Sociedade de Engenharia de Áudio

Artigo de Congresso

Apresentado no 12º Congresso de Engenharia de Áudio
18ª Convenção Nacional da AES Brasil
13 a 15 de Maio de 2014, São Paulo, SP

Programação dinâmica em *Pure Data* aplicada à *Wave Field Synthesis*

Marcio José da Silva,[1,2] Flávio Luiz Schiavoni[3] e Regis Rossi A. Faria[2,4]

[1] Universidade de São Paulo, Escola de Comunicações e Artes, Departamento de Música
São Paulo, SP, 05508-020, Brasil

[2] Lab. de Sistemas Integráveis da USP, Núcleo de Engenharia de Áudio e Codificação Sonora
São Paulo, SP, 05508-010, Brasil

[3] Universidade de São Paulo, Instituto de Matemática e Estatística, Departamento de Ciência da Computação
São Paulo, SP, Brasil

[4] Universidade de São Paulo, Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, Departamento de Música
Ribeirão Preto, SP, 14.040-900, Brasil

marcio.jose.silva@usp.br, fls@ime.usp.br, regis@usp.br

RESUMO

Analisando-se as possíveis implementações para processamento em tempo real no ambiente *Pure Data*, este artigo apresenta uma solução para a geração automática de *patches* aplicados à sonorização de um sistema de espacialização baseado em *WFS - Wave Field Synthesis* (ou Síntese de Campo de Onda). A solução emprega *patches* dinâmicos e uma arquitetura modular, permitindo flexibilidade e manutenabilidade do código, com vantagens particularmente para lidar com um número elevado de fontes e alto-falantes.

0 INTRODUÇÃO

Este artigo apresenta soluções para o desenvolvimento de um código de programação de estrutura modular dedicado à aplicação da técnica de sonorização de *WFS - Wave Field Synthesis* (ou Síntese de Campo de Onda). Esta técnica objetiva a criação de ambientes acústicos que permitem ao ouvinte a identificação da posição de um objeto sonoro numa grande região do espaço.

O desenvolvimento do programa foi proposto como

projeto de pesquisa do NEAC[1] (Núcleo de Engenharia de Áudio e Codificação Sonora), buscando-se a construção de uma implementação computacional modular e flexível para *WFS*, acoplado a um sistema desenvolvido neste mesmo núcleo, o AUDIENCE[2], uma biblioteca de objetos e abstrações para a plataforma *Pure Data (Pd)*[3]. Esta plataforma será apresentada na Subseção 0.2.

Dada a complexidade do desenvolvimento do sistema de sonorização para *WFS*, este artigo apresenta algumas soluções encontradas para a configuração do sistema e, também, uma solução adotada baseada em programação dinâmica no ambiente *Pure Data*.

A programação dinâmica é uma técnica computacional na qual o programa pode alterar seu próprio código em tempo de execução, permitindo assim que o mesmo se adeque dinamicamente a um problema específico [1].

Neste artigo apresentaremos os conceitos básicos de *WFS*, a visão geral do sistema e a implementação do mesmo, trazendo as escolhas computacionais feitas durante o processo de desenvolvimento.

0.1 O que é Wave Field Synthesis

Wave Field Synthesis foi introduzida em 1988 por A. J. Berkhout (Universidade TU Deft, Holanda). Esta técnica é uma aplicação do princípio introduzido no século XVII pelo físico holandês Christiaan Huygens [2]. Seu princípio fundamental é a modelagem física, através da propagação e superposição de várias pequenas frentes de onda, da síntese de ondas sonoras com os mesmos atributos físicos que seriam gerados por objetos reais.

A área de escuta, região que delimita onde os ouvintes devem estar para que tenham a percepção esperada dos sons gerados durante a espacialização sonora, tende a ser muito pequena (*sweet spot*) nas mais diversas técnicas. Já a *WFS* tem a capacidade de projetar objetos sonoros numa grande área de audição, atendendo um número maior de ouvintes simultaneamente, além de produzir imagens sonoras mais definidas e estáveis [3].

Num sistema prático *WFS*, com a utilização de alto-falantes densamente distribuídos na área de audição, espaçados em torno de 10 a 20 cm, ocorre a discretização da projeção do som [4]. Nesta técnica, cada alto-falante emite um sinal de áudio em instantes controlados de tempo, para que a soma das contribuições destes sinais possa sintetizar a frente de onda circular, correspondente à onda real [5].

A implementação de um protótipo para projeções sonoras com *WFS* requer uma formulação computacionalmente complexa devido à configuração do sistema, onde são exigidos muitos canais de áudio. A necessidade inicial de configuração era de 16 canais de saída, porém pode-se trabalhar com configurações de centenas de canais [5]. Para uma simulação de espacialização é necessário configurar a quantidade canais de entrada (ou objetos sonoros) e a quantidade de canais de saída.

0.2 *Pure Data*

O *Pure Data* [6] (conhecido por Pd) é um sistema que funciona como um ambiente gráfico de programação musical. Pode ser simultaneamente programado e operado em tempo-real, sendo amplamente utilizado por músicos, artistas e *sound designers*.

Além de preencher os requisitos que necessitávamos para a implementação do nosso projeto, o Pd ainda possui outras vantagens como ser uma ferramenta *opensource*, ser multiplataforma e funcionar em vários sistemas operacionais.

A programação em *Pure Data* também é bastante simples por utilizar o conceito de fluxos e blocos (como *pipes* e *filters*) permitindo a utilização de expressões matemáticas que alterem um fluxo de áudio. Um programa de Pd é chamado de *patch*.

Outra vantagem deste ambiente é a capacidade que o mesmo possui de aceitar extensões. O *Pure Data* pode ser extendido por meio da criação de novos objetos. Estes novos objetos podem ser feitos na formas de *externals* em linguagem C ou na forma de *abstractions* quando feitos em *patches*, com outros objetos do próprio ambiente.

Além disto, a escolha deste ambiente simplifica a integração do módulo de *WFS* ao sistema AUDIENCE, também implementado em tal plataforma.

1 VISÃO GERAL DO SISTEMA

No desenvolvimento deste sistema de auralização, deve-se definir as possíveis configurações de painéis de alto-falantes. Entretanto, durante o uso do sistema, as posições e a quantidade de caixas de som normalmente é fixa. Já o número de canais de entrada, usualmente, tanto pode ser fixo como pode ser alterado se, por exemplo, forem usados arquivos de áudio com número diferente de canais. Para uma dada configuração fez-se necessário que o programa pudesse permitir o uso de um número variável de canais de entrada e saída do sistema. Conforme apresentado na figura 1, o sistema possui uma configuração bastante complexa devido à quantidade de informações correspondentes às suas entradas e saídas.

Para cada sinal S_f de entrada é atribuída a posição (x_f, y_f). Para cada canal de saída C_n temos a configuração da posição da respectiva caixa no eixo X. Vale notar que a posição da caixa no eixo Y é fixa.

2 IMPLEMENTAÇÃO

A implementação do primeiro protótipo em Pd, apresentada na figura 2, mostrou-se eficaz para os objetivos do projeto, porém tal implementação com 12 módulos *WFS* desenvolvidos para um único objeto sonoro (uma entrada de áudio) demandou bastante tempo

[1] NEAC. Acesso em: www.lsi.usp.br/neac

[2] AUDIENCE: Sistema e Software para Imersão Sonora e Auralização. Acesso em: http://www.lsi.usp.br/neac/audience

[3] Acesso em: http://puredata.info/

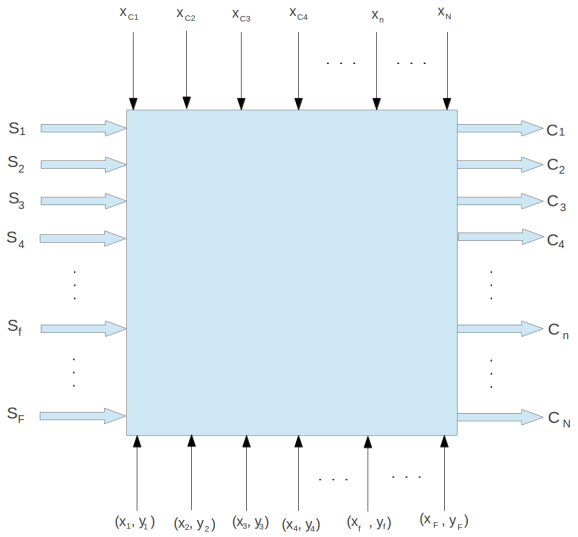

Figura 1: Diagrama geral *WFS*.

e se mostrou pouco flexível. Nesta figura tem-se uma visão geral do processamento, onde é possível verificar que a posição de um objeto sonoro irá alterar o sinal de todos os canais de saída.

Usar programação dinâmica para flexibilizar a configuração e aliviar a replicação de código é bastante atraente para o caso específico de *WFS* em que os sistemas podem chegar a centenas de alto-falantes.

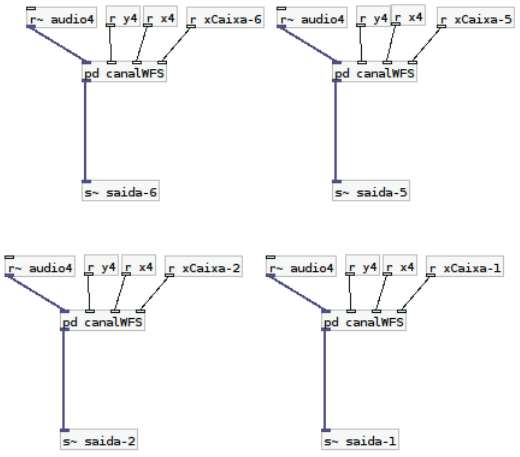

Figura 2: Parte do primeiro protótipo, com 4 dos 12 módulos para *WFS*.

2.1 Protótipo inicial

A baixa flexibilidade desta implementação pode ser notada pela quantidade de objetos e conexões necessários à implementação do sistema que inicialmente trabalha com apenas um objeto sonoro. Seguir tal linha de implementação implicaria replicar todos os objetos e conexões para tratar mais objetos sonoros. No caso mostrado, para cada novo canal de entrada teria que ser feita uma nova cópia destes 12 módulos e teriam que ser renomeados, de forma manual, vários objetos, a maioria deles não mostrados na figura 2. O mesmo ocorreria caso fosse necessário alterar algum índice, nome, equação e afins. O mesmo raciocínio valeria se houver alteração do número de canais de saída, pois uma mudança na posição ou quantidade de caixas de som implicaria em uma refatoração completa da implementação.

2.2 Buscando soluções

A implementação e testes de tal sistema nos levou à busca de uma escolha ferramental que permitisse a flexibilidade que o sistema exige. Entre as exigências de tal ferramenta estavam: permitir vários canais de entrada, permitir vários canais de saída, permitir alterar a posição de cada objeto durante a execução do algoritmo, permitir flexibilidade na codificação de cada função. Tais exigências nos levaram a buscar outras formas de implementar o sistema.

Como a experimentação do protótipo para a validação do sistema seria feita a partir de diferentes configurações com variações de parâmetros de entrada e saída, posição das fontes (ou objetos) sonoras e das caixas, adotamos algumas possibilidade de automatizar as configurações dos *patches*.

A primeira possibilidade de automatização da configuração foi feita por meio de uma ferramenta externa. O formato de arquivo do Pd é bastante simples e padronizado e pode ser visto como um arquivo texto. A figura 3 mostra um trecho de código que pode ser visto quando um arquivo .pd é aberto num editor de texto.

```
#N canvas 25 87 1270 684 10;
#X floatatom 640 58 5 0 0 0 - - -;
#X msg 541 52 0;
#X obj 409 317 +;
#X msg 406 459 \; pd-player\$1 vis 0;
#X obj 290 108 inlet;
#X obj 326 326 outlet;
#X obj 299 264 float;
#X text 340 233 reset;
#X connect 0 0 10 0;
#X connect 1 0 7 1;
#X connect 3 0 4 1;
```

Figura 3: Exemplo de arquivo do Pd.

Para gerar este tipo de arquivo foi desenvolvida uma ferramenta na linguagem Java que, a partir dos parâmetros, gerava os *patches* já configurados. Apesar de tal solução otimizar a criação do *patch* de *WFS*, ela demandava a incorporação de uma ferramenta externa no processo de trabalho, o que nem sempre é desejável.

Outra possibilidade de automatização estudada foi estender o Pd por meio de um *external* criado em linguagem C. O novo *external* poderia receber parâmetros

em sua criação, como a quantidade de fontes sonoras e de canais, e se adequar dinamicamente a esta configuração de maneira transparente ao usuário. Tal solução, apesar de trazer benefícios como o processamento em C de algumas funcionalidades, mostra-se pouco portável, pois faria que tal *external* fosse compilado para cada sistema operacional onde o Pd fosse executado. Novamente, isto incluiria ao processo novas ferramentas, como o compilador C, o que não é desejável. O protótipo inicial nos mostrou que os objetos necessários para a implementação já existiam e que construir um *external* não seria imprescindível.

Como o *Pure Data* permite sua extensão também por meio de abstrações, tal solução foi considerada mais adequada que as anteriores pelas seguintes razões:

- não há necessidade de adicionar novas ferramentas ao processo de desenvolvimento;
- o resultado obtido é totalmente compatível com o ambiente Pd e independente de onde ele foi compilado;
- a modificação da abstração pode ser feita durante a sua execução simplificando os experimentos e testes da ferramenta.

A seguir é apresentada esta solução.

2.3 Solução adotada

A partir das soluções encontradas, apresentadas na seção anterior, a solução adotada baseia-se em num conjunto de abstrações para o Pd.

A implementação permitiu que o sistema fosse visualizado de maneira que cada funcionalidade fosse implementada em blocos de funções interconectáveis pelas seguintes abstrações:

Módulos de entradas (L1) e saídas (L4): recebem as informações de posição dos objetos sonoros, seus respectivos sinais de áudio e as posições das caixas de som usadas pelo sistema. As saídas são os sinais de áudio já processados pelo sistema.

Módulos de cálculo para *WFS* (L2): recebem as configurações de entrada e calculam fatores de amplitude e atrasos que serão aplicados aos sinais de áudio.

Módulos de processamento de áudio (L3): aplicam, nos sinais de áudio, os atrasos e as relações de amplitude correspondentes à técnica de *WFS*.

A divisão das funções do programa em módulos se aproxima da proposta de arquitetura sugerida pelo sistema AUDIENCE [7]. Os módulos descritos podem ser associados às camadas específicas deste sistema.

Uma vez feita a divisão de responsabilidades do sistema, o próximo passo foi definir como seriam feitas as conexões entre os módulos. A solução encontrada para permitir flexibilidade para as conexões de áudio e controle foi fazê-las internamente no *Pure Data*, utilizando objetos *send* e *receive*, conforme ilustrado na figura 4. Com isto, foi possível que tais conexões funcionassem de maneira transparente para o usuário que, por sua vez, não precisaria criar cada conexão individualmente.

Desta maneira, o módulo de entrada L1 capta os sinais de áudio e posição das caixas e envia estes valores por meio de objetos *send*, por exemplo [send~ audio1] (ou [s~ audio1]). Qualquer outro objeto no *patch* que precisar receber tal informação pode acessá-la por meio de um objeto *receive*, por exemplo [receive~ audio1] (ou [r~ audio1]). Com esta solução não há mais a necessidade de conectar estes objetos explicitamente. Tal abordagem de implementação exigiu a definição de nomes para envio e recebimento de mensagens e a formalização na comunicação entre os módulos. Além disso, isto permite que os módulos sejam trocados e que exista mais de uma implementação para cada um deles.

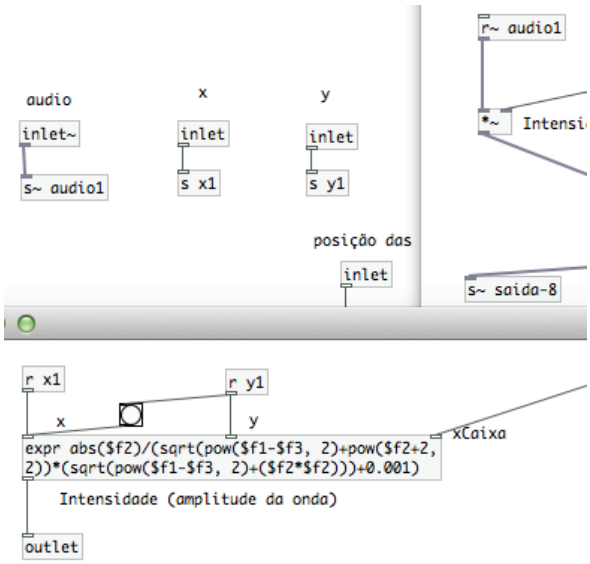

Figura 4: Conexões em feitas no *Pure Data* com o uso de *send* e *receive*.

A segunda parte da solução está na criação destes módulos como abstrações que utilizam programação dinâmica para a configuração do sistema. Através da passagem de parâmetros é possível informar a quantidade de canais de entrada e saída, ou conforme ilustrado na figura 5, a distância entre as caixas de som em sua criação. A técnica de programação dinâmica é bastante similar à técnica computacional chamada reflexão [8], por tratar do conhecimento que o código possui sobre si mesmo. Com isto, a abstração em si não traz apenas sua implementação, porém recria o *subpatch* toda vez em que é instanciado, baseando esta criação nos parâmetros informados. Em consequência disto, o número de canais pode ser alterado de forma simples e rápida.

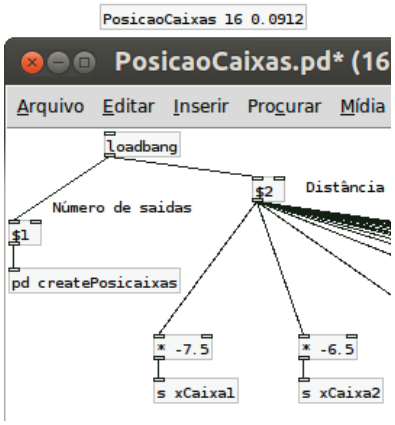

Figura 5: Exemplo de reflexão computacional. Quando foi criada a abstração *PosicaoCaixas* recebeu os parâmetros 16, correspondente ao número de canais de saída, e 0.0912, correspondente à distância entre as caixas de som.

3 RESULTADOS

Até o momento, o protótipo desenvolvido realiza a *WFS* fazendo a reprodução da frente de onda através dos cálculos da amplitude e do atraso do sinal de áudio que chega a cada uma das caixas de som.

A figura 6 apresenta o *patch* principal desenvolvido em Pd para o sistema de *WFS*.

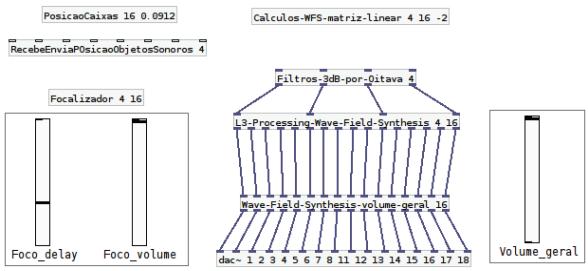

Figura 6: *Patch* principal desenvolvido em Pd para o sistema de *WFS*.

O desenvolvimento do protótipo proposto em *Pure Data* permite que o sinal de áudio enviado possa ser de um arquivo de áudio do computador, da entrada de som externa (via placa de som) ou um som sintetizado no próprio *patch* e que instrumentos eletrônicos, como os teclados controladores e os mais modernos computadores, também possam controlar os sons no espaço de audição em tempo real ou de forma programada.

A implementação do sistema como abstrações que utilizam programação dinâmica para se instanciarem garante uma simplicidade grande de manutenção do sistema, conforme mostra a figura 7. Tal solução também se mostrou eficaz devido à portabilidade e à facilidade de integração com outros sistemas ou outros *patches*. Como o sistema possui uma divisão clara de funcionalidades poderá ser modificado em partes, permitindo novas configurações e implementações em cada camada.

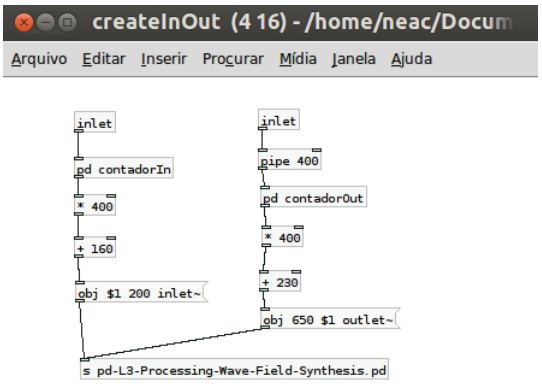

Figura 7: Exemplo de código de abstração funcional gerativa do sistema (criação dinâmica de objetos em patch).

4 CONCLUSÃO

Este artigo apresentou a solução encontrada para a sonorização de um sistema de espacialização com *WFS*. Apresentamos as possibilidades de implementação deste sistema e a solução adotada.

Esta solução traz facilidade de integração do módulo *WFS* com os diversos recursos do Pd, principalmente com outros *patches* e entradas e saídas de programas no universo do áudio e da música. O que facilita o uso desta técnica de sonorização em trabalhos de composição, gravações, apresentações, colaborações musicais interativas e aplicações que preconizem efeitos de espacialização sonora, alargando as possibilidades de auralização com instrumentações musicais.

A solução encontrada se mostrou bastante flexível e versátil pois cumpriu o objetivo de ser satisfatória no intuito de permitir a escolha da geometria final do sistema de alto-falantes.

Outra vantagem desta solução é a diminuição do esforço necessário para adequação e configuração do ambiente devido ao uso da técnica de reflexão. Tal solução vai além do escopo deste trabalho e pode ser utilizada para implementar outros tipos de processamentos sonoros no ambiente *Pure Data*.

A facilidade de uso determinada pela arquitetura modular e intercambiável possibilita conectar o algoritmo de *WFS* a algoritmos de simulação acústica, podendo vir a se tornar uma alternativa aos aplicativos que executam esta função.

Por fim, a solução aqui proposta é aberta, permitindo que outros pesquisadores utilizem-na em seus estudos. Após alguns acertos, este código será disponbilizado posteriormente junto à distribuição do OpenAUDIENCE no site do NEAC em `www.lsi.usp.br/audience`.

4.1 Trabalhos futuros

O *patch* desenvolvido não esta totalmente finalizado, e seu aperfeiçoamento resultará em novo trabalho

acadêmico. Pretendemos explorar outros algoritmos de espacialização e trabalhar tanto a documentação do sistema quanto seus testes e validações.

Para a avaliação e validação dos resultados do projeto, os módulos do sistema *WFS* serão testados em laboratório com matrizes de alto-falantes. Em uma segunda etapa procederemos à avaliação subjetiva por meio de voluntários e questionários sobre a percepção da localização de fontes sonoras em cenas sonoras de teste auralizadas em espaços auditivos controlados.

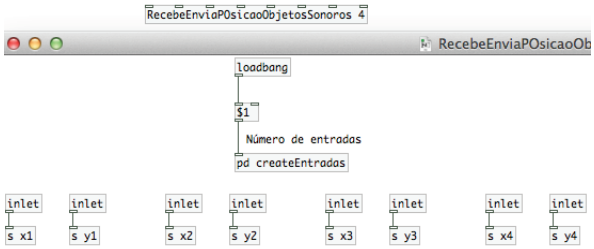

Figura 8: Qualquer dispositivo que envie as coordenadas *x* e *y* para as entradas da abstração *RecebeEnviaPOsicaoObjetosSonoros* altera a posição da fonte sonora no Pd.

Como o sistema é modular, a posição de cada objeto sonoro poderá ser informada por dispositivos externos (como sensores) ou mesmo por outros programas, podendo substituir ou trabalhar conjuntamente com qualquer interface de controle projetada para o protótipo desenvolvido, conforme ilustrado na figura 8. Isto permite que, como exemplo de aplicação, sejam colocados sensores adaptados a instrumentos musicais que variam suas coordenadas conforme sua localização num dado referencial.

5 AGRADECIMENTOS

Agradecemos ao Grupo de Pesquisa em Computação Musical do IME-USP, em especial ao pesquisador Thilo Koch, ao Núcleo de Pesquisa em Sonologia (NUSOM) da ECA-USP e ao Núcleo de Engenharia de Áudio e Codificação Sonora (NEAC) da POLI-USP.

Agradecemos também ao apoio dado pela FAPESP, através do processo 2012/17263-1.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Winfried Ritsch, *Bang: Pure Data*, chapter Does Pure Data Dream of Electric Violins?, Wolke Verlagsges. Mbh, Graz, Austria, 2006.

[2] Diemer de Vries, "Wave field synthesis - aes monograph," *Audio Engineering Society Inc*, 2009.

[3] M. A. J. Baalman, *On Wave Field Synthesis and The Electro-Acoustic Music: State of The Art 2007*, International Computer Music Conference 2007, 2007.

[4] Edo M. Hulsebos, *Auralization using Wave Field Synthesis*, Ph.D. thesis, Delft University of Technology. Delft, Holanda, 2004.

[5] M. A. J. Baalman, *On Wave Field Synthesis and Electro-acoustic Music: With a Particular Focus on the Reproduction of Arbitrarily Shaped Sound Sources*, Ph.D. thesis, Technischen Universität Berlin, 2008.

[6] Miller Puckette et al., "Pure data: another integrated computer music environment," *Proceedings of the Second Intercollege Computer Music Concerts*, pp. 37–41, 1996.

[7] Regis Rossi A. Faria, "Audience for pd, a scene-oriented library for spatial audio," *Proceedings of Pure Data Convention*, 2011.

[8] Iohannes Zmölnig, "Reflection in Pure Data," *Proceedings of the Linux Audio Conference*, 2009.