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

Uso de Processamento Digital de Áudio na Implementação de Efeitos em Instrumentos Musicais

André Wagner França

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

Orientador
Prof. Dr. Márcio da Costa Pereira Brandão

Brasília
2015

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

Coordenador: Prof. Dr. Homero Luiz Piccolo

Banca examinadora composta por:

Prof. Dr. Márcio da Costa Pereira Brandão (Orientador) — CIC/UnB
Prof. Dr. Alexandre Zaghetto — CIC/UnB
Prof. Dr. Camilo Chang Dórea — CIC/UnB

CIP — Catalogação Internacional na Publicação

França, André Wagner.

Uso de Processamento Digital de Áudio na Implementação de Efeitos em Instrumentos Musicais / André Wagner França. Brasília : UnB, 2015.
139 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2015.

1. áudio, 2. efeito, 3. digital, 4. som, 5. processamento, 6. sinal,
7. Csound

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

Uso de Processamento Digital de Áudio na Implementação de Efeitos em Instrumentos Musicais

André Wagner França

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

Prof. Dr. Márcio da Costa Pereira Brandão (Orientador)
CIC/UnB

Prof. Dr. Alexandre Zaghetto
CIC/UnB

Prof. Dr. Camilo Chang Dórea
CIC/UnB

Prof. Dr. Homero Luiz Piccolo
Coordenador do Bacharelado em Ciência da Computação

Brasília, 8 de Julho de 2015

Dedicatória

À minha família e amigos, pelo incentivo e apoio em todas as etapas do curso, e em especial à Simone, minha namorada, que me apoia a cada instante e sempre me traz motivação e força para continuar seguindo em frente.

Agradecimentos

Agradeço a todos que participaram do meu curso, direta ou indiretamente.

Primeiramente, a Deus, fonte de toda a minha alegria e força.

A todos os professores que se esforçaram para transmitir o conhecimento e a instrução.

Aos meus colegas e amigos, pelas ajudas nos trabalhos e estudos, pelas viradas de noites, e por caminharem junto comigo todos esses anos.

À minha família e à minha namorada, Simone, por sempre me motivarem, me confortarem, e apoiarem, em todas as situações.

Resumo

Este trabalho tem por objetivo apresentar ao leitor alguns dos principais efeitos de áudio presentes no mercado musical. Além de informações relacionadas ao surgimento e evolução do mundo dos efeitos de áudio, como os primeiros efeitos, a transição do domínio analógico para o domínio digital, será abordada a teoria por trás do processamento do sinal dentro de cada efeito, bem como a estruturação e uma implementação na linguagem Csound de cada um dos efeitos apresentados. Por fim, interliga-se todas as implementações de efeitos de forma a criar-se um sistema de processamento multi-efeitos em CSound.

Palavras-chave: áudio, efeito, digital, som, processamento, sinal, Csound

Abstract

This work has the objective of presenting to the reader some of the main audio processing effects in the musical industry. Along with the information about the creation and the evolution of the audio effects, like the first effects, the transition from analog to digital, the theory behind the signal processing within each effect will be explained, as well as the implementation of each presented effect in the Csound programming language. Lastly, all the effect implementation are linked to create a Csound multi-effects processing system.

Keywords: audio, effect, digital, sound, processing, signal, Csound

Sumário

Lista de Figuras

Capítulo 1

Introdução

O processamento de sinais digitais de áudio nos remete ao inicio da era digital, onde os sinais eram gravados para serem transmitidos através de rádios. Com o avanço da tecnologia, surgiram muitas outras áreas dentro do âmbito do processamento de sinais. Dentre estas áreas, pode-se destacar a área de efeitos de áudio.

De acordo com [15], efeitos digitais de áudio são caixas ou ferramentas de software cujas entradas são sinais de áudio ou sons que serão modificados por parâmetros de controle de som e cujas saídas são sons ou sinais.

Efeitos de áudio podem ser produzido tanto naturalmente quanto artificialmente. Cantores podem aplicar efeitos de vibrato e tremolo em suas vozes, da mesma forma que estes e outros efeitos podem ser aplicados em instrumentos musicais (como, por exemplo, um violinista que movimenta o dedo que pressiona a corda levemente para criar o efeito de vibrato).

Efeitos digitais, entretanto, são gerados artificialmente por uma variedade de circuitos e dispositivos de processamento de sinais, e são produzidos com o uso de técnicas de processamento digital de sinais [7].

Com o surgimento e crescimento da indústria musical e com o avanço da tecnologia, o processamento digital de sinais encontra-se cada vez mais presente na sociedade, sendo uma área que merece um estudo aprofundado para a devida compreensão de todas as suas propriedades e características.

Este fato pode ser comprovado com o crescimento da pesquisa e dos estudos sobre o assunto no meio acadêmico. No começo dos anos 80, o estudo do processamento digital de sinais era realizado apenas em cursos de pós-graduação em engenharia elétrica, e, num período de 10 anos, tornou-se parte convencional em cursos de graduação. Nos dias de hoje, o processamento digital de sinais é um conhecimento básico necessário a cientistas e engenheiros [13].

Pensando no futuro, é claro ver que o papel do processamento digital de sinais está crescendo aceleradamente, em parte pela convergência das comunicações, computadores e processamento de sinais, tanto na área do consumidor, quanto na indústria avançada e até mesmo em aplicações governamentais [9].

O foco deste estudo, entretando é na aplicação musical do processamento de áudio, sobretudo na aplicação de efeitos a sinais de áudio. Existe uma enorme variedade de efeitos de áudio que, por sua vez, podem ser caracterizados de diferentes formas. Uma

das formas de se caracterizar é pela natureza do efeito. Desta forma, podemos dividir em efeitos de distorção, de dinâmica, de modulação, filtros, dentre outros.

Grande parte dos efeitos de áudio pode ser criada tanto de maneira analógica quanto digital. Dependendo da qualidade da implementação e dos componentes utilizados, algumas simulações digitais de efeitos podem soar tão reais quanto o seu correspondente analógico.

Neste estudo serão implementados uma série de efeitos digitais com o uso da linguagem de programação Csound, sobretudo aqueles que costumam ser usados em guitarras, como distorções, *delays*, *flangers*, *chorus*, dentre outros.

O Csound permite que se trabalhe com entrada e saída de sinal de diversas maneiras. Da mesma forma que permite entrada e saída de som em tempo real, também é possível ler e escrever os sinais de entrada e saída de arquivos, bem como gerar os sinais de áudio a partir de funções no próprio código.

O código implementado permite que se trabalhe com entrada e saída em tempo real, conectando-se um instrumento na entrada de áudio do sistema e ouvindo o som processado pelos alto-falantes ou por fones de ouvido. Para fins de demonstração, entretanto, utiliza-se uma sequência de *plucks*, instruções Csound que sintetizam um sinal de áudio semelhante ao de uma corda pinçada.

Todas as implementações dos efeitos tratados neste estudo foram realizadas unicamente a partir dos circuitos descritos nos capítulos seguintes. Desta forma, o leitor tem acesso a um banco de efeitos de áudio implementados em Csound com a sua devida documentação.

Os capítulos que seguem trarão mais detalhes acerca dos efeitos de processamento de áudio. O capítulo 2 trará uma breve revisão sobre a teoria de processamento de sinais digitais e sobre a linguagem Csound; O capítulo 3 trará um breve histórico sobre o surgimento e evolução dos efeitos de áudio, abordando os efeitos de domínio analógico e digital, e, em seguida, trará as características dos efeitos tratados no estudo; O capítulo 4, por sua vez, trará uma análise mais detalhada da implementação de cada um dos efeitos; Por fim, o capítulo 5 tratará dos resultados obtidos pela implementação.

Capítulo 2

Revisão Teórica

2.1 Processamento Digital de Sinais

Neste capítulo serão apresentados alguns princípios básicos de processamento de sinais digitais, dando ao leitor uma base teórica para entender os conceitos e estruturas que serão apresentados nos capítulos subsequentes.

O termo sinal é geralmente aplicado à algo que carrega informação. Sinais geralmente carregam informação sobre o estado ou comportamento de um sistema físico, e, muitas vezes, sinais são sintetizados com o propósito de comunicar informações entre humanos ou entre humanos e máquinas [9].

Sinais normalmente são representados através de funções de uma ou mais variáveis independentes. A variável independente da representação matemática de um sinal pode ser contínua ou discreta. Grande parte dos sinais estudados tem como a variável independente o tempo. Sinais contínuos onde o tempo é a variável independente são ditos sinais contínuos no tempo e são muitas vezes ditos sinais analógicos. Por outro lado, sinais discretos no tempo são sinais onde a variável independente apresenta valores discretos, normalmente representados como uma sequência de números. Sinais ditos digitais são aqueles que são discretos tanto na amplitude quanto no tempo.

A classificação de sistemas de processamento de sinais é feita da mesma forma que os sinais. Sistemas ditos contínuos no tempo são sistemas em que a entrada e a saída são contínuas no tempo. Da mesma forma, sistemas discretos no tempo são sistemas onde a entrada e a saída são sinais discretos no tempo e sistemas digitais são aqueles onde a entrada e a saída são sinais digitais.

Sinais discretos no tempo são representados matematicamente com uma sequência de números x, onde o $n$ésimo número da sequência é denotado por $x[n]$. Dizemos que o valor da amostra $x[n]$ não está definido quando n não é um valor inteiro.

Podemos ter realizar diversas operações entre sinais. Definimos que a soma de dois sinais $x[n]$ e $y[n]$ é a soma amostra a amostra, enquanto o produto dos dois sinais é o produto amostra a amostra. A multiplicação de um sinal por um escalar a é a multiplicação de cada amostra do sinal pelo escalar a. Dizemos que uma sequência $y[n]$ é uma versão atrasada de $x[n]$ se temos que:

$$y[n] = x[n - n_0]$$

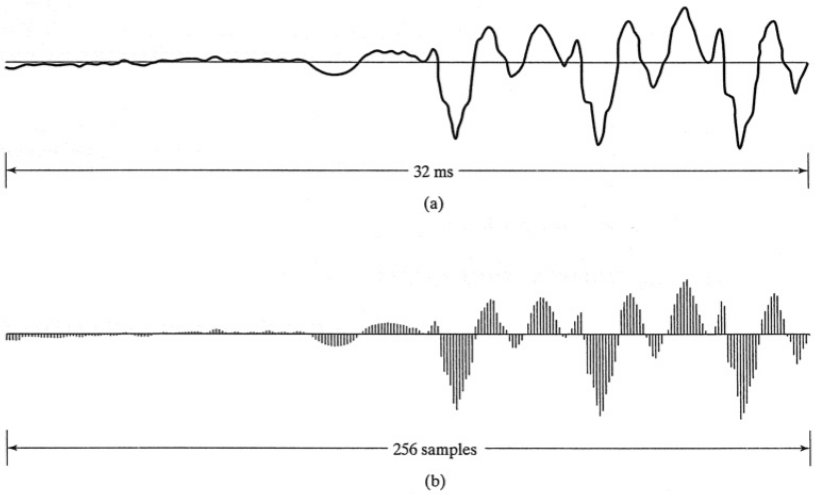

Figura 2.1: Sinal analógico e digital. [9]

2.1.1 Funções Úteis para Processamento de Sinais

Algumas funções básicas são utilizadas com frequência na análise teórica de processamento de sinais. Dentre elas, podemos citar a função de amostra unitária, ou a função de impulso unitário. A função de amostra unitária está para sinais discretos no tempo assim como a função Delta de Dirac está para sinais e sistemas contínuos no tempo. A função impulso é definida por:

$$\delta[n] = \begin{cases} 0, & n \neq 0 \\ 1, & n = 0 \end{cases}$$

Podemos dizer que, de forma geral, qualquer sequência pode ser descrita como um somatório de impulsos atrasados e multiplicados por escalares. Ou, escrevendo matematicamente:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot \delta[n-k];$$

Outra função bastante utilizada em processamento de sinais é a de degrau unitário, $u[n]$, definida por:

$$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

Podemos relacionar a função degrau com a função de impulso unitário da seguinte forma: a função degrau seria o somatório dos impulsos unitários deslocados a partir do zero. Escrevendo matemáticamente, temos:

$$u[n] = \sum_{k=-\infty}^{n} \delta[k];$$

Dentre as funções importantes na área de processamento de sinais, podemos citar também a função exponencial. Uma forma generalizada de se escrever uma sequência exponencial é:

$$x[n] = A\alpha^n;$$

Dizemos que a sequência numérica é real se A e α forem reais. No caso de A ser positivo e de $0 < \alpha < 1$, então os valores da sequência são positivos e vão decrescendo à medida que n cresce. Para um valor negativo de A, porém mantendo $0 < \alpha < 1$, temos uma sequência que alterna o sinal, mas diminiu em magnitude. Por fim, nos casos onde $\alpha \geq 1$, temos uma sequência que cresce juntamente com o n;

Um outro tipo de função muito comum em processamento de sinais é a função senoidal. Funções senoidais são caracterizadas por sua proporcionalidade com a função seno. Uma sequência senoidal pode ser descrita da seguinte forma:

$$x[n] = A \cdot \sin(\omega_0 n + \phi), \ \textit{para todo } n;$$

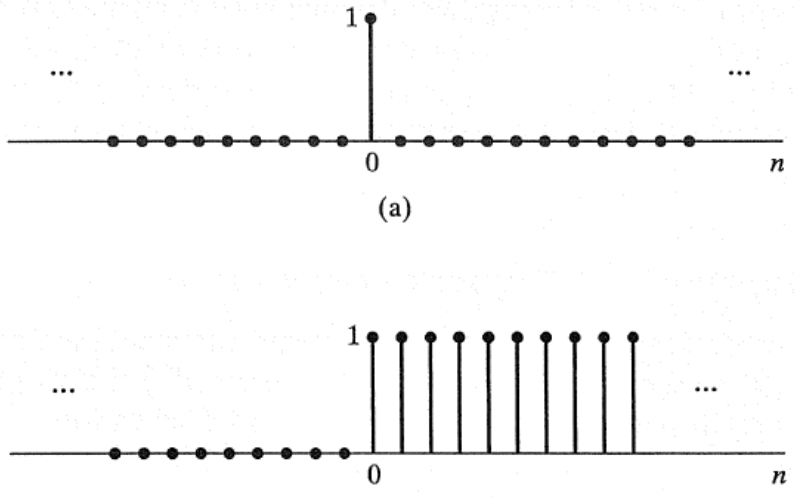

Figura 2.2: Sequência de Impulso e de Degrau. [9]

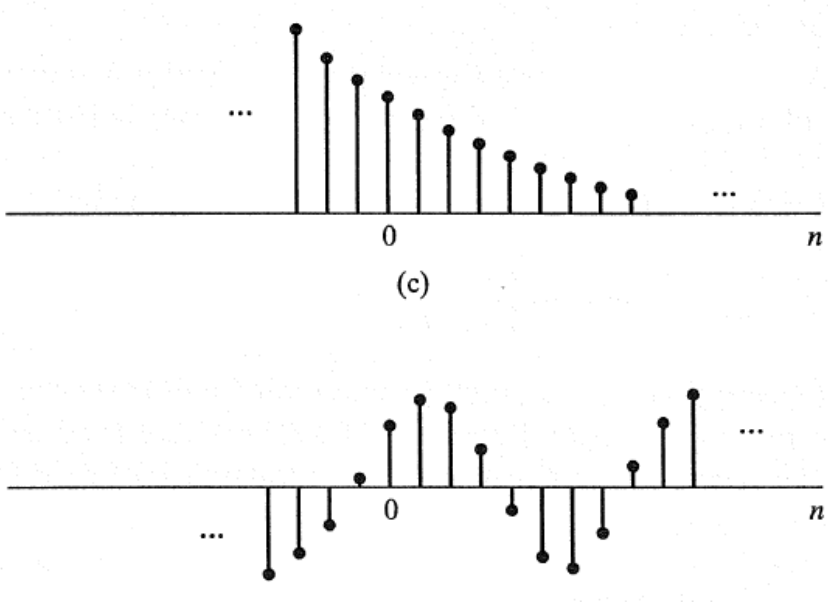

Figura 2.3: Sequência Exponencial Real e Senoidal. [9]

2.1.2 Sistemas

Definimos um sistema discreto no tempo como sendo uma transformação ou operação matemática que mapeia uma sequência de valores de entrada $x[n]$ a uma sequência de valores de saída $y[n]$. Os valores da saída $y[n]$ podem depender dos valores de $x[n]$ para qualquer n. Escrevemos da seguinte forma:

$$y[n] = T(x[n]);$$

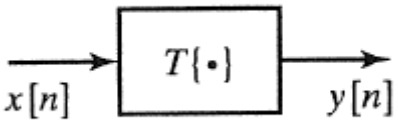

Figura 2.4: Um sistema discreto no tempo pode ser representado conforme a imagem acima . [9]

Podemos citar como exemplo um sistema de delay simples. Um sistema de delay simples é definido pela equação:

$$y[n] = x[n - n_d];$$

onde n_d é um inteiro fixo chamado atraso do sistema, é o número de amostras do sinal que serão atrasadas. O sistema de delay ideal é aquele que simplesmente desloca a sequência de entrada por n_d amostras, correspondendo a um atraso no tempo. No caso de um valor negativo para n_d, teríamos um deslocamento para a direita, o que resultaria num adiantamento no tempo.

Definimos como sistemas sem memória aqueles sistemas em que o valor de $y[n]$ depende apenas do valor de $x[n]$ para o mesmo valor de n. Um exemplo é o sistema abaixo.

$$y[n] = (x[n])^2;$$

Existe também a classe dos sistemas lineares. Sejam $y_1[n]$ e $y_2[n]$ as respostas de um sistema quando as entradas são $x_1[n]$ e $x_2[n]$, respectivamente. Dizemos que o sistema é *linear* se

$$T(x_1[n] + x_2[n]) = T(x_1[n]) + T(x_2[n]) = y_1[n] + y_2[n]$$
$$e\ T(ax[n]) = aT(X[n]) = ay[n]$$

onde a é uma constante arbitrária.

Outra classe de sistemas é a classe de sistemas invariantes no tempo. Dizemos que um sistema é *invariante no tempo* se um atraso ou adiantamento na entrada gera um atraso ou adiantamento correspondente na saída do sistema.

Definimos também um sistema como causal se, para cada escolha de n_0, a sequência de saída no índice n depende apenas dos valores de entrada de $n \leq n_0$. Podemos dizer que sistemas assim são não antecipativos.

O conceito de estabilidade também é aplicado à sistemas digitais. Dizemos que um sistema é *estável* se todas as suas entradas limitas produzem saídas limitadas. Dizemos que a entrada é limitada se existe um valor positivo finito B_x tal que:

$$|x[n]| \leq B_x < \infty,\ para\ todo\ n;$$

Então, temos um sistema estável se, para cada entrada limitada, existe um inteiro positivo fixo B_y tal que:

$$|y[n]| \leq B_y < \infty,\ para\ todo\ n;$$

2.1.3 Sistemas Lineares Invariantes no Tempo

Uma classe de sistemas muito importante para processamento de sinais é a classe dos sistemas lineares invariantes no tempo. Como o próprio nome já diz, tais sistemas são lineares e invariantes no tempo. No caso de o sistema ser linear, temos que

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]T(\delta[n-k]) = \sum_{k=-\infty}^{\infty} x[k]h_k[n]$$

No caso de o sistema ser invariante no tempo, temos que, se $h[n]$ for uma resposta a uma entrada $\delta[n]$, então a resposta de $\delta[n-k]$ é $h[n-k]$. Com isso, podemos reescrever a equação acima, para sistemas lineares invariantes no tempo, como sendo:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Um sistema linear invariante do tempo é completamente caracterizado por sua resposta ao impulso $h[n]$. Isso significa que, dado o $h[n]$, é possível computar a saída $y[n]$ para qualquer que seja a entrada $x[n]$.

Uma operação importante a ser definida quando se trata de sistemas lineares invariantes no tempo é a de soma de convolução, representada pela equação acima apresentada, ou, escrita de outra forma:

$$y[n] = x[n] * h[n];$$

A convolução em tempos discretos produz uma sequência $y[n]$ a partir das sequências $x[n]$ e $h[n]$. Podemos ressaltar algumas das propriedades da soma de convolução, dentre elas, a comutatividade:

$$x[n] * h[n] = h[n] * x[n]$$

e a distributividade sobre a adição:

$$x[n] * (h_1[n] + h_2[n]) = x[n] * h_1[n] + x[n] * h_2[n]);$$

A partir das características apresentadas acima, podemos chegar a algumas conclusões sobre sistemas lineares invariantes no tempo. Conectar tais sistemas em série, isto é, fazendo com que a saída de um seja a entrada do sistema seguinte, é equivalente a ter um único sistema linear invariante no tempo cuja resposta ao impulso é a convolução das respostas ao impulso dos sistemas pelo qual ele é formado. Seja $h_1[n]$ a resposta ao impulso de um sistema e $h_2[n]$ a resposta ao impulso de um outro sistema. Associando estes dois sistemas em série, temos que a resposta ao impulso $h[n]$ será:

$$h[n] = h_1[n] * h_2[n]$$

Como consequência da propriedade de comutatividade da convolução, podemos cascatear os sistemas em qualquer ordem.

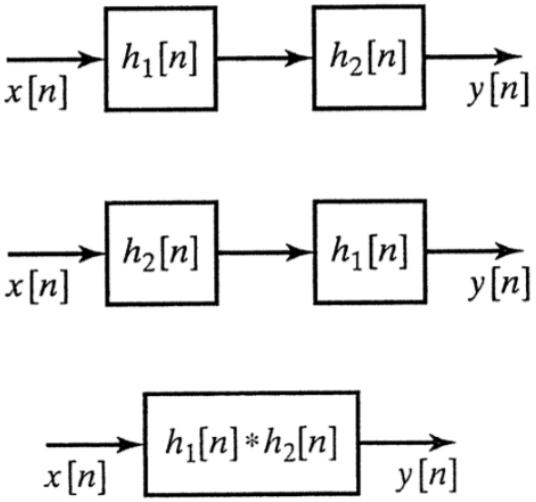

Figura 2.5: Sistemas lineares invariantes no tempo associados em série. [9]

Associar sistemas lineares invariantes no tempo em paralelo faz com que a resposta ao impulso do sistema resultante seja igual à soma das respostas ao impulso dos sistemas associados, isto é:

$$h[n] = h_1[n] + h_2[n]$$

Podemos dizer que um sistema linear invariante no tempo será estável se e somente se sua resposta ao impulso for somável em módulo, isto é:

$$S = \sum_{k=-\infty}^{\infty} |h[k]| < \infty$$

Sistemas com entradas limitadas que produzam $S = \infty$ são, portanto, sistemas não estáveis.

Para a causalidade de sistemas lineares invariantes no tempo, temos a seguinte propriedade:

$$h[n] = 0, n < 0;$$

Definimos dois sistemas $h_1[n]$ e $h_2[n]$ como *sistemas inversos* se a saída de um dos sistemas aplicada à entrada do outro gera a sequência de entrada do primeiro sistema. Sistemas inversos são úteis em várias situações, onde pode ser necessário compensar os efeitos de algum sistema linear.

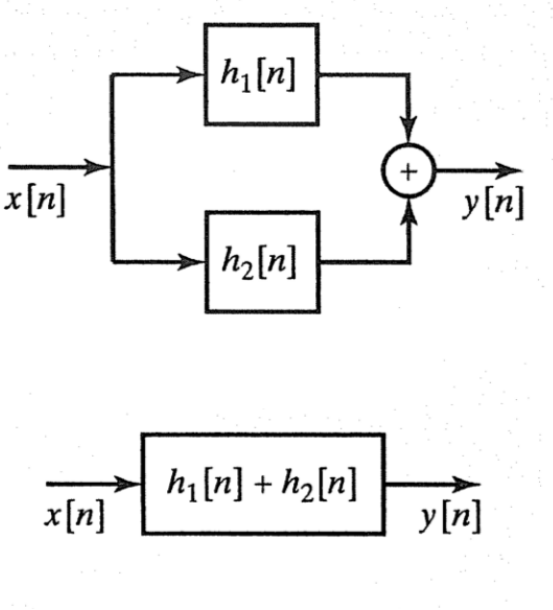

Figura 2.6: Sistemas lineares invariantes no tempo associados em paralelo. [9]

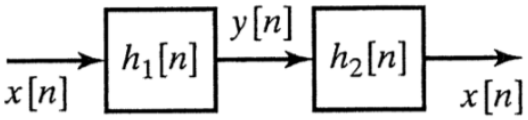

Figura 2.7: Sistemas inversos. [9]

2.1.4 Representação no Domínio da Frequência

O conceito de resposta em frequência para sistemas lineares invariantes no tempo é basicamente o mesmo para sistemas contínuos e discretos no tempo. A diferença está que, para sistemas lineares invariantes no tempo, a resposta em frequência sempre é uma função periódica em ω com período em 2π. Por exemplo, se considerarmos $x[n]$ como a exponencial complexa $e^{j\omega}n$. Substituindo ω por $\omega + 2\pi$, teremos:

$$H(e^{j(\omega+2\pi)}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j(\omega+2\pi)n}$$

Considerando que, para valores inteiros de n, temos que $e^{\pm j2\pi n} = 1$, temos que:

$$e^{-j(\omega+2\pi)n} = e^{-j\omega n}e^{-j2\pi n} = e^{-j\omega n}$$

Então, temos que:

$$H(e^{j(\omega+2\pi r)}) = H(e^{j\omega}),\ para\ r\ inteiro$$

Com isso, temos definida a periodicidade da resposta em frequência para sistemas lineares invariantes no tempo. Desta forma, só é necessário especificar $H(e^{j\omega})$ para intervalos $0 \leq \omega \leq 2\pi$ ou $-\pi \leq \omega \leq \pi$. Neste intervalo, temos que as *frequências baixas* são aquelas mais próximas de 0, e que as *frequências altas* são aquelas mais próximas de $\pm\pi$. Para o caso genérico, as *frequências baixas* são aquelas próximas aos múltiplos pares de π, enquanto as *frequências altas* são aquelas próximas aos múltiplos ímpares de π.

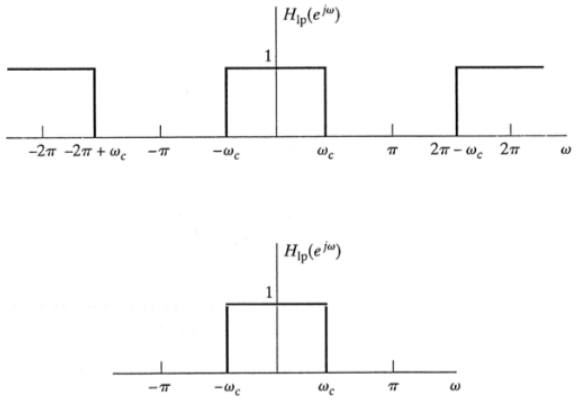

Figura 2.8: Um filtro passa-baixas ideal. Na primeira imagem, mostrando a periodicidade do filtro. Na segunda, apenas um período da resposta em frequência. [9]

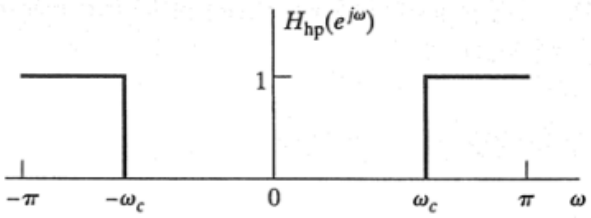

Figura 2.9: Um filtro passa-altas ideal, sendo exibido apenas um período da resposta em frequência. [9]

Uma das formas de representar sequências no domínio da frequência é através da Série de Fourier. Quando tratamos de sinais contínuos, todos os sinais periódicos em um período T podem ser representados como uma soma de senos e cossenos (exponenciais complexas) harmonicamente relacionados. $x[n] = x[n+N]$ é um sinal discreto periódico, $e^{jk\omega_0 n}$ é uma exponencial complexa discreta e $\omega_0 = \frac{2\pi}{N}$ é a frequência fundamental.

Como temos apenas N exponenciais complexas distintas em um período N, podemos representar o sinal $x[n]$ periódico em N através da soma de N exponenciais complexas.

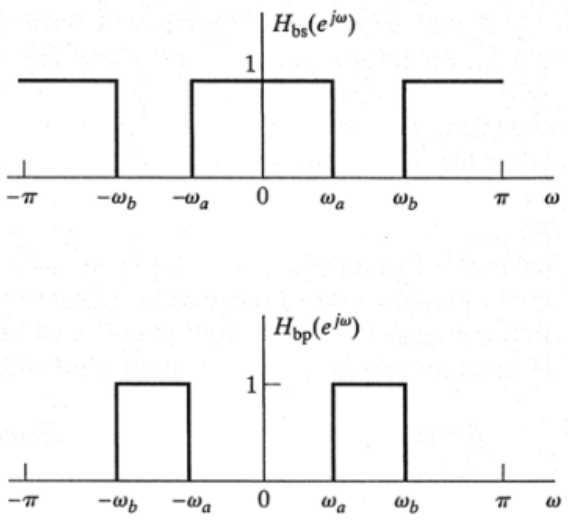

Figura 2.10: Um filtro rejeita-banda ideal, que rejeita frequências na faixa de $-\omega_a \leq \omega \leq \omega_b$ e deixa passar todas as outras frequências. Abaixo deste, um filtro passa-banda ideal, que passa as frequências na faixa de $-\omega_a \leq \omega \leq \omega_b$ e rejeita as outras. Na imagem, temos exibido apenas um período da resposta em frequência. [9]

$$x[n] = \sum_{k=k_0}^{k_0+N-1} a_k \cdot e^{jk\omega_0 n}, \omega_0 = \frac{2\pi}{N}$$

Os a_k serãos os coeficientes da série de Fourier, e são dados por:

$$a_k = \frac{1}{N} \cdot \sum_{n=\langle N \rangle} x[n] \cdot e^{-jk\omega_0 n};$$

Como a série de Fourier é periódica em N, temos apenas N coeficientes distintos, e, portanto, $a_k = a_{k+N}$

Quando tratamos de sinais discretos, utilizamos a DTFT, uma variação da transformada de Fourier para sinais discretos.

Com manipulações matemáticas, podemos obter a seguinte fórmula para a DTFT:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n] \cdot e^{-j\omega n}$$

$$e\ a_k = \frac{1}{N} \cdot X(e^{jk\omega_0}), \omega_0 = \frac{2\pi}{N}$$

Os coeficientes da série de Fourier do sinal $x_p[n]$ podem ser vistos como uma amostragem da transformada de Fourier em $k \cdot \omega_0$ do sinal x[n].

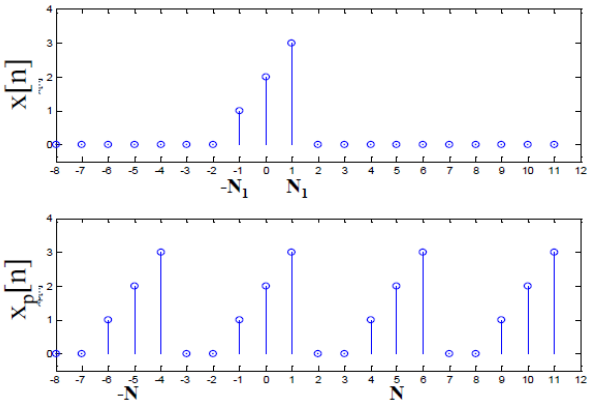

Figura 2.11: x[n] é um sinal discreto não periódico, e $x_p[n]$ é o seu sinal periódico associado com período N

2.1.5 Filtros Digitais

Podemos definir um filtro digital como sendo um sistema que deixa passar alguns componentes de frequência e modifica outros. Filtros digitais se comportar de 4 formas básicas: passa-baixas, passa-altas, passa-bandas e rejeita-bandas. Como os nomes indicam, os filtros passa-baixas deixam passar frequências abaixo de uma dada frequência de corte, atenuando ou até mesmo anulando as frequências maiores. Os filtros passa-alta, por outro lado, deixam passar as frequências acima da frequência de corte, atenuando ou anulando as inferiores. Os filtros passa-bandas deixam passar as frequências entre duas frequências de corte informadas, rejeitando todas as outras frequências fora deste intervalo. Os filtros rejeita-banda, por sua vez, tem comportamento inverso aos passa-banda, rejeitando todas as frequências entre as frequências de corte.

Existem várias formas de se implementar filtros digitais, cada qual com características distintas. Para o desenvolvimento de alguns dos efeitos que serão apresentados no decorrer do estudo será utilizado um tipo de filtro chamado Butterworth. Os filtros Butterworth são desenvolvidos de forma a ter uma resposta em frequência o mais plana o possível na banda passante, de forma a evitar as ondulações (*ripples*) presentes em outros tipos de filtros (Figura 2.12). O filtro Butterworth também se caracteriza pela resposta em frequência se aproximar a zero na banda rejeitada. Uma outra característica dos filtros Butterworth é o fato de que o formato do filtro se mantém para ordens mais elevadas, conforme pode ser visto na figura 2.13.

A magnitude da resposta em frequência de um filtro passa-baixas Butterworth pode ser dada pela seguinte equação:

$$G_n(\omega) = |H_n(j\omega)| = \frac{1}{\sqrt{1 + (\frac{\omega}{\omega_c})^{2n}}} \tag{2.1}$$

onde:

G é o ganho do filtro

H é a função de transferência

$\boldsymbol{j}$ é o número imaginário

$\mathbf{n}$ é a ordem do filtro

ω é a frequência angular do sinal, em radianos/segundo

ω_c é a frequência de corte

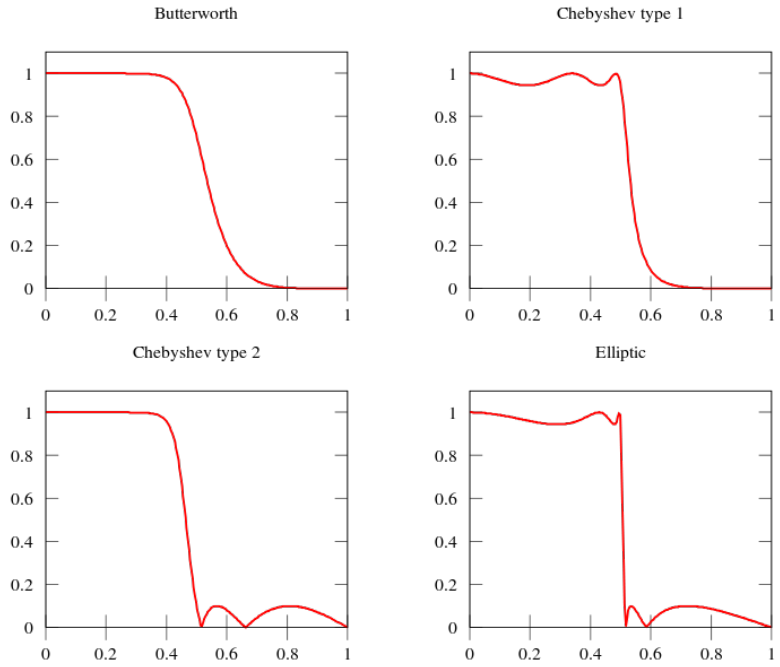

Figura 2.12: Comparação entre diferentes tipos de filtros. O filtro Butterworth apresenta um caimento mais lento ao se atingir a frequência de corte, porém não apresenta as ondulações presentes nos filtros Chebyshev e elíptico.

Com os conceitos apresentados, o leitor possui conhecimento suficiente para compreender um pouco sobre o processamento de sinais digitais. Vale ressaltar que esta área ainda tem muitos campos de pesquisa e que a bibliografia ainda apresenta muitos outros conceitos acerca do assunto. Na seção seguinte, será dada uma breve introdução sobre o Csound, a linguagem utilizada para o desenvolvimento dos efeitos no estudo.

2.2 A Linguagem Csound

Csound é uma das mais antigas e conhecidas linguagem de programação no campo da síntese e do processamento de áudio. Foi criada na década de 80 por Barry Vorcoe no Instituto de Tecnologia de Massachussetts (MIT), e tem crescido muito desde então.

Um arquivo Csound é composto de três seções: opções, orquestra e *score*. A seção de opções contém configurações para a execução do arquivo. A seção de orquestra é onde são definidos os instrumentos, que são as unidades que executam ações. Estes

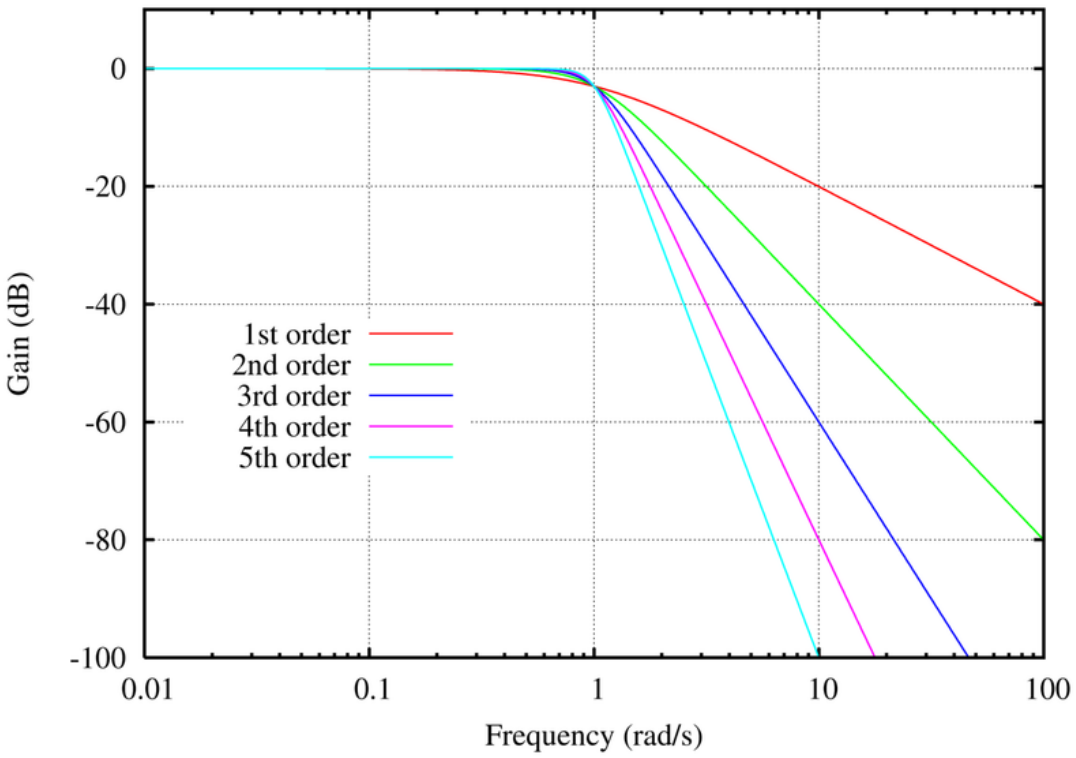

Figura 2.13: Filtros passa-baixas Butterworth, de ordens 1 a 5. Pode-se notar que o formato do filtro se mantém com a variação da ordem, variando apenas o decaimento com a ordem do filtro.

instrumentos são chamados, pela seção *score*, que vai dizer quando e como os instrumentos serão executados.

Cada uma das seções do arquivo Csound é iniciada com uma tag *<nomeDaSeção>* e encerrada com a tag *</nomeDaSeção>*. O arquivo Csound é iniciado com a tag *<CsoundSynthesizer>* e encerrado com *</CsoundSynthesizer>*. Segue abaixo um exemplo de estruturação de um arquivo Csound.

```
<CsoundSynthesizer>

<CsoundOptions>
  ;Csound configuration settings
</CsoundOptions>

<CsInstruments>
  ;instruments definition

  instr 1
    ;here goes the definition of instrument 1
    ;...
  endin

  instr 2
```

```
16      ;here goes the definition of instrument 2
17      ;...
18    endin
19  </CsInstruments>
20
21  <CsScore>
22    ;score definition
23  </CsScore>
24
25  </CsoundSynthesizer>
```

Um instrumento Csound é composto de instruções e é cercado pelas palavras-chave *instr* e *endin*. Os instrumentos são identificados por um número. As instruções dos instrumentos são chamados Opcodes, que são os principais blocos construtores do Csound, e são responsáveis por produzir sinais oscilatórios, filtrar sinais, executar funções matemáticas, entre outras funcionalidades. Dependendo da sua funcionalidade, opcodes podem necessitar de variáveis para tratar *inputs* e *outputs*. Um instrumento Csound segue este padrão:

```
1  instr X
2
3    output OPCODE input1, input2, input3, ..., inputN
4
5  endin
```

Um instrumento pode ter vários opcodes dentro de si, e eles serão executados em sequência enquanto o instrumento estiver ligado.

Assim como diversas linguagens de programação, o Csound trabalha com variáveis. As variáveis do Csound podem ser de diversos tipos, mas são três os mais comuns: As variáveis que trabalham com sinais de áudio tem nome iniciado com a letra a; As variáveis que trabalham com sinais de controle tem nome iniciado com a letra k, e são atualizadas com frequência menor do que as variáveis de áudio; Por fim, as variáveis que são inicializadas juntamente com a chamada do instrumento começam com a letra i. Uma forma interessante de se enxergar as variáveis do Csound é a de vê-las como cabos, fazendo as ligações nos inputs e outputs dos opcodes.

Os eventos do *score*, entretanto, são linhas de texto individuais que vão dizer em que instante e por quanto tempo um instrumento será ligado. Após estas informações, podem ou não ser informados parâmetros opcionais para os instrumentos, conforme pode ser verificado no exemplo abaixo:

```
1
2  <CsScore>
3  i 1 0 10
4  i 2 0 3 7.5 9
5
6  </CsScore>
```

No código descrito acima, o instrumento 1 é iniciado no instante 0 e tocado por 10 segundos, e o instrumento 2 é iniciado no instante 0, tocado por 3 segundos e são passados como parâmetros os valores 7.5 e 9, para serem tratados dentro do instrumento.

Alguns aspectos mais aprofundados do Csound, como a forma que os efeitos de áudio serão tratados e criados, serão descritos posteriormente, no capítulo de implementação dos efeitos.

Capítulo 3

História e Características de Efeitos de Áudio

Efeitos de áudio são utilizados pela humanidade há muito tempo. Seja para ser melhor ouvido, ou mesmo para passar diferentes tipos de sensações, os seres humanos já aplicavam efeitos em sons muito antes de se definir formalmente o que seria um efeito de áudio.

3.1 Histórico

Neste capítulo serão destacadas a história por trás dos efeitos de processamento de áudio moderno, seu surgimento, evolução e as principais mudanças trazidas por cada novo sistema de efeitos para o mercado de processamento de áudio.

3.1.1 Surgimento e Evolução dos Efeitos de Áudio

As guitarras elétricas surgiram em 1931, com o intuito de fazer com que o som de violões aparecesse mais nos contexto das *Big Bands* de Jazz. Junto com o surgimento das guitarras elétricas, surgiu uma busca crescente por modificar e variar seu som. Com isso, pode-se dizer que surgiram os primeiros efeitos. Os efeitos até então eram criados de forma puramente mecânica, e pode-se citar como exemplo o guitarrista Duanne Eddy, que, com o uso de um tanque de água vazio, criou um efeito de câmara de eco artificial para gravação, e da guitarra *Spanish Vibrola* da Rickenbacker (Figura 3.1), que apresentava roldanas móveis internas que moviam a ponte da guitarra para criar o efeito de vibrato.

Com o avanço da tecnologia, tornou-se possível a criação de efeitos cada vez mais complexos e sofisticados. No período de surgimento do rádio, os transmissores apresentavam muitos problemas devido à variação de níveis de áudio e ao grande número de transmissões ao vivo sendo feitas simultaneamente [11]. Tais problemas seriam resolvidos com a aplicação de limitadores picos e com outros sistemas de efeitos.

Pode-se dizer que o "processamento de áudio moderno" começou com o trabalho da equipe de design dos Laboratórios CBS no começo dos anos 60, com o Audimax e o Volumax, um compressor que eliminava ruído que os antigos compressores não conseguiam, e um aparelho que eliminava picos do sinal. O Audimax e o Volumax (Figura 3.2) foram os primeiros sistemas de efeito de áudio que utilizaram um processamento do sinal de áudio,

Figura 3.1: Perry Botkin com uma guitarra Rickenbacker Spanish Vibrola, em 1938.

tornando-se pioneiros de uma área que, em alguns anos, seria de extrema importância para a sociedade.

Figura 3.2: Volumax 4450A e Audimax, dos Laboratórios CBS, equipamentos utilizados para estabilização e melhora do sinal de rádio no período do seu surgimento

Os avanços da tecnologia possibilitaram a criação de circuitos elétricos cada vez menores e mais estáveis, fazendo com que a criação de sistemas de efeitos de áudio compactos pudessem ser criados e aprimorados. Em 1978, a Boss, uma das maiores marcas de pedais de efeitos do mundo, lançou seu primeiro pedal compacto, o OD-1 (Figuras 3.3 3.4), um pedal que trazia um efeito de *overdrive*, criando a partir de então um mercado de equipamento musical que perdura até os dias de hoje.

Figura 3.3: Pedal Boss OD-1, o primeiro pedal de efeito compacto, criado em 1978

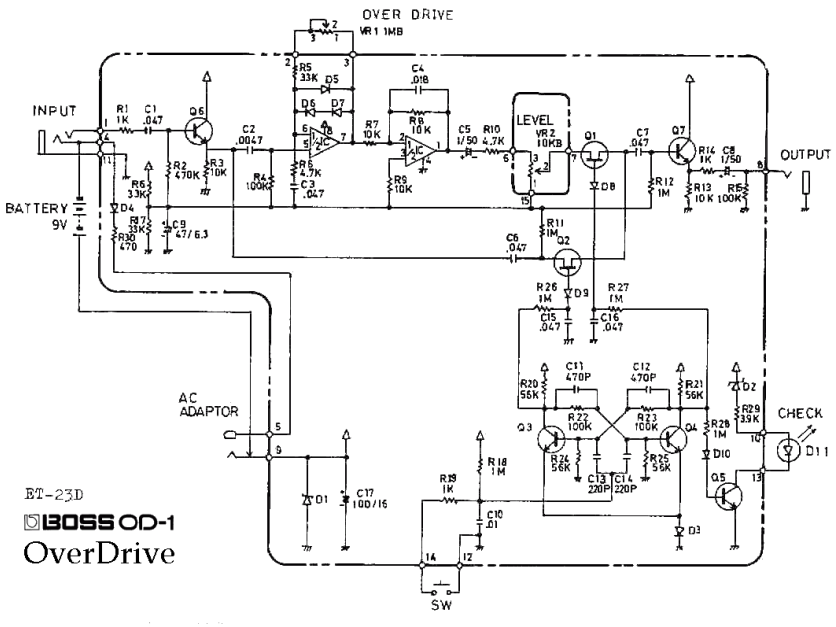

Figura 3.4: Esquema do cirucito elétrico do pedal OD-1, da Boss. O efeito de distorção produzido pelo pedal se dá de forma completamente analógica, através dos transistores, resistores, capacitores e das outras estruturas presentes no circuito.

Com o avanço da tecnologia na área da eletrônica, tornou-se possível a criação de sistemas digitais que processem sinais de áudio. Estações de Áudio Digitais (*Digital Audio Workstations*, ou DAWs) tiveram suas primeiras implementações nas décadas de 70 e 80. Entretanto, as limitações tecnológicas da época, como a baixa velocidade de processamento e de gravação de disco e o alto custo de armazenamento de dados tornaram as implementações extremamente ineficientes.

Com o avanço da tecnologia, no final da década de 80, um número relativamente grande de computadores pessoais já tinha capacidade para realizar edição de áudio digital. Em 1991, quando a *AVID Audio* lançou o *Pro Tools*, muitas grandes gravadoras migraram para o mundo digital, gerando uma verdadeira revolução no mercado musical. Até os dias de hoje, o *Pro Tools* é utilizado para gravação, edição e mixagem de aúdio em estúdios

profissionais (Figura 3.5).

Figura 3.5: Interface da Estação de Áudio Digital ProTools, em uma de suas versões mais recentes.O Pro Tools é uma ferramenta de processamento e mixagem de áudio profissional, utilizada por diversas gravadoras espalhados pelo mundo.

Em 1996, a Steinberg lançou o software *Cubase*, que permitia a gravação e reprodução de até 32 trilhas de áudio sem a necessidade de nenhum hardware externo de processamento de sinais. O *Cubase* modelava não apenas a interface de gravação e edição, mas também a mesa de mixagem e a bancada de efeitos presentes em estúdios. Com essas funcionalidades e com um preço muito abaixo dos softwares concorrentes, a Steinberg revolucionou o mercado de DAWs.

Com o tempo, sistemas de efeitos digitais portáteis foram surgindo e se tornando mais acessíveis, de forma que muitas pessoas, ao terem que escolher que tipo de sistema usar, optarem por sistemas digitais, devido à sua facilidade de uso, praticidade e preço. Com isto, o mercado de sistemas portáteis de efeitos digitais têm crescido muito.

3.1.2 Efeitos de Áudio na Atualidade

Com a popularização dos computadores, tem-se tornado cada vez mais comum o uso de softwares para processamento de áudio e aplicação de efeitos, não apenas para gravadoras e grandes organizações, mas também para pessoas comuns. Através de programas como DAWs e VSTs, *Virtual Studio Technology*, muitas pessoas têm tido a oportunidade de trabalhar com processamento de áudio a partir de seus computadores e estúdios caseiros.

Apesar da infinidade de softwares de processamento de áudio presentes no mercado, ainda existe uma busca muito grande por pedais de efeito e por sistemas de gravação e processamento, tanto digitais como analógicos. Isto se dá pelos diferentes usos de cada um deles.

DAWs são largamente utilizados em estúdios de gravação, tanto profissionais quanto caseiros, pois possibilitam o trabalho com múltiplas faixas simultâneamente, permitindo controle e mixagem de forma simples, rápida e eficiente.

Sistemas de processamentos de efeitos, como pedais e outros sistemas, são mais utilizados por artistas em si, de forma de que a escolha entre um sistema de efeitos digital ou analógico varia mais pelo gosto e necessidade do artista. Com a tecnologia atual é possível atingir uma qualidade sonora consideravelmente alta com efeitos digitais, de forma que a maioria das pessoas não consegue identificar diferenças, mesmo com questões que limitam a qualidade, como a taxa de amostragem e quantização do sinal. Sistemas digitais têm a vantagem de poder concentrar uma variedade grande de efeitos em apenas um sistema, em que se pode controlar uma grande quantidade de parâmetros.

Entretanto, muitos artistas ainda preferem o uso de pedais e sistemas de efeitos analógicos devido à maior pureza no timbre, que algumas vezes é possivel de ser identificada. Existem alguns efeitos analógicos que apresentam uma certa variação sonora baseada em variações na dinâmica de como o instrumento é tocado, ou em algumas variações nos componentes elétricos que compõem o circuito do efeito, como o aquecimento de transistores e válvulas.

3.2 Principais Características de Efeitos de Áudio

Nesta seção serão abordados alguns dos principais efeitos de áudio presentes no mercado, tratando de suas principais características audíveis e uma breve explicação de seu funcionamento. No capítulo seguinte serão abordadas as implementações dos efeitos descritos nesta seção.

3.2.1 Delays

O Delay é um efeito muito comumente utilizado com guitarras e consta na repetição do sinal de entrada com um certo atraso somado ao sinal de entrada original. Delays são a aplicação sonora mais simples do uso de linhas de retardo. Existem diversos tipos de delays, cada qual com suas características específicas. Neste trabalho, serão apresentadas três variações: O delay simples, aquele em que o sinal de entrada é repetido apenas uma vez; o delay infinito, onde o sinal de entrada é repetido infinitas vezes; e, por fim, o delay múltiplo, aquele em que o sinal de entrada é repetido por um número n de vezes.

Delay Simples

O delay simples é a forma mais básica de se aplicar o efeito de delay. O sinal de entrada, ao passar pela linha de retardo, é atrasado e então atenuado. Por fim, o sinal atrasado e atenuado é somado ao sinal de entrada.

O resultado audível da aplicação do efeito de delay depende do intervalo de tempo entre o sinal original e o sinal atrasado. Para intervalos de tempo menores do que 10 ms, o efeito de delay funciona como um filtro passa baixa de resposta finita, eliminando assim uma certa gama de frequências, que varia com o tempo de retardo. Para intervalos de tempo um pouco maiores, variando entre 10 e 50 ms, é gerado um efeito audível de ambientação, criando uma ilusão de preenchimento e de dobramento. Já para intervalos de tempo de mais de 50 ms, os sons produzidos pelo efeitos são ouvidos como repetições do som original, como ecos discretos [1].

No caso de um sistema de delay simples, podemos expressar o sinal de saída $y[n]$ em função do sinal de entrada $x[n]$ da seguinte forma:

$$y[n] = x[n] + \alpha * x[n - N]$$

Delay Infinito

O delay infinito é uma outra aplicação de linhas de retardo. Sua implementação é muito semelhante à implementação do delay simples, onde a única alteração é no sinal que alimenta a linha de retardo. No caso do delay simples, a linha de retardo era alimentada pelo sinal de entrada. Para a implementação do delay infinito, a linha de retardo deve ser alimentada pelo sinal de entrada somado com o sinal atrasado e atenuado.

As repetições do delay infinito são atenuadas com decaimento baseado no valor da atenuação. Eventualmente, a repetição do sinal terá uma amplitude tão baixa que deixará de ser audível. No caso de uma atenuação muito próxima de 1, as repetições não terão decaimento de amplitude aparente, fazendo com que o sinal de entrada se acumule na saída de forma que se tenha um excesso de volume e ruído no sinal.

Delay Múltiplo

O efeito de Delay Múltiplo, ou delay de múltiplos taps é uma outra aplicação das linhas de retardo. Com uma estrutura um pouco mais complexa do que o delay simples e o delay infinito, esta implementação nos permite um número fixo de repetições, definido pelo usuário. Desta forma, pode-se dizer que o efeito de delay múltiplo é uma variação do delay infinito, onde as repetições são cortadas após a Nésima repetição.

3.2.2 Vibrato

O efeito de vibrato é extremamente comum no mundo musical, e é aplicável a diversos instrumentos de diversas formas. É caracterizado pela variação da altura da nota de forma pulsante. O uso do efeito de vibrato produz uma percepção psicoacústica de preenchimento e calor [4].

A variação da frequência pelo efeito de vibrato pode ser obtida de uma forma simples através do uso de uma linha de retardo e de um Oscilador de Baixas Frequências (*Low Frequency Oscilator*, LFO). O intervalo dos taps da linha de retardo é controlado pela onda produzida LFO, de forma que a variação do tempo de tap causa a variação na frequência da nota tocada.

Para compreender com clareza o funcionamento do efeito de vibrato, é necessário compreender o funcionamento da variação da linha de retardo pelo LFO. Como visto anteriormente no efeito de delay, o uso da linha de retardo sozinha não provoca alterações na altura da nota. Seja M o tamanho (número de amostras de sinal) da linha de retardo. No caso de uma linha de retardo que não varie no tempo, cada amostra de entrada $x[n]$ gera uma amostra de saída $y[n] = x[n - M]$. Neste caso, o sinal de saída nada mais é do que uma cópia atrasada do sinal de entrada.

Suponha agora que o comprimento de M diminua de Δm a cada amostra, ou seja:

$$M[n] = M_{max} - (\Delta m)n$$

Aplicando $M[n]$ no sinal de saída, temos:

$$\begin{aligned} y[n] &= x[n - M[n]] = x[n - (M_{max} - (\Delta m)n)] \\ y[n] &= x[n - M_{max} + (\Delta m)n] \\ y[n] &= x[(1 + \Delta m)n - M_{max}] \end{aligned}$$

Pode-se ver que, devido ao aumento do tamanho da linha de retardo, a taxa de saída do sinal é $(1 + \Delta m)$ maior do que a taxa entrada. Isto significa que as frequências do sinal de entrada também serão multiplicadas por um fator de $(1 + \Delta m)$. Desta forma, tem-se que a diminuição no tamanho da linha de retardo gera um aumento na frequência do sinal.

O inverso ocorre da mesma forma, onde o aumento do tamanho da linha de retardo gera uma diminuição na frequência do sinal. Este fenômeno também explica a curiosidade de se obter frequências mais graves quando se toca um som a uma velocidade inferior à original e de se obter frequências mais agudas ao se tocar um som a uma velocidade superior.

O efeito de vibrato, portanto, consta na variação periódica do tamanho da linha de retardo, gerando assim uma variação da altura da nota tocada [10].

Pode-se relacionar também o efeito de vibrato com o efeito Doppler. No efeito Doppler, a variação na frequência do som é causada pela mudança na distância entre o emissor e o receptor do som. Esta variação na distância pode ser considerada equivalente à variação do tempo de delay no caso do efeito de vibrato [3].

3.2.3 Chorus

O efeito de chorus é um efeito muito conhecido no mundo musical, e se trata de um efeito semelhante a vários instrumentos com timbres semelhantes e tocando notas de frequências quase iguais ao mesmo tempo, como um coral, ou uma orquestra de cordas. Apesar de as frequências não serem absolutamente iguais, o resultado final não soa dissonante, mas soa como um som mais rico e preenchido.

Não existe uma maneira única de se implementar o efeito de chorus, existem vários algoritmos e diferentes formas de obtê-lo [12]. Uma forma extremamente simples de se obter o efeito é com o auxílio de um efeito de vibrato. Como o efeito de vibrato gera uma variação leve na frequência do sinal, basta somar o sinal processado com o vibrato ao sinal de entrada para obter-se o efeito de chorus.

A implementação do efeito de chorus com o auxílio de um efeito de vibrato gera um resultado simplificado. O efeito pode ainda ser melhorado com a adição de mais LFOs e linhas de retardo, podendo gerar um efeito que simula um som mais rico ainda,semelhante a uma quantidade maior de instrumentos.

Uma implementação interessante é aquela que usa uma outra linha de retardo, mas com o mesmo LFO. A segunda linha de retardo é alimentada pelo primeiro LFO, mas com

uma diferença de fase de 180°. Desta forma, enquanto um canal de áudio tem um ciclo que sobe, o outro tem um ciclo que desce, criando um efeito de ambientação interessante. Outra variação possível para o chorus é a criação de LFOs complexos, através da soma de múltiplos LFOs com frequências distintas.

3.2.4 Flanger

O efeito de flanging é obtido através da mistura de dois sinais de áudio identicos, onde um deles é atrasado por um intervalo de tempo curto e variável, normalmente menor do que 20 ms. O efeito audível é semelhante a uma turbina de avião por trás do sinal de entrada.

O termo flanging surgiu da forma como o efeito era produzido originalmente. O aúdio, até então gravado em fitas, era tocado a partir de duas fontes e simultaneamente. O engenheiro de áudio então pressionava uma das fitas com o dedo, fazendo com que uma das fontes ficasse com o sinal levemente atrasado em relação à outra. No período de desaceleração da fita, nota-se o efeito do flanging, devido à variação na altura do som. Pressionando a outra fita, o sinal original se atrasa também. A medida que o sinal se aproxima do outro, pode-se notar também o efeito de flanging, no outro sentido.

3.2.5 Distorção

O efeito de distorção é um efeito muito utilizado no mundo musical. Apesar de seu uso mais comum ser em efeitos de guitarras, também é comumente aplicado em baixos elétricos, teclados e até mesmo em vocais. O efeito digital é uma simulação do efeito obtido pela distorção gerada por válvulas, transistores e circuitos digitais.

Uma das formas de se produzir o efeito de distorção é através de amplificação. Um amplificador de som tem duas formas de operação: linear e não linear. No caso linear, o sinal de saída é simplesmente uma cópia amplificada do sinal de entrada. Já no caso não linear, o sinal de saída é uma cópia distorcida do sinal de entrada, ou seja, a forma de onda do sinal sofre alterações no processo de amplificação. Um caso de amplificação não-linear pode ser observado quando um amplificador tenta gerar uma voltagem de saída maior do que a sua capacidade máxima. Este tipo de variação no sinal é conhecido como *clipping*.

Outra forma de se produzir o efeito de distorção é através de *waveshaping*. O *waveshaping* nada mais é do que a alteração da forma de onda através de alguma operação matemática. Dependendo da operação, o efeito audível da distorção por *waveshaping* é semelhante ao efeito da amplificação não-linear, sendo a principal diferença o fato de não ser necessário um aumento do volume.

3.2.6 Equalizador

O processo de equalização consiste em alterar a resposta em frequência de um sistema de áudio utilizando filtros lineares. Pode-se dizer que os equalizadores são botões de volume para frequências específicas, uma vez que eles ajustam a amplitude do sinal em determinada frequência.

Equalizadores são utilizados em estúdios de gravação, de transmissão, ao vivo, e costumam ser utilizados para correção de frequência de microfones, de captadores de instrumentos, auto-falantes, acústica de ambientes, para eliminar sons não desejados, para deixar certas vozes ou instrumentos mais proeminentes, para ajustar o timbre de instrumentos individuais, dentre outros usos.

Existe uma variadede grande de tipos de equalizadores, podendo citar os paramétricos, semi-paramétricos, gráficos, programáveis, dentre outros. Equalizadores paramétricos permitem o controle de três parâmetros: A amplitude, a frequência central e a largura de banda. Desta forma, pode-se controlar a amplitude de cada uma das bandas, bem como a frequência central e a largura de banda. Equalizadores paramétricos costumam ser utilizados em gravação em estúdios e em shows ao vivo, pois são capazes de ajustes miuto mais precisos do que os outros tipos de equalizadores. Equalizadores semi-paramétricos são aqueles que possuem controle de amplitude e de largura de banda, porém possuem as frequências centrais fixas. Equalizadores gráficos são aqueles cujas frequências centrais e a largura de banda são fixas, mas a amplitude de cada uma das frequências é controlável.

Equalizadores podem ter quantas bandas forem necessárias. Para simplificação e melhor aplicabilidade no uso de sistemas de efeitos de guitarras, foi escolhido um equalizador gráfico de 3 bandas para a implementação deste estudo. Para tanto, foi utilizada uma implementação que faz uso de 3 filtros Butterworth: um passa-altas, um passa-baixas e um passa-banda.

Filtros Butterworth são interessantes pois são projetados para terem uma resposta em frequência o mais plana o possível na banda passante, não apresentando ondulações nas bordas como os outros tipos de filtros existentes. No caso do equalizador, esta escolha de filtros aparenta ser a mais indicada, uma vez que as ondulações dos outros tipos de filtro podem gerar resultados indesejados nas frequências onde elas ocorrem, e que pode ser feita uma leve superposição de frequências de forma que o decaimento mais lento dos filtros Butterworth não interfira no ganho desejado de determinada frequência pelo operador do equalizador (Figura 2.12).

Guitarras elétricas costumam ser afinadas de forma que a nota mais grave que se possa emitir seja o Mi (E) de 82Hz. A nota mais aguda, entretanto, varia de acordo com o modelo da guitarra, mas o mais convencional é que seja um Fá (F) 1.380 Hz. Como existe a possibilidade de se produzirem inúmeros harmônicos com a guitarra por se tocarem várias cordas simultaneamente, podem-se ter frequências de mais de 20 KHz.

3.2.7 Tremolo

O efeito de tremolo consta na oscilação do volume de uma dada nota. Muitas vezes, na cultura popular, o efeito de tremolo é confundido com vibrato. Este último, por sua vez, refere-se à oscilação na frequência da nota.

A oscilação do volume de uma nota é uma técnica extremamente simples de ser feita com a voz. Entretanto, os primeiros usos do efeito de tremolo em instrumentos nos remetem a um passado distante, no Império Bizantino do século IX e na Europa do Século XVI. No caso do Império Bizantino do século IX, podemos dizer que o efeito estava existia devido ao uso de instrumentos de arco e cordas, semelhantes a violinos e violoncelos. No caso de instrumentos de arco, o efeito de tremolo é produzido com a movimentação do arco, gerando também a sustentação da nota. Já no caso da Europa do século XVI, o

efeito existia nos órgãos de tubos, onde alguns deles possuiam um diafragma que varia va a pressão do ar dentro do tubo, variando, consequentemente, o volume do som [5].

A primeira aplicação do efeito de tremolo em guitarras surgiu em 1947 com um amplificador criado pela Danelectro, que aplicava os efeitos de tremolo e vibrato no som da guitarra, com regulagem de intesidade e de frequência do efeito.

Assim como nos casos mais antigos, o efeito de tremolo é utilizado para dar ideia de sustentação e persistência de determinada nota, fazendo crescer assim, a dinâmica do som.

3.2.8 Panner

O Panner é o último efeito a ser implementado neste estudo e está relacionado ao balanceamento e à quantidade de sinal destinado a cada um dos alto-falantes. Existem várias formas de se pensar e implementar um efeito de panner. A mais simples delas seria um único *knob*, com a posição de origem dele sendo a que aponta para cima. Ao girar o *knob* para a esquerda, o sinal passa a se tornar mais forte no canal da esquerda, e ao se girar para a direita, o sinal passa a se tornar mais forte no canal da direita.

Este efeito é extremamente utilizado na musica atual. Em várias músicas é possível notar que alguns instrumentos saem apenas em um dos alto-falantes, ou saem mais fortes em uns do que em outros. Isto gera, além da sensação de ambientação, uma sensação de equilíbrio, onde a presença de um certo som em um alto-falante é compensada por outro som no outro. Também pode-se destacar a existência de sistemas de som com vários canais e várias saídas, onde cada uma das saídas é específica para um certo tipo de frequências (mais graves, mais aguas, médias...).

Dentre as diversas formas de utilização do panner, existe aquela que produz a sensação de caminhamento do som pelos auto-falantes. Um panner que funcione desta forma, produz um efeito psicológico e de ambientação interessante.

Capítulo 4

Implementação de Efeitos de Áudio

Neste capítulo serão tratadas as implementações dos efeitos descritos no capítulo anterior. Será apresentado um circuito que caracteriza a implementação de cada um dos efeitos, bem como o código Csound que o gera. Em seguida,um breve comentário explicando como o código funciona, situando o leitor na implementação do efeito.

4.1 Encadeamento de Efeitos

Antes de se tratar da implementação dos efeitos em si, é importante descrever a forma de interação entre eles A utilização de forma separada dos efeitos já proporciona uma grande riqueza ao som processado. Seu uso de forma combinada, entretanto, pode acrescentar uma riqueza ainda maior ao som. Para permitir a combinação de efeitos, utilizou-se neste estudo um conjunto de opcodes do Csound para acessar diversos canais de áudio e permitir o processamento de cada um dos canais por cada efeito separadamente.

Os opcodes utilizados foram os opcodes *Zak*. Os opcodes *Zak* gerenciam diversos canais de aúdio e de controle globais, que podem ser acessados a partir de qualquer instrumento através de um índice inteiro. Para leitura de um canal *Zak* de áudio, utiliza-se o opcode *zar*, passando como parâmetro o canal desejado. O opcode retorna o sinal de áudio presente no canal. Para gravar um sinal de áudio num canal *Zak* é utilizado o opcode *zaw*, passando como parâmetros o sinal de áudio e a ser gravado.

Para inicialização do sistema *Zak* utiliza-se o opcode *zakinit*, chamado na seção de instrumentos, porém fora de um instrumento. O opcode *zakinit* recebe dois parâmetros: o primeiro deles é o número de canais de áudio a ser alocado pelo *Zak*, e o segundo é o número de canais de controle. O número de canais é inicializado no início da execução do programa csound e não pode ser alterado em tempo de execução.

Todos os instrumentos implementados neste estudo tem como parâmetros o seu canal de entrada e seu canal de saída. Desta forma, pode-se alterar o encadeamento dos efeitos na seção *CSscore* do arquivo Csound.

Foram criados instrumentos específicos para tratar os sons de entrada e de saída. Tais instrumentos são definidos da seguinte forma:

```
instr 1
  aIn    inch   1
  zaw    aIn,   p4
endin
```

```
instr 2
  aPluck   pluck 1, p5, 440, 0, 1
  zaw      aPluck, p4
endin
```

```
instr 3
  aOutL    zar      p4
  aOutR    zar      p4 + 1
  outs     aOutL,   aOutR
endin
```

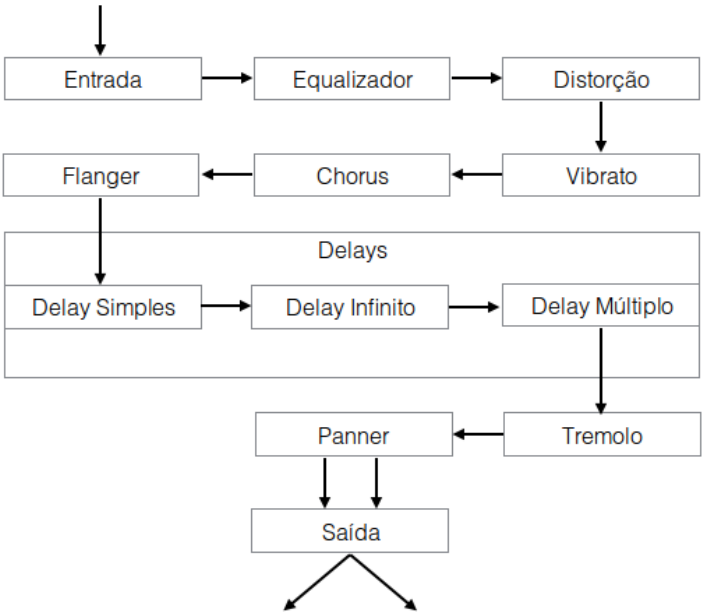

Figura 4.1: Forma como os efeitos foram encadeados no estudo. O sinal é recebido (ou gerado) na entrada, pelos instrumentos 1 ou 2. Em seguida, passa pelo equalizador, distorção, vibrato, chorus, flanger, pelos delays (simples, infinito e múltiplo), tremolo, panner e, por fim, é passado para o instrumento de saída, que leva o áudio aos alto-falantes.

Os instrumentos 1 e 2 estão relacionados ao sinal de entrada do sistema, armazenando o sinal no canal *Zak*. O instrumento 1 trata com entrada de sinal de áudio em tempo real, através do opcode *inch*, que recebe o sinal da entrada padrão do computador ou da entrada definida nas opções da seção *CsOptions*. O instrumento 2, por sua vez, trabalha com o opcode *pluck*. Em ambos os instrumentos, o primeiro parâmetro adicional é relativo ao canal *Zak* onde o sinal será escrito. No instrumento 2, entretanto, o segundo parâmetro adicional expressa a frequência da nota que será tocada.

O instrumento 3 cria duas variáveis de áudio que recebem o sinal dos canais de saída do último instrumento na cadeia. Os sinais de áudio são então tocados na saída do sistema como um som estéreo.

Os efeitos a serem trabalhados neste estudo foram encadeados conforme o esquema da figura 4.1. Esta sequência segue o padrão utilizados por guitarristas. Primeiramente, o sinal passa pelo equalizador, de forma a tratar as frequências da forma desejada. Distorções vêm logo adiante, seguidas de efeitos de modulação (vibrato, flanger e chorus). Após os efeitos de modulação tem-se os efeitos de delay. Por fim, os efeitos de ambientação, como tremolo e panner. Após o processamento pelo panner, o sinal é jogado no instrumento de saída, que é responsável por colocar o sinal nos alto-falantes do sistema.

Nas seções seguintes, serão explicadas as implementações de cada um dos efeitos tratados no capítulo anterior.

4.2 Delay

Primeiramente serão abordados os efeitos de delay, sendo eles dos três tipos apresentados anteriormente: Delay simples, delay infinito e delay múltiplo.

4.2.1 Delay Simples

O código abaixo implementa o efeito de delay simples, aquele que possui uma única repetição:

```
instr 58

  ;receive the input and output channel number
  iInChannel  = p4
  iOutChannel = p5

  ;sets aIn with the audio input
  aIn    zar    iInChannel

  ;receives the effect on/off switch
  kEffectOn    invalue "singledelay_on"

  ;if effect switch is on
  if (kEffectOn == 1) then

    ;receives the widgets values
    kDelay  invalue "singledelay_time"
    kLevel  invalue "singledelay_level"

    ;delay line
    aDelBuff  delayr  1        ;create a 1 second delay buffer
    aDelay    deltap  kDelay   ;tap on the delay line on the desired time
    delayw    aIn              ;write the delay buffer

    ;attenuates the delayed signal
    aDelay  =   aDelay * kLevel

    ;output signal is original signal + delayed signal
```

```
29      zaw     aDelay + aIn, iOutChannel
30
31
32    ;if effect switch is off
33    else
34
35      ;output signal is the original signal
36      zaw     aIn,  iOutChannel
37
38    endif
39
40  endin
```

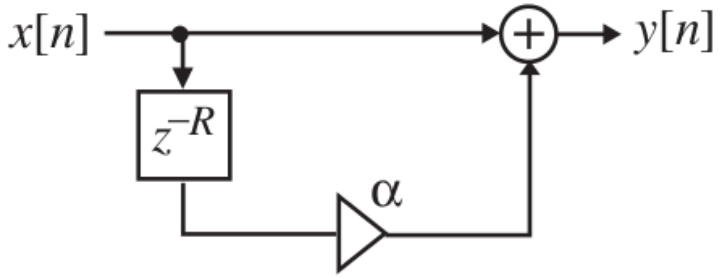

Figura 4.2: Esquema representativo de um circuito de delay simples. Esta implementação consta em um simples atraso do sinal de entrada com o uso de uma linha de retardo, seguido por uma atenação. Este sinal é então somado ao sinal de entrada, gerando assim o efeito de delay simples

O código acima descrito implementa o efeito de delay simples de uma única repetição, conforme o circuito descrito na figura 4.2. O começo do instrumento é padronizado para todos os instrumentos de efeitos deste trabalho: Atribuição dos canais de entrada e de saída (linhas 4 e 5), atribuição do sinal do canal de entrada para o sinal aIn (linha 8) e por fim atribuição da variável de controle que informa se o efeito está ligado ou desligado (linha 11). Após a verificação se o efeito está ligado (linha 14), passamos para a real implementação do efeito. Caso o efeito esteja desligado (linha 33), é sinal de entrada é escrito no canal de saída (linha 36).

Para este efeito, são lidos dos widgets do Csound os valores de atenuação (*kAlpha*) e de tempo (*kDelay*, em segundos) do efeito (linhas 17 e 18).

O sinal de entrada é então atrasado com o uso da linha de retardo, criada pelos opcodes *delayr* e *delayw*. Estes dois opcodes sempre aparecem aos pares. O opcode *delayr* (linha 21) cria uma linha de retardo no sinal de áudio *aDelBuff* do tamanho informado como parâmetro (1 segundo, no caso). O opcode *delayw* (linha 23), por sua vez, vai escrever o sinal passado em seu parâmetro no sinal criado pelo opcode *delayr* correspondente. Entre o *delayr* e o *delayw* temos os processamentos dos sinais das linhas de retardo e os opcodes *deltap*. Os opcodes *deltap* e suas variações são os responsáveis por capturar o sinal com

o atraso desejado informado em seus parâmtero. O sinal *aDelay* recebe então o sinal *aIn* atrasado de *kDelay* com o uso do opcode *deltap* (linha 22).

O sinal com atraso *aDelay* é então atenuado, através da multiplicação pelo valor *kLevel*, lido do widget (linha 26). Por fim, o sinal atrasado e atenuado é somado ao sinal de entrada, gerando assim o sinal de saída, com o efeito de delay aplicado (linha 29).

4.2.2 Delay Infinito

Segue abaixo o código Csound da implementação do delay de infinitos taps, conforme circuito apresentado na figura 4.3:

```
1  instr 59
2
3    ;receive the input and output channel number
4    iInChannel     = p4
5    iOutChannel    = p5
6
7    ;sets aIn with the audio input
8    aIn    zar    iInChannel
9
10   ;receives the effect on/off switch
11   kEffectOn    invalue "infinitedelay_on"
12
13   ;if effect switch is on
14   if (kEffectOn == 1) then
15
16     ;receives the widgets values
17     kDelay   invalue "infinitedelay_time"
18     kLevel   invalue "infinitedelay_level"
19
20     ;delay line
21     aDelBuff   delayr   1         ;create a 1 second delay buffer
22     aDelay     deltap   kDelay    ;tap on the delay line on the desired time
23     delayw     aIn + aDelay       ;write the delay buffer
24
25     ;attenuates the delayed signal
26     aDelay   = aDelay * kLevel
27
28     ;output signal is original signal + delayed signal
29     zaw      aDelay + aIn, iOutChannel
30
31
32   ;if effect switch is off
33   else
34
35     ;output signal is the original signal
36     zaw      aIn,     iOutChannel
37
38   endif
```

```
39
40 endin
```

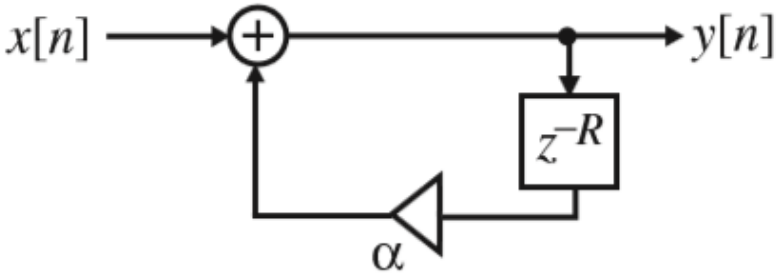

Figura 4.3: Esquema representativo de um circuito de delay infinito. Dado o sinal de entrada, a linha de retardo deve receber como alimentação o sinal de entrada somado ao sinal de saída da linha de retardo atenuado. O sinal de saída do efeito deve ser o mesmo sinal utilizado na alimentação da linha de retardo.

A implementação do efeito de delay de infinitas repetições é extremamente semelhante à implementação do delay simples de uma única repetição. A única diferença está no sinal que alimenta a linha de retardo. No caso do delay simples, o sinal que alimentava a linha de retardo era apenas o sinal de entrada. Neste caso, o sinal que alimenta é o sinal de entrada somado com o sinal já atrasado e atenuado (*aIn* + *aDelay*) (linha 23). Com isto, o sinal de saída apresenta infinitas repetições do sinal de entrada, onde cada repetição é atenuada com o valor de *kAlpha*.

4.2.3 Delay Múltiplo

Segue abaixo a implementação do efeito de delay múltiplo, conforme o circuito exemplificado na figura 4.4:

```
1  instr 60
2
3    ;receive the input and output channel number
4    iInChannel     = p4
5    iOutChannel    = p5
6
7    ;sets aIn with the audio input
8    aIn    zar    iInChannel
9
10   ;receives the effect on/off switch
11   kEffectOn    invalue "multidelay_on"
12
13   ;if effect switch is on
14   if (kEffectOn == 1) then
15
16     ;receives the widgets values
17     kRxSR    invalue "multidelay_time"
```

```
18      kAlpha  invalue "multidelay_atenuation"
19      kRep    invalue "multidelay_reps"
20
21      ;get the integer value of repetitions
22      kN      =    int (kRep)
23      outvalue "multidelay_intreps", kN
24
25      ;getting -1 * (aplha ^ N)
26      kNegAlphaN  pow     kAlpha, kN,     -1
27
28      ;first delay line
29      aDBuff1   delayr  1                       ;create a 1 second delay buffer
30      aD1       deltap  kRxSR                   ;tap on the delay line
31                delayw  aIn + (aD1 * kAlpha)    ;writes the delayed signal
32
33      ;second delay line
34      aDBuff2   delayr  1                       ;create a 1 second delay buffer
35      aD2       deltap  kRxSR * (kRep - 1)      ;tap on the delay line
36                delayw  aD1                     ;writes the delayed signal
37
38
39      ;output signal is:
40      ;original signal + first delay line signal + second delay line signal
41      zaw aIn + (aD1 * kAlpha) + (aD2 * kNegAlphaN), iOutChannel
42
43
44    ;if effect switch is off
45    else
46
47      ;output signal is the original signal
48      zaw     aIn,    iOutChannel
49
50    endif
51
52  endin
```

O código acima descreve o instrumento que implementa o efeito de delay de múltiplos taps. Recebemos dos widgets os valores da atenuação em *kAplha*, do tempo entre reptições em *kRxSR* e do número de repetições em *kRep* (linhas 17, 18 e 19). Como os widgets não trabalham com valores inteiros, é necessário fazer uma conversão explícita através do opcode *int* (linha 22). Com o valor inteiro de repetições, podemos calcular $z^{-(N-1)R}$, atribuindo isso à variável *kNegAlphaN*.

Como a implementação necessita de duas linhas de retardo, foram utilizadas dois blocos *delayr/delayw* (linhas 29-31 e 34-36), necessitando também de duas variáveis de áudio auxiliares, *aD1* e *aD2* (linhas 30 e 35). O primeiro bloco de delay é alimentado pelo sinal de entrada somado com o sinal resultante do primeiro bloco de delay (linha 31). Já o segundo bloco é alimentado pelo sinal de saída do primeiro bloco (linha 36). O sinal de saída é a soma do sinal de entrada com o sinal do primeiro bloco atenuado de *kAlpha* somado com o sinal do segundo bloco atenuado de $-kAlpha^N$ (linha 41).

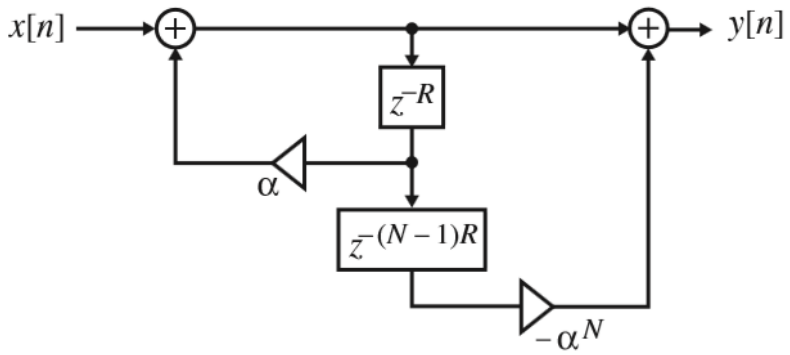

Figura 4.4: Esquema representativo de um circuito de delay múltiplo. O código mostrado acima implementa este circuito. $x[n]$ é o sinal de entrada. N é o número de repetições do efeito. z^{-R} e $z^{-(N-1)R}$ são as linhas de retardo, que atrasarão o sinal de entrada. Os blocos α e $-\alpha^N$ são os blocos de atenuação do sinal.

O sinal de saída é o sinal original de entrada somado aos sinais provenientes dos outros dois blocos de delay, com suas devidas atenuações. Desta forma, temos um efeito de delay com um número de repetições, intervalo entre as repetições e valor de atenuação definido pelo usuário.

4.3 Vibrato

Segue abaixo a implementação do efeito de vibrato. A implementação segue conforme a figura 4.5, porém com um pequeno acréscimo: a regulagem de intensidade do efeito, através do parâmetro kDepth. O parâmetro kDepth é utilizado para modificar a onda do LFO de forma que o sinal da onda varie sempre entre 1 e kDepth.

```
instr 34

  ;receive the input and output channel number
  iInChannel  = p4
  iOutChannel = p5

  ;sets aIn with the audio input
  aIn   zar   iInChannel

  ;receives the effect on/off switch
  kEffectOn   invalue "vibrato_on"

  ;if effect switch is on
  if (kEffectOn == 1) then

    ;receives the widgets values
    kDepth  invalue "vibrato_depth"
    kRate   invalue "vibrato_rate"
```

```
19      kMix     invalue "vibrato_mix"
20
21      ;generate the LFO wave and creates the modulation wave
22      aOsc     oscili    kDepth, kRate
23      aMod     = 1 - ((1-kDepth) * (1+aOsc) / 2)
24
25      ;variable tap on the delay line, based on the modulation wave
26      aTemp    delayr   1
27      aOut     deltapi  aMod
28               delayw   aIn
29
30      ;outputs the signal
31      zaw      aOut, iOutChannel
32
33    else
34      zaw      aIn,     iOutChannel
35    endif
36
37  endin
```

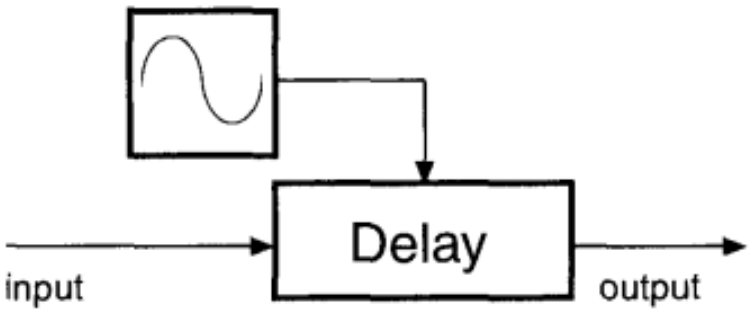

Figura 4.5: Esquema que representa o efeito de vibrato. O sinal de entrada passa por uma linha de retardo modulada por um LFO. A modulação do LFO faz com que o sinal de saída varie sua frequência.

O processamento do efeito de vibrato começa com a leitura dos parâmetros do efeito através dos widgets (linhas 17-19). É gerada então a onda do LFO por meio do opcode *oscili*, com amplitude *kDepth* e frequência *kRate* (linha 22). A onda moduladora é criada com base na onda do LFO, entretanto seus valores variam entre *kDepth* e 1 (linha 23). O sinal de saída então é o sinal gerado pelo tap variável na linha de retardo (linha 27), criando assim o efeito de variação na frequência característico do vibrato.

4.4 Chorus

O código da implementação do efeito de chorus é apresentado abaixo, implementando o circuito apresentado na figura 4.7. Esta versão do chorus é uma versão simplificada,

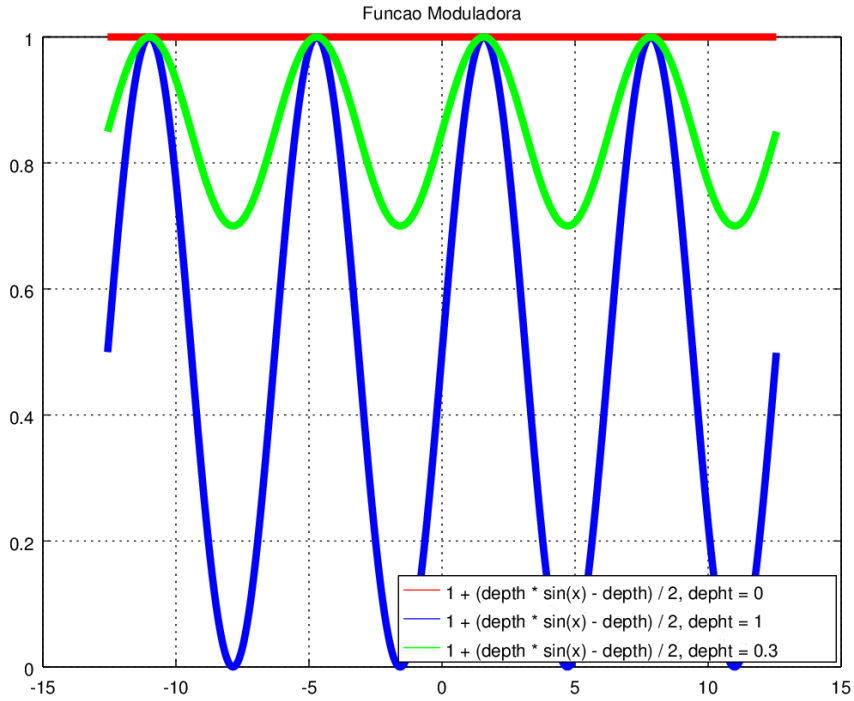

Figura 4.6: Gráfico de três variações da onda moduladora. Todas as três variações apresentam a mesma frequência, estando o diferencial entre as ondas no parâmetro *depth*. A primeira delas, em vermelho, apresenta $depth = 0$, ou seja, a onda moduladora não varia no tempo. A segunda, azul, apresenta o valor máximo de profundidade, isto é, $depth = 1$. Já a terceira, a onda de cor verde, possui um valor de *depth* intermediário, de forma que a onda varie entre 1 e o valor do parâmetro *depth*. No exemplo acima, $depth = 0.3$.

com apenas um LFO e uma linha de retardo, isto é, aquela que faz uso de um bloco de efeito de vibrato.

```
1  instr 35
2
3    ;receive the input and output channel number
4    iInChannel  = p4
5    iOutChannel = p5
6
7    ;sets aIn with the audio input
8    aIn    zar iInChannel
9
10   ;receives the effect on/off switch
11   kEffectOn invalue "chorus_on"
12
13   ;if effect switch is on
14   if (kEffectOn == 1) then
15
16     ;receives the widgets values
```

```
17      kDepth  invalue "chorus_depth"
18      kRate   invalue "chorus_rate"
19      kMix    invalue "chorus_mix"
20
21      ;generate the LFO wave and creates the modulation wave
22      aOsc    oscili   kDepth, kRate
23      aMod    = 1 - ((1-kDepth) * (1+aOsc) / 2)
24
25      ;variable tap on the delay line, based on the modulation wave
26      aTemp   delayr  1
27      aDel1   deltapi aMod
28              delayw  aIn
29
30      ;output signal is the delay line output atenuated by kMix
31      ;added to the input signal
32      aOut  balance (aDel1*kMix)+aIn, aIn
33
34      ;outputs the signal
35      zaw aOut, iOutChannel
36
37    else
38      zaw   aIn,  iOutChannel
39    endif
40
41  endin
```

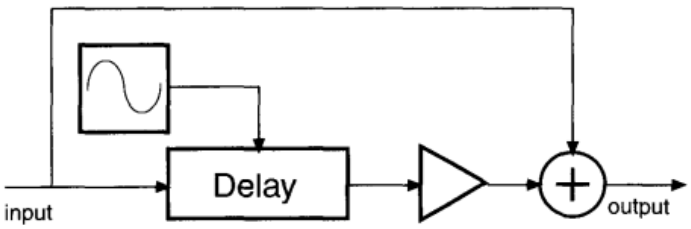

Figura 4.7: Esquema representativo de um circuito de chorus, na sua versão mais simplificada. O sinal de entrada é dividido e uma das partes é atrasada por uma linha de retardo controlada por um LFO. A parte que sofre o atraso é então atenuada e, então, somada ao sinal de entrada original, sendo então enviada à saída do sistema [1]. Pode-se notar através do circuito apresentado que esta implementação do chorus em um efeito de vibrato somado ao sinal de entrada do sistema.

A implementação do efeito de chorus descrita acima é extremamente semelhante à implementação do efeito de vibrato descrito anteriormente. A única e principal alteração está no sinal de saída. No efeito de vibrato, o sinal de saída era o próprio sinal gerado pela linha de retardo. No chorus, o sinal de saída consta na soma do sinal gerado pela linha de retardo, atenuado pela variável *kMix* somado ao sinal de entrada do sistema (linha 32).

4.5 Flanger

Segue abaixo a implementação do efeito de flanging, de acordo com o circuito apresentado na figura 4.8:

```
instr 40

  ;receive the input and output channel number
  iInChannel  = p4
  iOutChannel = p5

  ;sets aIn with the audio input
  aIn    zar    iInChannel

  ;receives the effect on/off switch
  kEffectOn invalue "flanger_on"

  ;if effect switch is on
  if (kEffectOn == 1) then

    ;receives the widgets values
    kDepth  invalue "flanger_depth"
    kRate   invalue "flanger_rate"
    kMix    invalue "flanger_mix"
    kFeed   invalue "flanger_feed"

    ;generate the LFO wave and creates the modulation wave
    aOsc    oscili   kDepth, kRate
    aMod    = 1 - ((1-kDepth) * (1+aOsc) / 2)

    ;variable tap on the delay line, based on the modulation wave
    ;delay line input is it's own output attenuated
    ;and added to the input signal
    aTemp   delayr  1
    aDel    deltapi aMod
            delayw  aIn + ((aDel * kMix) + aIn)* kFeed

    ;output signal is the delay line output atenuated by kMix
    ;added to the input signal
    aOut    balance aIn + (aDel * kMix), aIn
    zaw     aOut, iOutChannel

  ;if effect switch is off
  else

   ;output signal is the original signal
   zaw     aIn,    iOutChannel

  endif

```

```
46 endin
```

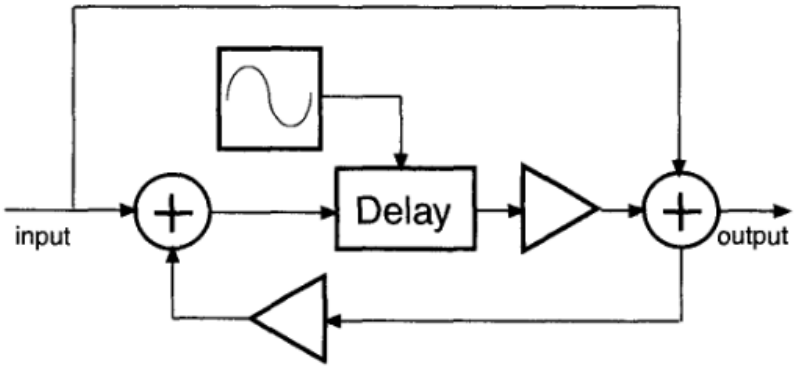

Figura 4.8: Circuito representativo do efeito de flanging. O sinal de entrada é dividido, e então passa por um bloco de retardo modulado por um LFO. O sinal atrsado é então somado com o sinal original de entrada, gerando o sinal de saída. Este sinal é então enviado à saida do sistema e também é somado com o sinal a ser atrasado, como uma retro-alimentação. Observando o circuito de implementação do efeito, pode-se perceber uma forte semelhança com os efeitos de chorus e de vibrato. Pode-se dizer que o efeito de flanging nada mais é do que um chorus com uma retro-alimentação no sinal que será atrasado.

O código que implementa o flanger é semelhante ao código que implementa o chorus. A principal variação está no sinal que alimenta a linha de retardo. No caso do chorus, apenas o sinal de entrada alimenta alinha de retardo. No flanger, o sinal de entrada é somado ao sinal de saída do efeito (linha 31), funcionando como uma retro-alimentação. Esta retro-alimentação é a responsável por produzir o efeito audível característico do flanger.

4.6 Distorção

Para implementação do efeito de distorção em Csound foi utilizado o opcode *distort*. O opcode *distort* retorna um sinal de áudio e possui 3 parâmetros obrigatórios (o sinal de entrada, a quantidade de distorção e a função de modelamento de onda) e mais dois parâmetros opcionais (sendo o primeiro deles relacionado à frequência de corte de um filtro passa-baixas interno ao opcode e o outro relacionado à disposição dos dados internos ao opcode).

O sinal de entrada do opcode *distort* é primeiramente comprimido com o auxílio de uma função RMS (*Root Mean Square*, uma média quadrática) e então é passado pela função de modelamento de onda, que modificará sua forma e seu espectro. Após isto, a onda é reajustada para sua amplitude original. O tanto que o sinal é distorcido depende tanto da função de modelamento quanto do parâmetro de quantidade de distorção, que costuma variar entre 0 e 1.

Valores mais próximos de zero no parâmetro de quantidade de distorção fazem com que o sinal de entrada passe pela função de modelamento com poucas mudanças, como se a função fosse uma reta com inclinação positiva passando pelo 0 no meio da tabela. A medida que este parâmetro cresce, o sinal comprimido é expandido mais e mais até se tornar cada vez mais parecido com a função modeladora. Quando o valor da quantidade de distorção se torna grande o suficiente, *clipping* começa a ocorrer na onda. Isso se dá por que a leitura da tabela não retorna ao início quando se atinge seu fim, fazendo com que permaneça nos valores das bordas da função quando os atinge. O ponto de *clipping* varia de acordo com a função de modelagem. Algumas funções podem começar a apresentar o *clipping* com quantidade de distorção em 0.7 (no caso de uma sinusóide simples), em 0.5 (no caso de algumas funções mais complexas), ou até em valores menores ainda, dependendo da complexidade da função.

As funções de modelamento podem ser arbitrárias, mas é recomendado que elas sejam contínuas, bem-comportadas perto do centro da tabela e que as partes positivas e negativas sejam bem balanceadas. Para a implementação no estudo, foi utilizada uma função sigmóide, conforme ilustra a imagem. A função sigmoide (Figura **??**graphSigmoid) é dada pela equação:

$$sig(x) = \frac{1}{1 + e^{-k*x}}$$

Segue abaixo a implementação realizada no estudo para o efeito de distorção em Csound, conforme o circuito da figura 4.9.

```
;this goes in the CsScore section
;defining the sigmoid wave
f 2     0    257      9    0.5 1    270

;this goes in the CsInstruments section
instr 30

  ;receive the input and output channel number
  iInChannel   = p4
  iOutChannel  = p5

  ;sets aIn with the audio input
  aIn    zar    iInChannel

  ;receives the effect on/off switch
  kEffectOn invalue "dist_on"

  ;if effect switch is on
  if (kEffectOn == 1) then

    ;receives the widgets values
    kdist        invalue "dist_dist"
    kpostgain    invalue "dist_postgain"

```

```
25      ;output is the distorted version of the input signal
26      aOut distort aIn, kdist, gifn
27
28      ;applying postgain
29      zaw aOut*kpostgain, iOutChannel
30
31
32    ;if effect switch is off
33    else
34
35      ;output signal is the original signal
36      zaw aIn, iOutChannel
37
38    endif
39
40  endin
```

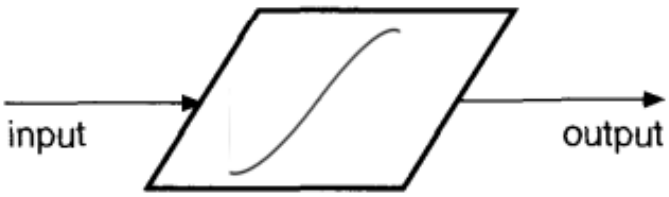

Figura 4.9: Circuito que implementa um efeito de distorção simplificado. Aqui, o sinal de entrada passa pela função de modelamento, gerando como sinal de saída o sinal distorcido.

A implementação do efeito de distorção é extremamente simples. Após a verificação se o efeito está ou não ligado e as atribuições de valores dos widgets às variáveis (linhas 22 e 23), aplica-se o opcode *distortion* ao sinal de entrada, com base no parâmetro de quantidade de distorção (linha 26). Por fim, aplica-se um pós-ganho, que multiplica o sinal distorcido por um valor, de forma a aumentar ou reduzir seu volume (linha 29). O sinal de entrada, portanto, sofre uma distorção com base na sigmoide e é multiplicado por um valor escalar de pós-ganho, gerando assim o sinal de saída.

4.7 Equalizador

Para a implementação do equalizador, foi escolhido uma variação do tipo gráfico com três bandas, onde as frequências de corte escolhidas foram de 500 Hz e de 4000 Hz. Desta forma, as bandas criadas foram: uma de menos de 500 Hz, para notas mais graves, uma de frequências entre 500 4000 Hz para notas médias e uma para frequências mais agudas, de mais de 4 KHz.

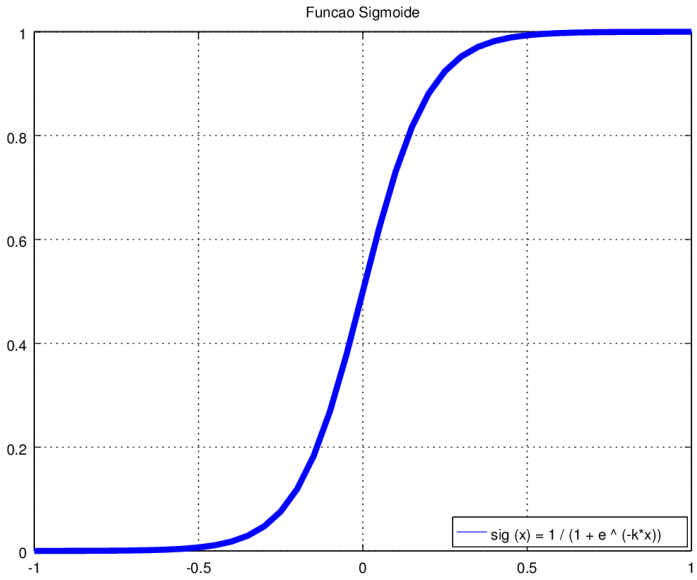

Figura 4.10: Função sigmoide

O código abaixo implementa o equalizador descrito anteriormente e com esquema indicado na figura 4.11:

```
instr 10

  ;receive the input and output channel number
  iInChannel  = p4
  iOutChannel = p5

  ;sets aIn with the audio input
  aIn    zar    iInChannel

  ;receives the effect on/off switch
  kEffectOn invalue "eq_on"

  ;if effect switch is on
  if (kEffectOn == 1) then

   ;receives the widgets values
   kLowGain     invalue "eq_lowgain"
   kMidGain     invalue "eq_midgain"
   kHighGain    invalue "eq_highgain"

   ;the cutoff value for each band
   iLowCutOff    = 500
   iHighCutOff   = 4000
   iCentralFreq = (iHighCutOff - iLowCutOff)/2 + iLowCutOff
   iBandWidth    = ((iHighCutOff - iLowCutOff) / 2) * 1.1
```

```
26
27      ;obtaining the bands signal
28      aLowSig      butterlp aIn, iLowCutOff
29      aMidSig      butterbp aIn, iCentralFreq, iBandWidth
30      aHighSig     butterhp aIn, iHighCutOff
31
32      ;output are the sum of bandsSig * bandGain
33      aOut    = aLowSig*kLowGain + aMidSig*kMidGain + aHighSig*kHighGain
34      aOut    balance aOut, aIn
35
36      ;output the signal
37      zaw      aOut,    iOutChannel
38
39
40    ;if effect switch is off
41    else
42
43      ;output signal is the original signal
44      zaw      aIn,     iOutChannel
45
46    endif
47
48  endin
```

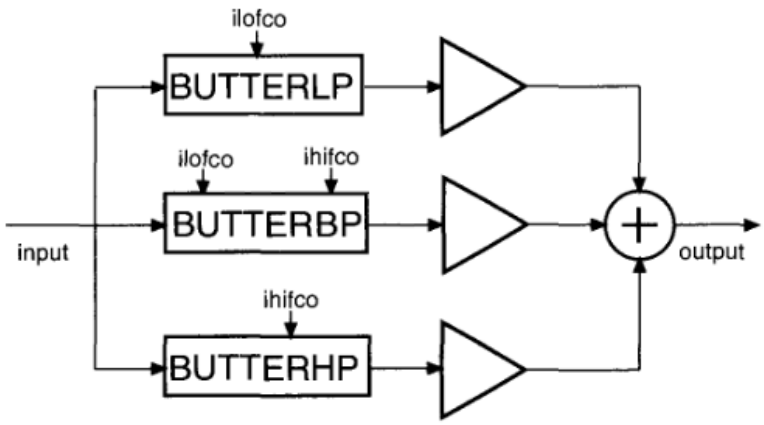

Figura 4.11: Circuito representativo do equalizador de 3 bandas implementado no estudo.

No código descrito acima, o sinal de entrada passa pelos três filtros em paralelo, então, os sinais resultantes são atenuados pelos valores informados pelos widgets e então somados, dando origem ao sinal de saída (linhas 33 e 34).

Para a implementação, foram utilizados os opcodes *butterlp* (linha 28), *butterbp* (linha 29) e *butterhp* (linha 30). Estes opcodes funcionam como os filtros passa-baixas, passa-banda e passa-altas, respectivamente. Os opcodes *butterlp* e *butterhp* possuem apenas dois parâmetros: o sinal de entrada e a frequência de corte, respectivamente. Já o opcode

butterbp possui 3 parâmetros: o sinal de entrada, a frequência central e a largura de banda do filtro. Todos os três opcodes possuem como saída o sinal filtrado.

No caso do filtro passa-bandas implementado acima, foi colocado um fator multiplicativo de 1.1 na largura de banda. Isto foi feito para que todas as frequências pudessem ser atingidas e trabalhadas pelos filtros, uma vez que os filtros Butterworth apresentam um curva com uma inclinação mais atenuada do que outros tipos de filtros.

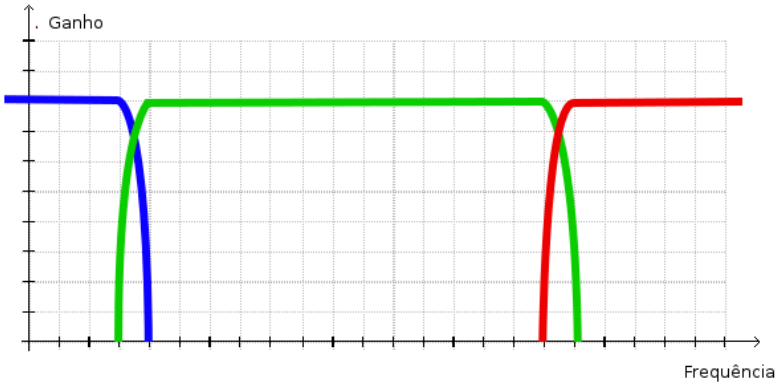

Figura 4.12: Gráfico de ganho x frequência dos filtros passa-baixas (azul), passa-bandas(verde) e passa-altas(vermelho). Com o fator multiplicativo de 1.1, existem frequências em que os filtros se interceptam. Desta forma, todas as frequências são atingidas pelas 3 bandas do equalizador.

4.8 Tremolo

O código abaixo implementa o efeito de tremolo, conforme o circuito mostrado na figura 4.13:

```
instr 70

  ;receive the input and output channel number
  iInChannel  = p4
  iOutChannel = p5

  ;sets aIn with the audio input
  aIn    zar    iInChannel

  ;receives the effect on/off switch
  kEffectOn invalue "tremolo_on"

  ;if effect switch is on
  if (kEffectOn == 1) then
```

```
15
16      ;receives the widgets values
17      kFreq    invalue "tremolo_freq"
18      kDepth   invalue "tremolo_depth"
19
20      ;creates a sine wave with kDepth amplitude and kFreq frequency
21      kOsc     oscili  kDepth, kFreq
22
23      ;output signal is the original signal multiplied by the modulating wave
24      kMod     = 1 - ((1-kDepth) * (1+kOsc) / 2)
25      aOut     = aIn * kMod
26
27      zaw      aOut, iOutChannel
28
29    ;if effect switch is off
30    else
31
32      ;output signal is the original signal
33      zaw      aIn,     iOutChannel
34
35    endif
36  endin
```

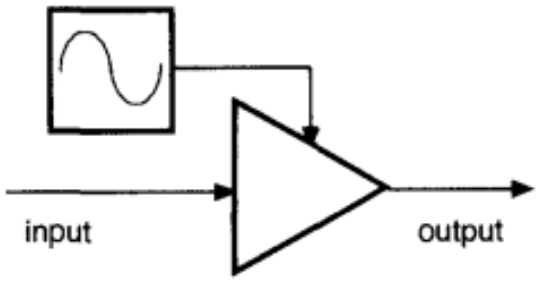

Figura 4.13: Circuito que implementa o efeito de tremolo. O circuito é extremamente simples. O sinal de entrada é atenuado pelo valor de um LFO, dando origem ao sinal de saída.

A aplicação descrita acima apresenta apenas dois widgets: O de intensidade do efeito (kDepth) e o de frequência do LFO (kFreq). Primeiramente, é criada uma senoide que tem como parametros os dados dados dos widgets (linha 21). Cria-se então uma onda moduladora da mesma forma que a onda que controla a intensidade dos efeitos de vibrato, chorus e flanger, variando entre *kDepth* e 1 (linha 24). Esta onda moduladora multiplicará a onda do sinal de entrada, gerando assim a alteração na amplitude da onda com base no LFO (linha 25).

4.9 Panner

O efeito de panner implementado a seguir se comporta de uma maneira peculiar. Faz-se uso de um LFO para definir a quantidade de sinal para cada um dos canais de audio. Desta forma, temos a sensação de que o som caminha do alto-falante esquerdo para o direito e vice-versa, criando um efeito sonoro e psicológico interessante.

Temos abaixo a transcrição do código que implementa o efeito, conforme o circuito exemplificado pela figura 4.14.

```
instr 100

;receive the input and output channel number
  iInChannel  = p4
  iOutChannel = p5

  ;sets aIn with the audio input
  aIn    zar    iInChannel

  ;receives the effect on/off switch
  kEffectOn invalue "panner_on"

  ;if effect switch is on
  if (kEffectOn == 1) then

    ;receives the widgets values
    kFreq    invalue "panner_freq"

    ;creating a sine wave o amplitude = 1 and frequency = kFreq
    kOsc     oscili   1, kFreq

    ;creating the panner levels
    ;left panner level is the normalization of the sine wave
    ;right panner level is the inverse of the left panner value
    kPanL    = (kOsc + 1) / 2     ;left  panner
    kPanR    = 1 - kPanL          ;right panner

    ;the output signals are the input signals
    ;multiplied by each channel panner levels
    aOutL    = aIn * kPanL
    aOutR    = aIn * kPanR

    ;writing the output signals to separated channels
    zaw      aOutL,  iOutChannel
    zaw      aOutR,  iOutChannel + 1


  ;if effect switch is off
  else

    ;output signal are the original signals
```

```
42      zaw     aIn,    iOutChannel
43      zaw     aIn,    iOutChannel + 1
44
45    endif
46
47  endin
```

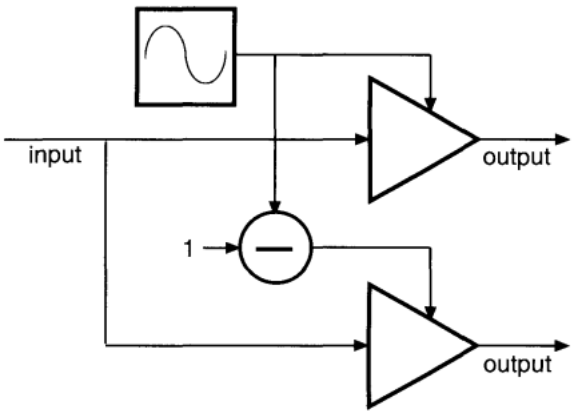

Figura 4.14: Circuito representativo do efeito de panner. A senoide gerada pelo LFO controla quanto do sinal será distribuído para um canal, enquanto o inverso da senoide controla quanto do sinal será distribuído para o outro canal.

Para a implementação do efeito, a primeira coisa a ser feita é a leitura da frequência do LFO a partir do widget (linha 17). Cria-se então a senóide com a frequência informada (linha 20). A partir daí, grava-se nas variáveis *kPanL* e *kPanR* os valores relativos à intensidade dos sinais nos canais esquerdo e direito (linhas 25 e 26). O valor do *Pan* do canal esquerdo será a normalização da senoide, enquanto o valor do *Pan* do canal direito será o inverso do valor do *Pan* do canal esquerdo. O sinal de saída de cada um dos canais será então o sinal de entrada multiplicado pelo valor de *Pan* do respectivo canal (linha2 30 e 31).

Pela necessidade de se trabalhar com dois canais, o efeito de panner foi deixado como último na cadeia de efeitos implementada no estudo, de forma que todos os outros efeitos trabalham com sinal mono e apenas o panner trabalha com sinal estéreo.

No capítulo a seguir serão tratados os resultados da implementação destes efeitos, analisando as formas de onda geradas pela sua aplicação, bem como um breve comentário acerca da sensação audível produzida através deles.

Capítulo 5

Análise de Resultados

Neste capítulo serão comentados e descritos os resultados da implementação dos efeitos do capítulo anterior. Para melhor compreensão, sugere-se que o leitor teste os efeitos em seu computador. Serão exibidos as formas de onda dos sinais puros e processados através dos efeitos.

5.1 Delays

A verificação visual do efeito de delay é de fácil visualização. Para demonstração do efeito de delay foi utilizado um sinal de *pluck*. O sinal de *pluck* é um sinal gerado pelo algorítmo Karplus-Strong, um algorítmo simples e efetivo para a modelagem de instrumentos de corda. A figura 5.1 mostra a forma de onda de um sinal gerado pelo opcode *pluck* do Csound.

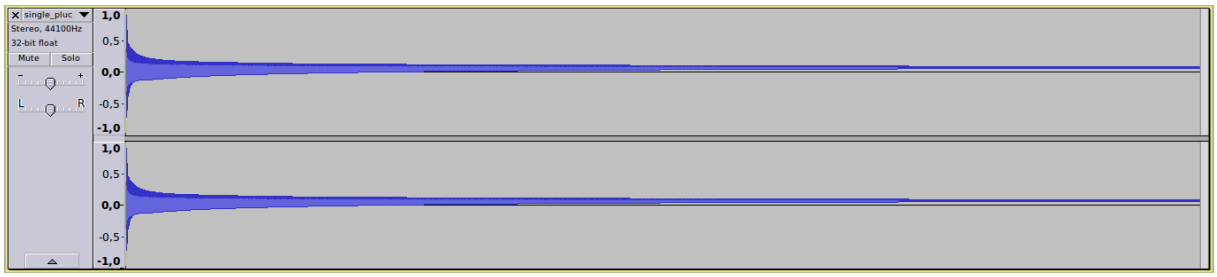

Figura 5.1: Onda gerada pelo opcode *pluck* do Csound. O som é gerado com início no instante 0 e tem duração de 10 segundos. Foi gerado com uma frequência de 440Hz.

5.1.1 Delay Simples

Ao se aplicar o efeito de delay simples na onda gerada pelo *pluck*, tem-se a onda da figura 5.2. Nota-se que o sinal processado é o sinal de entrada somado com uma cópia atenuada e atrasada no tempo. Neste caso, como o tempo de delay é considerávelmente alto, a recepção do sinal é perceptivel. Para tempos de delay mais curtos (tempos menores do que 50 ms), a sensação audível é diferente, uma vez que se percebe um efeito de ambientação e preenchimento.

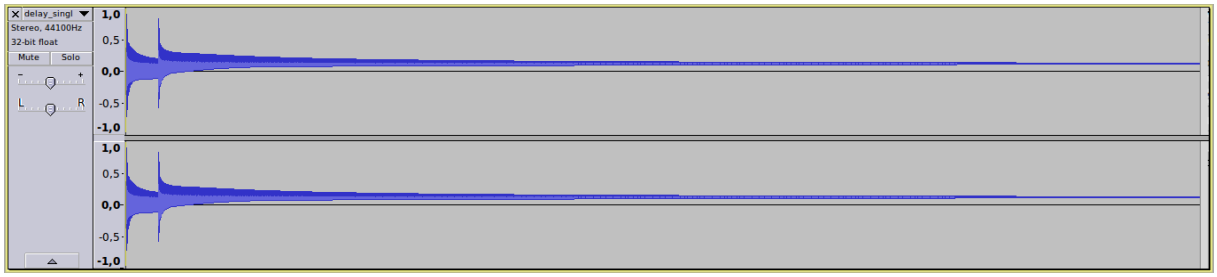

Figura 5.2: Onda gerada pelo opcode *pluck* do Csound e processada pelo efeito de delay simples descrito no capítulo anterior. Os parâmetros do efeito foram: 0.3 segundos de atraso e 0.7 de atenuação.

5.1.2 Delay Infinito

Assim como no caso anterior, a aplicação do efeito de delay infinito gera a onda descrita na figura 5.3. O sinal processado é dado pelo somatório de infinitas cópias de si mesmo, atenuadas por uma constante α e igualmente espaçadas no tempo.

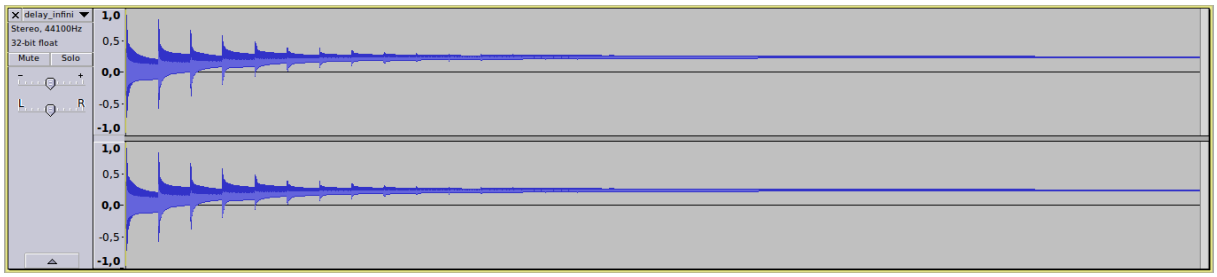

Figura 5.3: Onda processada pelo efeito de delay infinito implementado no capítulo anterior. Os parâmetros do efeito também foram: 0.3 segundos de atraso e 0.7 de atenuação.

5.1.3 Delay Múltiplo

O efeito de delay múltiplo se comporta de forma semelhante aos efeitos de delay descritos anteriormente. A principal diferença está no número definido de repetições. Para um caso de N repetições, o efeito se comporta como um delay infinito até a Nésima repetição, onde os sinais vão decaindo de amplitude com uma taxa de α. A partir da repetição $N+1$, entretanto, o circuito cancela as repetições seguintes, conforme pode ser observado na figura 5.4.

5.2 Vibrato

Os efeitos a serem tratados a partir desta seção envolvem alterações no domínio da frequência. Desta forma a vizualização da onda sonora não acrescenta muita informação ao leitor.

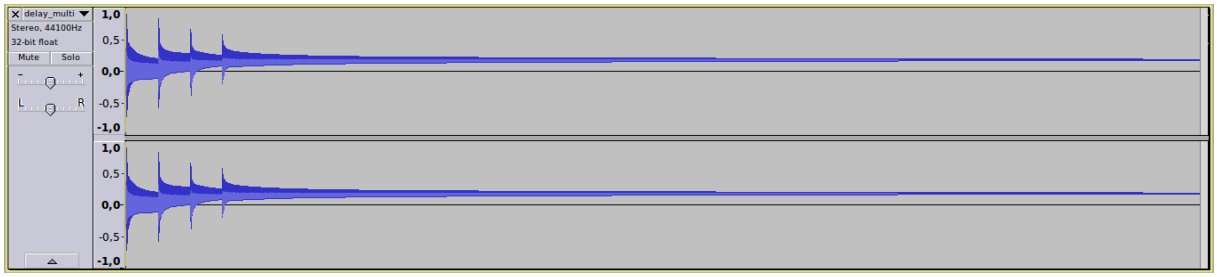

Figura 5.4: Onda gerada pela aplicação do efeito de delay múltiplo. Para esta onda, os parâmetros selecionados foram: 0.3 segundos de atraso, 0.7 de atenuação e 3 repetições

O efeito de vibrato é caracterizado pela variação da frequência de forma periódica no sinal. A variação na frequência é obtida através da variação do tamanho da linha de retardo, conforme visto na seção 4.3.

Para a aplicação do efeito de vibrato e de alguns efeitos seguintes, foi utilizado como sinal de entrada uma sequência de *plucks*, gerados pelo Csound, que pode ser conferida na figura 5.5. A onda do sinal processado pelo efeito de vibrato está indicada na figura 5.6.

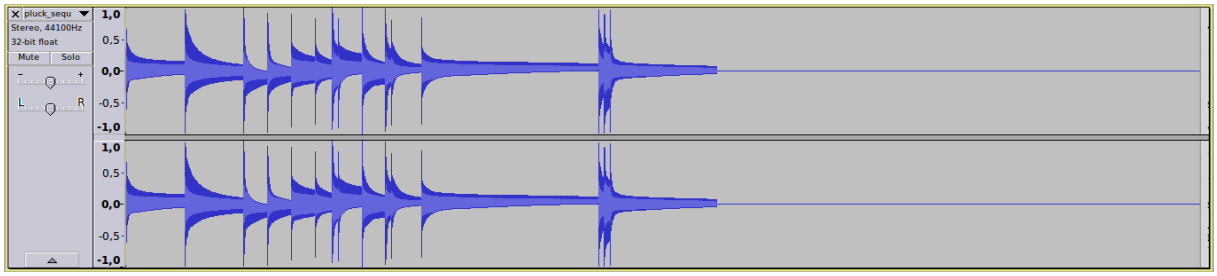

Figura 5.5: Onda gerada pela sequência de *plucks*.

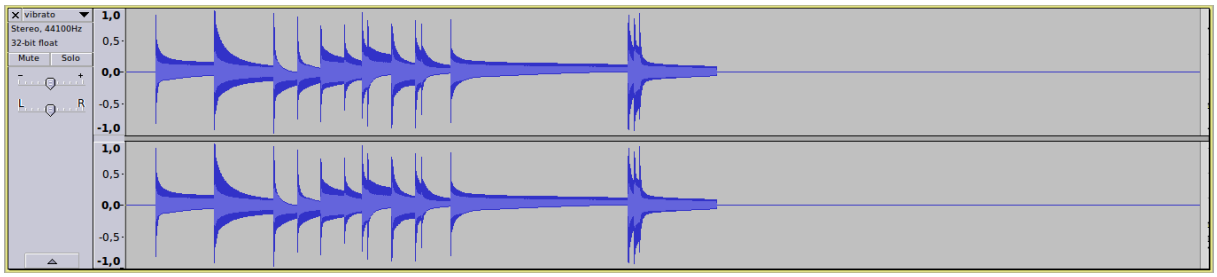

Figura 5.6: Aplicação do efeito de vibrato na sequência de plucks da figura 5.5. Por se tratar de um efeito que provoca alterações no domínio da frequência, a observação da forma de onda não trás informações esclarescedoras acerca do efeito.

5.3 Chorus

Assim como o efeito de vibrato, a principal variação do efeito de chorus se dá no domínio da frequência, desta forma, a vizualização do efeito não é claramente visível através da onda de sinal processado. Pode-se notar que o sinal gerado pela sequência de *plucks* do Csound foi o sinal utilizado como sinal de entrada para o teste do efeito.

O efeito de chorus exemplificado pela figura 5.7 não soa muito semelhante ao efeito de chorus original devido à associação entre o sinal gerado pelos *plucks* e a passagem do sinal pelos canais *Zak*. O sinal gerado pelos *plucks*, ao ser escrito/lido de um canal *Zak*, perde um pouco da sua continuidade e preenchimento, fazendo com que seu som se torne mais seco. Desta forma, o efeito do chorus deixa de soar como deveria, soando mais como a combinação dos efeitos de delay simples com vibrato. No caso da aplicação deste mesmo efeito em um sinal proveniente de um instrumento musical, através da entrada de áudio do computador, o efeito soa conforme deveria, produzindo um som mais rico e preenchido.

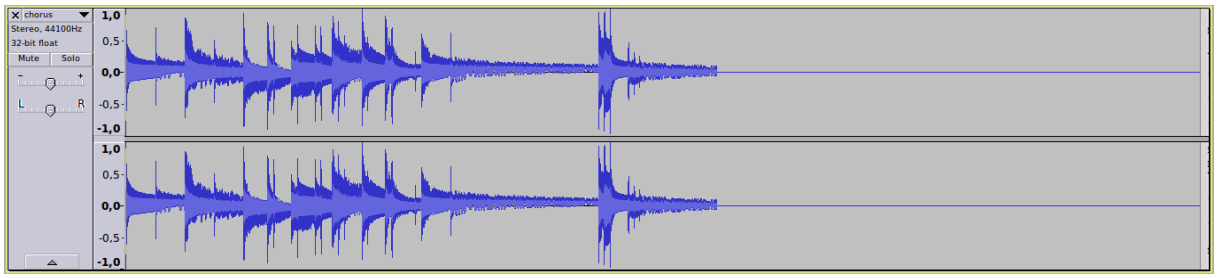

Figura 5.7: Aplicação do efeito de chorus na sequência de plucks da figura 5.5. Por se tratar de um efeito que provoca alterações no domínio da frequência, a observação da forma de onda não trás informações esclarescedoras acerca do efeito. Os parâmetros utilizados neste efeito foram: 0.007 de depth, 1.5 de rate e 1.0 de mix

5.4 Flanger

O efeito de flanger, assim como o os efeitos de vibrato e chorus, apresenta as suas principais variações no domínio da frequência. Entretanto, por apresentar um resultado com mais ruído e variações do que os efeitos descritos anteriormente, a sua forma de onda apresenta algumas variações quando comparada à onda do sinal da figura 5.5. A onda do sinal com efeito de flanger pode ser conferida na figura 5.8.

5.5 Distorção

O efeito de distorção desenvolvido neste estudo consta principalmente no *waveshaping* (modelagem de onda). O para fins de demonstração, o efeito foi aplicado no sinal da sequência de *plucks* exemplificado na figura 5.5. O sinal processado pelo efeito de distorção pode ser visualizado na figura 5.9.

Para melhor compreensão, foram aproximadas as figuras 5.5 e 5.9. Nota-se que o sinal distorcido apresenta uma amplitude bem reduzida com relação ao sinal original, mesmo

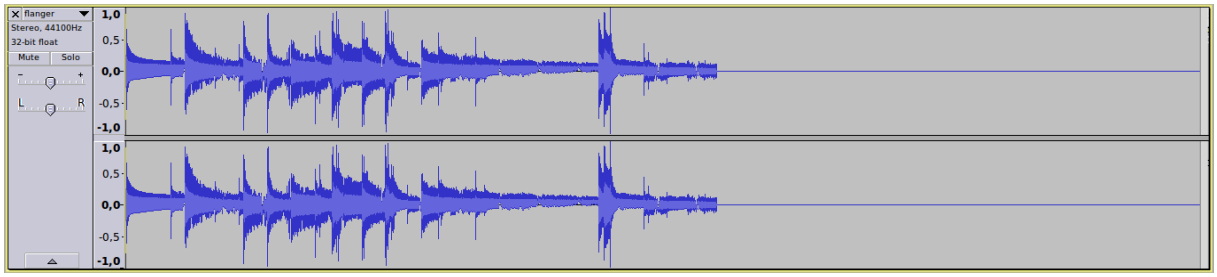

Figura 5.8: Aplicação do efeito de flanger na sequência de *plucks*, com *depth* de 0.7, frequência de 1.5 Hz, mix de 0.5 e feed de 0.75.

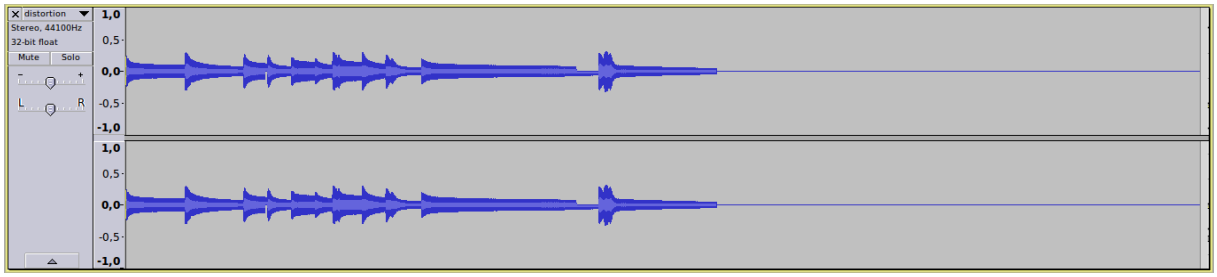

Figura 5.9: Aplicação do efeito de distorção no sinal de entrada descrito na figura 5.5.

com a aplicação de um valor de pós-ganho. Entretanto nota-se que a onda distorcida apresenta um formato mais brusco do que o sinal original, tornando-a mais parecida com ondas triagulares, quadradas ou dentes de serra, fazendo com que o som tenha seu timbre característico do efeito de distorção, conforme pode ser observado nas figuras 5.10 e 5.11.

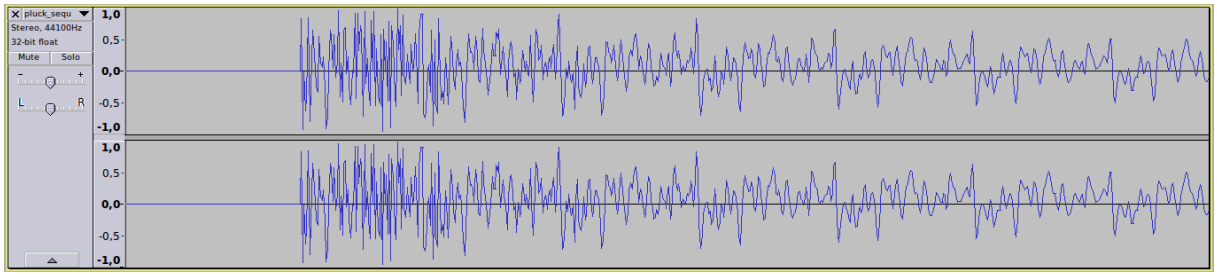

Figura 5.10: Aproximação da onda do sinal da figura 5.5.

5.6 Tremolo

O efeito de tremolo consta na variação da amplitude do sinal entre 1 e um valor definido pelo usuário através do parâmetro *depth* numa frequência também definida pelo usuário. Por se tratar de uma variação na amplitude do efeito, percebe-se com facilidade a aplicação do efeito no sinal.

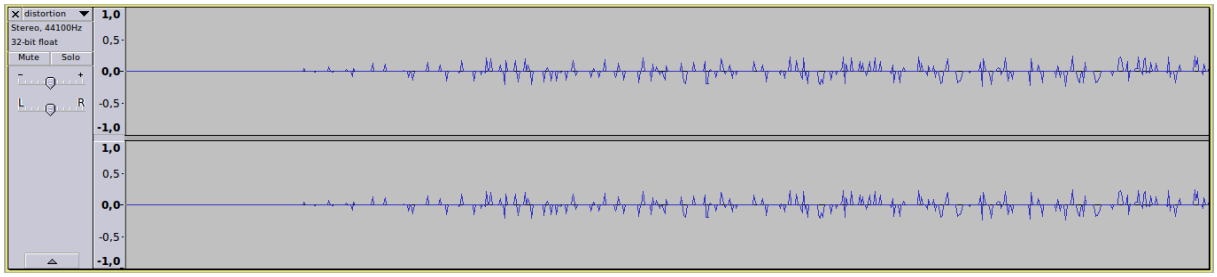

Figura 5.11: aproximação da onda da figura 5.9, de forma a se verificar as diferenças na forma de onda provocadas pela aplicação do efeito de distorção.

A figura 5.12 descreve uma senóide simples, com início no instante 0 e fim no instante 10 s, a uma frequência de 440 Hz. A aplicação do efeito de tremolo na senoide tem uma sensação audível interessante, uma vez que a variação na amplitude da onda é fácil de se perceber quando o sinal de entrada não apresenta variações. Da mesma forma, a vizualização da variação da amplitude do sinal também é perceptível com a observação da onda.

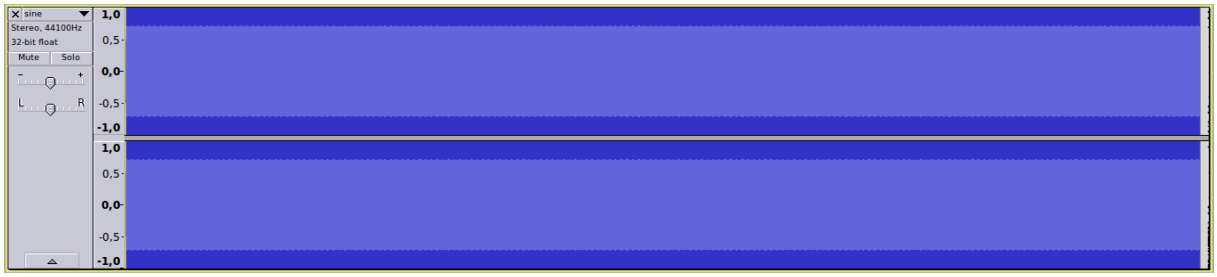

Figura 5.12: Senoide de 440 Hz gerada pelo opcode *oscil* do Csound.

A figura 5.13 trás a aplicação do efeito na sequência de plucks e a figura 5.14 trás a aplicação do efeito na senóide simples.

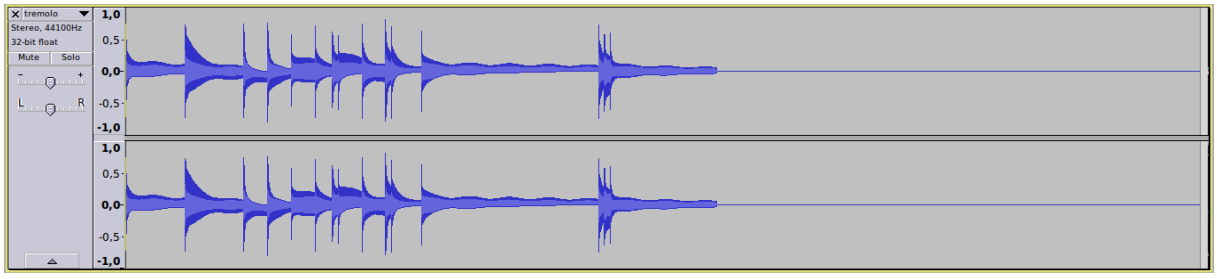

Figura 5.13: Aplicação do efeito de tremolo na sequência de *plucks*, com *depth* de 0.5 e frequência de 1.5 Hz.

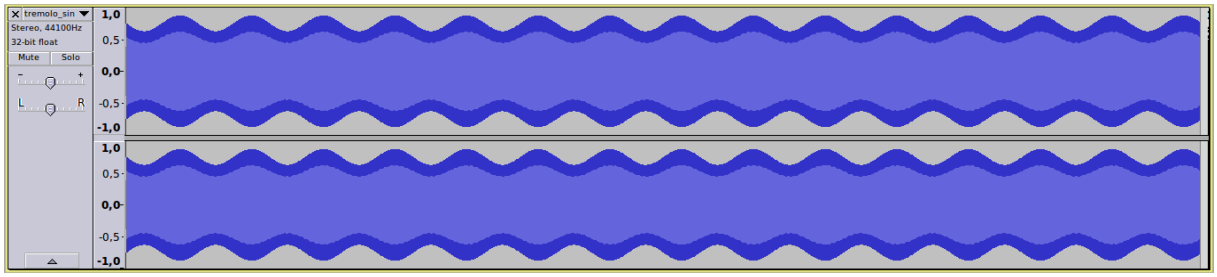

Figura 5.14: Aplicação do efeito de tremolo na senóide simples, para melhor vizualização do resultado do efeito. Assim como no caso anterior, *depth* vale 0.5 e a frequência vale 1.5 Hz.

5.7 Panner

O efeito de panner é caracterizado pela alternância do sinal entre os canais de áudio. No caso do efeito implementado no capítulo anterior, a alternância atingia uma amplitude de 100%, isto é, num dado instante, o sinal tinha amplitude 0 em um canal e 1 no outro. A medida que a amplitude de um dos canais aumenta, a do outro diminui, na mesma proporção, variando sempre entre 0 e 1. Para ilustração deste efeito, pode-se verificar o seu uso na sequência de plucks (Figura 5.15) e na senóide pura (Figura 5.16).

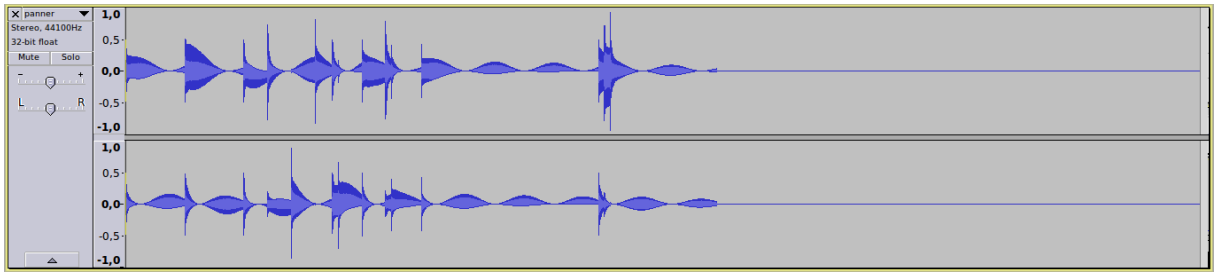

Figura 5.15: Onda gerada pela aplicação do efeito de panner na sequência de *plucks*. Nota-se que, enquanto a amplitude da onda decresce em um dos canais, ela cresce no outro, garantindo a consistência do efeito. Na onda da figura, o efeito foi aplicado com uma frequência de 1 Hz.

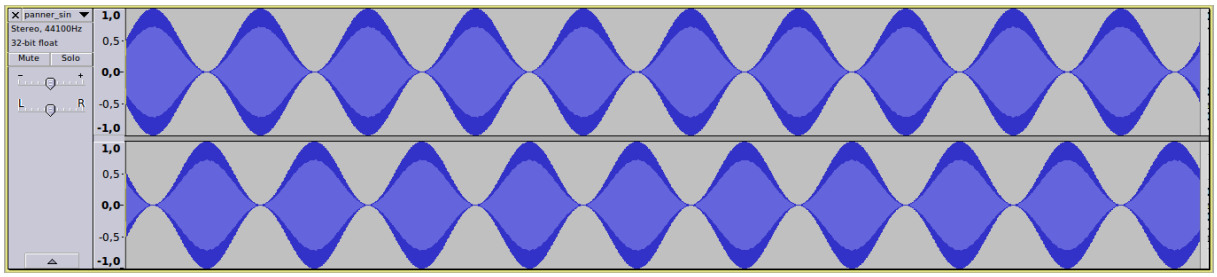

Figura 5.16: Onda gerada pela aplicação do efeito de panner na senóide simples. Como a onda do sinal de entrada é uma senóide simples, com a amplitude sendo mantida inalterada durante todo o tempo, a onda do sinal processado permite uma fácil vizualização do resultado do efeito. Nesta onda, a frequência do efeito de panner também foi de 1 Hz.

Capítulo 6

Conclusão

A partir deste estudo puderam ser conhecidos e compreendidos diversos dos efeitos de áudio existentes no mercado. Muitos dos efeitos descritos são bastante conhecidos por músicos e entendedores da área. A sua origem e implementação, entretanto, são pouco difundidas e, muitas vezes desconhecidas.

Com o desenvolvimento deste estudo, percebeu-se que o entendimento dos efeitos analógicos é de grande importância para a implementação dos seus correspondentes digitais. Estruturas como linhas de retardo, osciladores de baixa frequência, modeladores de ondas e filtros estão presentes em ambos os domínios, analógico e digital. Compreender como estes e outros elementos se comunicam e interagem em cada um dos efeitos foi primordial para a implementação dos efeitos digitais deste estudo.

Através da implementação dos códigos e estruturas propostos neste estudo, foi atingido o objetivo da criação de um banco de efeitos variados, bem como a devida explicação acerca de seu funcionamento.

Cada um dos efeitos implementados pode ter seu código comparado com uma esquematização, de forma a se verificar a sua estrutura, sendo possível comparar cada linha do código da sua implementação com a esquematização. Os efeitos desenvolvidos no estudo tem sonoridade semelhante aos efeitos existentes no mercado, tanto as simulações de software, quanto os equipamentos de hardware, conforme a discussão dos resultados no capítulo referente à análise.

Ainda existem muitos efeitos digitais que não foram implementados neste estudo. Muitos deles não foram implementados devido à sua complexidade, outros devido ao uso de estruturas que o Csound não suporta. Efeitos como *wah-wah*, *talkboxes*, filtros de ruídos, *pitch-shifters*, dentre outros, foram deixados de fora por estes motivos.

Muitas dificuldades foram enfrentadas na implementação dos efeitos. Uma bibliografia relativamente restrita, somada à íngreme curva de aprendizado da linguagem de programação tornou complexo o desenvolvimento do sistema de efeitos. A grande maioria dos efeitos foi implementada a partir do seu circuito de esquematização somente, aplicando os conceitos e opcodes conhecidos da linguagem para a devida implementação.

O estudo ainda pode progredir em diversos aspectos. A implementação de efeitos mais complexos, bem como a busca por diferentes formas de implementação podem fazer com que o banco de efeitos aqui apresentado se expanda, de forma que o usuário tenha uma experiência cada vez mais completa dos variados tipos de efeitos existentes no mercado.

Uma outra possível evolução para o estudo seria a portabilização do banco de efeitos para outras plataformas e situações. A configuração de um computador ou de um hardware para executar somente o sistema de efeitos e com alguma facilidade para alteração de seus parâmetros pode tornar viável o uso do sistema em eventos, ensaios ou apresentações.

Com a popularização de dispositivos móveis e com as bibliotecas Csound já portáveis para os sistemais operacionais móveis mais comuns, pode ser interessante a implementação de um sistema de efeitos que funcione nestes dispositivos. A integração do código de implementação dos efeitos com um aplicativo móvel pode facilitar o uso do sistema de efeitos por parte de músicos e curiosos da área.

Apesar de o uso de efeitos de processamento de áudio se aplicar a uma parcela restrita da população, esta área específica apresenta uma grande importância para a indústria musical mundial. Desta forma, a pesquisa acerca de efeitos, seu funcionamento e diferentes formas de se produzir e aplicar persistirão enquanto a indústria musical existir. Com isto, pesquisas na área poderão ser realizadas, de forma a expandir ainda mais o conhecimento acerca do assunto de efeitos de áudio.

Referências

[1] R. Boulanger. *The Csound book: Perspectives in software synthesis, sound design, signal processing, and programming*. The MIT Press, 2000. viii, 22, 38

[2] A. Cabrera. An overview of csound variable types. 2009.

[3] A. Clarke. Digital implementation of vibrato. 2012. 24

[4] T. Drozdowski. Vibrato versus tremolo: Two classic effects, December 2012. http://www2.gibson.com/News-Lifestyle/Features/en-us/vibrato-versus-tremolo-1204-2012.aspx. 23

[5] D. Formosa. A brief history of tremolo, October 2013. http://www.premierguitar.com/articles/19777-a-brief-history-of-tremolo. 27

[6] M. Kahrs and K. Brandenburg. *Applications of Digital Signal Processing to Audio and Acoustics*. Springer, 1998.

[7] S. K. K. Mitra. *Digital Signal Processing: A Computer-Based Approach*. McGraw-Hill Higher Education, 2nd edition, 2000. 1

[8] M. J. Morrell and J. D. Reiss. Dynamic panner: An adaptative digital audio effect for spatial sound. October 2009.

[9] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-time Signal Processing (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999. vii, 1, 3, 4, 5, 6, 9, 10, 11, 12

[10] J. D. Reiss and A. McPherson. *Audio Effects: Theory, Implementation and Application*. Taylor & Francis, 2014. 24

[11] D. Roach. History of broadcast audio processing. 2007. 18

[12] C. Roads, J. Strawn, C. Abott, J. Gordon, and P. Greenspun. *The Computer Music Tutorial*. MIT Press, 1996. 24

[13] S. W. Steven. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, CA, USA, 1997. 1

[14] D. T. Yeh, J. S. Abel, A. Vladimirescu, and J. O. Smith. Numerical methods for simulation of guitar distortion circuits. 2008.

[15] U. Zölzer, editor. *DAFX: Digital Audio Effects*. Wiley, second edition, 2011. 1