Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

SaWerkraut: Sintetizador VST implementado em Csound com o uso do framework Cabbage.

Herman Ferreira Militão de Asevedo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.ª Dr.ª Carla Denise Castanho

Coorientador
Prof. Dr. Carlos Eduardo Vianna de Mello

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Rodrigo Bonifacio de Almeida

Banca examinadora composta por:

Prof.ª Dr.ª Carla Denise Castanho (Orientadora) — CIC/UnB
Prof. Dr. Carlos Eduardo Vianna de Mello — MUS/UnB
Prof. Dr. Pedro de Azevedo Berger — CIC/UnB

CIP — Catalogação Internacional na Publicação

Asevedo, Herman Ferreira Militão de.

SaWerkraut: Sintetizador VST implementado em Csound com o uso do framework Cabbage. / Herman Ferreira Militão de Asevedo. Brasília : UnB, 2016.
107 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. Sintetizador, 2. Csound, 3. Cabbage, 4. VST, 5. Digital Audio Workstation.

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

SaWerkraut: Sintetizador VST implementado em Csound com o uso do framework Cabbage.

Herman Ferreira Militão de Asevedo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Carla Denise Castanho (Orientadora)
CIC/UnB

Prof. Dr. Carlos Eduardo Vianna de Mello
MUS/UnB

Prof. Dr. Pedro de Azevedo Berger
CIC/UnB

Prof. Dr. Rodrigo Bonifacio de Almeida
Coordenador do Bacharelado em Ciência da Computação

Brasília, 5 de agosto de 2016

Agradecimentos

Primeiramente gostaria de agradecer à minha orientadora Carla Denise Castanho e à coorientação do professor Carlos Eduardo Vianna de Mello, que demonstraram apoio e deram conselhos sobre o percurso do projeto, além de anteriormente dar a motivação inicial para os primeiros passos nas áreas de desenvolvimento e sonorização de jogos digitais. Também gostaria de agradecer os professores Márcio da Costa Pereira Brandão e Aluízio Arcela Junior, que tiveram grande importância na minha graduação, apresentando de forma detalhada a área de computação sonora e me elucidando em grande parte de seus conceitos. Agradeço também o professor Pedro de Azevedo Berger por ter aceito o convite para participar da banca de avaliação deste trabalho.

Agradeço fortemente a minha família e amigos por terem me auxiliado durante toda a minha vida, principalmente nesse percurso de graduação. Agradeço especialmente a minha mãe Dorcas e a minha irmã Karen pelo apoio constante e sem fim nesse e em demais períodos da minha vida, e pela paciência e compreensão nos demais momentos que estive presente.

Agradeço também a todos os integrantes que fazem parte da história da empresa Fira Soft e continuam, junto comigo, escrevendo ela. Agradeço principalmente pelo motivo de que ela, desde a sua concepção até os dias de hoje, está presente constantemente em meu cotidiano, com muitos momentos de alegria e trabalho pesado, e me dando oportunidades para demonstrar em nossas criações as minhas composições musicais.

Agradeço aos demais colegas de curso e de universidade pela presença nessa jornada de graduação, por todos os momentos de estudo compartilhados, pelas aulas em conjunto e pelos momentos de lazer proporcionados.

Resumo

Na produção sonora, o uso de instrumentos musicais virtuais fazem parte do cotidiano de compositores e *sound designers*. Esses instrumentos são executados na maioria dos casos em um dos estúdios de áudio virtuais existentes, softwares denominados *DAWs* (*Digital Audio Workstation*), que tem o papel de ser uma ferramenta completa na área de criação sonora. Com um conhecimento básico de programação e sinal digital, é possível utilizar linguagens de programação e sistemas voltados a áudio, com o objetivo de se desenvolver novos instrumentos virtuais ou efeitos sonoros. O *Csound* faz parte dessa categoria de sistemas, e recentemente com o aparecimento do *framework Cabbage* para ele, fez-se possível a utilização dos instrumentos desenvolvidos no *Csound* em *DAWs*, com interface gráfica personalizável.

É nesse contexto que esta monografia apresenta o instrumento musical virtual *SaWerkraut*, um sintetizador com uma arquitetura envolvendo os tipos de síntese aditiva, subtrativa e *FM*. Foi desenvolvido na linguagem *Csound* com o uso do *framework Cabbage*, para ser executado em *DAWs* como um instrumento virtual. O projeto tem como um dos focos as características e possibilidades sonoras do instrumento, e a compatibilidade do conjunto *CSound/Cabbage* no ambiente do *DAW*. Uma demonstração é realizada em uma composição utilizando diversas instâncias do *SaWerkraut* no *DAW FL Studio*.

Palavras-chave: Sintetizador, Csound, Cabbage, VST, Digital Audio Workstation.

Abstract

In sound production, the use of virtual musical instruments are part of the daily lives of composers and sound designers. These instruments are performed in most cases on one of the existent virtual audio studios, software called DAWs (Digital Audio Workstation), which have the role of being a complete tool in the area of sound creation. With a basic knowledge of programming and digital signal, it is possible to use audio dedicated programming languages and systems in order to develop new virtual instruments or sound effects. Csound is a part of that category of systems, and recently with the appearance of the Cabbage framework, the use of instruments developed in Csound with customizable GUI on DAWs became possible.

In this context, this monograph presents the *SaWerkraut* virtual musical instrument, a synthesizer with an architecture involving additive, subtractive and FM synthesis types. It was developed in the Csound language using the Cabbage framework, to run in DAWs as a virtual instrument. The project has one of its focuses the features and sonic possibilities of the instrument, and the compatibility of the CSound/Cabbage pair on a DAW environment. A demonstration is provided through a musical composition using multiples instances of *SaWerkraut* in the DAW FL Studio.

Keywords: Synthesizer, Csound, Cabbage, VST, Digital Audio Workstation.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Na produção musical e sonora no cenário atual, é constante o uso de instrumentos musicais virtuais, que são geradores de sons a partir do computador [37]. Os instrumentos virtuais em questão tem um vasto histórico de evolução, partindo de sintetizadores analógicos desde os anos 1920 até os dias de hoje, envolvendo o uso de novas técnicas de síntese, amostragem de instrumentos reais, novas arquiteturas de fluxo de sinal, *hardware* mais poderoso, entre outros elementos [21]. Com o aparecimento da necessidade do uso de tais sintetizadores no meio digital, eles então migraram ou foram adaptados para esse meio. Um dos exemplos dessa relação de migração/adaptação é o caso do clássico teclado/sintetizador *Yamaha DX7* [9] e o instrumento virtual *FM7* da *Native Instruments* [8].

Os instrumentos virtuais normalmente são dispostos para serem utilizados em conjunto com um *Digital Audio Workstation* (*DAW*). Em resumo, um *DAW* é um estúdio virtual voltado ao áudio, com a presença de instrumentos musicais virtuais, trilhas de áudio, banco de amostras, mesa de mixagem e diversas outras funcionalidades [20]. Seu uso consta na presença de sequenciamento de notas e eventos, a serem tocadas pelos instrumentos virtuais.

Em geral, os instrumentos são carregados nesses estúdios virtuais como bibliotecas dinâmicas dos sistemas operacionais utilizados, a partir do uso de um interfaceamento entre o algoritmo que gera/processa o áudio do instrumento virtual e o software de composição *DAW* que se comunica com ele. Existem alguns tipos de interfaceamentos. Um dos exemplos desse interfaceamento é a arquitetura *VST* desenvolvida pela *Steinberg* [2, 16]. *VST* trabalha com a presença de diversas funcionalidades que facilitam e padronizam esse processo. O algoritmo deve ser desenvolvido de forma a se comunicar com o *DAW* utilizando a interface apropriadamente. Nesse contexto, linguagens, sistemas ou *softwares* voltados à geração de sinal, como o *Csound* [7] por exemplo, podem ser utilizadas junto a tal ferramenta.

Csound é uma linguagem voltada para geração e processamento de áudio [27]. Devido ao grande número de funcionalidades e comandos presentes na linguagem que foram provenientes de anos de uso e evolução dela, a linguagem *Csound* se demonstra poderosa e robusta em se tratando de geração de sinal de áudio digital. O *Csound* compila e executa o instrumento a partir de eventos e notas descritos no seu código ou via mensagens *MIDI*, não se relacionando diretamente com algum *DAW*. Com o aparecimento do *CsoundVST* [10] e, em seguida, da ferramenta *Cabbage* [29], todo o ferramental que a

linguagem *Csound* possui passou a ser utilizável de forma amigável como um instrumento virtual em *DAWs* a partir de interfaceamento.

O *framework Cabbage* é uma ferramenta desenvolvida recentemente, voltada a geração de *plugins* de áudio com interface gráfica personalizável, no uso da linguagem *Csound* [31]. Seu papel fundamental proposto é realizar o interfaceamento entre o instrumento virtual desenvolvido na linguagem *Csound* e o *DAW*. Os *plugins* gerados são exportados em demais formatos reconhecidos por *DAWs*, sendo um desses formatos, por exemplo, o *VST*. Existem diversos elementos de interface gráfica que podem ser utilizados no controle do instrumento virtual desenvolvido, incluindo por exemplo *sliders* de diferentes tipos, botões, tabelas editáveis, *comboboxes*, entre outros. Anteriormente, o *CsoundVST* fornecia uma interface semelhante, porém sem a presença de edição dos elementos gráficos [10].

1.1 Problema

A linguagem *Csound* fornece uma estrutura suficiente para que sintetizadores de formatos e arquiteturas diversas possam ser desenvolvidos. Porém, estes sintetizadores estão limitados a serem utilizados dentro do escopo do *Csound* apenas, ou com a possibilidade de serem utilizados sem uma interface gráfica própria em *DAWs*, pelo uso do *CSoundVST*.

1.2 Hipótese

O *framework Cabbage* oferece a possibilidade da construção de uma interface gráfica para o sintetizador desenvolvido em *Csound* e o interfaceamento com *DAWs*. Além de diversos elementos de interface gráfica, o *Cabbage* também torna possível a exportação do instrumento em formatos de interfaces diferentes, de acordo com os sistemas operacionais. A ferramenta juntamente com o *Csound* executa o sintetizador com estabilidade e responsividade em relação a eventos provenientes do *DAW*.

1.3 Objetivo

Partindo desse pressuposto, o objetivo desse trabalho é o desenvolvimento de um sintetizador, denominado *SaWerkraut*, com características específicas, a partir da linguagem *Csound* e a ferramenta de interface *Cabbage*. O sintetizador trabalha com técnicas de síntese dos tipos aditiva, subtrativa e frequência modulada (FM), com o uso de blocos modulares de síntese que se comunicam. Tal sintetizador será compilado em um *plugin VST* via *Cabbage*, sendo utilizável em *softwares* do tipo *DAW*, voltados especificamente para a produção de áudio. A ferramenta *Cabbage* em conjunto com o *Csound* será utilizada para a implementação de toda parte relativa ao processamento de sinal e o interfaceamento do *software*, com exportação dele em um *plugin VST* ao final.

1.4 Metodologia

A metodologia segue um processo baseado em etapas. Inicialmente, foi feito um levantamento bibliográfico dos fundamentos e tecnologias que estão presentes e se relacionam

com o projeto. Foi feito um estudo dos exemplos de implementações em *Csound* e *Cabbage* fornecidos a fim de auxiliar a etapa seguinte, que consta na definição da arquitetura do sintetizador proposto e da sua implementação. Testes posteriores a fim de validar as características da ferramenta foram realizados, incluindo a composição de uma música de demonstração da ferramenta no uso em conjunto com o *DAW FL Studio*.

1.5 Organização do documento

A monografia está organizada da seguinte forma. Em sequência ainda neste capítulo, uma breve descrição dos exemplos e referências para o desenvolvimento são descritas. Continuando, o Capítulo 2 traz diversos conceitos estudados e aplicados nesse trabalho. Tais conceitos envolvem uma base de sinal digital, os tipos de sínteses que estão presentes no projeto, o protocolo *MIDI*, *Digital Audio Workstation* (*DAW*), A *SDK VST* da *Steinberg*, a linguagem *Csound*, a ferramenta *Cabbage* e os trabalhos correlatos. Já em seguida o Capítulo 3 traz a descrição e implementação do sintetizador, descrevendo o fluxo de sinal e de controle, além de figuras ilustrativas da interface gráfica. Ao final desse capítulo são descritos testes com o projeto desenvolvido e um caso de uso, que envolve a composição de uma música com o uso do sintetizador no *DAW FL Studio*. Por fim, o Capítulo 4 contém os resultados do trabalho, as conclusões observadas e propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Esse capítulo traz alguns conceitos em relação ao som no meio digital. Tais conceitos envolvem sinal analógico, sinal digital, profundidade de bits, taxa de amostragem, *SNR* (*Signal Noise Ratio*), Teorema de *Nyquist* e *FFT* (*Fast Fourier Transform*).

2.1 Sinal digital

O som é uma vibração de um meio físico, envolvendo compressão e dispersão das moléculas desse meio, que é percebida pelo ser humano a partir do canal auditivo e interpretada pelo cérebro.

No meio analógico, a representação de um som é feita a partir de um sinal gerado variando a voltagem no decorrer do tempo de forma periódica, gerando variações de *amplitude* $\times$ *tempo* de forma contínua, entre regiões de maior e menor voltagem em relação a um valor central, refletindo respectivamente o que seriam momentos de compressão e dispersam do meio de propagação. A geração desse sinal é feita pelo processo de transdução. Uma representação gráfica demonstra esses valores com picos e vales, de acordo com a onda na Figura 2.1. Nesse exemplo, a onda descreve um tom puro (senóide) que possui sua frequência incrementada continuamente no decorrer do tempo.

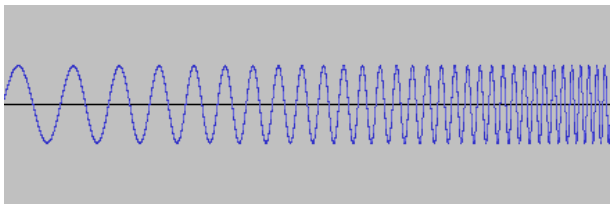

Figura 2.1: Representação digital de uma onda.

É importante notar que essa demonstração na Figura 2.1 se refere a uma onda que na realidade é digital e não analógica. A principal diferença entre o contexto analógico e digital está na discretização do segundo em relação a continuidade do primeiro. Computadores somente trabalham internamente com material digital, o que no caso do uso de hardwares externos que trabalham de forma analógica, requerem conversão entre os dois meios para haver comunicação, analógico para digital e digital para analógico.

No caso de representações de ondas sonoras no meio digital, ela é feita de forma a se obter uma onda resultante o mais próximo possível do sinal original analógico, sem as perdas de representação devido o processo de conversão. No caso de uma geração ou síntese no meio digital, ela deve ser o mais próximo possível do formato do sinal que se deseja alcançar. Um dos métodos de representação mais comum é o *PCM* (*Pulse-code modulation*), em que pontos de amplitude distribuídos uniformemente no decorrer do tempo (domínio do tempo) são amostrados de forma a representar fielmente a onda que foi convertida a partir do meio analógico, ou que foi gerada no meio digital. Dois fatores são importantes na amostragem de qualquer onda em relação a fidelidade do resultado, no uso desse método, sendo elas:

- **Profundidade de Bits (*Bit Depth*)**: Número de bits de informação de cada amostra, determinando a quantidade de valores distintos que podem ser atribuídos a um ponto de amplitude. Cada ponto na onda discreta é um número que representa valores positivos ou negativos no decorrer do tempo. Com um *bit depth* de 16 bits temos 2^{16} possíveis valores distintos que podem ser atribuídos entre 32767 e -32768, por exemplo. A Figura 2.2 representa uma onda com 4 bits de profundidade, a qual é representada com 16 possíveis valores de amplitude.

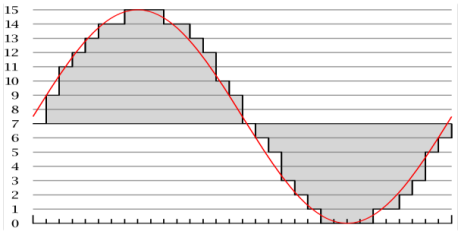

Figura 2.2: Onda senoidal amostrada com uma profundidade de bits igual a 4, possuindo então 16 possíveis valores de amplitude.

- **Taxa de Amostragem (*Sample Rate*)**: Taxa que indica a quantidade de amostras de sinal em um determinado período de tempo. Devido a essa uniformidade, a taxa é constante durante o processo de amostragem, em que com uma taxa de 44100 amostras por segundo, obtém se um resultado sonoro que consegue representar as faixas audíveis pelo ser humano com boa precisão, devido o teorema de amostragem, citado mais a frente. A Figura 2.3 demonstra um processo de amostragem de uma onda, a qual o resultado final a partir da onda senoidal ao fundo é demonstrado visualmente pela onda quadriculada, discretizado pelos pontos pretos (amostras) no gráfico.

Ainda no meio analógico, existe uma medida de fidelidade do sinal denominada SNR (*signal to noise ratio*), e pode ser calculada pela razão:

$$SNR = \frac{\text{amplitude média do sinal}}{\text{amplitude média do ruido}}$$

A razão ocorre entre o sinal desejado e o ruído gerado devido as configurações da amostragem/sistema. O cálculo para as duas amplitudes (sinal e ruído) é feito em três

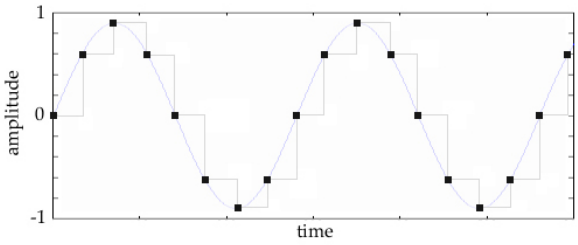

Figura 2.3: Onda senoidal amostrada com 8 amostras por período.

passos, sendo eles o quadrado das amplitudes calculadas, a média desses valores e a raiz quadrada do resultado. Quanto maior o *sample rate* e o *bit depth*, maior é esse valor SNR após o processo de conversão do sinal analógico para o digital, sendo maior também o esforço computacional de processamento do sinal e armazenamento dos dados com maiores resoluções.

Os valores de 16 bits de profundidade de bits e $44100Hz$ da taxa de amostragem são valores padrões, comumente utilizados. Esse valores são utilizados em diversos sistemas de áudio digitais, por representarem com fidelidade considerável um sinal analógico para o ouvido humano, sem grande custo computacional.

O motivo da frequência de amostragem ser padronizada para demais sistemas com o valor de $44100Hz$ amostras por segundo se dá por dois fatores, o teorema de amostragem de Nyquist e o alcance de frequências audíveis para o ser humano ser cerca de $20000Hz$.

O teorema de amostragem de Nyquist define que para se amostrar um sinal que possui conteúdo de frequência máxima em um valor X, a taxa de amostragem tem que ser de pelo menos $2X$ para representar essa frequência corretamente [35]. Como o alcance do ouvido humano é de cerca de $20000Hz$, com uma taxa de $44100Hz$, a amostragem é suficiente para abranger com precisão essas frequências abaixo da metade desse valor ($22050Hz$).

O que ocorre quando a taxa de amostragem é insuficiente está exemplificado na Figura 2.4, em que com uma frequência de onda de $\frac{3SR}{4}$, com SR sendo a taxa de amostragem do sistema, o resultado é exatamente uma onda de frequência igual a $\frac{SR}{4}$. O que ocorre é que as ondas são representadas corretamente quando a frequência varia de 0 a $\frac{SR}{2}$ e são representadas erroneamente quando a frequência varia de $\frac{SR}{2}$ a SR, sendo representadas nesse segundo caso como ondas de frequência que variam de $\frac{SR}{2}$ a 0, resultado do cálculo $SR - \frac{3SR}{4}$.

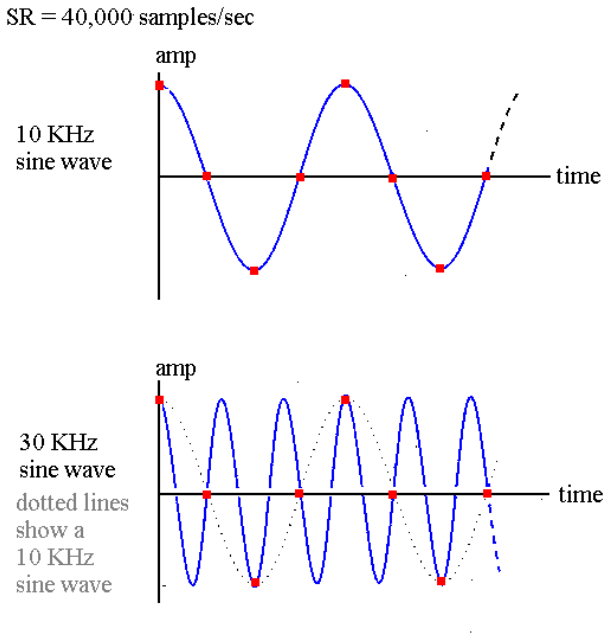

Figura 2.4: Duas ondas amostradas com frequências $\frac{SR}{4}$ acima e $\frac{3SR}{4}$ abaixo, em que SR é a taxa de amostragem.

No domínio do tempo, a onda é representada pela variação da amplitude das amostras no decorrer do tempo. No domínio da frequência, a onda agora é representada a partir da aplicação da série de *Fourier*, uma soma de termos senoidais infinitos da forma:

$$\begin{aligned} f(t) = & A_0 + \\ & A_1 cos(2\pi tf) + B_1 sin(2\pi tf) + \\ & A_2 cos(2(2\pi tf)) + B_2 sin(2(2\pi tf)) + \\ & A_3 cos(3(2\pi tf)) + B_3 sin(3(2\pi tf)) + \\ & A_4 cos(4(2\pi tf)) + B_4 sin(4(2\pi tf)) + \\ & \ldots \end{aligned}$$

O teorema de *Fourier* nos diz que qualquer função periódica pode ser descrita pela soma de senóides, com a possibilidade de infinitos termos dependendo da forma da onda [34]. Os termos, denominados parciais, são múltiplos inteiros da frequência fundamental f, sendo que os valores de A_n e B_n, com $n \geq 1$, são os coeficientes de *Fourier* que definem as intensidades de cada uma das parciais para a onda resultante, definindo a sua forma a partir da soma desses termos.

A análise de *Fourier* em meios digitais é feita para a geração do espectro da onda, a partir do sinal de áudio. Em tempo real, ela é feita por um algoritmo denominado *FFT* (*Fast Fourier Transform*). O algoritmo *FFT* faz a transformação discreta de *Fourier* de forma otimizada e rápida, desconstruindo um bloco de amostras no domínio do tempo em um espectro no domínio da frequência. O processo inverso, denominado *IFFT* (Inverse FFT) passa do domínio da frequência para o domínio do tempo, construindo (sintetizando) o sinal a partir da análise que foi feita previamente, ou a partir de valores arbitrários que

podem ser definidos para esses coeficientes. A Figura 2.5 demonstra simplificadamente as diferenças entre os números de parciais e frequências do espectro de algumas ondas.

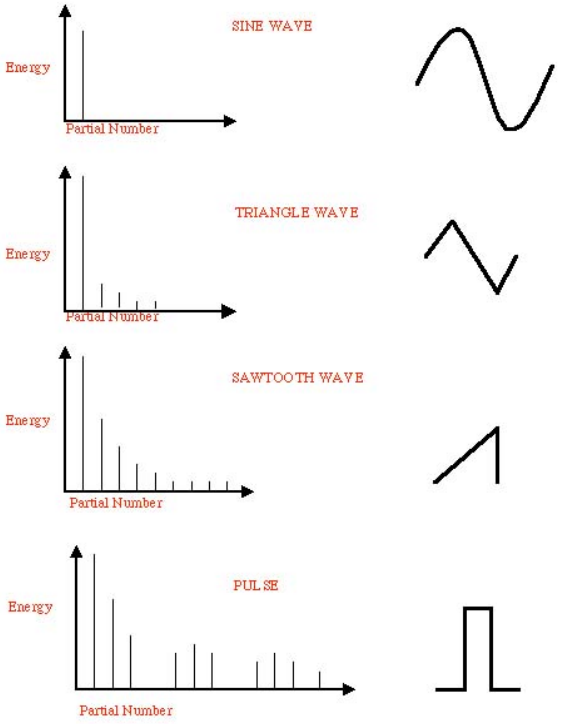

Figura 2.5: Exemplo de ondas distintas e seus espectros.

A Figura 2.6 descreve em relação aos cinco gráficos de ondas os três passos que são feitos sobre um sinal qualquer $f(t)$, não necessariamente periódico, a fim de se obter os coeficientes de *Fourier*. Como a transformação de *Fourier* ocorre apenas sobre sinais periódicos, o procedimento deve ser feito para que um pequeno bloco do sinal seja periódico. Os passos desse procedimento são:

- ***Janelamento***: Uma função de janelamento $j(t)$ é definida para ser utilizada no sinal, a fim de se selecionar apenas uma porção do sinal a se aplicar à análise. Uma das formas da função e aplicação é demonstrada na Figura 2.6, com uma forma de onda retangular como janela, demonstrando nos quatro primeiros gráficos tal processo. O sinal após a aplicação do janelamento ($f(t) \times j(t)$) compreende agora apenas a uma porção do sinal inicial.
- ***Transformação de Fourier***: A transformação é feita agora sobre uma onda periódica, que tem como período essa porção do sinal, resultando nos coeficientes de *Fourier*. O procedimento continua para novos blocos do sinal em sequência.

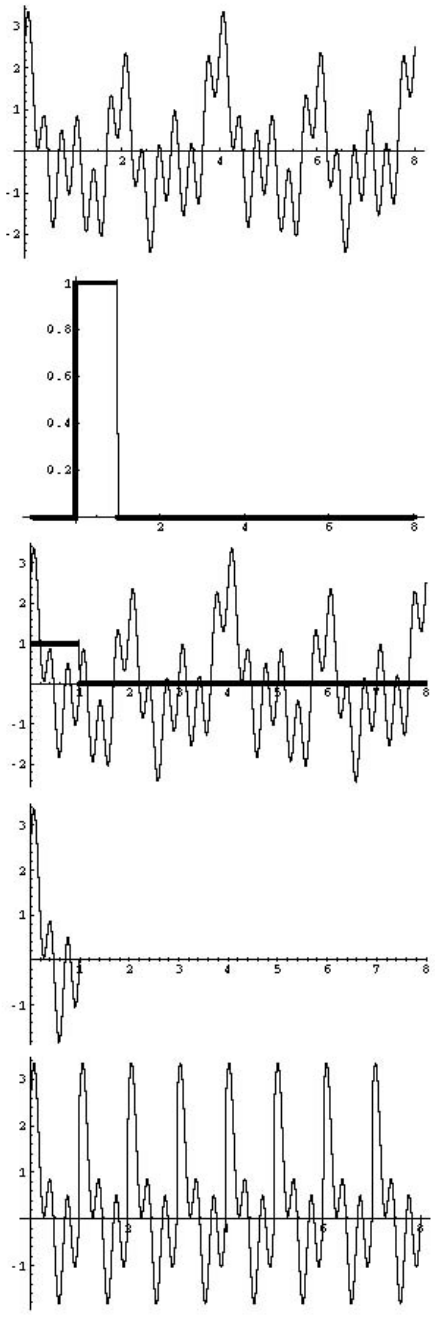

Figura 2.6: Sequência de passos executada por uma *FFT* para o computo dos coeficientes de *Fourier* [3].

O gráfico para a representação de uma onda nesse domínio, exemplificada na Figura 2.5, é construído a partir da frequência × amplitude das parciais, descrevendo um *frame* relativo a um período curto de tempo da análise. As frequências são discretizadas e subdivididas em bandas (*bins*), com espaçamento fixo e constante igual a largura de banda. As parciais de uma onda são representadas nas intensidades desses *bins* nas frequências exatas ou aproximadas.

A largura de banda é dependente de dois fatores, a taxa de amostragem e o tamanho do *frame*. O cálculo desse valor é feito pela formula:

$$\text{largura do } bin = \text{taxa de amostragem/tamanho do frame}$$

Valores comuns para esses dois elementos são respectivamente 44100 hertz e 1024 amostras, gerando uma largura de cerca de 43 hertz. Devido a distribuição desses *bins* ser de forma linear, as diferenças entre as frequências mais graves são representadas com baixa precisão enquanto que as mais agudas com alta precisão. A diferença de uma onda de frequência 43 para 86 é exatamente o dobro (uma oitava) enquanto que, por exemplo de 10000 para 10043, a diferença é perceptivelmente menor (menor que 1 semitom). De um ponto de vista sonoro, as frequências mais graves perdem sua distinção, ocupando os mesmos *bins* próximos que representam outras frequências.

A distribuição de frequências do espectro (número de *bins*) é calculada a partir do tamanho do frame/2, sendo 512 o valor adquirido utilizando os mesmos dados do cálculo anterior.

Com o decorrer do tempo, uma sequência de *frames* é gerado, o que requer uma representação gráfica em três dimensões (frequência dos *bins*, amplitude e tempo). Devido os *bins* serem discretos e limitados em relação a quantidade, a representação gráfica pode ser feita em duas dimensões apenas, com o uso de elementos visuais de maior ou menor intensidade para representar essa amplitude de cada sinal.

Esse capítulo foi baseado em material didático fornecido pelas matérias cursadas na Universidade de Brasília e do livro online de *Burk* [3].

2.2 Tipos de Sínteses

Esse capítulo traz uma breve descrição de três tipos de sínteses comumente utilizadas em sintetizadores analógicos e digitais, sendo elas a síntese aditiva, a subtrativa e a síntese por frequência modulada (*FM*).

2.2.1 Síntese aditiva

A síntese aditiva se baseia na construção de um sinal complexo a partir da soma de sinais simples de diferentes frequências e diferentes amplitudes, sendo essas frequências normalmente harmônicas entre si e as amplitudes com a possibilidade de serem variáveis e independentes entre si, garantindo uma sonoridade mais dinâmica ao sinal final. Ela é semelhante ao conceito descrito sucintamente na Seção 2.1, *Inverse FFT*. Nesse caso ela é um tipo de síntese aditiva, porém lida com a análise de um sinal original e não na geração a partir de outros parâmetros. A síntese aditiva dinâmica, em que um número

considerável de parciais são geradas com variações dependentes ou independentes tanto na frequência quanto na amplitude, pode ser custosa para o computador.

Em geral a síntese aditiva trabalha com o uso de ondas senoidais para a construção das ondas complexas. No caso de uma síntese aditiva harmônica, essas ondas são múltiplas da frequência fundamental. A equação a seguir descreve uma síntese aditiva harmônica:

$$X(t) = \sum_{k=1}^{N} a_k \sin(2\pi(kf_0) + \phi_k)$$

Nessa equação, $X(t)$ representa a saída da síntese aditiva a partir da soma das N parciais. O termo a_k é a amplitude da parcial de índice k, que pode ser descrita por um valor fixo ou uma função variável ao longo do tempo ($a_k(t)$), adicionando dinamicidade à síntese. O termo kf_0 descreve um múltiplo inteiro da frequência fundamental f_0. Nesse caso, é possível trabalhar com amplitude e frequência dinâmicas em relação ao tempo no uso de funções para esses dois termos ($K_k(t)f_k(t)$). Ao final é adicionado a fase ϕ_k a parcial [25].

2.2.2 Síntese subtrativa

A síntese subtrativa tem uma grande importância e uso em geradores de sinal, desde sua concepção até os dias de hoje, sendo presente em conjunto com diversas outras técnicas de síntese em muitos sintetizadores. Seu conceito se baseia, como seu próprio nome indica, na subtração (filtragem) de informações de um sinal complexo, a fim de se gerar um novo sinal reduzido em relação as quantidade de frequências do espectro.

A síntese subtrativa se relaciona muito bem com diversos tipos de síntese, como a aditiva por exemplo, devido a possibilidade de frequências agudas, graves ou centrais de um espectro gerado serem amenizadas ou destacadas, alterando as características da onda de acordo com o propósito escolhido para o sinal final no uso de diferentes tipos de filtros. De uma certa forma esse tipo de síntese não trabalha com a geração do sinal, apenas com a operação sobre um sinal gerado a partir de um outro processo de síntese, sendo ela então dependente de uma fonte de sinal anterior ao seu processo.

Existem diversos tipos de filtros utilizáveis em síntese subtrativa, sendo 3 deles comumente utilizadas em diversos sintetizadores, operando com a presença dos parâmetros frequência de corte f_c e ressonância res. Os 3 tipos são:

- **Passa Baixa**(***Low Pass***): A filtragem tem por objetivo reduzir informações de frequências mais altas do sinal, no caso, sobre as frequências acima de f_c. O efeito sonoro é que a cada passo de se reduzir f_c continuadamente entre o maior valor (frequência de amostragem, por exemplo) a 0, o sinal se demonstra menos "brilhante", devido as frequências de valores mais altos do sinal, acima dessa frequência de corte, serem filtrados.

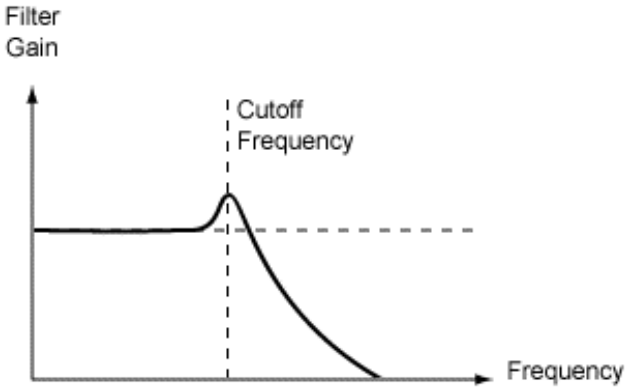

Figura 2.7: Exemplo de um filtro passa baixa no espectro.

- **Passa Alta**(***High Pass***): A filtragem nesse caso tem por objetivo reduzir informações de frequências mais baixas do sinal, no caso, abaixo de f_c. O efeito sonoro agora é que incrementando f_c de 0 a taxa de amostragem, o sinal perde informações graves, eliminando constantemente a característica grave do sinal tornando-o mais "brilhante"a cada passo, devido as parciais (e fundamental) do espectro de valores abaixo dessa frequência serem filtrados.

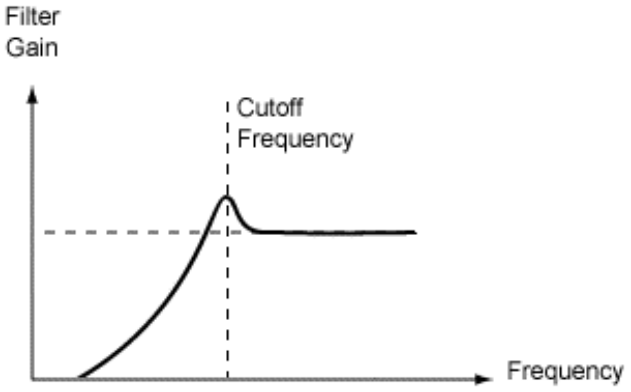

Figura 2.8: Exemplo de um filtro passa alta no espectro.

- **Passa Banda**(***Band Pass***): Finalmente o filtro passa banda realiza algo semelhante aos 2 filtros passa baixa e passa alta em conjunto sobre uma mesma f_c. O resultado visual no espectro é um banda em relação a essa frequência, enfatizando o material sonoro que se encontra interno a ela. Filtros desse tipo geralmente possuem também um terceiro parâmetro largura de banda, que define a área em que o filtro atua em relação ao centro em f_c.

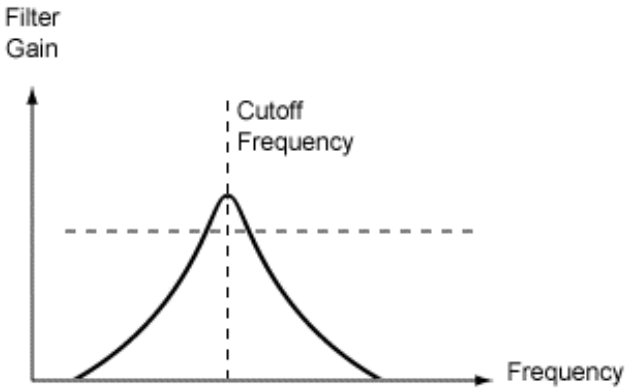

Figura 2.9: Exemplo de um filtro passa banda no espectro.

A região em relação as bandas que o filtro opera sobre são denominadas "para banda", devido a elas serem filtradas e suas bandas no espectro não estarem presentes na resultado pós-filtragem. A região que o filtro deixa passar as frequências são denominadas "passa-banda".

Um filtro perfeito em termos analógicos seria aquele em que dada uma frequência de corte, o sinal seria separado em exatamente duas partes: uma região que possui toda a informação não amenizada do sinal e uma segunda que possui nenhuma informação, pois toda ela foi filtrada completamente. Isso é implementado utilizando um parâmetro denominado qualidade do filtro, mas que ao passo que este valor aumenta, uma ressonância do sinal ocorre sobre a frequência f_c, o que faz esse parâmetros ser comumente denominado *res*. Esse efeito pode ser indesejável em filtragem de sinais no contexto não sonoro, porém ele é utilizado em sinais complexos musicais, devido ao resultado desse efeito de ressonância ser interessante no contexto de um instrumento virtual.

A síntese subtrativa ocorre principalmente com o uso de filtros dinâmicos, que variam a frequência de corte f_c e a ressonância *res* durante a execução de uma ou mais notas. Com o uso de por exemplo uma variação dinâmica sobre o parâmetro f_c durante a execução de uma nota, o sinal gerado varia em relação a quantidade de parciais presentes, se tornando mais ou menos “brilhantes” no decorrer do tempo, de acordo com o tipo de filtro e a forma dessa variação [24, p. 99-115].

2.2.3 Síntese por frequência modulada

Síntese por frequência modulada é um tipo de síntese que de certa forma possui uma implementação simples, mas que é uma poderosa forma de se gerar sons complexos, harmônicos e inarmônicos, a partir de operações entre ondas, mesmo que ondas simples, como senóides por exemplo. Esse procedimento requer a existência de dois sinais para sua execução que podem inclusive variar dinamicamente em tempo real sem muito custo. Com a descoberta de seu algoritmo por John Chowning em 1967, se tornou popular com o uso dele no sintetizador da *Yamaha DX7* em 1983, que continuou com a aprimoração de seu algoritmo nos modelos seguintes [9].

Basicamente a síntese por FM (frequência modulada) ocorre quando uma onda, denominada carregadora, possui sua frequência somada por uma outra onda, denominada

moduladora. Começando por um tom puro, uma onda senoidal de frequência f_c e amplitude A, temos que sua forma é expressa pela seguinte fórmula:

$$A\sin(2\pi f_c t)$$

Ao passo que a partir de uma nova onda de frequência f_m e forma $I\sin(2\pi f_m t)$, com I sendo o índice de modulação, especificamente a amplitude da onda moduladora, temos como resultado uma onda de forma:

$$A\sin(2\pi f_c t + I\sin(2\pi f_m t))$$

A Figura 2.10 exemplifica o resultado de uma onda modulada, descrita na parte superior dela, com a sua moduladora logo abaixo, com $f_c = 100$ e $f_m = 4$ e duração de 0.5 segundos.

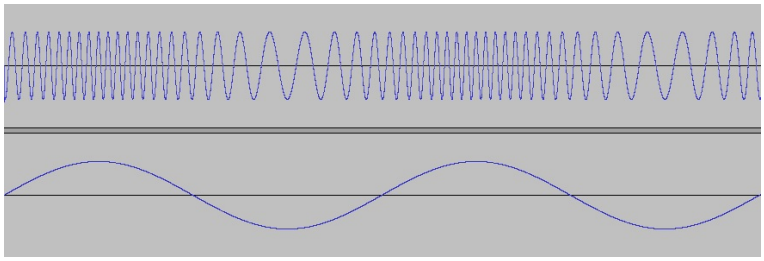

Figura 2.10: Onda modulada por frequência seguida de sua onda moduladora.

Note que um efeito de vibrato, em que a onda é percebida como um tom com rápidas variações na sua altura, é percebido caso f_m seja um valor abaixo da faixa audível, cerca de 20 *Hz*. Note também que o índice de modulação I garante um controle sobre a influência da onda moduladora, em que com $I = 0$ não ocorre modulação.

A onda resultante varia sua frequência instantânea, isto é, a frequência resultante entre o processo de modulação em instantes da função ao longo da variação do tempo t, entre os valores $f_c + I$ e $f_c - I$, já que a amplitude da segunda onda é o próprio índice de modulação. Repare que caso $I > f_c$, a frequência instantânea é negativa em alguns valores de t, o que leva o espectro a representá-la como uma onda de mesma frequência defasada em 180 graus.

O espectro da onda é, após o resultado sonoro, a parte mais interessante do processo de síntese por *FM*. Ele é composto da frequência da carregadora f_c e de um número de bandas laterais de frequências $f_c \pm Kf_m$, onde K representa valores inteiros positivos. A Figura 2.11 exemplifica um espectro nessa situação. Ocorrem situações em que a subtração de f_c por Kf_m geram frequências negativas, relativo a amplitude da moduladora. Estas por sua vez representam ondas com a fase invertida na mesma frequência das ondas positivas, em que a soma dessas bandas invertidas de mesma frequência pode gerar cancelamento do sinal naquela frequência.

A relação das frequências f_c e f_m tem um papel importante nesse processo, devido as bandas resultantes nas frequências fazerem parte ou não do conjunto de frequências harmônicas, caso a relação entre as duas frequências sejam números racionais, isto é:

$$f_c/f_m = N_1/N_2$$

Em que N_1 e N_2 são números inteiros. Com todos seus fatores em comum divididos, a frequência fundamental da onda resultante é dada por:

$$f_0 = f_c/N_1 = f_m/N_2$$

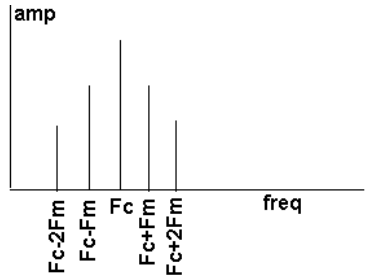

Figura 2.11: Exemplo de um espectro de uma onda modulada, com $K = 2$.

A quantidade de bandas laterais (Os valores de K) são dependentes e proporcionais ao valor do índice de modulação I. Quanto maior for este valor, mais bandas laterais são geradas. As amplitudes de todas as frequências, incluindo f_c, também variam de acordo com a variação desse índice. I controla então a distribuição de frequências e de energia do espectro, o que ao final representa uma variação no timbre em relação a complexidade.

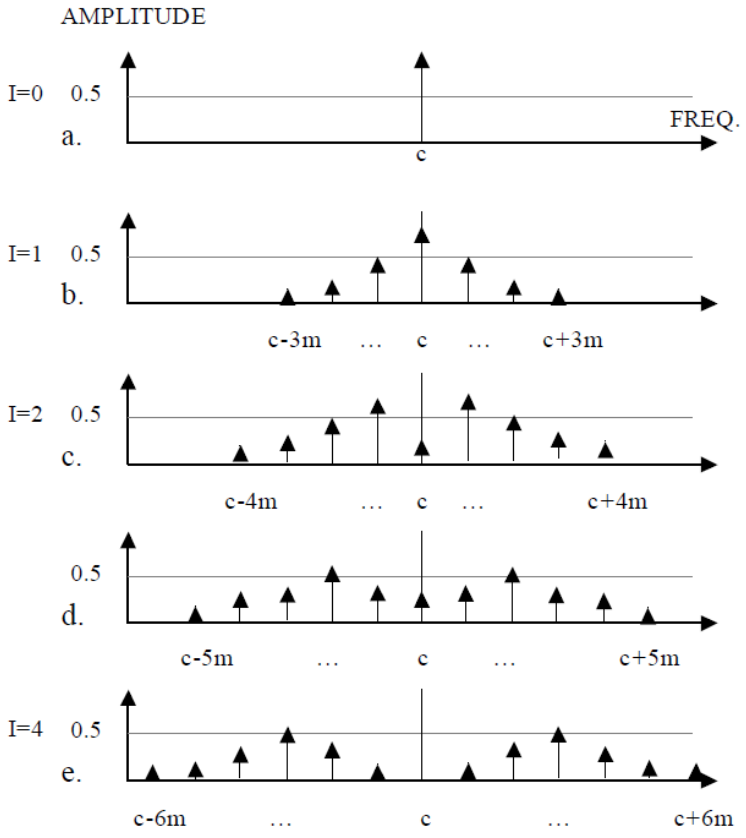

Figura 2.12: Exemplo de espectros de ondas moduladas com diferentes índices I.

O poder de tal síntese é considerável, em que apenas na alteração de um parâmetro, o espectro se demonstra cada vez mais rico e o som cada vez mais brilhante, como a Figura 2.12 demonstra. Vale relembrar que esse processo pode ocorrer de forma dinâmica.

Com o uso de múltiplas ondas moduladoras sobre uma mesma carregadora de forma paralela ou serial, temos a possibilidade de trabalhar com a geração de sinais cada vez mais complexos, timbres mais ricos e detalhados. Essa opção de modulação múltipla nesse tipo de síntese é essencial para a obtenção de resultados mais diferenciados.

Chowning em seu artigo sobre a descoberta e implementação da *FM*, além de todo o conteúdo demonstrado anteriormente, descreve com detalhes o processo matemático que explica a geração das bandas laterais, a forma das ondas resultantes a partir de séries de Fourier e funções de Bessel [6].

2.3 MIDI e Digital Audio Workstation

Estúdios digitais voltados a áudio são cada vez mais simples e fáceis de se montar hoje em dia, se comparado as dificuldades existentes antigamente. É fácil criar um estúdio digital avançado, com a presença de um completo arsenal de sintetizadores, banco de efeitos, mesa de som, amostradores, entre outros, todos estes podendo ser executados em paralelo, com a necessidade de apenas um computador para isso. Simuladores dos sintetizadores analógicos clássicos, ou da distorção de modelos específicos de amplificadores, ou da amostragem de uma bateria, são presentes digitalmente para serem interligados e mixados em projetos a partir de um software apenas, um software do tipo *DAW* (*Digital Audio Workstation*) [37].

Diversos modelos de *DAWs* existem atualmente no mercado, com alguns dos modelos existentes sendo inclusive de graça. Com exemplos de nomes como *Ableton Live*, *Logic Pro*, *Cubase*, *FL Studio*, entre outros, eles basicamente funcionam de uma forma geral semelhante. O intuito é ser fácil de se utilizar e ao mesmo tempo eficiente. Um usuário de um programa desse tipo normalmente tem a possibilidade de escolher e configurar um entre diversos instrumentos musicais virtuais e/ou efeitos sonoros digitais que possui, gravar/importar, editar e processar áudio externo, inserir notas e eventos musicais via *MIDI*, piano visual ou notação musical, executar o projeto e escutar o resultado do áudio processado e mixado, podendo exportar o áudio, reimportar e trabalhar em cima dele, entre diversos outros procedimentos [20].

É impossível mencionar *DAWs*, instrumentos e efeitos virtuais, música digital em geral, sem mencionar *MIDI*. *MIDI* é um protocolo de mensagens voltado a eventos musicais, que existe a mais de 30 anos e é um elemento indispensável, padrão para praticamente qualquer software musical existente hoje no mercado. A arquitetura é baseada no envio e recebimento de mensagens (*bytes*) a partir de dispositivos externos, como os clássicos teclados *YAMAHA*, ou os modernos controladores diversos, ou em formatos que definem um sequenciamento de mensagens.

MIDI possui um vasto número de mensagens previamente definidas possíveis, com mensagens que são principalmente execuções de notas pelo instrumento e controles de parâmetros durante a execução destas. Esses eventos podem ocorrer de duas formas, em tempo real e em sequência. Um evento em tempo real pode ser gerado a partir de, por exemplo, um pressionamento de uma tecla de um teclado musical que contém essa arquitetura, enviando esse evento de nota para ser interpretado e processado a fim de se

gerar um timbre como saída. Eventos em sequência servem como uma partitura de uma música a ser executada, e ocorrem quando sequenciadores/*DAWs* enviam essas mensagens para instrumentos e efeitos virtuais, para o próprio DAW, organizados de acordo com os parâmetros definidos ou inclusive para controle de outros elementos como luzes em um show, por exemplo [1].

MIDI também pode ser executado a partir de um arquivo próprio, que tem como propósito o sequenciamento de uma música, contendo um cabeçalho com definições da execução, canais síncronos ou assíncronos entre si que possuem uma sequência de eventos de notas e controles em um formato definido, a ser executado (tocado) por um interpretador MIDI. Esses arquivos são criados por sequenciadores *MIDI* que existem para esse propósito e tem no maior dos casos sua origem anterior aos demais *DAWs* existentes atualmente. *DAWs* vieram de sequenciadores *MIDI*, portanto muito do que é existente neles se baseiam nessa arquitetura fortemente, o que demonstra o quão robusta ela é e o quão importante é sua presença na tecnologia musical atual [11].

2.4 VST

Muitos *DAWs* proprietários trabalhavam sobre seus próprios instrumentos, fabricados pela mesma companhia de acordo com a arquitetura de seu software. Isso acarreta em um número restrito de instrumentos e efeitos úteis por *DAW*, já que portabilidade desses instrumentos era cercada de complicações. A necessidade que companhias, que trabalham especificamente com criação de softwares de áudio como instrumentos virtuais ou efeitos, teriam de portar seus softwares para diversas arquiteturas diferentes era inviável de um ponto de vista econômico.

É sobre esse conceito que é criada em 1996 a arquitetura *VST* pela *Steinberg*, a fim de se estabelecer um padrão de interface de troca de dados entre instrumentos e efeitos com os diversos *DAWs* em diferentes sistemas, o que evita esse trabalho a mais e facilita a comunicação entre eles. Entre as arquiteturas de propósitos semelhantes, *VST* é atualmente a mais utilizada para esse fim, sendo possível inclusive, a partir de um conhecimento prévio de processamento de sinal digital e de programação, criar um *plugin*, o instrumento/efeito, diretamente a partir da SDK apenas.

VST (*Virtual Studio Technology*) possibilita a criação de softwares componentes de áudio a serem executados em aplicações hospedeiras. As aplicações o tratam como uma caixa preta, oferecendo a ele entradas (áudio, eventos, configurações) e recebendo saídas após o processamento [36]. O número de entradas e saídas vai de acordo com o que a implementação do componente define, a cargo do programador, com um limite alto de até 2^{32} parâmetros de entrada.

Os softwares são divididos em três categorias, podendo ser geradores de áudio (*VST instruments*, *VSTi*), sendo estes sintetizadores e *samplers*; *plugins* de efeitos (*VST effects*), recebendo, processando o áudio e tendo como resultado esse áudio modificado ou uma visualização específica dele, como uma análise espectral por exemplo; e processamento de mensagens *MIDI* (*VST MIDI effects*), recebendo e alterando de acordo as mensagens e enviando as como saída [28].

Em geral, *VST* permite instrumentos e efeitos virtuais serem integrados com diversos *DAWs* nas plataforma Windows/Mac/Linux, ao qual na presença de comandos e eventos do *DAW*, executam comandos que direcionam a geração e processamento de áudio. Em

uma comparação, temos como se os instrumentos *VST* e os efeitos *VST* fossem respectivamente instrumentos musicais e efeitos pós-microfone como amplificadores, pedais de efeitos, entre outros. O *DAW* seria o estúdio onde esses instrumentos e maquinário são gravados, com a presença de mesa de som virtual, garantindo mixagem digital e controle sobre as trilhas, aos quais as notas e eventos diversos são executados pelos músicos no estúdio, no caso, mensagens *MIDI* no *DAW*. Com a presença de automações é possível também controle de parâmetros em tempo real [16].

Atualmente a *SDK* do *VST* se encontra na versão 3.6, sendo ela acessível no site oficial da *Steinberg* [26]. Devido a sua compatibilidade à diversos softwares, sejam via presença de traduções, "*wrappers*" ou outros meios, e devido a sua acessibilidade sem custo, *VST* é a arquitetura mais utilizada como interface de implementação nesse meio, concorrendo com diversas outras semelhantes como os *Audios Units* (*AU*), da *Apple* e o *Real Time Audio Suite* (*RTAS*), da *Digidesign*, por exemplo [2].

A interface *VST* foi desenvolvida na linguagem C++, o que permite a integração de diversas funcionalidades da linguagem ao projeto, como bibliotecas matemáticas ou até projetos inteiros envolvendo *DSP*. Basicamente é necessário apenas adaptar o código a ser executado ao escopo da arquitetura, o que se resume simplesmente em adaptar um dos exemplos fornecidos na *SDK* ao código, sem a necessidade de muita experiência no uso da *SDK*, ou até mesmo começar dele diretamente na tentativa de se entender os elementos básicos dela.

A partir de um dos códigos de exemplo fornecidos pela *SDK*, como o ***again***, temos alguns métodos principais a serem utilizados. O *again* é um *VST effect* que tem um parâmetro apenas que controla o ganho do sinal de saída a partir do sinal de entrada. Os métodos *processReplacing()* e *processDoubleReplacing()* descrevem o mesmo procedimento, um utilizando *float* e o outro utilizando *double*. Eles compõem a região principal voltada ao processamento do sinal, a ser implementado de acordo com as requisições do programador no desenvolvimento do sintetizador ou efeito. No caso do *again*, serve para a aplicação do ganho no sinal. Os *setters* e *getters* descritos são responsáveis pela troca de informações entre o *DAW* e o *plugin*. A partir do método *setParameter()* com o fornecimento dos parâmetros *index* e o valor *value*, o canal relativo a aquele índice é atribuído, geralmente a partir de uma interação feita pelo usuário sobre o *DAW* [17, 23].

2.5 Csound

A implementação do projeto descrito nessa monografia se baseia em uma linguagem poderosa para programação sonora, o *Csound*. *Csound* é um sistema com diversas funcionalidades voltadas a áudio, referenciado como linguagem apenas no texto para simplificar. Essa linguagem tem um vasto histórico, baseado na série de programas *Music-N*, de Max Mathews, começando com o programa *Music 4* desenvolvido na *Bell Telephone Laboratories* no início dos anos 1960 [22]. *Csound* está atualmente na versão 6.05.

A linguagem *Csound* foi desenvolvida originalmente por Barry L. Vercoe em 1985, e é uma linguagem voltada especificamente para a geração de áudio via unidades geradoras e um sistema de execução de eventos musicais. Ela é escrita completamente em C, o que facilita compatibilidade em relação a diversos sistemas. Sua execução pode ser feita direta via terminal, com comandos e *flags* de compilação dos códigos fontes, ou pode ser implementada e executada direto de uma *front-end* servindo como um programa gráfico

de auxílio na edição do código, como o exemplo do *QuteCsound*, contida no download do *Csound* [27].

O *Csound* está disponível como *freeware* no seu próprio site em versões para Windows, Mac e Linux [7].

2.5.1 Estrutura

A estrutura de um código fonte atual se baseia em subdivisões do código por *tags* de marcação de inicio e fim de escopo, como "*<CsInstruments>*" finalizada por "*</CsInstruments>*". O código pode ser completamente implementado direto em um arquivo somente, de extensão "csd" ou separado em arquivos para cada um dos escopos do código. Os escopos principais são:

- ***CsOptions***: região do código em que as *flags* de compilação se encontram. Elas podem fazer parte de um arquivo separado de extensão ".csoundrc" ou ainda inclusive ser definidas pela *front-end*;
- ***CsInstruments***: região do código que contém a estrutura principal, voltada a implementação da geração e controle do sinal sonoro. Essa região contém o *header* descrevendo informações sobre o estado do sistema (taxa de amostragem, número de canais, taxa de controle...) seguido de um número arbitrário de instrumentos. Cada instrumento é um bloco definido a partir dos comandos *instr*, rotulados com o número do instrumento e finalizados por *endin*, finalizando o seu bloco. Toda a geração do sinal sonoro é processada nos instrumentos, que recebem parâmetros tanto do escopo *CsScore*, descrito a frente, quanto de mensagens MIDI, se assim for definido na codificação. Inicializações são feitas logo abaixo do *header*, fora de um bloco definido de um instrumento.
- ***CsScore***: região do código que define a sequência de eventos/mensagens a serem enviados para os instrumentos definidos. Funciona como um certo tipo de partitura, em que "notas" são os principais eventos a serem descritos e enviados, interagindo com os instrumentos a partir dos parâmetros que compõe as mensagens. Serve também para a geração de tabelas de ondas, *f-tables*.
- ***CsoundSynthesizer***: define o escopo geral do arquivo, de extensão ".csd".

O *Csound* tem como base de seu funcionamento a utilização de diversos comandos, denominados *opcodes*, que possuem uma estrutura geral do tipo "*output(s)*" "*opcode*" "*parameter(s)*", além de cálculos matemáticos com o uso de variáveis. Basicamente "*output(s)*" armazenam o resultado do bloco de operações "*opcode*", que opera a partir do(s) parâmetro(s) "*parameter(s)*". Essas operações envolvendo o uso de *opcodes*, cálculos matemáticas, descritas em sequência dentro do bloco de um ou mais instrumentos, com argumentos dos parâmetros provenientes do *Score* ou de mensagens MIDI, descrevem um funcionamento geral de um programa em *Csound*. Os comandos subsequentes tendem a direcionar o sinal a alguma saída de áudio do sistema, mas não necessariamente, podendo se optar por utilizar a saída de um instrumento como entrada de outro, para processamento subsequente.

Partindo de um evento no *CsScore* ou via *MIDI*, a execução dos comandos de um instrumento é feita. Existem comandos/variáveis que são executados/atribuídos somente

uma vez, no início da execução do instrumento, ou que são atualizadas a uma taxa constante. Isso ocorre devido ao *Csound* trabalhar com passagens de interpretação para ambos os casos. A passagem de inicialização *i-pass*, faz com que valores sejam atribuídos apenas uma vez após o início do evento, enquanto que a passagem de performance *p-pass*, faz com que os valores sejam atualizados uma vez a cada bloco de amostras. A quantidade de amostras contidas em um bloco é determinada diretamente pelo parâmetro *ksmps*, definido no *header*. Quanto menor esse parâmetro, menor o tamanho do bloco, o que leva a um maior uso de processamento na geração do sinal, ao mesmo tempo que menor é o atraso gerado na geração do sinal na execução de um evento. O inverso para os dois casos também ocorre quando esse parâmetro aumenta, menor processamento e mais atraso. O equilíbrio entre performance e tempo de resposta deve ser levado em consideração na implementação no uso desse parâmetro *ksmps*.

Os tipos de variáveis diferentes no *Csound* tem ligação com essas passagens, e são diferenciadas por prefixos específicos. Os prefixos mais utilizados são o "i", para valores de inicialização, "k", para valores de controle, que são atribuídos uma vez por execução de um bloco de amostras, e "a", para o armazenamento de valores voltados a geração de sinal, armazenando-os como vetores de tamanhos equivalentes ao do bloco de amostras. O uso de cada tipo fica a cargo da implementação. Existem também o prefixo "S" para *strings*, e "g" para globais, utilizados antes dos prefixos citados anteriormente para que tal variável seja global, acessados por qualquer instrumento [5].

2.5.2 Código

```
1  <CsoundSynthesizer>
2  <CsOptions>
3  --env:SSDIR+=../SourceMaterials -odac
4  </CsOptions>
5  <CsInstruments>
6  ;Example by Alex Hofmann
7  instr 1
8  aSin     oscils   0dbfs/4, 440, 0
9           out      aSin
10 endin
11 </CsInstruments>
12 <CsScore>
13 i 1 0 1
14 </CsScore>
15 </CsoundSynthesizer>
```

Código 2.1: Código de exemplo do *Csound*.

Em resumo, o Código 2.1 descreve a execução do instrumento "*instr* 1", que executa uma nota senoidal de frequência 440 *Hz*, começando no instante 0 e durando 1 segundo.

A linha 3, interior ao *CsOptions* define 2 *flags*, sendo que o comando *-odac* determina que o sinal seja mandado para a saída de som principal do sistema.

As linhas 7 a 10, interno ao CsInstruments, representam o bloco do instrumento "*instr* 1", que possui apenas 1 *opcode*, *oscils*. O *opcode* em questão é um gerador de sinal senoidal, com parâmetros que são em sequência a amplitude, a frequência e a fase da onda. O sinal gerado é armazenado na variável *aSin*. O comando *out* transfere então *aSin* para a saída de áudio.

A linha 13, interno ao *CsScore*, descreve um evento do tipo "nota", relativo ao instrumento 1, indicado no seu primeiro parâmetro. Os outros 2 parâmetros numéricos definem em sequência o tempo de início do evento e sua duração.

Existem também comentários que começam a partir de ';' e finalizam no fim de linha, e comentários em blocos, no estilo da linguagem *C*, separandos-os com os caracteres '/*' e '*/'.

Este exemplo básico demonstra a simplicidade de se gerar um sinal mínimo sem muito esforço de implementação no *Csound*. A linguagem permite a criação de diversos instrumentos interno ao escopo *CsInstruments*, que podem inclusive se relacionarem a partir de chamadas ou variáveis globais. A quantidade de *opcodes* é enorme, envolvendo as demais formas de interação que o computador pode fornecer internamente, como geradores de sinal, filtros, processos matemáticos, incluindo o uso de hardware externo, na execução de processos com sinal, variáveis, controles, entre outros. A quantidade de argumentos que o *Score* pode transmitir para os instrumentos também é enorme, sendo estes três inclusos no evento apenas o mínimo necessário para a execução dele.

Além disso, a linguagem permite expressões complexas, desvios condicionais, criação de opcodes, uso de macros e demais outros comandos não citados ou não desenvolvidos em detalhes nesse exemplo.

Note que este é um exemplo bem simples que não faz o uso de todo o ferramental da linguagem e não descreve os demais elementos presentes nela. O manual canônico do *Csound* contém a referencia de toda a lista de *opcodes*, com exemplos específicos para cada. Uma leitura mais aprofundada com execução dos diversos exemplos é útil para familiarizar-se melhor com a linguagem e com as operações dos *opcodes* [27].

Em consequência da evolução da linguagem *Csound*, e da evolução da computação musical, a idéia de se ter um instrumento *Csound* executando em um *DAW* foi cogitada e de fato implementada, inicialmente por Michael Gogins, que criou o *CsoundVST* [10]. *CsoundVST* é uma *frontend* que basicamente apresenta um *plugin VST* que pode carregar código pertencente a linguagem *Csound* e executar *opcodes* específicos para a interação deste com eventos *MIDI* provenientes do *DAW*. Atualmente, o outro projeto que trabalha com o mesmo conceito de interagir *Csound* e *VST* é o *Cabbage*.

2.6 *Cabbage*

Cabbage é uma *frontend* em desenvolvimento por Rory Walsh que, no uso do *Csound*, gera instrumentos e efeitos compatíveis com *DAWs* ou *standalones* para os sistemas *Windows*, *Mac* e *Linux*[32]. Ele utiliza-se da arquitetura para *Windows VST*, da arquitetura para *Linux LADSPA* (*Linux Audio Developer's Simple Plugin API*) [18] e da arquitetura para *Mac AU* (*Audio Units*) para gerar a partir do arquivo *Csound* um *plugin* de áudio compatível com o sistema e com interface gráfica definida a partir do código, com seu fluxo de acordo com a Figura 2.13.

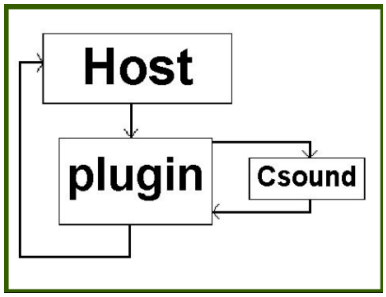

Figura 2.13: Fluxo de execução do *Cabbage* em relação ao *host* (*DAW*).

Por ser uma *frontend*, possui diversos elementos próprios de interface gráfica, como *knobs* circulares, *sliders* verticais e horizontais, botões, textos, tabelas editáveis, entre outros, para serem utilizados pelo programador no processo de desenvolvimento da parte gráfica do *plugin* [29]. Sua execução pode ocorrer direto no *DAW*, sendo este o objetivo do interfaceamento, ou em modo *standalone* para teste ou uso externo [33].

Ele está em seu próprio *site* atualmente na versão 1.1 para as plataformas *Windows*, *Mac*, *Linux* e *Android* [4]. Sua *API* possui comandos de fácil compreensão para geração do *plugin*, além de um próprio editor de texto, e um editor da estrutura visual do *plugin*. A documentação presente no site exemplifica muito bem os elementos gráficos que podem ser utilizados e a variabilidade de comandos identificadores para cada um. Alguns dos principais comandos que podem ser adicionados aos elementos de interface gráfica são descritos a seguir:

- *bounds(x, y, width, height)*: define a posição (x,y) do elemento e seu tamanho nos dois eixos (width, height);
- *channel("chan")*: Define o nome do canal de acesso ao valor do elemento pelo *Csound*, para ser acessado via o *opcode chnget* ou atualizado via o *opcode chnset* nos instrumentos a partir desse nome.
- *identchannel("channel")*: Define o nome do canal de acesso aos identificadores do elemento pelo *Csound*. Isso dá a possibilidade de se atualizar o estado de algum identificador de um elemento gráfico, como por exemplo a posição e tamanho dele enviando no *Csound* uma *string* composta por *bounds(1, 2, 3, 4)* para o canal com esse nome.
- *range(min, max, value, skew, incr)*:Define o alcance mínimo e máximo do elemento, valor inicial, formato de atualização de valores entre linear e exponencial, e a quantidade a ser incrementada a cada passo da interação do usuário. Utilizado por exemplo em *sliders*
- *text("name")*: Define o nome do elemento, geralmente demonstrado próximo ou interno a ele. Pode conter mais *strings* para casos de elementos com estados diferentes (botão ligado e desligado, por exemplo).
- *value(val)*: Define o valor inicial do elemento. Utilizado por exemplo em botões.
- *visible(val)*: Define a visibilidade do elemento na tela.

- *active(val)*: Define se o elemento esta ativo para o usuário.

Além desses principais, existem mais identificadores na ferramenta, voltados a modificar características como por exemplo o aspecto geral do *plugin*, exemplificados mais a frente, ou a aparência dos elementos. A lista completa se encontra na sua documentação [29].

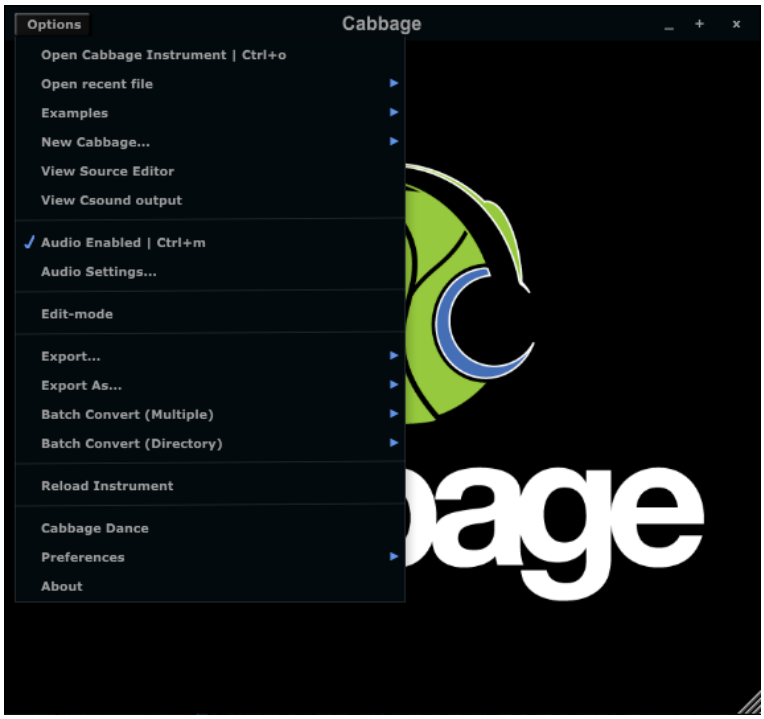

Figura 2.14: Janela do *Cabbage*, com alguns comandos básicos presentes na lista de opções.

De início você pode criar um novo instrumento ou efeito com o comando "*New Cabbage*" a partir da lista de opções demonstrada na Figura 2.14, editar seu código via "*View Source Editor*", e exportá-lo com o tipo correto com "Export". O código referente ao *plugin VST* somente precisa ser exportado uma vez, sendo que o arquivo ".csd", que deve ser posicionado junto ao *plugin* (*dll*) no caso do *Windows*, é o único a ser alterado após a primeira exportação. Isso dá a possibilidade de se editar o instrumento *VST* sem a necessidade de exportá-lo novamente.

O *Cabbage* vem com um grande número de exemplos que podem ser acessados via a opção "*Examples*". Grande parte desses exemplos são compostos de diversos instrumentos implementados no *Csound* ao longo de sua existência principalmente por *Iain Mccurdy* e portados para a interface do *Cabbage* [19]. Eles são agrupados de acordo com sua categoria, alguns voltados a serem instrucionais para o usuário, outros sintetizadores ou efeitos finalizados.

O código que é gerado a partir do "*New Cabbage*" →"*Effect*" descreve o mínimo necessário para se gerar um *VST effect* que recebe o sinal e não o altera, com o código de direcionamento do sinal a saída comentado pelo caractere ";". A descrição da ferramenta terá um formato mais prático. Então, com pequenas alterações se cria um código demonstrativo, voltado a alterar o ganho de um sinal de entrada, com interface própria

do *Cabbage*. O plugin se denomina *again* devido a ser equivalente a um dos exemplos de mesmo nome encontrado na *SDK* do *VST*.

2.6.1 *Again*

```
1  <Cabbage>
2  form size(160, 160), caption("aGain"), pluginID("plu1")
3
4  rslider bounds(30,30,100,100), text("GAIN"), channel("gain"), range(0, 1,
       .1)
5
6  </Cabbage>
7  <CsoundSynthesizer>
8  <CsOptions>
9     -n -d
10 </CsOptions>
11 <CsInstruments>
12    sr = 44100
13    ksmps = 64
14    nchnls = 2
15    0dbfs=1
16
17    instr 1
18       aL inch 1
19       aR inch 2
20
21       kgain chnget "gain"
22
23       outs aL * kgain, aR * kgain
24    endin
25
26 </CsInstruments>
27 <CsScore>
28    f1 0 1024 10 1
29    i1 0 28800
30 </CsScore>
31 </CsoundSynthesizer>
```

Código 2.2: Código do *again* implementado em *Csound* no *Cabbage*.

Em relação ao Código 2.2, o primeiro elemento a se perceber é a nova tag *Cabbage*, definindo o escopo da interface gráfica do *plugin*. O *Cabbage* precisa definir a janela do *plugin* e seu único parâmetro de ganho, ambos definidos respectivamente nas linhas 2 e 4. O formato das definições ocorrem com um entre os diversos tipos de comandos relativos a geração de um elemento da interface, seguidos de configurações para este elemento a partir de identificadores.

O comando "*form size()*" na linha 2 define o tamanho da janela do *plugin* em 160 $pixels^2$, seguidos de nome do *plugin* e *pluginID*, ambos que comunicam com a *SDK* do *VST* para gerar esses elementos de forma correta no *DAW*. Nesse caso, o *pluginID* deve

ser uma *string* de 4 caracteres única, em relação aos demais *plugins VST* existentes, caso se deseje publicar o instrumento desenvolvido.

A linha 4 possui o comando *rslider()*, que serve para gerar um *knob* circular posicionado de acordo com seus argumentos. Os identificadores seguintes "*text()*" e "*channel*" definem o nome do parâmetro e o nome de um canal de comunicação entre o valor do parâmetro e o escopo *CsInstruments*, no caso "*gain*", com alcance e valor inicial definidos pelo identificador "*range()*". A comunicação entre os elementos de interface do *Cabbage* e o *Csound* é feita a partir de canais, como o canal "*gain*" definido, que normalmente são únicos para cada elemento gráfico.

As linhas 12 a 15 inicializam o *Csound* definindo a taxa de amostragem em 44.1*Khz*, número de amostras em um bloco de controle em 64 e 2 números de canais de saída.

As linhas 17 a 24 definem o instrumento 1, que no caso opera a partir de sinal estéreo de entrada. Ele recebe as entradas esquerda e direita utilizando os comandos "*inch*" e armazenando-as em *a1* e *a2* nas linhas 18 e 19, multiplicando-as pela variável *kgain* na linha 23.

Essa variável "*kgain*" é alterada pelo comando "*chnget*" na linha 21, que roteia o valor emitido pelo canal de nome "*gain*" a partir de alterações nesse *knob* definido pelo *rslider()*.

Figura 2.15: *again* implementado no *Cabbage* executando no *FLStudio*.

A simplicidade do *plugin* de efeito representada no código, tanto na criação da interface gráfica quanto no processamento do sinal demonstram muito bem a facilidade de geração dele, tendo seu visual exemplificado na Figura 2.15. Em sequência temos a implementação de um sintetizador simples, que usufrui de alguns dos elementos do *Cabbage* e do *Csound*.

2.6.2 Sintetizador Básico

Começando pelo *plugin* antes do código, podemos perceber na Figura 2.16 que seu conteúdo é mais complexo e com maior número de elementos que o anterior.

O instrumento é polifônico, possui duas ondas e possui um filtro passa-baixa simples configurável, com a possibilidade de definir uma *ADSR* para sua frequência de corte, como era comum em sintetizadores analógicos, além de um *ADSR* padrão para a amplitude.

Note que existem diversos "*rsliders*" presentes no *plugin*, assim como diversos outros tipos de elementos que podem ser interagidos, como *sliders* verticais e horizontais, botões, *combo-box* e até um teclado visual e um terminal do *Csound*, para se visualizar as mensagens *MIDI* recebidas e possíveis erros que podem ocorrer durante sua execução. Cada um dos elementos gráficos são descritos por comandos específicos do *Cabbage* e roteados para as variáveis corretas.

Figura 2.16: Sintetizador simples demonstrativo implementado no *Csound/Cabbage* executando no *FLStudio*.

É perceptível também o agrupamento de alguns elementos, uma possibilidade que a interface do *Cabbage* permite, facilitando o posicionamento dos elementos que são agora relativos a posição do elemento agrupador. O identificador *plant()* é utilizado para esse procedimento.

Em relação ao código, os elementos tendem a ser descritos de forma semelhante. Analisando somente alguns deles, como o "Volume *ADSR*" e "Detune" por exemplo, exemplificamos o uso de grupos e outros elementos gráficos.

```
12 ;;;;; VOL-ADSR ;;;;;
13
14 groupbox bounds(150, 0, 300, 100), text("Volume ADSR"), preset("preADSR"),
       plant("plantADSR"){
15     rslider bounds(.05,.33, .6, .6), text("ATK"), channel("atk"), range(0,
           1, .1)
16     rslider bounds(.3, .33, .6, .6), text("DEC"), channel("dec"), range(0,
           1, .1)
17     rslider bounds(.55,.33, .6, .6), text("SUS"), channel("sus"), range(0,
           1, .75)
18     rslider bounds(.8, .33, .6, .6), text("REL"), channel("rel"), range(0,
           1, .3)
19 }
```

Código 2.3: Código referente a *ADSR* do *plugin*.

O comando "*groupbox bounds*" gera o grupo com os argumentos definindo posição e tamanho, nomeado em sequência pelo identificador "*text*". As linhas 15 a 18 geram *knobs* circulares a partir do mesmo comando utilizado no *again* feito no *Cabbage*. O posicionamento e tamanho dos elementos agora são relativos a posição e tamanho do grupo ao qual estão contidos. O mesmo ocorre nos outros casos em que os elementos estão visualmente internos a grupos.

```
36 groupbox bounds(0, 100, 150, 100), text("Detune"), preset("preDetune"),
       plant("plantDetune"){
37     hslider bounds(0, .33, 1, .33), channel("detune1"), textBox(0),
           range(-1, 1, 0), fontcolour("blue")
38     hslider bounds(0, .66, 1, .33), channel("detune2"), textBox(0),
           range(-1, 1, 0), fontcolour("blue")
39 }
```

Código 2.4: Código referente ao *detune* do *plugin*.

No caso do "*Detune*", o *hslider* é utilizado, representando um controle horizontal. A codificação da síntese por sua vez utiliza-se de alguns *opcodes* do *Csound*.

```
74
75 instr 1
76
77 iwaveType1 chnget "waveType1"
78 iwaveType2 chnget "waveType2"
79
80 iatk chnget "atk"
81 idec chnget "dec"
82 isus chnget "sus"
83 irel chnget "rel"
84
85 ifAtk chnget "fAtk"
86 ifDec chnget "fDec"
87 ifSus chnget "fSus"
88 ifRel chnget "fRel"
89 kcutOff chnget "cutOffFreq"
90 kQ chnget "quality"
91
92 kdetune1 chnget "detune1"
93 kdetune2 chnget "detune2"
94
95 kvol chnget "volume"
96 kvolW1 chnget "volumeW1"
```

Código 2.5: Roteamento dos "*channels*" para variáveis de controle.

Primeiramente todos os "*channels*" definidos no escopo gráfico devem ser roteados para suas devidas variáveis. O próximo passo é a utilização delas no processamento do

sinal no resto da descrição do instrumento.

```
 99
100 kADSR madsr iatk, idec, isus, irel, 0 , 1
101 kFADSR madsr ifAtk, ifDec, ifSus, ifRel, 0 , 1
102
103 a1 oscili p5 * kADSR * kvolW1 * kvol, p4 + (p4 * kdetune1 * 0.2),
        iwaveType1
104 a2 oscili p5 * kADSR * kvolW2 * kvol, p4 + (p4 * kdetune2 * 0.2),
        iwaveType2
105
106 a1 lowpass2 a1, kFADSR * kcutOff, kQ
107 a2 lowpass2 a2, kFADSR * kcutOff, kQ
108
109 outs a1 * 0.8 + a2 * 0.2, a1 * 0.2 + a2 * 0.8
```

Código 2.6: Processamento geral do sinal e roteamento para a saída de áudio.

Os *opcodes* utilizados são "*madsr*", que tem como função gerar uma *ADSR* a partir dos parâmetros inseridos, "*oscili*", semelhante ao *oscils* na geração de onda, agora com escolha de tipo de onda via acesso a tabelas *f-tables* e "*lowpass2*", filtro passa-baixa com frequência de corte e qualidade (ressonância), todos esses que geram/operam sobre os sinais *a1* e *a2*, mandados para a saída estéreo com o comando *outs*.

No caso do *oscili*, a amplitude para a primeira ocorrência é definida pela equação "$p5 * kADSR * kvolW1 * kvol$" e a frequência da onda por "$p4 + (p4 * kdetune1 * 0.2)$". O *Csound* está em um estado em que a execução de eventos são provenientes de mensagens *MIDI* ao invés do uso de eventos do escopo *CsScore*. Nesse caso, os parâmetros *p5* e *p4*, que naturalmente são o 5º e 4º parâmetros de um evento no CsScore agora representam a *velocity* e *cps* da nota ativada já convertida do *MIDI*, devido as flags de comando "–midi-key-cps=4 –midi-velocity-amp=5" descritas no *CsOptions*.

O último parâmetro escolhe o tipo de onda a ser gerada, a partir das definições de ondas descritas no *CsScore*, nas linhas 114 a 116, utilizando *f-tables* para gerar tabelas de ondas.

```
112 </CsInstruments>
113 <CsScore>
114 f1 0 4096 10 1 .5 .3 .25 .2 .167 .14 .125 .111
115 f2 0 4096 10 1 0 .3 0 .2 0 .14 0 .111
116 f3 0 4096 10 1
117 f0 28800
```

Código 2.7: *Score* definindo as *funtion Tables* para as três ondas.

Utilizando a linha 114 como exemplo, temos uma tabela com 4096 pontos representando a partir do 5º valor as parciais que representam a onda dente-de-serra, com apenas 8 harmônicos. O mesmo ocorre com as linhas 115 e 116, só que agora com outras for-

mas de onda, o pulso e a senóide. A última função na linha 117 serve para programar o instrumento para tocar por 8 horas, representadas em segundos.

Para esses dois *plugins* descritos, a execução no *DAW FLStudio* ocorreu sem problemas, com apenas algumas configurações necessárias no programa em relação ao tamanho e tipo de *buffer*, definido como estático.

2.7 Trabalhos Correlatos

O *Cabbage* possui uma lista de instrumentos exemplos que implementam as demais técnicas de síntese de diversas formas. A gama de instrumentos relacionados ao descrito nesse trabalho é alta devido a ele ser baseado em tipos de síntese presente nos exemplos do *Cabbage*.

O que diferencia tais instrumentos que se baseiam nessas mesmas técnicas entre si e entre o desenvolvido neste trabalho, são as características da forma como o sintetizador opera, como ele se comporta, sua arquitetura e *design*.

Grande parte dos instrumentos desenvolvidos no *Cabbage* foram disponibilizados como exemplos que acompanham a instalação dele no sistema. Tais exemplos, com a adição de diversos outros voltados somente ao *Csound*, também se encontram disponibilizados no site do Iain McCurdy [19] e do próprio fórum da ferramenta *Cabbage* [30].

Uma das principais referências do sintetizador, principalmente para a organização da interface e estrutura dos módulos foi o sintetizador *Sytrus*, um dos sintetizadores desenvolvido pela *Image-Line* na linguagem *Delphi*, contido no *DAW FL Studio* [15]. Em resumo, sua arquitetura de síntese se baseia no uso de diversas técnicas em conjunto, como frequência modulada, modulação em anel, síntese aditiva e síntese subtrativa. Ele é composto por 6 osciladores (operadores), 3 filtros de múltiplos tipos, mapa de modulação de frequência e de modulação em anel, além de um banco de efeitos, com *chorus*, reverberação, *delay* e *unison*. Devido a ser um *plugin* privado de código fechado, muito do seu funcionamento interno foi especulado a partir do uso do instrumento, da visualização e do diagrama de processamento fornecido, o que, apesar de somente demonstrar uma visão de mais alto nível, serviu para nortear o *design* do sintetizador *SaWerkraut*.

Capítulo 3

O Sintetizador SaWerkraut

Esse capítulo descreve o sintetizador *SaWerkraut*, desenvolvido a partir da linguagem *Csound* em conjunto com a ferramenta *Cabbage*, permitindo que ambos em conjunto possam ser utilizados em softwares do tipo *DAW*, voltados especificamente para a produção de áudio. O sintetizador trabalha com técnicas de síntese dos tipos aditiva, subtrativa e frequência modulada (*FM*) e é compilado em um *plugin VST*. A linguagem *Csound* e a ferramenta *Cabbage* foram utilizadas em conjunto para, respectivamente, implementar toda a parte relativa ao processamento e geração de sinal e implementar a interface gráfica para o usuário e o interfaceamento entre o *Csound* e o *DAW*.

3.1 Visão Geral

O sintetizador tem uma estrutura baseada em 3 osciladores independentes entre si e um filtro. Todos esses possuem Envoltória *ADSR* e *LFO* configuráveis para alguns de seus parâmetros. Também possui um mapa de frequência modulada entre os 3 osciladores, além de reverberação e alguns controles globais. A seguir temos uma enumeração e resumo dos componentes que estão presentes no sintetizador:

- 3 osciladores de formato equivalente, independentes entre si;
  - síntese por tabela de onda, com cinco tipos disponíveis (seno, triângulo, serra, quadrado e pulso);
  - volume, multiplicador de frequência, fase e quantidade de modulação *FM*;
  - envoltória por *ADSR* (*attack*, *decay*, *sustain* e *release*) e *LFO* independentes para 4 parâmetros;
    - amplitude;
    - espaçamento estereofônico (*Pan*);
    - altura;
    - modulação por *FM*;
- filtragem dinâmica pelo filtro *SVF* (*State Variable Filter*);
  - frequência de corte (Cutoff) e ressonância;
  - níveis dos filtros passa-baixa, passa-banda e passa-alta;

  - envoltória por *ADSR* (*attack*, *decay*, *sustain* e *release*) e LFO independentes para 2 parâmetros;
    * frequência de corte (Cutoff);
    * ressonância;

- Efeitos
  - *Delay*
    * Volume
    * *FeedBack*
    * Tempo
    * *Offset* estereofônico
  - Reverberação
    * Volume;
    * Nível de *Feedback*;
    * Frequência de corte;

- mapeamento de modulação *FM* para cada um dos 3 osciladores (9 relações);
  - Parâmetros de saída para os 3 sinais modulados;
    * espaçamento estereofônico (*Pan*);
    * volume;

- mapeamento de filtragem para cada oscilador pós modulação *FM*;
  - parâmetros de saída para cada sinal filtrado;
    * espaçamento estereofônico (*Pan*);
    * volume;

- controles globais do sintetizador;
  - volume;
  - pan;
  - desvio de altura em semitons;
  - desvio de altura fina;
  - multiplicador do *Pitch Bend*;

A síntese é construída a partir de uma série de procedimentos em sequência. Eles tem início primordialmente a partir de um evento MIDI de nota tocada, e envolvem a leitura de todos os estados dos elementos de interface, seguidos então pelo cálculo e geração dos parâmetros que são armazenados, e por fim são processados. Finalmente com todos os elementos calculados é feita a geração do sinal. Os procedimentos são divididos em etapas separadas, módulos que são descritos em seções posteriores.

Com a ferramenta *Cabbage* a interface foi construída e possui seu formato como demonstra a Figura 3.1. A interface pode ser dividida em cinco regiões:

- Selecionador de módulo;
- Parâmetros globais;
- Módulo atual;
- Mapa de *FM*, filtro e saída;
- Teclado virtual;

Figura 3.1: Sintetizador desenvolvido. Visão geral com foco no módulo do oscilador 1.

Uma das características do sintetizador utilizando tal ferramenta foi uma arquitetura envolvendo troca de módulos de interface, o que permite em determinada área da tela obter mais elementos gráficos. Portanto o módulo demonstrado na Figura 3.1 é referente ao oscilador 1, que está posicionado no grupo de painéis sob o texto "*Oscillator 1*" até o teclado virtual. Existem 6 módulos principais que podem ser selecionados a partir do selecionador de módulo, localizado no topo do sintetizador demonstrado na Figura 3.2. São eles o módulo principal, os 3 osciladores, o filtro e o módulo de efeitos. Cada módulo modifica as abas referentes ao mesmo espaço descrito no oscilador 1, mais ao centro do sintetizador.

Figura 3.2: Selecionador de módulos, ao topo do sintetizador.

Essas trocas de módulos são feitos a partir do uso de *triggers* no *Csound* para os canais respectivos aos elementos de interface. Quando esses *triggers* são acionados a partir de alguma alteração em um desses canais, uma chamada relativa é feita e o *Csound* atualiza a partir do *identchannel* daquele elemento gráfico. O *identchannel* é uma das propriedades de elementos gráficos do *Cabbage* e descreve um *link* entre o *Csound* e esse elemento. Isso serve para o *Csound* atualizar parâmetros do elemento, como por exemplo a visibilidade dele, de acordo com o valor escolhido pelo usuário, o que é feito no caso das trocas de módulos. Existem 232 canais de acesso definidos via *Cabbage* para todo o sintetizador, sendo eles distribuídos entre os *sliders* circulares, seletores de módulos, botões, entre todos os outros elementos fornecidos pelo *Cabbage*.

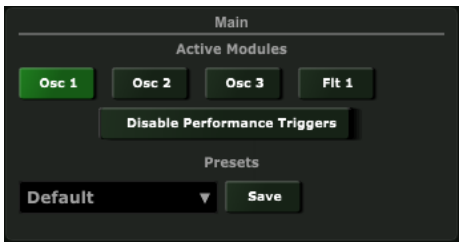

Figura 3.3: Visão com foco no módulo *main*.

A Figura 3.3 contém o primeiro módulo, o módulo principal. Ele possui botões referentes a ativar e desativar algumas funcionalidades do sintetizador, relativas a performance. Os botões superiores servem para ativar e desativar a execução dos 3 osciladores e do filtro. Inicialmente, somente o oscilador 1 está ativo, como demonstra o estado da Figura 3.3. Nesse caso somente ele produz sinal e pode ser utilizado no processo de modulação de frequência.

O botão "Disable Performance Triggers" logo abaixo tem a função de desabilitar a execução dos *triggers* de performance. Tais *triggers* ficam em modo de espera e são ativados quando algum parâmetro é modificado enquanto uma ou mais notas estão ativas, o que gera a chamada da reinicialização de um grupo de comandos via o comando *reinit*, gerando a modificação do sinal durante esse período de acordo com esse parâmetro modificado. Desabilitar esses *triggers* diminui o impacto da performance na síntese, mas elimina a possibilidade de se alterar qualquer parâmetro via eventos do *DAW* ou do usuário após a execução de uma nota que reflete nessa síntese dessa nota. Caso um parâmetro seja alterado, seu novo valor somente será utilizado na próxima execução de nota. O uso dessa funcionalidade é recomendado quando não há a intenção de se alterar parâmetros durante a execução de notas, gerando um ganho de performance.

Finalmente, abaixo do texto "Presets", existe uma área destinada a carregar e salvar o estado atual de todas as configurações do sintetizador. O *checkbox* ao lado contém a lista de *presets*, que estão armazenadas em arquivos de extensão "*.snaps*" localizados no diretório "SaWerkraut Presets", junto do *csd*. O botão *Save* ao lado abre um janela para que seja inserido o nome do *preset* a ser salvo, que então é adicionado a lista ao lado. Para carregar um *preset*, basta selecioná-lo na lista. A implementação do sistema de *presets* é feita de acordo com o que é descrito pela própria ferramenta, com apenas o uso de 2 comandos.

A esquerda temos um painel com os parâmetros globais, demonstrados na Figura 3.4, e acessados e utilizados por seções do sintetizador. Partindo do topo da Figura 3.4 para

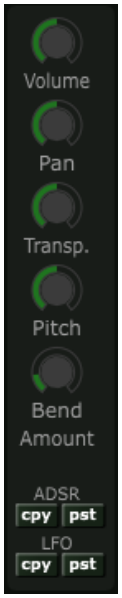

Figura 3.4: Região a esquerda do sintetizador, dedicada aos parâmetros globais configuráveis.

baixo, os parâmetros são respectivamente: Volume da saída geral, *Pan* geral, transposição das notas em semitons com alcance de 2 oitavas abaixo e acima, altura fina com variação de 50 centésimos abaixo e acima, e o multiplicador do efeito de *Pitch Bend*, realizado via teclado *MIDI* ou por comandos do *DAW*. Grande parte dos elementos gráficos do projeto são constituídos de *sliders* circulares, denominado *rslider* no *Cabbage*. Ainda nesse painel, temos também abaixo 4 botões destinados a copiar e colar estados da *ADSR* e do *LFO*. A cópia é feita a partir do pressionamento dos botões a esquerda, que armazenam os parâmetros do módulo atual de acordo com o tipo (*ADSR* ou *LFO*), e a colagem é feita no pressionamento dos botões a direita, a partir dos parâmetros armazenados para qualquer outro módulo atual que contenha essas duas regiões, como os osciladores e o filtro.

A direita do sintetizador, na Figura 3.1, temos o Mapa de *FM*, mapa do filtro e as saídas correspondentes a cada uma. Essa região controla justamente os níveis de intensidade de modulação de frequência que ocorre entre os osciladores do sistema. Também possui a quantidade a ser filtrada, e o *pan* e volume das demais saídas do sinal. Osciladores, filtros e o mapa de *FM* serão descritos mais detalhadamente em seções posteriores.

Finalmente localizado abaixo temos um teclado virtual, com uso focado em testes.

3.2 Fluxo e Controle

Em relação ao fluxo do sinal, o diagrama apresentado na Figura 3.5 descreve sucintamente como ele ocorre, demonstrando as etapas partindo da leitura dos valores dos elementos da interface do *Cabbage* até o processamento do código em Csound.

O *Cabbage* com seus elementos de interface, a partir da interação do usuário ou de comandos provenientes do *DAW* com acesso aos canais, configura os parâmetros e os transfere para os demais módulos do sintetizador, os blocos do diagrama. Cada bloco

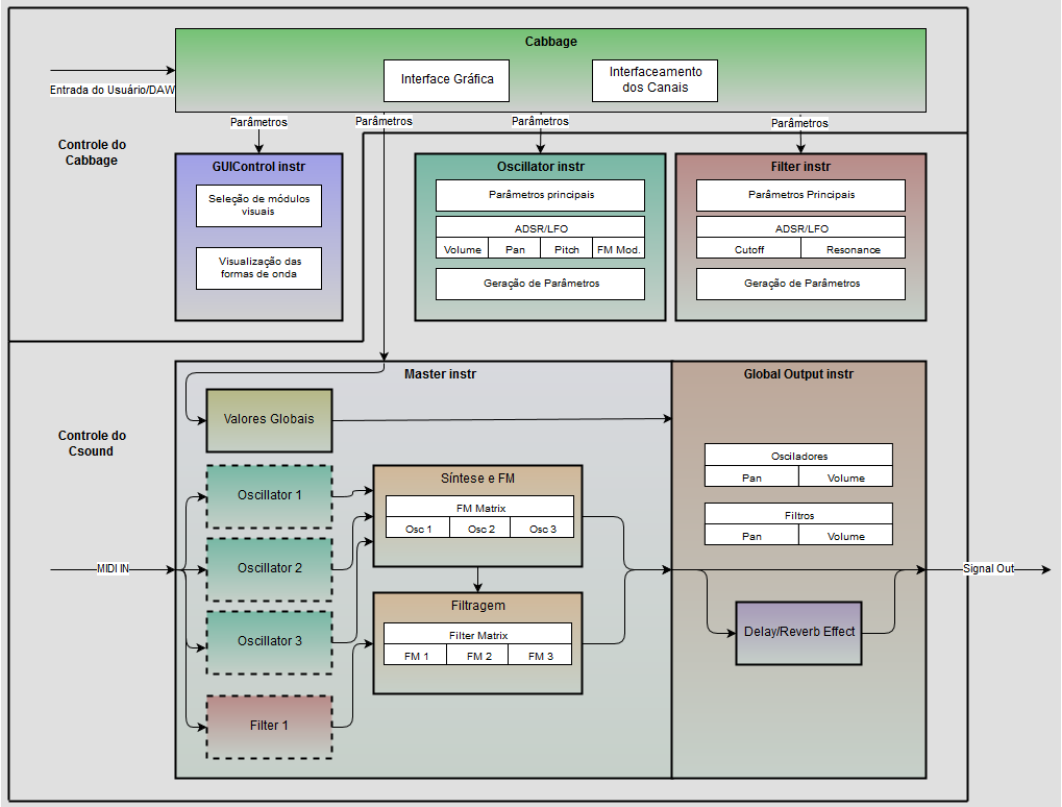

Figura 3.5: Diagrama básico da estrutura do sintetizador.

separado do diagrama, com exceção do bloco *Cabbage*, equivale a um instrumento no *Csound*.

O bloco/instrumento "*GUIControl instr*" é um instrumento apenas de inicialização e de controle dos elementos da interface. Ele trabalha, junto ao *Cabbage*, trocando módulos, definindo o que visualmente está ativo e o que não está, e modificando a onda demonstrada do oscilador após a troca, todas elas a partir da interação do usuário nos demais elementos da interface. As principais trocas ocorrem no uso dos painéis de seleção de módulo, que modificam o módulo atual de acordo com a escolha, e os painéis internos a esses módulos (oscilador e filtro), que trocam os módulos da *ADSR* e do *LFO* de acordo com a seleção. Ele também é utilizado nas operações de cópia e colagem da *ADSR* e *LFO* entre os módulos. Esse instrumento e o instrumento "*GlobalOutput*" são executados a partir de eventos no *Score* do *Csound*, de duração extremamente alta, mantendo-os sempre ativos e não somente na execução de notas e eventos.

O bloco "*Master*" recebe os parâmetros globais e realiza as principais funções e chamadas do sintetizador, sendo ativado e instanciado a partir de um evento de nota *MIDI* executado. Os blocos "*Oscillator*" e "*Filter*" são instrumentos que são instanciados e controlados pelo instrumento "*Master*". Há 3 instâncias do oscilador e 1 do filtro, e é importante citar que esses 2 instrumentos geram como saída parâmetros e configurações, e não sinal. Todo o sinal é gerado no instrumento "*Master*" devido ao fator de que os osciladores instanciados não terem a possibilidade de executar o processo de *FM* antes de toda a geração de sinal dos osciladores ter sido feita. Essa escolha de arquitetura trouxe a necessidade de armazenamento dos valores retornados por essas instâncias em matrizes

globais, que são indexadas, no caso dos osciladores, de acordo com o número do oscilador correspondente. Existe também matrizes de *strings* para os canais do *Cabbage* que o instrumento "*Oscillator*" acessa ao invés de *strings* definidas diretamente nele, devido a ele ser instanciado 3 vezes com a necessidade de acessar canais diferentes para os módulos osciladores diferentes.

As 3 instâncias dos osciladores e a instância do filtro, demonstradas como bloco pontilhados no diagrama da Figura 3.5, enviam seus parâmetros para regiões destinadas a síntese do sinal e aplicação da FM, e filtragem. Em sequência, no instrumento "*GlobalOutput*", é aplicado o efeito de *delay* e reverberação sobre o sinal final e é feito o processo de direcionamento do sinal a saída com controle de volume e *pan* a partir dos valores globais.

3.3 Oscilador

O oscilador possui sua interface composto por 3 regiões separadas. A primeira, superior, descreve os elementos principais dele. A segunda em conjunto com a terceira, descrevem 2 processos, *ADSR* e *LFO* para 4 parâmetros do sinal, volume, *pan*, altura e modulação por *FM*.

Figura 3.6: Região superior do oscilador.

A Figura 3.6 descreve os elementos principais de um oscilador. É possível escolher uma entre as 5 ondas predefinidas (seno, triângulo, serra, quadrado e pulso), que atualiza a tabela ao lado a cada escolha. O tipo de onda pulso possui um parâmetro adicional, que se torna visível ao lado da forma de onda, que é utilizado para modificar a largura do pulso. Também é possível configurar o volume, fase, desvio de frequência e a quantidade de modulação *FM* que a onda opera em relação ao mapa de modulação.

O desvio de frequência ocorre de duas formas, relativa e absoluta. Os dois parâmetros a direita do volume controlam o desvio relativo, o primeiro variando em inteiros de 0 a 32, inicializado em 1, e o segundo em frações de 0 a 1. Ambos são somados e em seguida multiplicados a frequência da nota tocada. O terceiro é uma variação absoluta de frequência somada a frequência original da nota. Esses três controles dão a possibilidade de osciladores distintos operarem sobre frequências diferentes de razões iguais, fator importante caso se deseje adicionar os sinais ou principalmente na aplicação de *FM* entre os osciladores, ou adicioná-los a mixagem de saída.

A quantidade de modulação indica quanto de modulação o oscilador receberá em relação ao mapa de modulação na linha correspondente a ele. Esse parâmetro será melhor esclarecido na Seção 3.5

Em sequência temos as 2 regiões responsáveis pelo *ADSR* e *LFO* dos quatro parâmetros. As regiões correspondem respectivamente a seleção do módulo (parâmetro) a ser configurado e ao grupo de parâmetros daquele módulo, como demonstra a Figura 3.7. Os 4 módulos são referentes ao:

- *Volume*: Volume do sinal, variando de mudo a total (0 a 1);
- *Pan*: Variação estereofônica do sinal, variando da esquerda a direita até o centro (0 a 1) ou vice versa (0 a -1);
- *Pitch*: Altura da nota, podendo variar até 4 oitavas;
- *FM Mod*: Quantidade de modulação funcionando de forma semelhante ao parâmetro na primeira região, desta vez com variação dinâmica;

Figura 3.7: Selecionador de módulos de *ADSR* e *LFO* do oscilador seguidos por ambos, com foco no parâmetro volume.

A *ADSR* possui 5 parâmetros, 4 para a determinação da forma da envoltória (*Attack*, *Decay*, *Sustain* e *Release*), e o quinto para a quantidade de *ADSR* a ser aplicada a onda (*Amount*). Um valor 0 de *Amount* indica que nenhuma envoltória é criada para o parâmetro, e é valor padrão de inicialização do *Pan*, *Pitch* e *FM Mod*. A implementação utiliza-se do *opcode madsr*.

O *LFO* possui 5 parâmetros também. O *Attack* determina o período em que o processo de *LFO* varia de 0 até o valor determinado pela intensidade. *Speed* indica a velocidade e *Intensity* a intensidade deste, amplitude da onda utilizada no *LFO*. *Type* define o tipo de onda que o *LFO* faz sobre o parâmetro do módulo, possuindo 4 tipos, seno, triângulo, serra e quadrado. Intensidades negativas invertem a onda escolhida em *Type*. O *LFO* é executado se estiver ativo, o que é realizado a partir do botão *On/Off* a direita. O *opcode lfo* é utilizado para implementar essa operação.

Como foi mencionado, um instrumento "Oscillator" é instanciado três vezes a partir de chamadas do instrumento "Master". Para cada chamada, é passado ao oscilador os parâmetros *MIDI* recebidos, a nota e a *velocity* dela, seguidos do índice a ser utilizado na indexação da matriz que contêm os parâmetros a serem gerados pelo oscilador e na indexação do vetor de nomes dos canais a serem acessados por aquela instância. A execução se baseia inicialmente no acesso aos parâmetros definidos no *Cabbage* seguidos do cálculo e armazenamento nas matrizes, a serem utilizados pelo instrumento "Master" na síntese.

A geração do sinal a partir dos parâmetros calculados no instrumento "*Oscilattor*" é feita utilizando o *opcode poscil3*, com o uso de tabelas de onda com tamanho igual a 8192 amostras. Como há 3 instâncias do instrumento "*Oscillator*", há 3 chamadas do *opcode* com os seus respectivos parâmetros, que geram os sinais a serem utilizados na modulação de frequência.

3.4 Filtro

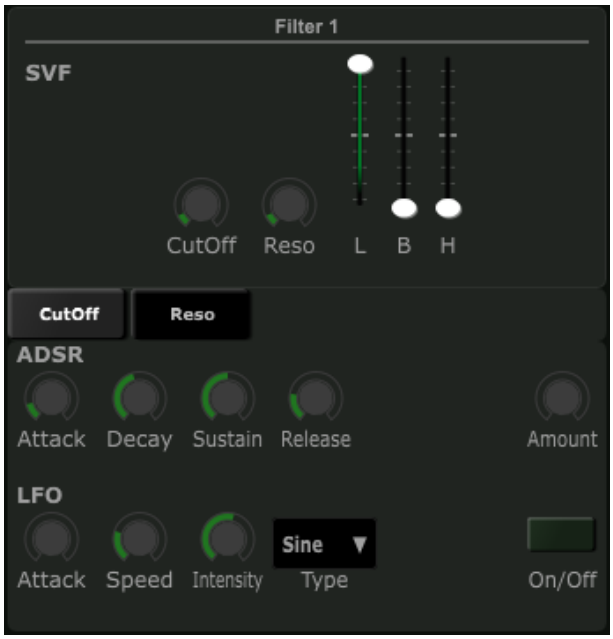

Figura 3.8: Visão com foco no módulo do filtro.

O filtro possui uma estrutura de subdivisões semelhante a do oscilador, como demonstra a Figura 3.8. Possui 3 regiões, sendo a primeira voltada aos elementos principais e a segunda e terceira voltadas a *ADSR* e *LFO*. O filtro implementado é o *SVF* (*State Variable Filter*), devido a fornecer a filtragem de sinal para os três tipos de filtros simultaneamente, passa-baixa, passa-banda e passa-alta. Foi utilizado para isso o *opcode* "*svfilter*", que retorna os três sinais filtrados.

Os níves de cada um dos filtros são definido pelos 3 *sliders* verticais, que foram definidos com o comando *vslider* no *Cabbage*, indicando o volume do sinal filtrado por aquele tipo de filtro. Respectivamente temos o slider "L" para passa-baixa, "B" para passa banda e "H" para passa alta. Para tais filtros, é possível se configurar a frequência de corte e ressonância base nos *sliders* circulares "*CutOff*" e "*Reso*".

A segunda e terceira regiões, voltadas a *ADSR* e *LFO*, se comportam de forma equivalente à descrita no oscilador. A diferença está nos parâmetros aqui associados, sendo eles a frequência de corte e a ressonância, podendo variar de acordo com as configurações definidas no executar de uma nota.

O filtro opera nos osciladores após a aplicação de FM, como demonstra o diagrama da Figura 3.5. O nível de entrada do sinal dos osciladores no filtro e o nível da saída do

filtro são definidos no *Output Map*, a partir dos *sliders* referentes a eles. Essa parte será melhor descrita na Seção 3.5.

3.5 Mapa de *FM*, Filtragem e Saída de sinal

Figura 3.9: Mapa de FM, filtragem e saída.

A região mais a direita do sintetizador, demonstrada na Figura 3.9, contém a área destinada a 3 funções:

- Matriz de Modulação de Frequência para os 3 osciladores;
- Mix de Filtragem dos sinais pós modulação;
- Pan e Volume das 4 saídas (3 osciladores e filtro);

A matriz de modulação de frequência serve para que cada oscilador possa modular e receber modulação de todos os 3 osciladores (o que inclui a si mesmo). Isso envolve 9 relações, com a presença de 9 *sliders* de controle de cada valor da modulação, dispostos abaixo do texto "*Oscillators FM/OUT*" até o texto "*Filters FM IN/OUT*".

Inicialmente 3 osciladores são gerados a partir do instrumento "*Master*". Em sequência 3 novos osciladores, as ondas carregadoras, são gerados e recebem a aplicação da modulação de frequência a partir dos 3 osciladores gerados anteriormente, as ondas moduladoras, levando em consideração os níveis de cada um dos 9 *sliders* de intensidade. A Figura 3.10 demonstra o fluxo de geração e modulação das ondas carregadoras e moduladoras.

Os números 1, 2 e 3 dispostos verticalmente a esquerda na Figura 3.9 representam o número do oscilador em questão, que recebe a modulação dos osciladores 1, 2 e 3 de

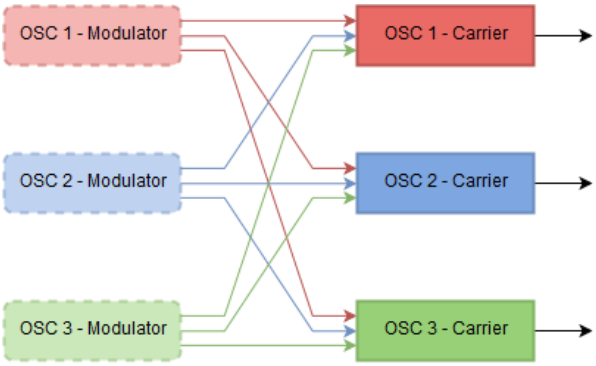

Figura 3.10: Geração das ondas moduladoras e das carregadoras, que recebem a modulação logo em sequência.

acordo com os valores dos *sliders* a direita de seu número, e abaixo dos números dispostos horizontalmente.

Usando a própria Figura de exemplo, temos a situação em que o oscilador 1 recebe modulação do oscilador 2 com máxima intensidade, e não recebe modulação dos osciladores 1 e 3. O oscilador 2 recebe modulação do 1 com cerca de 1/3 de intensidade, nada do 2 e intensidade máxima do 3. O oscilador 3 não recebe modulação de nenhum dos 3 osciladores.

É importante destacar que como são gerados novos osciladores para receberem a modulação de outros gerados anteriormente, essa modulação não é cumulativa. A modulação que, por exemplo, o oscilador 1 recebe do 2, não interfere quando o 2 recebe do 1.

A filtragem então ocorre nos osciladores pós modulação. Os níveis dos sinais que recebem a filtragem são determinados pelos *sliders* sobre os números 1, 2 e 3, localizados abaixo do texto "*Filters FM IN/OUT*". Usando a mesma Figura 3.9, temos que somente o oscilador 2 pós modulação está sendo inserido no filtro, com intensidade máxima.

Finalmente temos os controles de *pan* e volume, a direita dos 3 osciladores e do filtro. No caso da Figura temos volume máximo para o oscilador 1 com *pan* centralizado e metade do volume do filtro, com pan total à direita.

3.6 Efeitos

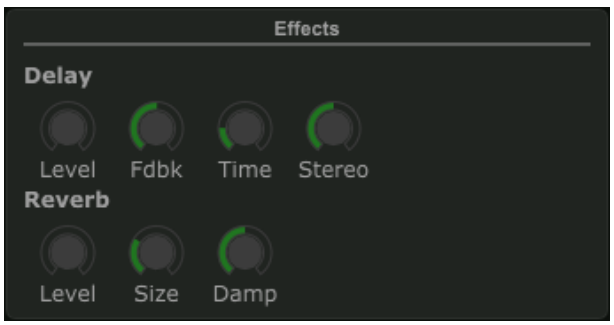

Figura 3.11: Visão com foco no módulo de efeitos.

O sexto e último módulo, demonstrado na Figura 3.11, possui dois efeitos globais, que operam sobre o sinal após toda a modulação e filtragem, o *Delay* e a Reverberação. Os 4 sliders superiores são referentes ao *Delay*, podendo configurar respectivamente o volume do efeito, *feedback*, tempo desse atraso e atraso diferenciado para os dois canais. Da mesma forma, os 3 inferiores são referentes a reverberação, controlando o volume do efeito, o "tamanho" do ambiente e o "abafamento" deste.

Foram utilizados os *opcodes delay* e *reverbsc* para toda a geração desses efeitos. A geração desse sinal ocorre em série, isto é, um efeito opera sobre o resultado de outro, com a execução do delay primeiro e reverberação em sequência.

3.7 Testes

Esta seção trata do uso do sintetizador *SaWerkraut* em um *DAW* escolhido. Isso envolve testes relativos a capacidade do sintetizador em termos da síntese proposta e a compatibilidade dele com o *host*. Ambos os testes foram feitos principalmente no processo de composição de uma música demonstrativa, que utiliza-se de timbres e sonorizações criados pelo sintetizador desenvolvido e sequenciamento de notas e configurações inseridas no *DAW*. A música demonstrativa se encontra no site "`https://bitbucket.org/hermanfma/sawerkraut`", junto aos *presets* exportados dos instrumentos gerados via o sintetizador e o código fonte do *SaWerkraut*.

A música contém diversas instâncias do *SaWerkraut*, trabalhando com a combinação das características dele na síntese de alguns tipos de sonoridades. Existem sonoridades semelhantes a instrumentos percussivos tonais, outras próximas a instrumentos de sopro como flautas e oboés, e alguns exemplos de sons sintéticos não baseados em instrumentos reais. Alguns desses instrumentos possuem seus parâmetros atualizados durante a execução da música, o que permite maior dinamicidade do som gerado.

O *DAW* escolhido para esses testes foi o *FL Studio* versão 12.1.2 64*bits* [12]. Algumas configurações iniciais foram necessárias para que o sintetizador operasse corretamente nesse *DAW* e obtivesse ganho na performance.

3.7.1 Configurações no *FL Studio*

Inicialmente, o *plugin* do instrumento deve ser inserido e carregado no *FL Studio* da mesma forma que *plugins VSTs* são carregados no programa [13]. O *plugin*, após inserido no diretório de instrumentos *VST*, deve ser adicionado via o *Plugin Manager*.

Em relação as configurações no *FL Studio*, na aba de configurações do instrumento *VST*, foi necessário ativar o *checkbox* "*Use fixed sized buffers*". Essa configuração serve para resolver o problema de, em alguns casos, notas não receberem o evento de *note-off* quando deveriam, não terminando sua execução. O *FL Studio* troca dados com *plugins* a partir de blocos de dados, os *buffers*, de tamanhos variáveis, padrão do *VST*, e essa configuração faz com que sejam utilizados blocos de tamanho fixo [14], que no caso do *Csound*/*Cabbage*, resolve tal problema, apesar de não ser recomendado.

Quando um projeto é salvo no *FL Studio*, o estado das configurações de todos os instrumentos e efeitos carregados é armazenado em conjunto no projeto, que são recuperados no momento em que tal projeto é aberto novamente. O mesmo ocorre com o sintetizador desenvolvido, porém todos os canais dos elementos da interface que, na inicialização

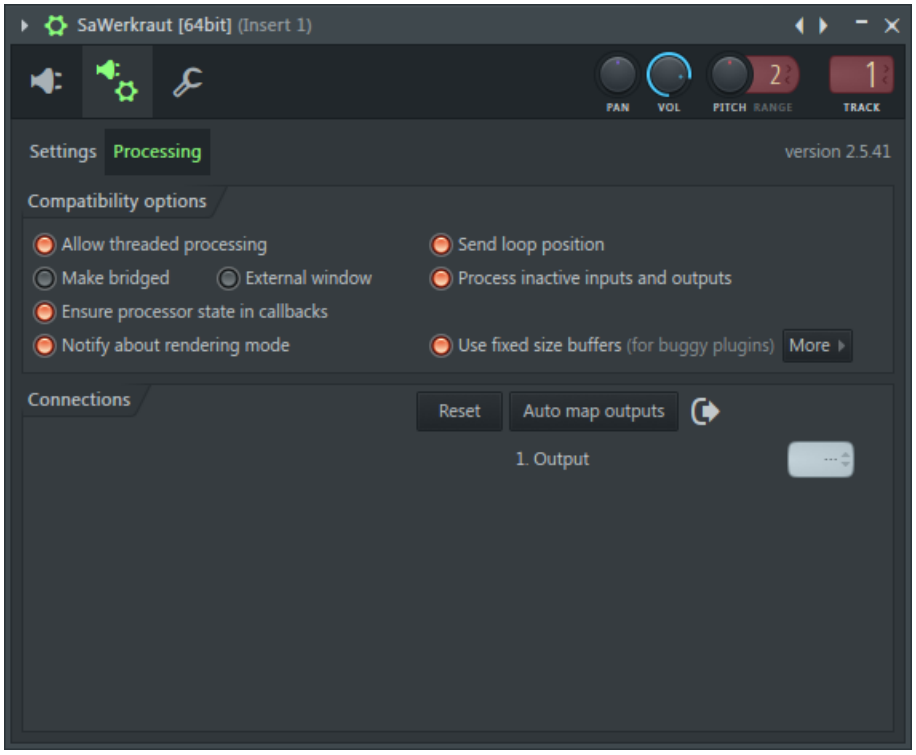

Figura 3.12: Configurações no *Wrapper* de *VST* no *FL Studio*.

do sintetizador via o identificador "*range()*" do *Cabbage* possuíam valor inicial diferente de 0 e foram modificados para algum valor, retornam para esse valor inicial quando o projeto é aberto novamente. Isso não ocorre nos casos dos canais dos elementos que são inicializados em 0. Tal problema é grave no fluxo de trabalho de um usuário e precisou de uma medida auxiliar, que envolve exportar o estado das configurações do instrumento e reimportá-los novamente depois que o projeto foi recarregado. Essas configurações são exportadas no formato "*.fxp*", que é um formato desenvolvido pela *Steinberg* voltado a armazenar um conjunto de configurações (*preset*) de um *plugin VST*, que podem ser recarregadas novamente [36]. O *FL Studio* provê tal funcionalidade na aba de configurações do instrumento. Com a atualização da ferramenta *Cabbage* da versão 1.0 para a versão 1.1, tal problema foi corrigido.

É possível também salvar e carregar essas configurações via o uso dos *presets* implementados no sintetizador, mas eles só funcionam corretamente no modo *standalone*, a partir da execução no *Cabbage*. Tais *presets* ficam localizados junto ao *plugin* exportado no diretório "*SaWerkraut presets*", no formato *xml*.

Em relação a performance, ainda nessa aba, os *checkboxes* "*Allow threaded processing*" e "*Notify about rendering mode*" foram ativadas. O primeiro item serve para garantir que o sintetizador utilize as *threads* do sistema de forma a melhorar a distribuição de processamento entre os núcleos do processador. O segundo faz com que o instrumento consiga distinguir entre síntese em tempo real e síntese para exportação, em que no segundo caso ela não sofre com os problemas de limitação de performance em tempo real. A Figura 3.12 ilustra o estado das configurações.

3.7.2 Configurações no *SaWerkraut*

A performance e responsividade do sintetizador estão diretamente ligadas a uma variável do *Csound*, o *ksmps*. O *ksmps* determina o tamanho do bloco de controle que o *Csound* opera, sendo que quanto menor esse valor, mais responsivo o instrumento fica porém mais custoso é o processamento, e o inverso para valores mais altos. Testes foram feitos em cima desse valor, a fim de se tentar equilibrar essas duas qualidades.

Foram criadas então duas versões desse mesmo sintetizador, uma com o valor *ksmps* de 15 e outra de 60, ambas com apenas essa variação. Nesse caso a versão com valor de 15 requer maior uso de processamento para a síntese, mas possui maior responsividade da interface e de trocas de valores dos parâmetros em tempo real enquanto notas estão ativas (automações). O segundo caso é mais leve, porém as trocas de valores em notas ativas ocorrem a uma taxa alta, o que gera saltos bruscos de valores intermediários em, por exemplo, uma variação linear de 0 a 1 de um parâmetro como o volume de uma onda no período de 1 segundo. No caso da variação do *pitch bend*, que ocorre via *MIDI* e não via parâmetros criados via *Cabbage*, esses saltos bruscos não ocorrem mesmo no uso de *ksmps* igual a 60, o que demonstra que existe uma diferença nas taxas de atualizações dos dois casos. As duas versões se denominam respectivamente "*SaWerkraut*" e "*SaWerkraut_ 60*". O uso do identificador "*guirefresh()*", voltado a modificar a taxa de atualização da interface gráfica e dos canais do *Cabbage*, somente demonstrou resultados no modo *standalone*, não fazendo diferença em relação a execução no *DAW*. Os *presets* salvos no formato *fxp* em uma versão do instrumento pode ser reimportado na outra versão, já que ambos compartilham os mesmos canais com os mesmos nomes. Porém, ambos os *plugins* tem o *PluginID* diferentes, a versão "*SaWerkraut*" sendo "sK15" e a é versão "*SaWerkraut_ 60*" sendo "sK60". O arquivo é binário, então um editor desse tipo é necessário para se alterar esses valores, que estão localizados perto do início do arquivo.

O sintetizador demonstrou uma estabilidade razoável e uma síntese interessante dentro do seu escopo, porém a performance está mais alta do que o esperado. Alguns testes exploratórios foram feitos para tentar descobrir a fonte ou fontes de tal uso de processamento. Tais testes envolveram:

- Modificar valores do *ksmps*;
- Desabilitar módulos;
- Desabilitar *triggers* de execução;
- Utilizar geradores de sinal mais leves, sem interpolação;
- Utilizar tabelas de ondas até 4 vezes menor que a atual.

Nos testes exploratórios, os elementos que geraram o maior ganho de performance foram os 3 primeiros da lista, que levaram a inserção de elementos configuráveis pelo usuário no sintetizador a fim de que ele tenha um controle em algum nível sobre a performance. Em relação ao caso do *ksmps*, foi então criado as duas versões do instrumento. Houve pouca variação no quarto e quinto, o que indica que o gargalo não se encontra na execução dos *opcodes* de síntese.

Capítulo 4

Conclusão

A linguagem *Csound* existe há um certo tempo, mas ela ainda é uma linguagem robusta e poderosa atualmente no meio da síntese de sinal digital. Combinada com a ferramenta *Cabbage*, ela passa a ser utilizável ainda com a presença de interface gráfica pelos demais *DAWs* existentes. *Cabbage* é uma ferramenta recente e está em contínuo desenvolvimento, com a possibilidade de no futuro serem adicionados novos elementos de interface ou novas funcionalidades. A estabilidade da ferramenta tanto no processo de implementação e teste quanto no *plugin* gerado ainda não está ideal, mas em futuras atualizações é esperado melhorias significativas. O fórum destinado a ferramenta está constantemente ativo e é aberto para qualquer usuário tirar dúvidas, demonstrar seus trabalhos e se comunicar com outros usuários.

Em relação ao sintetizador *SaWerkraut* desenvolvido, é possível a geração de sonoridades variadas, que músicos e *sound designers* possam utilizar em seus projetos voltados a diversos tipos de mídias, como jogos eletrônicos ou cinema. A presença principalmente do mapa de *FM* entre os osciladores, em conjunto com os controles de modulação com a possibilidade de serem modificados em tempo real, adicionam uma diversidade de configurações na geração de sons complexos.

O código fonte do *SaWerkraut*, junto ao *plugin dll* exportado estarão localizados no domínio "`https://bitbucket.org/hermanfma/sawerkraut`", junto a *presets* de alguns instrumentos gerados para ele, nos formatos *fxp* e nos formatos *snaps* no diretório "SaWerkraut presets".

4.1 Trabalhos Futuros

Devido a construção do *plugin* ter sido baseada em módulos independentes que se relacionam, o *SaWerkraut* é expansivo, podendo ser atualizado com novas modalidades. A seguir, é apresentada uma lista de módulos adicionais e possíveis melhorias para o instrumento:

- Modo monofônico com portamento configurável (*slide*);
- Adição de um efeito geral composto pela geração de ondas levemente diferenciadas em fase, altura, intensidade, efeito semelhante a um *Chorus*;
- Utilização do BPM do *host* em parâmetros baseados em tempo (*ADSR*, *LFO*, *delay*) tornando-os relativos ao andamento da música;

- Adição de novas formas de onda aos osciladores, além de um modo aditivo no próprio oscilador com um número significativo de parciais;
- Adição de síntese por linhas de retardo (pluck) nos osciladores;
- Adição de novos tipos de filtros e do uso de filtros em série;
- Adição de mais módulos de osciladores e filtros.

Existe também a possibilidade de se aplicar os conceitos adquiridos nesse trabalho em novos projetos similares, como o desenvolvimento de novos instrumentos ou efeitos a partir do *Csound/Cabbage*. Com o progresso de atualização da ferramenta *Cabbage*, os projetos desenvolvidos serão atualizados afim de se compilar versões mais recentes e estáveis dos instrumentos desenvolvidos, além de utilizar novos elementos de interface e comandos adicionados. Testes de compatibilidade da ferramenta em outros *DAWs* também são fatores a se considerar.

Referências

[1] MIDI Manufacturers Association. Learn About MIDI. http://www.midi.org/aboutmidi/index.php. [Online; acessado 05-Dezembro-2014]. 17

[2] Plugin Boutique. Interface types explained (VST, RTAS, AU, etc.). http://support.pluginboutique.com/knowledgebase/articles/51119-interface-types-explained-vst-rtas-au-etc. [Online; acessado 05-Dezembro-2014]. 1, 18

[3] Phil Burk. *Music and Computers: A Theoretical and Historical Approach: Course Guide*. Key College Pub., 2005. v, 9, 10

[4] Cabbage. Cabbage url. www.cabbageaudio.com. [Online; acessado 15-Abril-2016]. 22

[5] Andrés Cabrera. An overview of csound variable types. *Csound Journal*, Issue 10, January 2009. http://csoundjournal.com/issue10/CsoundRates.html. 20

[6] John Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society*, pages J. Audio Eng. Soc. 21 (7), 526–534., 1973. 16

[7] Csound. Csound url. www.csounds.com. [Online; acessado 15-Abril-2016]. 1, 19

[8] Vintage Synth Explorer. Native Instruments FM7. http://www.vintagesynth.com/misc/fm7.php. [Online; acessado 02-Junho-2015]. 1

[9] Vintage Synth Explorer. Yamaha DX7. http://www.vintagesynth.com/yamaha/dx7.php. [Online; acessado 02-Junho-2015]. 1, 13

[10] Michael Gogins. Michael Gogins's tumblr web page. http://michaelgogins.tumblr.com/CsoundVST. [Online; acessado 05-Dezembro-2014]. 1, 2, 21

[11] David Miles Huber. *The MIDI manual: a practical guide to MIDI in the project studio*. Taylor & Francis, 2007. 17

[12] Image-Line. FL Studio. https://www.image-line.com/flstudio/. [Online; acessado 22-Junho-2016]. 41

[13] Image-Line. FL Studio - Installing Plugins. https://www.image-line.com/support/FLHelp/html/basics_externalplugins.htm. [Online; acessado 24-Junho-2016]. 41

[14] Image-Line. FL Studio Fruity Wrapper. https://www.image-line.com/support/FLHelp/html/plugins/wrapper_2_processing.htm. [Online; acessado 20-Junho-2016]. 41

[15] Image-Line. Sytrus. https://www.image-line.com/support/FLHelp/html/plugins/Sytrus.htm. [Online; acessado 01-Junho-2016]. 29

[16] Derek Johson and Debbie Poyser. Steinberg cubase vst. *Sound on Sound*, July 1996. http://www.soundonsound.com/sos/1996_articles/jul96/steinbergcubase3.html. 1, 18

[17] Chris Larsen. Tremolo VST Plugin, 2003. http://www.euphoriaaudio.com/tutorials/tremolo/EA_Tremolo_paper.pdf. 18

[18] Victor Lazzarini and Rory Walsh. Developing ladspa plugins with csound. In *Proceedings of Linux Audio Conference*, pages 60–63. Citeseer, 2007. 21

[19] Iain Mccurdy. Csound Examples. http://iainmccurdy.org/csound.html. [Online; acessado 12-Junho-2016]. 23, 29

[20] Dennis Miller, Brian Smithers, and Geary Yelton. Sequencing on a shoestring. *Electronic Musician*, September 2005. http://www.emusician.com/gear/1332/sequencing-on-a-shoestring/35453. 1, 16

[21] Jussi Pekonen and Vesa Välimäki. The brief history of virtual analog synthesis. In *Proc. 6th Forum Acusticum, Aalborg, Denmark*, pages 461–466, 2011. 1

[22] Miller Puckette. Max at seventeen. *Computer Music Journal*, 26(4):31–43, 2002. 18

[23] randomgoogleproof. C++ VST Plugin Tutorials. http://learnvst.wordpress.com/. [Online; acessado 05-Dezembro-2014]. 18

[24] B.K. Shepard. *Refining Sound: A Practical Guide to Synthesis and Synthesizers*. OUP USA, 2013. 13

[25] Julius O. Smith. *Spectral Audio Signal Processing*. http://ccrma.stanford.edu/~jos/sasp/, 2015. [Online; acessado 1-Julho-2015]. 11

[26] Steinberg. 3rd Party Developer. http://www.steinberg.net/en/company/developers.html. [Online; acessado 29-Junho-2015]. 18

[27] Barry L. Vercoe. *The Canonical Csound Manual*. MIT Media Lab, 2013. 1, 19, 21

[28] Martin Walker. Steinberg cubase vst 3.7. *Sound on Sound*, September 1999. http://www.soundonsound.com/sos/1996_articles/jul96/steinbergcubase3.html. 17

[29] Rory Walsh. *Cabbage documentation*. http://cabbageaudio.com/docs/introduction/. 1, 22, 23

[30] Rory Walsh. Cabbage Forum. http://forum.cabbageaudio.com/. [Online; acessado 12-Junho-2016]. 29

[31] Rory Walsh. Developing Audio Plugins with Cabbage and Csound. http://lac.linuxaudio.org/2011/download/rw_pligin_dev_with_cabbage_and_csound.pdf. [Online; acessado 05-Dezembro-2014]. 2

[32] Rory Walsh. Cabbage, a new gui framework for csound. In *Proceedings of the Linux Audio Conference KHM Cologne, Germany*, 2008. 21

[33] Rory Walsh. Audio plugin development with cabbage. In *Proceedings of the Linux Audio Conference, Maynooth, Ireland*, pages 47–53, 2011. 22

[34] Wikipedia. Fourier Series. https://en.wikipedia.org/wiki/Fourier_series. [Online; acessado 29-Junho-2015]. 7

[35] Wikipedia. Nyquist Shannon sampling theorem. https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem. [Online; acessado 22-Junho-2015]. 6

[36] Wikipedia. Virtual Studio Technology. http://en.wikipedia.org/wiki/Virtual_Studio_Technology. [Online; acessado 05-Dezembro-2014]. 17, 42

[37] Yoemun Yun and Si-Ho Cha. Designing virtual instruments for computer music. *International Journal of Multimedia and Ubiquitous Engineering*, 8(5):173–178, 2013. 1, 16