

Renewed architecture of the *sWONDER* software for Wave Field Synthesis on large scale systems

Marije A.J. Baalman Torben Hohn Simon Schampijer
Thilo Koch Daniel Plewe Eddie Mond

Institute for Audio Communication
Technische Universität Berlin

24.03.2007

Outline

Introduction

Hardware setup

Software architecture

- Renderer

- Control

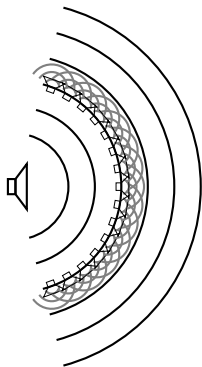
- Caching filter coefficients

Synchronisation

Current Tools

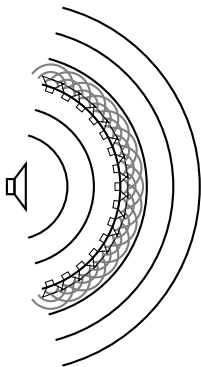
What is Wave Field Synthesis?

Huygens' principle

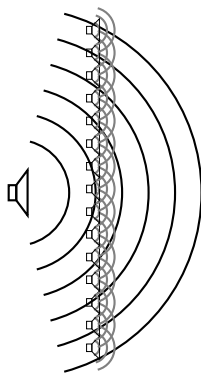


What is Wave Field Synthesis?

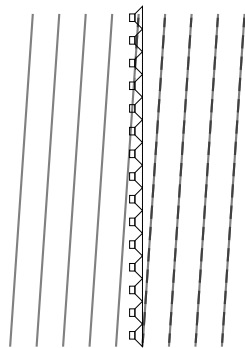
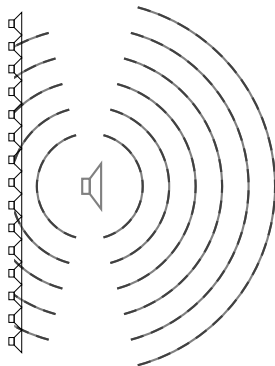
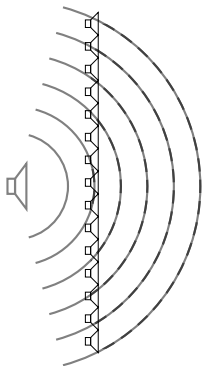
Huygens' principle



WFS



Basic source types



Wave Field Synthesis for lecture hall H0104

Project part of renovation of the lecture halls, including the renewal of the media facilities

Participants:

Planning Christoph Moldrzyk

Loudspeakers Anselm Görtz, Christoph Moldrzyk

Software Marije Baalman

control Simon Schampijer

render Torben Hohn

GUI Eddie Mond

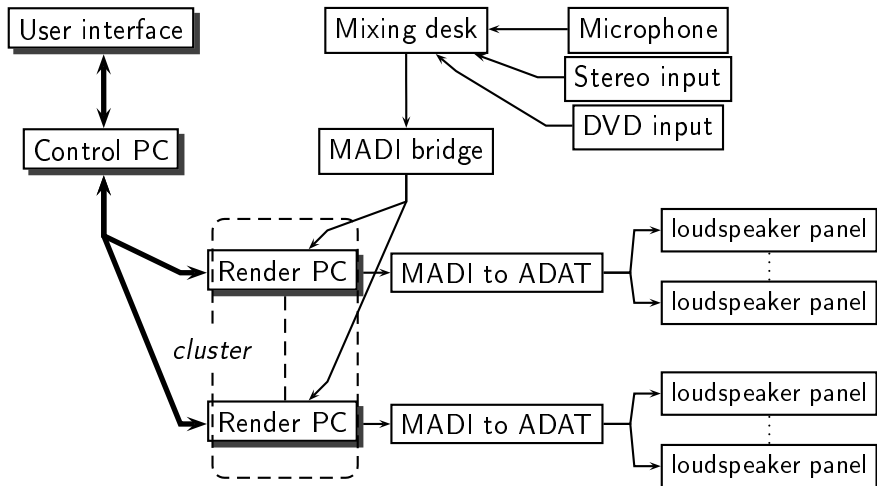
score Daniel Plewe

cluster / offline render Thilo Koch

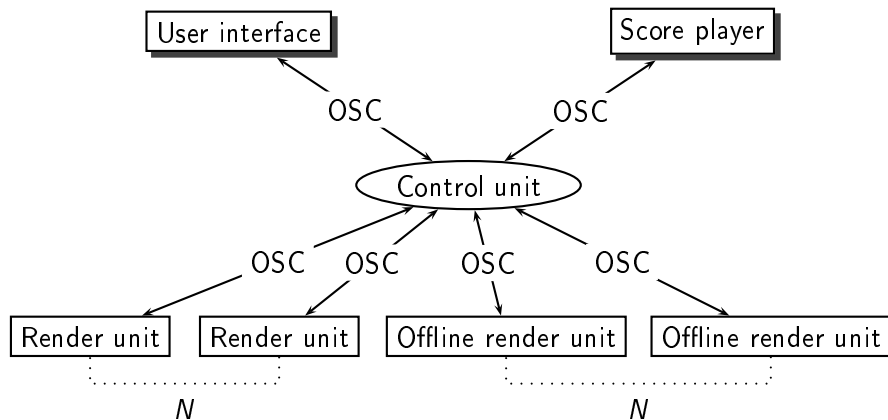
Some numbers

- ▶ ca. 700 seats
- ▶ 840-channel WFS system
- ▶ 2730 loudspeakers
- ▶ speaker distance 10 cm.
- ▶ ca. 100 meter total wall length

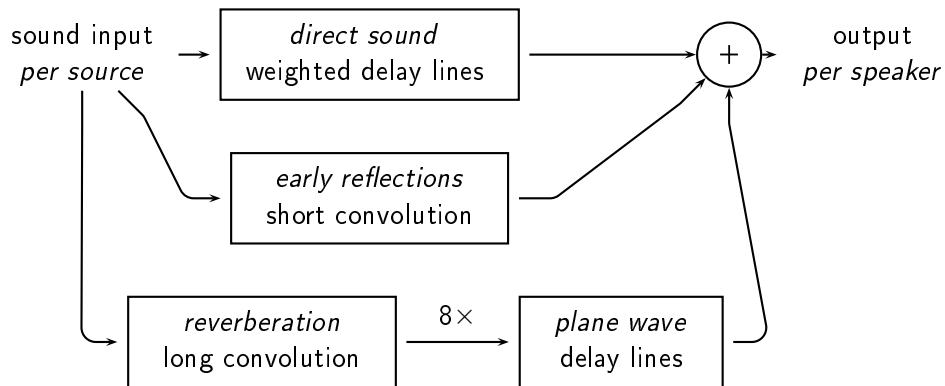
Hardware setup



Software architecture



Rendering architecture



Control unit

The control unit reacts on messages from a user interface, parses these and gives subsequent commands to the renderer and offline renderer, and informs the user interface of the current status of the system.

Renderer just executes commands, i.e. the control unit does everything that needs some kind of logic and scheduling, such as:

- ▶ Loading and unloading of filter coefficients for early reflection or reverberation
- ▶ Conversion from score time to frame/sample time

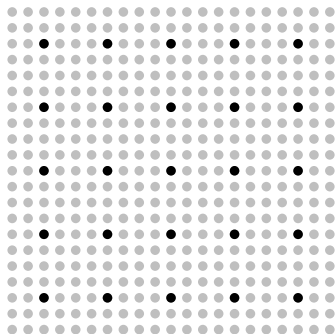
Example:

- ▶ User moves a sound source in a room with early reflections and reverberation
- ▶ Control unit sends the renderers:
 1. Position of the sound source
 2. The corresponding grid point ID for the early reflections
 3. Grid point ID's of neighbouring points to load in memory, so that future position updates can be followed quickly with appropriate early reflections
- ▶ Control unit sends the user interface information about the data sent to the renderers, so that the user interface can display the status to the user

Filter coefficients loading and unloading

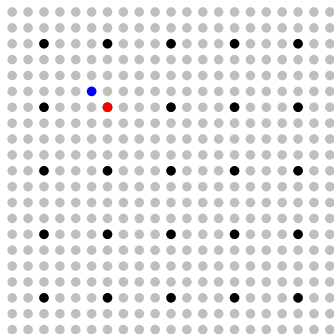
Grid layout

Black points are always loaded in memory



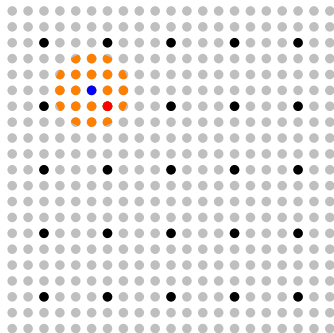
Filter coefficients loading and unloading

Source moves
Base point is chosen for early
reflections



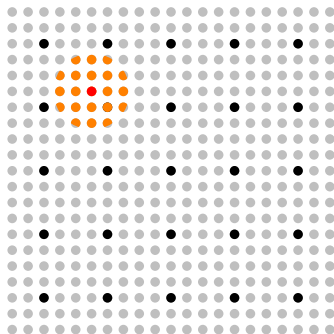
Filter coefficients loading and unloading

Actual points in neighbourhood
of source location are loaded



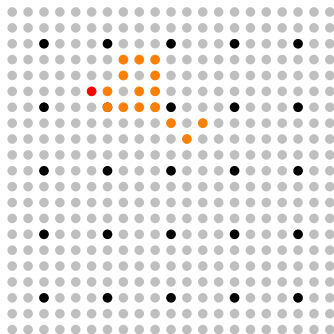
Filter coefficients loading and unloading

Early reflections of actual source point are used



Filter coefficients loading and unloading

When a score is present, we know where sources will be in advance, and we can preload all the needed filter coefficients.



Dealing with time

Internally, between control unit and renderers

External: between control unit and user interface

Internal synchronisation

- ▶ Important as changes in source position need to take place at the same time between all renderers.
- ▶ The only reliable clock we have is the audio clock.
- ▶ In order to make use of this, we synchronise on the audio blocks, by starting JACK simultaneously. A small program *jWONDER* (or *jackframetime*) then keeps the render units and the control unit up-to-date about the current time within the system.

External synchronisation

Commands are given to the control unit in seconds from *now*.

The score player can synchronise to Midi Time Control (MTC) to play a score of source movements.

Current Tools

- ▶ Control unit
- ▶ Render unit
- ▶ Score player
- ▶ LADSPA plugin for single source control
- ▶ LADSPA plugin for source group control
- ▶ 3D Visualiser (implemented in Fluxus)
- ▶ simple GUI (*gWONDER*)
- ▶ SuperCollider class