Master Thesis: SoundScape Renderer Networking

David Runge
Audiokommunikation und -technologie
Fachgebiet Audiokommunikation
Technische Universität Berlin
dave@sleepmap.de

October 13, 2016

Abstract

Wave Field Synthesis (WFS) as a technological concept has been around for many years now and all over the world several institutions run small and some even large scale setups ranging from single speaker lines to those facilitating a couple of hundred loudspeakers respectively.
The still evolving implementations are driven by several rendering engines, of which two free and open-source ones, namely sWONDER and SoundScape Renderer, have (partially) been developed at TU Berlin.
The latter due to its current design is not yet able to render for large scale setups, ie. those using several computers to render audio on a loudspeaker setup, due to the high amount of channels.
Its solid codebase however, which additionally offers a framework for many more renderering types, and the ongoing development, deems further work on this application a good future investment.
This work is about the extension of the SoundScape Renderer's functionality to turn it into a networking application for large scale WFS setups.

0.1 Introduction

Wave Field Synthesis (WFS) describes a spatial technique for rendering audio. As such it aims at synthesizing a sound field of desired acoustic preference in a given listening area, assuming a planar reproduction to be most suitable for most applications.
WFS is typically implemented using a curved or linear loudspeaker array surrounding the listening area.
Several free and open-source renderer applications exist for WFS environments, with varying stages of feature richness.
The proposed work will focus on one of them and its extension towards WFS on large scale systems.

0.2 Free and open-source wave field synthesis renderers

To date there exist three (known of) free and open-source Wave Field Synthesis renderers, which are all JACK Audio Connection Kit (JACK) [**?**] clients:

- sWONDER [**?**], developed by Technische Universität Berlin, Germany
- WFSCollider [**?**], developed by Game Of Life Foundation [**?**], The Hague, Netherlands
- SoundScape Renderer (SSR) [**?**], developed by Quality & Usability Lab, Deutsche Telekom Laboratories and TU Berlin and Institut für Nachrichtentechnik, Universität Rostock

Currently only WFSCollider and the SSR are actively maintained and developed, thussWONDER, although used in some setups, loses significance. Generally it can be said, that different concepts apply to the three renderers, which are about to be explained briefly in the following sections.

0.2.1 WONDER

sWONDER [**?**] consists of a set of C++ applications that provide binaural and WFS rendering. In 2007 it was specifically redesigned [**?**] to cope with large scale WFS setups in which several (computer) nodes, providing several speakers each, drive a system together.
In these setups each node receives all available audio streams (which represent

one virtual audio source respectively) redundantly and a master application signals which node is responsible for rendering what source on which speaker. It uses Open Sound Control (OSC) for messaging between its parts and for setting its controls. Apart from that, it can be controlled through a Graphical User Interface (GUI), that was specifically designed for it. Unfortunately sWONDER has not been actively maintained for several years, has a complex setup chain and many bugs, that are not likely to get fixed any time soon.

0.2.2 WFSCollider

WFSCollider was built on top of SuperCollider 3.5 [**?**] and is also capable of driving large scale systems. It uses a different approach in doing so, though: Whereas withsWONDER all audio streams are distributed to each node, WFSCollider usually uses the audio files to be played on all machines simultaneously and synchronizes between them.
It has a feature-rich GUI in the "many window" style, making available time lines and movement of sources through facilitating what the sclang (SuperCollider programming language) has to offer.
As WFSCollider basically is SuperCollider plus extra features, it is also an OSC enabled application and can thus also be used for mere multi-channel playback of audio.
Although it has many useful features, it requires MacOSX (Linux version still untested) to run, is built upon a quite old version of SuperCollider and is likely never to be merged into it, due to many core changes to it.

0.2.3 SoundScape Renderer

SoundScape Renderer (SSR), also a C++ application, running on Linux and MacOSX, is a multi-purpose spatial audio renderer, as it is not only capable of Binaural Synthesis and WFS, but also Higher-Order Ambisonics and Vector Base Amplitude Panning.
It can be used with a GUI or headless (without one), depicting the virtual sources, their volumes and positions, alongside which speakers are currently used for rendering a selected source. SSR uses TCP/IP sockets for communication and thus is not directly OSC enabled. This functionality can be achieved using the capapilities of other applications such as PureData [**?**] in combination with it though.
Unlike the two renderers above, the SSR is not able to run large-scale WFS setups, as it lacks the features to communicate between instances of itself on several computers, while these instances serve a subset of the available loudspeakers.

0.3 Extending Sound Scape Renderer functionality

The SSR, due to its diverse set of rendering engines, which are made available through an extensible framework, and its clean codebase, is a good candidate for future large scale WFS setups. These type of features are not yet implemented though and will need testing.
Therefore I propose the implementation and testing of said feature, making the SSR capable of rendering on large scale WFS setups with many nodes, controlled by a master instance.
The sought implementation is inspired by the architecture of sWONDER, but instead of creating many single purpose applications, the master/node feature will be made available through flags to the ssr executable, when starting it. This behavior is already actively harnessed eg. for selecting one of the several rendering engines. While the SSR already has an internal logic to

Figure 1: A diagram displaying the SSR master/node setup with TCP/IP socket connections over network (green lines), audio channels (red dashed lines) and OSC connection (blue dashed line). Machines are indicated as red dashed rectangles and connections to audio hardware as outputs of SSR nodes as black lines below them.

know which loudspeaker will be used for what virtual audio source, this will have to be extended to be able to know which renderer node has to render what source on which loudspeaker (see Figure 1). To achieve the above features, the SSR's messaging (and thus also settings) capabilities have to be extended alongside its internal logic concerning the selection of output channels (and the master to node notification thereof). To introduce as little redundant code as possible, most likely a "the client knows all" setup is desirable, in which each node knows about the whole setup, but is also set to only serve its own subset of loudspeakers in it. This will make sure that the rendering engine remains functional also in a small scale WFS setup.
The lack of a direct OSC functionality, as provided by the two other renderers, will not be problematic, as master and nodes can communicate through their builtin TCP/IP sockets directly and the master can, if needed, be controlled via OSC.

0.4 Prelimenaries

In preparation to the exposé I tried to implement a side-by-side installation, using Arch Linux on a medium scale setup, facilitating the WFS system of the Electronic Studio at TU Berlin. Unfortunately the proprietary Dante

driver, that is used in that system is very complex to be built, as well as underdeveloped and thus keeps the system from being easily updated, which is needed for testing purposes (finding a suitable real-time, low-latency Linux kernel), trying out new software features, building new software and keeping a system safe. The driver will most likely require changes to the hardware due to implemention of hardware branding by the vendor and dire testing before usage.
Although eventually using a proper WFS setup for testing will be necessary, it is luckily not needed for implementing the features, as they can already be worked out using two machines running Linux, JACK and the development version of SSR.
The hardware of the large scale setup at TU Berlin in H0104 is currently about to be updated and therefore a valuable candidate for testing of the sought SSR features.

0.5 Schedule

I propose a six month schedule for the implementation and testing of the changes to the source code and writing of an accompanying thesis. The following rough schedule should serve as a guideline for the realization of the work:

Schedule			
Week	**Implementation**	**Tests**	**Thesis**
1	Reading into codebase		
2	Reading into codebase		
3	Reading into codebase		
4	Reading into codebase		
5	Assessing changes		Documentation
6	Assessing changes		Documentation
7	Implementing changes		
8	Implementing changes		
9	Implementing changes		
10	Implementing changes		
11	Implementing changes		
12	Implementing changes		
13	Implementing changes		Preparation
14	Implementing changes		Preparation
15		Small scale setup	Writing
16		Large scale setup	Writing
17		Large scale setup	Writing
18		Large scale setup	Writing
19	Large scale setup (scripts)		Writing
20	Large scale setup (scripts)		Writing
21	Large scale setup (scripts)		Writing
22			Writing
23			Writing
24			Writing

Bibliography