	5_Clap	2.5	30	0	-1.25	2.17	0	☐
5	6_Loop	2.5	-30	0	1.25	2.17	0	☐
6	7_Snare	2.5	-20	0	0.86	2.35	0	☐
7	8_FX	2.5	-10	0	0.43	2.46	0	☐

Figure 4. Selecting sources for playback and editing positions.

How can I change the set-up?

First, put your WAV files in a folder within the /audioscenes directory.

The script 'gConfig.m' lets the user create a valid configuration structure. Please, navigate throughout the script to create a configuration fitting your needs. You can also open the 'gConfig.html' page to see a published version of the script.

Once you have modified and run 'gConfig' according to the desired configuration, a new 'conf.mat' file will be in the /configurations directory. Then, you can specify this new configuration in SART3D:

```
>> SART3D('conf.mat')
```

Can I see an example?

Yes, let's create a 5 loudspeaker setup instead of the default stereo setup.

Go to 'gConfig.m' and specify the spherical coordinates of the loudspeakers in the 'Loudspeaker Locations' section:

```
% *Loudspeaker Locations*:
% Loudspeaker locations are defined in spherical coordinates. For example,
% for a two-loudspeaker (or headphones) set-up:
conf.LS.coord = {
        [1.75, +30, 0],...
        [1.75, 0, 0],...
        [1.75, -30, 0],...
        [1.75, -150, 0],...
        [1.75, +150, 0]};
```

Specify the correct audio channel mapping in your set-up, i.e. which audio channel in your audio interface corresponds to loudspeaker 1, which one to loudspeaker 2, etc. By default, loudspeaker 1 goes to audio channel 1, loudspeaker 2 to channel 2, etc. However you can specify the desired mapping as follows:

```
conf.driver.ChannelMapping = [5 4 3 2 1].
```

This would tell the software that loudspeaker one is driven by the output audio channel 5, loudspeaker 2 by audio channel 4 and so on.

Save and run the script to overwrite the 'conf.mat' file. Now, run again SART3D specifying the new configuration:

```
>> SART3D('conf.mat')
```

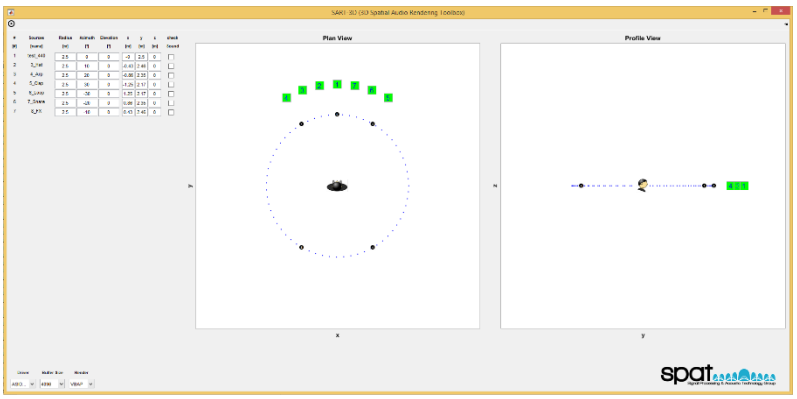


Figure 5. Five loudspeaker set-up.

Which spatial audio rendering methods are supported?

Currently, the toolbox incorporates:

- VBAP Vector Base Amplitude Panning
- Stereo Panning Sine Law
- Stereo Panning Tangent Law
- Ambisonics Amplitude Panning
- Binaural Rendering
- Wave Field Synthesis (makes use of the SFS Toolbox)
- Near-Field Compensated Higher Order Ambisonics (makes use of the SFS Toolbox)

How do I create a new rendering method?

To create a new rendering method, you have to modify some of the functions in SART3D.

Let us create a very simple invented rendering method that consists in selecting the closest loudspeaker to the virtual source. Let's call it CLOSEST.

First, specify the new method in gConfig.m:

```
conf.methods.names = {'VBAP'; 'AAP'; 'HRTF'; 'StTL'; 'StSL'; 'WFS'; 'NFCHOA'; 'CLOSEST'};

% The initially selected method:
conf.methods.selected = 8;
```

Now edit gCheckConfig.m to let the GUI know how to initialize the method. In the 'Rendering Method Initialization' section , enter a new case for the method within the `switch conf.methods.selected` statement:

```
    case 8 % CLOSEST
        % -----------------------------------------------------------------
        % Initialization routine
        % -----------------------------------------------------------------
        CLOSESTstart;
```

Create a new folder 'CLOSEST' for the method in the /algorithms directory and create the initialization script 'CLOSESTstart.m'. In this case, the initialization script just specifies that the rendering filter is just a gain (one coefficient):

```
% Number of coefficients (just a gain)
conf.nCoeffs = 1;

% Add all the necessary parameters of the method in the conf structure.
conf.CLOSEST = '';
```

Now, we have to specify the function that calculates the filter coefficients and selects the contributing loudspeakers in our new method. We call it 'gCLOSEST.m':

```
function [H,I] = gCLOSEST(sph)
%GCLOSEST Example of Algorithm. Selects the closest loudspeaker for the
%rendering.
%
% Usage:
%   [H,I] = gCLOSEST(sph)
%
% Input parameters:
%   sph - Spherical coordinates of the virtual source [r,Az,El].
```

```
%
% Output paramters:
%   H - Gain of the selected loudspeaker.
%   I - Selected loudspeaker.
%
% See also: StSLstart, gStTL

global conf;

car = gSph2Car(sph);

dist = sqrt((conf.LS.car(1,:)-car(1)).^2 + (conf.LS.car(2,:)-car(2)).^2 + (conf.LS.car(3,:)-car(3)).^2);

[d,I] = min(dist);

% Distance attenuation
r = max(d, conf.rMin);
H = conf.rMin/r;
```

We save the above rendering function in /algorithms/CLOSEST. Then, we include in the filter updating function 'gRefreshH.m' the new case within the `switch conf.methods.selected` statement:

```
    % INCLUDE HERE MORE CASES FOR NEW RENDERING METHODS!
    case 'CLOSEST'
        [H,I] = gCLOSEST(data.vSSph(:,ii));
```

Finally, to let the user change among this method and the rest by using the pop-up menu, go to 'GPopupmenu.m' object and add a case for the new algorithm within the `switch conf.methods.names{get(handles.pmMethod, 'Value')}` statement:

```
                    case  'CLOSEST'
                        conf.nCoeffs = 1;
                        if isfield(conf,'CLOSEST')==0
                            CLOSESTstart;
                        end
```