CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB

CURSO DE ENGENHARIA DE COMPUTAÇÃO

GUSTAVO MAGALHÃES DOS SANTOS

EFEITOS DIGITAIS PARA GUITARRA ELÉTRICA

Orientador: Prof. Ms.C. Francisco Javier De Obaldía Díaz

Brasília

junho, 2011

GUSTAVO MAGALHÃES DOS SANTOS

EFEITOS DIGITAIS PARA GUITARRA ELÉTRICA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. **Ms.C. Francisco Javier De Obaldía Díaz**

Brasília

junho, 2011

GUSTAVO MAGALHÃES DOS SANTOS

EFEITOS DIGITAIS PARA GUITARRA ELÉTRICA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. **Ms.C. Francisco Javier De Obaldía Díaz**

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas -FATECS.

_____
Prof. Abiezer Amarilia Fernandez
Coordenador do Curso

Banca Examinadora:

_____
Francisco Javier De Obaldía Díaz
Mestre em Engenharia Elétrica

_____
Prof. João Marcos Souza Costa
UniCEUB

_____
Prof. Vera Lúcia Farine Alves Duarte
UniCEUB

_____
Prof. José Julimá Bezerra Júnior
UniCEUB

DEDICATÓRIA

Primeiramente, dedico este projeto a Deus, por toda ajuda que me deu. Ao meu pai Rogério dos Santos, que me ajudou com tudo em minha vida me mostrando o melhor caminho a seguir, que descanse em paz. Minha mãe Jair Magalhães e minha irmã Juliana Magalhães, que sempre estiveram me apoiando em todo o curso. E a Lauren Lisieux, minha fiel companheira que sempre me ajudou me amparando nas dificuldades, compartilhando minhas alegrias e tristezas, sempre ao meu lado.

AGRADECIMENTOS

A todos aqueles que de alguma forma contribuíram para a realização deste projeto, especialmente:

Aline Magalhães
Altair Magalhães
Carleuza Medeiros
Felipe Kern
Giovane Godoi
Hugo Santiago
Janaína Medeiros
John Herbert
Lúcia Magalhães
Mara Magalhães
Márcio Nobrega
Marco Terada
Pedro Mundim
Prof. Francisco Javier
Prof. Maria Marony
Solange Godoi

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE EQUAÇÕES

RESUMO

Neste projeto é apresentado um modelo de sistema para manipular ondas sonoras provenientes de uma guitarra elétrica. O sistema a ser apresentado compõe-se de um instrumento musical, no caso a guitarra elétrica, uma placa de som externa para captar as ondas sonoras produzida pelo instrumento convertendo-as de analógico a digital, um software para manipular essas ondas sonoras produzidas pela guitarra e uma caixa de som como saída de áudio dessas ondas manipuladas. O sistema a ser apresentado se assemelha aos hardwares encontrados também em acessórios periféricos de mesmo objetivo - a produção de efeitos no som do instrumento. Tais efeitos podem ser de múltiplas naturezas como no caso da distorção, delay, reverb, modulação, dentre outros. O software foi desenvolvido na linguagem de programação C utilizando a API PortAudio (biblioteca de I/O de áudio) executado em plataforma Windows. Tal software irá captar o som do instrumento e manipular as ondas de acordo com o objetivo de cada efeito.

Palavras Chave: Captação de onda, Manipulação de onda, Clipagem de onda, Atraso de onda, Modulação de onda, PortAudio, Efeitos Digitais, Guitarra Elétrica, Timbragem.

ABSTRACT

In this project is presented a model of system to manipulate sound waves from an electric guitar. The system to be presented consists of a musical instrument, in this case a electric guitar, a external sound card to get the sound waves produced by the instrument converting it from analog to digital, a software to manipulate these sound waves produced by the guitar and a sound box as a audio output of these manipulated waves. This system to be presented resembles to hardware also found in peripheral accessories of same goal – the production of effects in the instrument's sound. Such effects can be of natures multiply as in the case of distortion, delay, reverb, modulation, among many others. The software was developed on C programming language using the PortAudio API (audio I/O library) executed on windows platform. Such software will go to get the instrument's sound and manipulate the waves according with the each effect's objective.

CAPÍTULO 1 - INTRODUÇÃO

Uma das grandes dificuldades do músico profissional é a aquisição de um bom equipamento para a realização de seu trabalho, sendo de extrema importância, para ele, uma aparelhagem de boa qualidade para atender às suas necessidades, pois é dessa aparelhagem que o músico colocará qualidade em seu trabalho. Tanto em apresentações ao vivo quanto em gravações do áudio proveniente de tal equipamento.

A aquisição de um bom equipamento tem um custo muito elevado. Além do instrumento em si, o músico necessita de periféricos para um bom aproveitamento do sinal proveniente do instrumento. Esses periféricos vão desde efeitos musicais até o amplificador. Cada efeito, assim com cada amplificador, tem sua peculiaridade que os diferenciam dos demais. A boa qualidade que o músico pode retirar desse conjunto vai de acordo com o seu gosto. Outro problema é que existem inúmeros periféricos e se forem acrescentados na escolha contribuem com mais parcelas de gastos.

No mercado existem milhares de marcas e modelos de equipamentos musicais à disposição dos músicos. Para se chegar a certo tipo de qualidade sonora, ao gosto do músico, chamada aqui de timbre, é necessária a combinação de alguns dispositivos que possuem certas características. Cada dispositivo, que trata o sinal que vem do instrumento, tem suas características próprias que os diferem dos demais.

Às vezes surgem muitas dúvidas sobre qual equipamento usar, pois cada equipamento musical possui várias características. A qualidade destas é que diferencia um periférico de outro. O que para um equipamento são caracteristicas positivas, e outras negativas, em diferente equipamento isso pode se inverter. Por isso é difícil dizer qual equipamento é o melhor ou o pior. Pode-se dizer que um periférico é bom para certo tipo de efeito musical e ruim para outro.

É devido a essa diferença entre as qualidades dos equipamentos que o músico tem um gasto maior para adquirir periféricos que atendam ao seu gosto e que possa ser usado sem desperdício.

Como fazer para amenizar consideravelmente o custo, para se adquirir uma boa combinação de timbragens e efeitos, sem perder a qualidade sonora?

1.1 Objetivos do Trabalho

Os músicos enfrentam a dificuldade de encontrar uma combinação de periféricos a um preço acessível, assim como com uma qualidade de sinal de áudio que atenda às necessidades de cada um. Para atenuar essa dificuldade é necessário criar uma plataforma, de baixo custo, que simule esses periféricos musicais necessários aos trabalhos dos músicos.

Entendem-se aqui como equipamento musical somente os periféricos usados para tratar o sinal de áudio vindo do instrumento. O instrumento musical em si e a saída desse áudio em caixa acústica não fazem parte deste trabalho, deixando assim estas escolhas para cada músico.

Este projeto tem como objetivo especificar e criar uma plataforma para a manipulação das ondas sonoras provenientes de um instrumento musical, simulando o processo de manipulação das ondas sonoras concebidas por circuitos analógicos via software para que se possa então submeter tais ondas às alterações objetivadas. Serão utilizados os efeitos mais comuns usados pelos músicos, tais como Distorção, Reverb, Delay e Modulação.

Não somente estes efeitos musicais em si são de grande importância para a qualidade de som que os músicos procuram. É necessário um tratamento adequado do sinal de áudio, que vem do instrumento dos músicos, para que a saída deste sinal tenha esta boa qualidade. Esse tratamento de sinal pode ser chamado de timbre, que varia no gosto de cada músico. Então este projeto não somente simulará os efeitos musicais, oferecendo a possibilidade de combinação dos mesmos, como também a de escolhas de timbres, ao gosto do músico, para que se possa comportar com um equipamento analógico.

A aplicação do tratamento do timbre será utilizada somente no efeito de distorção, já que este é o único que faz uma alteração considerável no sinal de áudio. Este tipo de efeito cria várias outras freqüências dentro do sinal, fazendo com que o sinal de saída seja de má qualidade. Para isso será necessário o tratamento deste sinal, retirando assim essas freqüências indesejáveis.

1.2 Justificativa e Importância do Trabalho

É de grande importância para o músico um equipamento que atenda todas as suas necessidades. Disponibilizado a ele um leque de opções de timbragens e efeitos, ele pode simular a combinações destes ao seu gosto sem ter um grande custo para tal. Para um músico é importante ter um equipamento com uma boa qualidade sonora.

1.3 Escopo do Trabalho

O escopo deste trabalho é dispor ao músico um conjunto de aparelhagem a um baixo custo e com boa qualidade de sinal de áudio

Neste trabalho pretende-se manipular ondas sonoras, via software, provenientes de um instrumento musical específico. Tais manipulações serão efetuadas por um software desenvolvido em linguagem C utilizando uma biblioteca de I/O de áudio. As manipulações, chamadas aqui de efeitos, será a de Distorção.

Já que será utilizada uma placa de som externa para a conversão de sinais de áudio de analógico a digital, esse processo não será detalhado no escopo deste trabalho.

1.4 Resultados Esperados

Espera-se que, com a manipulação de ondas pelo software, os sinais de entrada do áudio sejam modificados de acordo com cada efeito pré-definido, em tempo real, e sem perda na qualidade do sinal. Tais resultados podem ser demonstrados auditivamente por comparação de amostras dos áudios de entrada com os de saída.

1.5 Estrutura do Trabalho

Este trabalho está dividido em cinco capítulos. O capítulo 1 apresenta a introdução, objetivos gerais, justificativa do trabalho e a estrutura da monografia como aqui apresentado. O capítulo 2 trata da apresentação dos problemas deste

projeto assim como as soluções sugeridas para o tratamento desses. O capítulo 3 trata tanto do funcionamento da guitarra elétrica como o funcionamento do amplificador. Também uma primeira fase de testes realizados não somente com a parte fundamental do código do programa como o código completo e seu funcionamento detalhado. No capítulo 4 é apresentado o desenvolvimento do projeto de efeitos digitais para guitarra elétrica em seu modelo proposto, etapas do modelo e sua implementação. O capitulo 5 trata da aplicação do projeto, implementação, simulação e resultados obtidos com os testes simulados e testes na implementação. Finalmente o capítulo 6 trata das conclusões e propostas de projetos futuros

CAPÍTULO 2 - APRESENTAÇÃO DO PROBLEMA

Como a aquisição de um bom equipamento, que atenda às necessidades de um músico, possui um custo um pouco elevado, uma das soluções dos problemas apresentados aqui é a disposição de um sistema que simule os equipamentos analógicos utilizados pelos músicos, a um baixo custo. Nesse caso, todas as funcionalidades das aparelhagens serão substituídas por um software que simulará todo tipo de efeitos musicais utilizados pelos músicos. Com a criação de um software para suprir as necessidades básicas de um músico, o custo do equipamento é consideravelmente reduzido. Neste caso o músico só necessitará de um computador com configurações suficientes para suportar o software sem nenhum problema e uma placa de som que tenha uma boa qualidade para a conversão de sinal analógico/digital e um auto-falante externo para a saída de áudio. Este último componente pode variar dependendo do local de uso do equipamento pelo músico.

O computador pode ser um Desktop ou um Laptop. Sugere-se usar um Laptop pois será mais cômodo na locomoção do músico. Já que ele também carregará seu instrumento musical. A máquina necessitará de uma configuração mínima de um processador de 1 GHz, memória RAM de 512MB, Disco Rígido de 40G e pelo menos uma entrada USB. Deve conter qualquer versão do sistema operacional Linux. A placa de som externa mais simples que pode ser utilizada é a M-Audio Fast Track USB. Esse periférico já possui entrada específica para instrumentos músicas. E por fim dois cabos P10 de 5 metros para instrumentos musicais. Segue tabela 2.1 o custo de um equipamento que atenda essas especificações:

Equipamento	**Quantidade**	**Preço**
M-Audio Fast Track	1	R$ 450,00
Netbook Acer	1	R$ 500,00
Cabos P10	2	R$ 60,00
Total		**R$ 1.010,00**

Tabela 2.1. Custo de equipamento

Mas o principal problema deste projeto será na etapa da distorção. A distorção aqui tratada não é nada mais do que a clipagem da onda do sinal de entrada do áudio. Essa clipagem será uma simulação digital de um processo analógico, em amplificadores ou aparelhagem para o mesmo sinal, onde o sinal de entrada sofre achatamento nas extremidades das ondas de áudio. O que provoca esse achatamento é o fato do canal de saída não suportar a amplitude do sinal de entrada. A tensão do sinal de entrada é muito maior do que a do canal de saída pode suportar. Daí que ocorrem esses achatamentos nas extremidades das ondas. Esse processo é proposital, pois modifica o sinal vindo do instrumento para um sinal bastante utilizável em ambiente musical.

Mas não somente o processo de clipagem é suficiente para criar o efeito de distorção, pois, os achatamentos das extremidades das ondas sonoras, no canal de saída do áudio, acabam criando novos componentes dentro desse sinal e isto é um problema.. A clipagem do sinal de áudio fará com que surjam novas frequências dentro do mesmo sinal. Essas novas frequências podem sujar o som vindo do instrumento.

O propósito da distorção é saturar o som vindo do instrumento. Essa saturação é muito usada pelos músicos em seus trabalhos. Mas não somente a saturação do áudio é suficiente, pois ela precisa ter boa qualidade. Essa qualidade é conhecida como timbre. É exatamente nessa parte que será tratada a timbragem do sinal do instrumento.

A timbragem do áudio é de gosto pessoal de cada música e ela tem que ser tratada de acordo com o que cada um quer. Os timbres são bastante variados, pois cada músico tem preferência por um tipo de som. É o tipo de equipamento utilizado pelo músico e o que esse equipamento pode oferecer para tratar do sinal do instrumento o que vão definir o tipo de som que sairá dele.

Esse tipo de problema somente acontecerá na clipagem da onda sonora. Isso ocorrerá no efeito de distorção. Os demais efeitos não terão influência na qualidade sonora do áudio trabalhado.

Basicamente o som de um instrumento, após o efeito de distorção, pode vir em seu timbre um reforço nas frequências graves, médias ou agudas. Vai depender de como será a reação do sinal após a clipagem desse. Este é o momento em que o músico pode escolher como ele quer que seja o sinal de saída do equipamento. Para isso, essas novas freqüências serão tratadas com filtros

digitais. Esses filtros vão trabalhar entre os três campos de frequências, graves, médios e agudos, audíveis pelo ouvido humano. Sendo assim, o sinal de saída será manipulado de acordo com o gosto do músico.

As freqüências audíveis pelo ouvido humano variam de 20Hz a 20KHz, onde estas estão dividias em quatro bandas. Grave (de 20 a 125 Hz), Médio Grave (de 125Hz a 1,25kHz), Médio Agudo (de 1,25kHz a 6,3kHz) e Agudo (de 6,3kHz a 20KHz). Os filtros irão trabalhar dentro dessas faixas, mudando o timbre do sinal proveniente do instrumento musical.

Não se deve confundir filtro de bandas com equalização do áudio. Já que o filtro irá tratar das freqüências indesejáveis enquanto a equalização faz a atenuação ou reforço de frequências dentro de uma extensão de onda do canal de entrada. Este projeto não tem como escopo o tratamento da equalização do áudio.

Os filtros digitais são usados em duas ocasiões: separar sinais que estão combinados e restaurar sinais que foram distorcidos em algum ponto. Os filtros digitais que serão utilizados neste projeto somente serão utilizados no domínio da freqüência, que podem ser de três tipos: Filtros passa-baixa; Filtros passa-faixa; ou Filtros passa-alta. A utilização desses filtros neste projeto será somente o de separar sinais combinados, bloqueando a passagem de certas freqüências indesejáveis.

Chama-se aqui de freqüências indesejáveis as criadas pelo processo de clipagem do áudio que acabam não deixando o som vindo do instrumento muito agradável ao ouvido do músico. Por isso a necessidade de tratar o sinal de saída do áudio com filtros após o processo de clipagem. O tipo de filtro vai depender de quais novas freqüências foram criadas com a clipagem. Existe também a escolha do filtro pelo gosto do músico onde o sinal de saída pode ser tratado simplesmente por preferência de timbre pelo músico não sendo necessário retirar certos tipos de freqüências criadas no processo de clipagem.

Como já citado, no momento de clipagem do sinal de áudio ocorre o achatamento das extremidades das ondas. Nesse achatamento são geradas novas freqüências dentro do mesmo sinal. As novas freqüências podem acarretar em um timbre não muito desejável pelo músico. Para o melhoramento do timbre o músico poderá definir alguns filtros para tratar o sinal de saída do áudio, deixando assim a qualidade do mesmo de acordo com o gosto do músico.

O filtro passa-baixa deixa somente passar as freqüências mais baixas, bloqueando as freqüências médias e as mais altas. Esse filtro é muito utilizado em casos onde se quer tirar ruídos de um sinal. Os demais filtros serão derivados do filtro passa-baixa. O filtro passa-alta, ao contrário do filtro passa-baixa, já impede a passagem das freqüências médias e mais baixas. Por fim o filtro passa-faixa bloqueia freqüências mais altas e mais graves deixando passar somente as freqüências da faixa média.

Toda manipulação de áudio será efetuada por uma biblioteca específica para isso. A biblioteca PortAudio é multi-plataforma e de código aberto. Ela é uma biblioteca programada em linguagem C e será utilizada em plataforma Linux configurado com o subsistema de áudio ALSA. Será utilizado um código pré-definido já existente dentro dessa biblioteca e a partir dele serão feitas as complementações de tratamento do áudio com os filtros.

Toda captação do sinal de áudio vindo do instrumento será efetuada por uma placa de som externa M-AUDIO fast track. Esta placa realizará toda a conversão do sinal, tanto de analógico/digital, como de digital/analógico. O projeto do filtro digital se inicializará pelo estudo das características do sinal produzido pelo instrumento, após o processo de clipagem de áudio, tendo em vista quais freqüências são consideradas sujas dentro do espectro sonoro. Em seguida serão definidos parâmetros da frequência de corte (fc), a partir da qual se define a atenuação da banda passante e da banda rejeitada, que serão utilizadas na filtragem, o comprimento do kernel do filtro (M) e o tipo de filtro a ser utilizado. Após isso se inicia o projeto do filtro digital, em seu algoritmo de filtragem, e por fim a implementação em software. A característica do sinal produzida pelo instrumento é analisada pela Transformada de Fourier já que ela é uma transformada integral que expressa uma função como soma ou integral de funções sinusoidais multiplicadas por coeficientes, sendo eles amplitudes (UFRJ fonte: http://www02.lps.ufrj.br/~sergioln/theses/bsc19diegosilva.pdf, acessado em 10/06/11). A Série de Fourier é a representação de uma função periódica como a soma de funções periódicas. São formas de representar funções como soma de exponenciais ou senóides (UFPE fonte: http://www2.ee.ufpe.br/codec/series1.pdf, acessado em 10/06/11).

A aplicação da Transformada de Fourier em processamento de sinais é utilizada para decompor um sinal em suas componentes em frequência e suas

amplitudes, pois pela Série de Fourier um sinal é o somatório de vários outros sinais (UFRJ fonte: http://www02.lps.ufrj.br/~sergioln/theses/bsc19diegosilva.pdf). Logo, aplicando a Transformada de Fourier Inversa sobre um sinal, vamos obter os sinais derivados deste e suas amplitudes.

Como o sistema a ser considerado é um filtro digital, que é um sistema invariante no tempo discreto, a resposta ao impulso é chamada de Filtro Kernel. *O filtro tem como objetivo de selecionar determinada faixa de freqüência de uma entrada qualquer* (IFES – fonte: ftp://ftp.cefetes.br/Cursos/EngenhariaEletrica/Salomao/Sinais_e_Sistemas/Aulas_ppt/aula_Filtros1.pdf, acessado em 10/06/11). E já que a saída do sistema depende de um número finito de entradas, o filtro digital que será utilizado nesse projeto será o filtro de Resposta Finita ao Impulso (FIR – Finite Impulse Response). A função desse sistema no domínio da freqüência será:

$$H(e^{j\omega}) = H_c(j\frac{\omega}{T}), \quad |\omega| < \pi. \qquad \text{EQUAÇÃO 2.1}$$

No entanto a resposta ao impulso deste filtro é representada pela Transformada de Fourier Inversa:

$$h_d[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d\left(e^{j\omega}\right)e^{j\omega n}d\omega = \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} 1e^{-j\omega\alpha}e^{j\omega n}d\omega =$$

$$= \frac{\omega_c}{\pi}\frac{sen\left[\omega_c(n-\alpha)\right]}{\omega_c(n-\alpha)} \qquad \text{EQUAÇÃO 2.2}$$

Na Transformada Inversa, a resposta é uma função de comprimento infinito quando observado no tempo. Logo um filtro ideal é impossível de ser aplicado na prática. O Filtro FIR tem a característica de a resposta ao impulso ser nula após o tempo finito. Por ter resposta de fase linear com aritmética real, não tem problemas de estabilidade e ter implementação eficiente através da utilização de Transformada de Fourier é vantajoso utilizar esse tipo de filtro ao invés do filtro de Resposta Infinita ao Impulso (IIR – Infinite Impulse Response). Num filtro ideal, se aplicarmos a TDF inversa sobre a resposta em freqüência, obtém-se a resposta ao impulso

correspondente (figura 2.1). No MatLab a função fir1() é responsável pela determinação de parâmetros do filtro FIR. Para se obter a resposta ao impulso do filtro utilizado é necessário o uso da Transformada Rápida de Fourier sobre a convolução do sinal de saída com o filtro. Para a convolução temos a função conv() e para a transformada, a função fft(). Logo, com plot(abs(fft(conv(saida,fir1())))) obtemos o gráfico da resposta ao impulso do filtro.

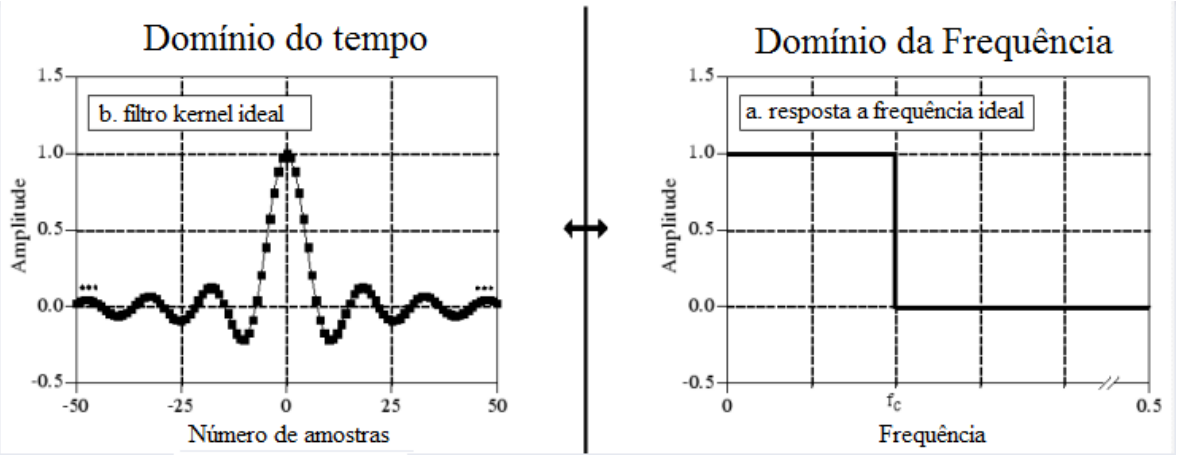

FIGURA 2.1 – Filtro Kernel ideal - (http://www.dspguide.com/CH6.PDF, acessado em 10/04/11)

Mas essa resposta ao impulso é definida em –∞ a +∞. Neste caso é preciso truncá-lo para representá-lo no computador. Porém esse truncamento faz com que a resposta ao impulso tenha um final abrupto e isso vai gerar ondulações na resposta em freqüência, tanto na banda passante quanto na banda de corte (figura 2.2) (fonte: 5871-DSP-Guide)

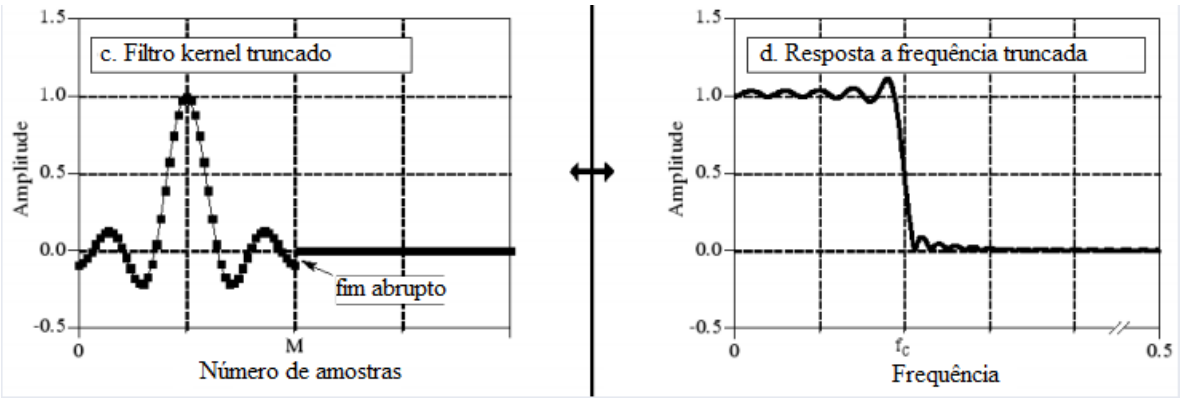

FIGURA 2.2 – Filtro Kernel Truncado - (http://www.dspguide.com/CH6.PDF, acessado em 10/04/11)

Para diminuir essa ondulação é necessário truncar o Filtro Kernel utilizando uma janela mais suave. Para este caso poderia multiplicar o sinal por uma janela de Hamming ou a Janela Blackman (figura 2.3) (fonte: 5871-DSP-Guide).

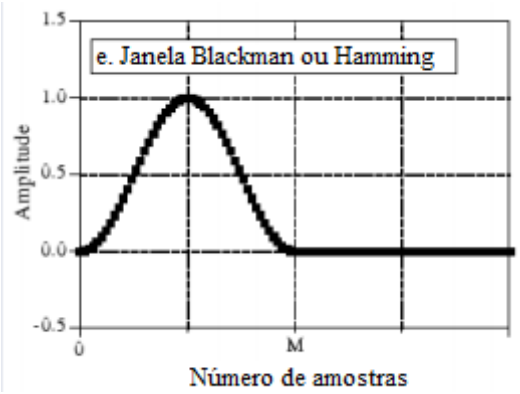

FIGURA 2.3 – Janela Blackman ou Hamming - (http://www.dspguide.com/CH6.PDF, acessado em 10/04/11)

Com isso a resposta ao impulso não teria um final muito abrupto. Logo, a freqüência de resposta não sofreria ondulações (figura 2.4) (fonte: 5871-DSP-Guide).

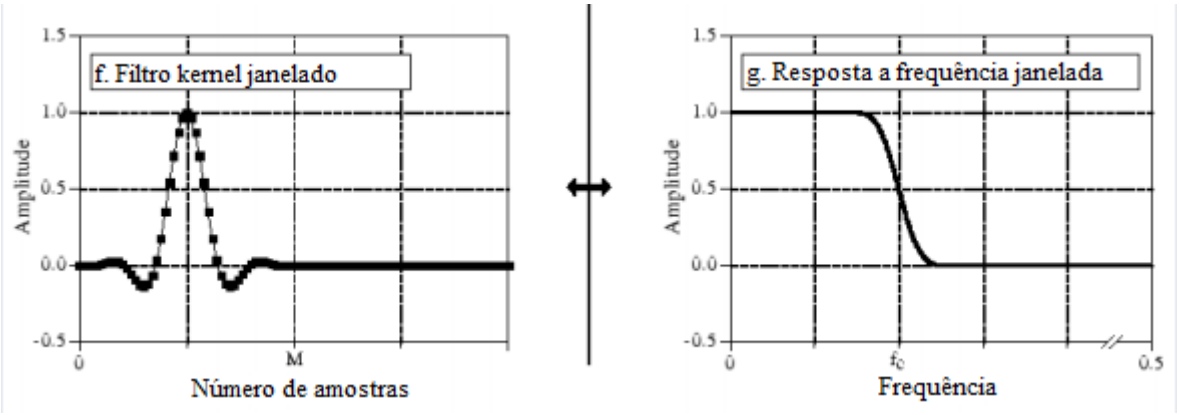

FIGURA 2.4 – Filtro Kernel Janelado - (http://www.dspguide.com/CH6.PDF, acessado em 10/04/11)

Logo, para implementar um filtro na prática basta multiplicá-lo por uma janela *w*[n] de comprimento N (equação 2.3). Com isso o truncamento da reposta ao impulso seria suavizado pela janela que fosse utilizada.

$$H_d\left(e^{j\omega}\right) \Leftrightarrow h[n] = h_d[n]w[n] \qquad \text{EQUAÇÃO 2.3}$$

A metodologia escolhida neste projeto para se projetar o filtro kernel será á Janela de Blackman, devido a sua atenuação maior da banda rejeitada:

$$w[n] = 0.42 - 0.5 \cos(2\pi n / M) + 0.08 \cos(4\pi n / M)$$ EQUAÇÃO 2.4

CAPÍTULO 3 – BASES METODOLÓGICAS PARA RESOLUÇÃO DO PROBLEMA

Neste capítulo serão apresentadas todas as bases metodológicas para a resolução do problema apresentado no capítulo anterior. Tanto como o funcionamento da guitarra elétrica, desde a criação da onda, da captação até a transmissão dela ao amplificador, o funcionamento de amplificador em uma onda criada pela guitarra, como dos periféricos analógicos e toda a sua aplicação no ambiente digital.

3.1 Guitarra Elétrica

A guitarra elétrica é originada do violão, que é um instrumento acústico com origem Européia. O violão era um instrumento utilizado em orquestras e veio perdendo seu espaço nela devido a seu baixo volume. Por ser um instrumento acústico com pouca ressonância, comparado com os outros instrumentos de uma orquestra, tais como piano, instrumentos de sopro, violinos dentre outros, ele não conseguia se destacar tão bem quanto os outros.

Aos poucos o violão foi deixando de ser executado em orquestras e passando a ser executado em música de câmara, que é formada por um número menor de instrumentos, assim por conseqüência se destacando melhor dentro da massa sonora. Pelo seu formato, tamanho e timbre, o violão acabou caindo no gosto popular de diversas culturas européias, sendo usado bastante em músicas cantadas.

Não só o violão, como muitos instrumentos de cordas, estava perdendo espaço nas orquestras devido ao seu volume não ser alto suficiente para se destacar dentre os outros instrumentos musicais. Por conta disso várias pessoas começaram uma busca para o desenvolvimento de algum sistema que pudesse amplificar o som desses instrumentos musicais.

Na década de 30, já se tinha o conhecimento de que uma corrente elétrica, em uma bobina magnética, poderia ser criada movendo-se um metal dentro desse campo magnético. Devido a isso, Beauchamp, após várias tentativas, criou um captador usando dois ímãs dentro de uma bobina com seis pólos magnéticos onde por cada um deles passava uma corda de metal do instrumento. Nesse momento foi

criado o primeiro captador de som de instrumentos de cordas. A partir daí várias empresas decidiram desenvolver captadores, para instrumentos de cordas, onde influenciaram todas as construções das guitarras. (Revista Souza Lima – 31/05/2010)

No início, todas as guitarras fabricadas eram acústicas ou semi-ácusticas. Foi devido a Charlie Christian, guitarrista de jazz, que a guitarra elétrica, por meio do modelo ES-150, se popularizou e se transformou. Deixando assim de ser um instrumento de acompanhamento e se tornando também um instrumento solista.

A guitarra elétrica (figura 3.1) é um instrumento musical formado por três partes comumente feitas de madeira ou de material sintético que compõem um sistema. São elas: mão, braço e corpo.

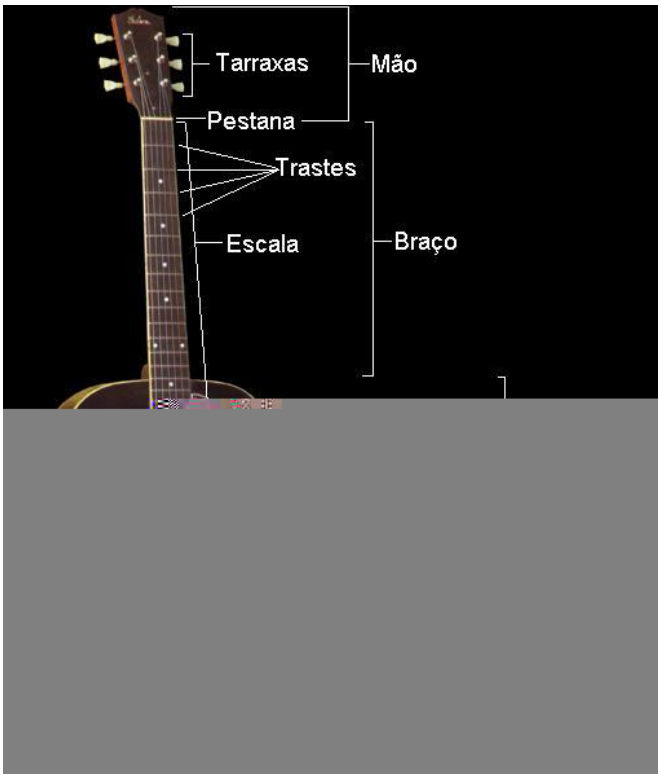

FIGURA 3.1 - Guitarra Elétrica

A mão é composta pelas tarraxas onde são presas as cordas. Essas tarraxas têm como função tencionar a corda do instrumento. Tal tensão é utilizada para a afinação das cordas, uma em relação à outra.

O braço do instrumento é a área onde existem marcações de metal
separadas por distâncias distintas.
Essas marcações, chamadas trastes, são utilizadas para mudar as frequências em cada corda, dando assim mais disponibilidade de notas musicais.

O corpo é a parte do instrumento onde se encontra todo o circuito que é utilizado para transformar as vibrações das cordas em sinais elétricos.

Cada característica da guitarra influencia em seu timbre. Essas características podem ser o tipo e forma do material que compõe o instrumento, madeira ou sintético, o tipo de cordas utilizadas, o tipo dos captadores e circuito que fazem parte do sistema (fonte: TIMBRES E EFEITOS PARA GUITARRA - Por

Márcio Rocha - http://www.seradorador.com.br/index.php?pag=artigos&acao=Ler&id=12, acessado em 22/04/11).

3.1.1 Funcionamento

A guitarra elétrica funciona basicamente transformando ondas mecânicas, formadas por vibrações propositais das cordas, em ondas elétricas. As freqüências das vibrações são determinadas pelo comprimento e tensão de cada corda. O comprimento é determinado pela fricção da corda em determinado traste da escala do braço do instrumento.

Diferentes dos violões, que necessitam de um microfone para captar seu som, as guitarras possuem captadores que transformam as vibrações mecânicas de suas cordas em sinais elétricos. O captador é o coração do instrumento. Ele é o único responsável pela transformação das ondas mecânicas em sinais elétricos. Tais sinais são transmitidos para um equipamento específico, conhecido como amplificador, que faz amplificação desse sinal, lançando-o a um alto-falante e transformando-o em ondas sonoras.

Os captadores encontrados nas guitarras elétricas são na verdade transdutores, nome dado ao dispositivo que converte energia física em energia elétrica. Eles transformam a energia das vibrações das cordas em impulsos elétricos de corrente alternada. O captador magnético não é nada mais do que uma bobina formada por um ímã permanente envolto em um enrolamento de fio de cobre. Uma corda quando friccionada vibra e essa vibração perto de uma bobina magnética é transformada em impulsos elétricos de corrente alternada. São esses impulsos que são enviados ao amplificador que tem a função de amplificar os sinais e transformá-los em energia sonora novamente.

Existem diversos tipos de captadores de diferentes qualidades sonoras. Essas qualidades podem diferencia-se de acordo com o tipo de captador escolhido. Os captadores podem ser ou não ser alimentados por uma fonte de energia adicional, exemplo, uma bateria de 9V. Quando não alimentados por uma fonte de energia, esses sistemas são passivos, enquanto os alimentados

Figura 3.2 Captador Single-coil

são sistemas ativos. A diferença entre esses dois sistemas é que no sistema passivo todo material do instrumento influencia na captação das vibrações das cordas e no ativo o instrumento quase não influencia no timbre. Mas nesse último o sinal captado é tratado, amplificando, intensificando e equalizando as freqüências capturadas pelas vibrações das cordas. Qualquer um dos sistemas pode ser combinado com circuitos ativos para melhoria do timbre do instrumento.

Outra influência na qualidade sonora dos captadores é pela quantidade de bobinas. Elas podem ser simples (single-coil) (figura 3.2), duplas (humbucker) (figura 3.3) ou quádruplas (quad-rail) (figura 3.4). Foram criados os captadores de bobinas duplas, pois os de bobinas simples captam muitas interferências. As bobinas em humbuckers são enroladas em série, mas defasadas. Com as correntes elétricas viajando a sentidos opostos todo o ruído captado era anulado. Mas devido a isso também há um corte nas freqüências mais altas captadas por eles criando assim a diferença em se utilizar captadores single-coils e humbuckers.

Figura 3.4
Captador
Quad-rail

Figura 3.3
Captador
Humbucker

Há também a influência do tipo de ímã utilizado pelos captadores que podem ser cerâmicos ou de alnico (Luighi, Edmar - Guia Ilustrado do Baixo, Edição 1).

3.2 Amplificadores

Neste projeto, os amplificadores serão definidos como equipamentos utilizados com a guitarra. O amplificador tem a função de pegar o sinal elétrico produzido pela guitarra e amplificá-lo. Isto é necessário, pois a guitarra produz um sinal fraco indispensável de amplificação.

Não somente o amplificador é utilizado para amplificar as ondas produzidas pela guitarra assim como para modificá-las. Os amplificadores têm como padrão a característica de saturação do sinal da guitarra. Amplificadores mais modernos já possuem alguns efeitos musicais a mais que permitem ao músico a escolha desses efeitos.

Válvulas e transistores são os principais componentes de um amplificador.

3.3 Simulação

Neste ponto será abordado o processo de vibração de uma das cordas da guitarra, seus efeitos e a aplicação da clipagem da onda sobre essa nota. Será simulada a vibração da corda lá da guitarra elétrica e seus harmônicos.

Como já citado a Série de Fourier é a soma de um conjunto de funções periódicas, chamadas de harmônicas, onde estas são representadas como funções de seno e co-seno multiplicados por um coeficiente, a amplitude. Na Série de Fourier uma função é representada pela sobreposição de seus harmônicos.

Cada corpo tem uma característica peculiar de freqüências. Isto é, a freqüência de uma nota criada em um instrumento musical, em conjunto com as características físicas deste instrumento, acabam excitando outras freqüências da cadeia harmônica da nota fundamental, chamada de Série Harmônica. Essa série obedece à ordem de múltiplos inteiros da freqüência fundamental.O que define a diferença de amplitudes de uma série harmônica não é somente a corda que é tocada, mas sim a dimensão, o material, tanto da corda quanto do instrumento, e o tipo de captação utilizado na guitarra elétrica. A variação de amplitude da Série Harmônica é o que caracteriza o timbre do instrumento. É por causa disso que conseguimos diferenciar, para uma mesma nota, o som de uma guitarra elétrica ao de um piano.

A onda produzida pela guitarra elétrica será representada por uma função seno, tanto para sua frequência fundamental como para seu harmônicos, já que as vibrações das cordas representam funções periódicas. Assim, a vibração da corda da guitarra será o somatório de todos os seus harmônicos.

Usando a fórmula $x(i)=A.sen(2p.f.t+f)$, onde A é a amplitude da onda (em radianos), f é a freqüência utilizada, t é o tempo (em segundos) e f é a fase da onda, será simulado em MatLab a vibração da corda lá. Para se obter um bom resultado nas simulações, ao invés de utilizar somente um sinal senoidal, serão utilizados também os harmônicos da freqüência fundamental. Logo, como já foram citados, esses harmônicos são os que definem o timbre do instrumento. Chega-se assim a forma de realizar uma simulação bem próxima da realidade.

A vibração de uma corda da guitarra obedece a Série de Fourier, onde é o somatório de várias funções periódicas. Cada harmônico aqui será representado por uma função seno. Se tratando de Série de Fourier e utilizando-se a Transformada Inversa de Fourier sobre a função da vibração de uma das cordas da guitarra, podemos identificar as componentes da Série de Fourier, isto é, podemos identificar todas as funções seno de todos os harmônicos que compõe o sinal.

Usam-se os harmônicos gerados pela excitação da corda lá de uma guitarra Yamaha modelo Eterna EG-303, onde foi utilizada no experimento da Revista Brasileira de Ensino de Física, vol. 24, no. 2, Junho, 2002. Nesse experimento foi aplicada uma transformada rápida de Fourier sobre onda produzida pela guitarra e simultaneamente registrada por um osciloscópio. E assim lhes proporcionando os dados numéricos (tabela 3.1) referente aos harmônicos criados pela excitação da corda lá do instrumento. (Revista Brasileira de Ensino de Física, vol. 24, no. 2, Junho, 2002).

Tabela 3.1 Harmônicos da corda lá.
(http://www.sbfisica.org.br/rbef/pdf/v24_129.pdf acessado em 20/04/11)

Neste projeto, com os dados extraídos da tabela será gerado por MatLab a onda fundamental com o somatório dos seus harmônicos. Assim, cada harmônico está representado pelas equações: 0,42.sen(2.pi.107,4.(t-25,4)) (figura 3.5), 0,81.sen(2.pi.214,8.(t-125)) (figura 3.6), sen(2.pi.322,3.(t-220)) (figura 3.7), 0,83.sen(2.pi.429.7.(t-300)) (figura 3.8), 0,5.sen(2.pi.537,1.(t-376)) (figura 3.9), 0,17.sen(2.pi.644,5.(t-457)) (figura 3.10), 0,1.sen(2.pi.751.9.(t-272)) (figura 3.11), 0,19.sen(2.pi.859,4.(t-332)) (figura 3.12), 0,2.sen(2.pi.966,8.(t-702)) (figura 3.13) e 0,1.sen(2.pi.1074,2.(t-678)) (figura 3.14).

Figura 3.5 – 1º Harmônico

Figura 3.6 – 2º Harmônico

Figura 3.7 – 3º Harmônico

Figura 3.8 – 4º Harmônico

Figura 3.9 – 5º Harmônico

Figura 3.10 – 6º Harmônico

Figura 3.11 – 7º Harmônico

Figura 3.12 – 8º Harmônico

Figura 3.13 – 9º Harmônico

Figura 3.14 – 10º Harmônico

Para uma melhor aproximação da realidade, a simulação da corda lá será feita com a onda completa, isto é, com o somatório de todos os seus harmônicos (figura 3.15).

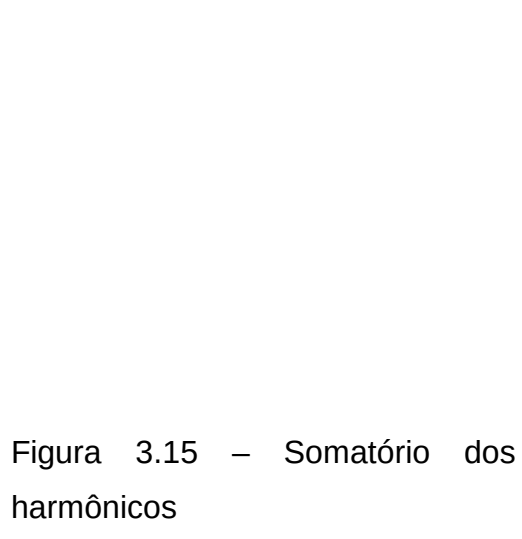

Figura 3.15 – Somatório dos harmônicos

A distorção é o resultado da não-linearidade do sinal amplificado. Um dentre vários métodos utilizados para gerar distorção é a clipagem de sinal (onde parte superior do sinal fica achatado) (figura 3.16). Isto ocorre quando se excede a quantidade de energia suportada por um sistema. Em um sistema analógico a clipagem é obtida por saturação do sinal, onde o ganho do sinal de entrada deixa de ser linear com o ganho do sinal de saída. A linearidade acaba quando o sistema não suporta mais o ganho do sinal de entrada.

No domínio digital tal clipagem é resultado do achatamento das cristas da onda do sinal de entrada. Para isso é necessário aplicar uma limitação na amplitude do sinal de entrada.

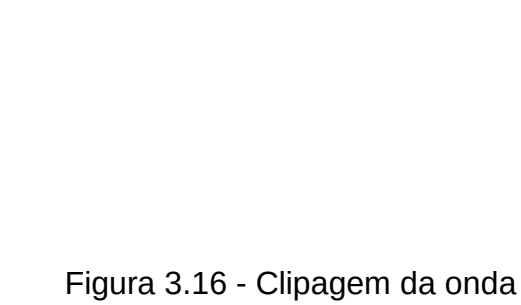

Figura 3.16 - Clipagem da onda

O processo de distorção abrange vários processos como o de amplificação, o de clipagem e o de filtragem. Esses processos podem ser realizados repetidas vezes. Chama-se aqui cada repetição de fase. Além da clipagem, a distorção pode ser realizada por alteração da onda, tanto em determinados valores das ondas, quanto alteração para uma onda quadrada ou triangular.

Devido à qualidade sonora do áudio, a clipagem será combinada com filtros para atenuar a nova gama de freqüências criadas pela clipagem e distorção do sinal.

A amplificação da onda se dá por a onda perder amplitude no momento da clipagem dela. E com o achatamento das cristas, são criadas quinas nas ondas, onde essas geram novas freqüências que afetam na sonoridade. Essas novas freqüências são tratadas por filtros para melhorar a qualidade do som.

A extensão das frequências fundamentais das notas da guitarra elétrica variam entre 82,5 Hz a 1K Hz. Cada nota tem sua séria harmônica distinta. Devido ao processo de distorção e clipagem do sinal de áudio, no sinal de saída surgem frequências mais agudas assim como o reforço de frequências médias. Frequências mais baixas não sofrem muitos reforços. Todos esses resultados serão demonstrados neste projeto.

Devido a essas gama de frequências criadas e reforçadas por conta do processo de amplificação e clipagem do áudio, o sinal de saída chega carregado dessas novas frequências. O sinal de saída, para o músico, acaba ficando saturado de frequências indesejáveis. Para amenizar esses resultados, os filtros são usados para atenuar algumas frequências indesejáveis.

Como o sinal de saída depende do sinal de entrada, filtros não-recursivos serão utilizados neste projeto. Filtros FIR se adequam perfeitamente neste projeto pois são filtros estáveis. Como filtros ideais trabalham com tempo infinito, é necessário o truncamento da resposta ao impulso para se projetar um filtro digital. Consequentemente esse truncamento gera variações perto da frequência de corte deixando o filtro não trabalhar próximo a uma faixa ideal de frequência de corte. Essas ondulações serão amenizadas usando-se uma janela para esse filtro digital

3.4 Conceitos básico de PortAudio

PortAudio é um biblioteca I/O de áudio, gratuito e de código aberto. Sua programação é feita em Linguagem C onde pode ser compilado e rodado em várias plataformas tais como Windows, Macintosh, Unix, SGI e BeOS. O PortAudio se destina a promover a troca de softwares de áudio entre programadores de diferentes plataformas.

Essa biblioteca permite fazer qualquer manipulação no sinal recebido pela entrada de áudio do computador. Seu principal benefício é que ela é uma simples API de gravação e reprodução de som que usa uma função de callback. Com ela

pode-se criar programas que sintetizam ondas sonoras ou até mesmo sobre ondas de instrumentos musicais (http://www.portaudio.com/, acessado em 20/02/11).

O sistema será executado em plataforma Linux, versão Ferdoa 15, usando subsistema de áudio ALSA. O funcionamento básico da API é armazenar uma parte do áudio captado pelo canal de entrada em um buffer e dividir esse buffer em frames. Dentro de cada frame ocorre a manipulação do áudio e o envio para o canal de saída dele.

O código da blibioteca a ser usado será o pa_fuzz.c, onde se encontra no anexo do projeto. Este código será responsável pela distorção do sinal. Podendo ser alterado o algoritmo para tal.

CAPÍTULO 4 – EFEITOS DIGITAIS PARA GUITARRA ELÉTRIC A

Neste capítulo será apresentado todo o funcionamento do projeto proposto, assim como suas conexões e interfaces. O objetivo do trabalho é produzir um som a um nível desejável pelo músico, tratando todos os ruídos indesejáveis, criados pelo processo de distorção, por filtros digitais. O resultado pretendido com o modelo é cortar as freqüências mais altas criadas pela distorção do sinal usando o filtro passa-baixas.

4.1 Apresentação Geral do Modelo Proposto

Neste projeto serão utilizados: uma guitarra elétrica (1) ligada a uma placa de som externa M-AUDIO (2) por meio de um cabo de áudio P10. A placa é ligada ao notebook (3) por meio de cabo USB. No notebook será instalada a biblioteca PortAudio utilizanda em plataforma Linux, versão Fedora 15. A saída de áudio da placa de som será ligada em uma saída externa adequada de áudio (4), por cabo de áudio P10 ou RCA.

Figura 4.1 – Modelo do projeto

4.1.1 Descrição das Etapas do Modelo

a) Produção da onda sonora

Nessa etapa, com a utilização de uma guitarra elétrica, como na figura 4.1, será produzida uma onda sonora com freqüência de 440 Hz. Para isto basta, simplesmente, friccionar a quinta corda (nota lá) do instrumento. Essa corda vibrará exatamente a 440 Hz, quando bem tensionada, e com o auxílio das bobinas (captadores) encontradas na guitarra, a vibração é convertida em ondas elétricas.

b) Captação da onda sonora pela placa de som:

Nessa etapa, o sinal produzido pela guitarra elétrica é recebida pela placa de som via cabo P10. A placa, devidamente regulada, captará o som produzido pela guitarra e o converterá ao universo digital, conversão Analógica/Digital (A/D). Somente com essa conversão será possível a manipulação da onda pelo software da máquina.

c) Manipulação das ondas sonoras:

É nesta etapa que o sinal convertido pela placa de som é enviado à porta de input de áudio do computador. Após isso o software armazenará, seqüencialmente, as ondas digitais em um buffer. Cada buffer será dividido em pequenas partes, chamadas frames. O software trabalhará com um frame de cada vez, seqüencialmente. Depois de trabalhado, cada frame é enviado à porta de output de áudio.

d) Envio do sinal para saída externa de áudio

Nesta etapa, o sinal é recebido novamente pela placa de som e convertido em universo analógico. Esse sinal analógico será transmitido ao dispositivo de áudio externo.

4.1.2 Descrição da Implementação

4.1.2.1 Simulação em MatLab

A simulação inicializará pelo processo de distorção da onda e amplificação em determinados pontos. Esse processo será realizado em 4 fases. O código abaixo é uma simulação em MatLab da corda lá da guitarra com seus 10 harmônicos na primeira fase.

```
fa=44100; % Frequência de amostragem
Ta=1/fa; % Período de amostragem
t=0:Ta:2; % Tempo de amostragem (em segundos)
loop = length(t);

%Somatório da freqüência fundamental e seus harmônicos
output_clean =
                    0.42*sin(2*pi*107.4*t-25.4) + 0.81*sin(2*pi*214.8*t-125) +
                    sin(2*pi*322.3*t-220) + 0.83*sin(2*pi*429.7*t-300) +
                    0.5*sin(2*pi*537.1*t-376) + 0.17*sin(2*pi*644.5*t-457) +
                    0.1*sin(2*pi*751.9*t-272) + 0.19*sin(2*pi*859.4*t-332) +
                    0.2*sin(2*pi*966.8*t-702) + 0.1*sin(2*pi*1074.2*t-678);

%Amplificação do sinal com leve distorção.
for i=1:1:loop
input(i) =
                0.42*sin(2*pi*107.4*t(i)-25.4) + 0.81*sin(2*pi*214.8*t(i)-125) +
                sin(2*pi*322.3*t(i)-220) + 0.83*sin(2*pi*429.7*t(i)-300) +
                0.5*sin(2*pi*537.1*t(i)-376) + 0.17*sin(2*pi*644.5*t(i)-457) +
                0.1*sin(2*pi*751.9*t(i)-272) + 0.19*sin(2*pi*859.4*t(i)-332) +
                0.2*sin(2*pi*966.8*t(i)-702) + 0.1*sin(2*pi*1074.2*t(i)-678);

if input(i)<0
      temp=input(i)+1;
```

```
        output(i)=(temp*temp*temp)-1;
    else
        temp=input(i)-1;
        output(i)=(temp*temp*temp)+1;
    end
end;
```

As outras três fases de amplificação e distorção da onda serão efetuados pelo código abaixo:

```
input = output;

for i=1:1:loop
        temp=input(i)+1;
        output(i)=(temp*temp*temp)-1;
    else
        temp=input(i)-1;
        output(i)=(temp*temp*temp)+1;
    end
end;
```

O processo de simulação da distorção na onda passou por 4 fases: Fase 1 (figura 4.2), Fase 2 (figura 4.3), Fase 3 (figura 4.4) e Fase 4 (figura 4.5). As ondas foram atenuadas somente para melhor visualização dos gráficos.

Figura 4.2 – Distorção Fase 1

Figura 4.3 - Distorção Fase 2

Figura 4.4 – Distorção fase 4

Figura 4.5 – Distorção fase 5

Foram criados cinco arquivos de áudio, com dois segundos de duração cada um, em todas as fases do processo de distorção. Desde a simulação da corda lá às quatro fases de amplificação e distorção. No processo de criação dos arquivos, o áudio passou por um processo de clipagem devido ao MatLab só suportar valores de amplitude até 1 (um). Acima desse valor, as ondas sofreram clipagem nas cristas.

Aplicando-se a Transformada de Fourier no áudio, constatamos que com o processo de distorção e clipagem da onda surgiram novas freqüências dentro do sinal. Sendo assim necessário o tratamento desse sinal para retirar essas freqüências indesejáveis (figuras 4.6 e 4.7).

Figura 4.6 – Onda limpa - FFT

Figura 4.7 - Onda distorcida - FFT

Foram realizados testes em sinais de áudio da vibração de uma das cordas da guitarra. Nota-se também aqui o surgimento de novas freqüências dentro do espectro do áudio (figuras 4.8 e 4.9).

Firgura 4.8 – Áudio Limpo - FFT

Figura 4.9 – Áudio distorcido - FFT

Nota-se tanto na figura 4.7 quanto na 4.9 o reforço das frequências fundamentais e o surgimento de frequências mais altas. Neste caso, na saída do áudio para o músico surge um áudio mais agudo um pouco incômodo aos ouvidos dele. Para isso é necessário o uso de um filtro digital passa-baixas para a atenuação das frequências mais altas. O corte dessas frequências são arbitrárias, podendo ser escolhidas pelo músico as que lhe agradam.

Nesta etapa da simulação será aplicado o filtro passa-baixas para tratamento do sinal de áudio após o processo de distorção. O filtro utilizado possui ordem N=110, tendo 20K Hz como freqüência de corte usando a janela de Blackman (N+1) (figuras 4.10 e 4.11).

Figrua 4.10 – Onda filtrada

Figura 4.11 - Áudio filtrado

Figura 4.12 – Onda filtrada – Janela 110

Figura 4.13 – Onda filtrada - Janela 330

Nas figuras 4.12 e 4.13 nota-se que o atraso do sinal é diretamente proporcional ao tamanho utilizado no filtro digital. Um atraso de até 5 milissegundos é tolerável, já que o ouvido humano não consegue distinguir atrasos abaixo desse tempo (fonte: Valle, Sólon do – Manual Prático de Acústica, 1ª edição, 2006)

Para uma melhor qualidade de sinal, uma frequência de amostragem de 44,1K Hz, em formato 32 bits e buffer de tamanho 64 serão utilizados no programa.

No programa o processo de distorção é feito pela função CubicAmplifier(), onde a função pega cada ponto do áudio e faz cálculos sobre esses valores obtendo assim uma distorção do sinal. A função FUZZ() é a realimentação da função de distorção em três vezes. Totalizando quatro fases de distorção. Toda essa parte é realizada pela função fuzzCallback(), que é uma função própria da biblioteca PortAudio.

Para uma melhor percepção sonora do áudio trabalhado a saída do sinal será em stereo, onde o canal L será da distorção já com o tratamento com filtro e o canal R com o som limpo. Sendo necessário somente um canal mono para entrada de sinal. Tais parâmetros são setados em intputParametres.channelCount e outputParametres.channelCount.

O filtro, que se encontra no anexo do projeto, será gerado pela função wsfirLP() com a janela Blackman e será adicionado ao sinal de saída da distorção. Esta função gera o filtro em convolução com a janela escolhida, necessário somente passar como parâmetro o tipo de janela, o tamanho do filtro e a frequência de corte.

A ordem de todo processo é: Abertura da stream com Pa_OpenStream, onde é setado a stream, parâmetros de input, parâmetros de output, frequência de amostragem, tamanho de buffer, opção de clipagem de áudio e chamada da função fuzzCallback; Em seguida indiciamento da stream com a função Pa_StartStream() para liberação do áudio; Para parar o programa é utilizado a função Pa_CloseStream() e em seguida o fechamento da stream de áudio com a função Pa_Terminate().

CAPÍTULO 5 - APLICAÇÃO DO MODELO PROPOSTO

5.1 Apresentação da área de Aplicação do modelo

O projeto apresentado é uma ferramenta bastante utilizada pelos músicos guitarristas. A área de aplicação do modelo proposto é somente em ambiente musical. Podendo ser aplicada em estúdios de gravações ou ensaios, escolas de música ou até em espetáculos ao vivo.

5.2 Descrição da Aplicação do Modelo

A distorção é um efeito musical muito comum nos trabalhos dos músicos. A timbragem é determinada não somente pela equalização do instrumento pelo amplificador mas também pela qualidade de componentes de tratamento de áudio do equipamento.

Esta qualidade também é tratada por meio de filtros de sinais. Como já citado no trabalho o processo de distorção acaba sujando o som do instrumento de modo que não possa ser apreciado pelo músico. Cabe aqui o tratamento do sinal do instrumento por meio de filtros. A diferença entre os equipamento analógicos, em relação à distorção, é o tipo de ciclo de amplificação e a aplicação de filtro sobre cada um.

5.3 Avaliação Global do Modelo

O tratamento do áudio pode ser feito de muitas maneiras distintas. Neste projeto o tratamento do áudio pelo filtro foi realizado somente com um tipo de filtro, passa-baixas, no final de todos os ciclos de amplificação do sinal. O resultado não foi totalmente satisfatório podendo ter melhor desempenho usando filtros digitais, passa-faixa ou rejeita-faixa no final de todo o ciclo de amplificação como dentro de cada um.

CAPÍTULO 6 - CONCLUSÃO

6.1 Conclusões

O sistema em si apresentou erros na implementação. No código do PortAudio somente um dos canais de saída apresentava sinal. Teve-se que mudar o código da distorção para o canal que estava funcionando. Também não foi possível incorporar o código do filtro dentro do código do PortAudio, pois no momento de compilar o código completo surgiram erros.

O filtro FIR com o método da Janela de Blackman obteve boa resposta sobre o sinal de saída. Os cortes das freqüências altas foram eficazes. Nota-se um pequeno atraso do sinal devido ao dimensionamento da janela utilizada. Neste caso, pode ser corrigido usando uma janela de tamanho menor.

De acordo com análise do espectro do sinal de saída, nota-se um grande quantidade de novas freqüências produzidas pelo processo de distorção e clipagem do áudio

A qualidade do timbre não teve seu objetivo atingido, pois somente foi usado filtros digitais para cortes de frequências agudas. Como se obteve reforço das frequências fundamentais da onda original pelo processo de distorção, seria adequado o uso de filtros digitais passa-faixa ou rejeita-faixa. Tais filtros pode ser utilizados também em cada estágio de distorção do sinal ao invés de ser usado somente no sinal de saída após todo o processo de distorção, como foi neste projeto.

Não é necessário tratar com filtros passa-altas, pois não se teve muito ganho nas freqüências mais baixas, ao contrário das freqüências médias e agudas que tiveram um ganho considerável.

6.2 Sugestões para Trabalhos Futuros

O processo de filtragem foi utilizado somente no sinal de saída após todo o processo de distorção e clipagem do áudio. Existem várias possibilidades de filtragens que podem ser utilizadas durante o processo ao invés de se apenas usar no final. Amplificadores de guitarras costumam usar filtros analógicos em cada fase

de amplificação do sinal. Isto é, o sinal captado no primeiro ciclo de amplificação já é filtrado, quando realimentado é novamente filtrado e assim sucessivamente. Este é um modelo de projeto que pode ter tratamento de áudio mais eficiente, podendo ser usado tipos diferentes de filtros em cada estágio com cortes de freqüências distintos.

No processo de distorção também pode ser usado diferentes tipos de algoritmos para tal, tais como algoritmo de onda quadrada, de onda triangular ou de onda em serra. Além disso podemos implementar diferentes efeitos musicais dentro do código podendo assim todo o projeto pode ser implementado para outro instrumento musical como o baixo elétrico.

REFERÊNCIAS

ARAÚJO, Leonardo C. de. Filtros Digitais. Centro de Estudos da Fala, Acústica, Linguagem e Música. 2006 em <www.cefala.org/~leoca/ee/filtros/filtros.pdf> Acessado em 15/04/11.

DSP GUIDE, http://www.dspguide.com/. Disponível em <http://www.dspguide.com/CH6.PDF> Acessado em 10/04/11.

GONÇALVES, João. MARQUES, Libânia. SALVADO, José. SOUSA, Leonel. Ferramenta para Síntese e Implementação de Filtros em DSPs. Escola Superior de Tecnologia de Castelo Branco. Instituto Superior Técnico (DEEC) e Instituto de Engenharia de Sistemas e Computadores (SIPS). Lisboa - Portugal, 2010.

INGLE, Vinay K. PROAKIS, John G. Digital Signal Processing Using MATLAB V.4. PWS Publishing Company, 1997.

LUIGHI, Edmar. Guita Ilustrado do Baixo: Manual de Conhecimentos e Reparos Essenciais. 1 ed. São Paulo: hmp, 2004.

OSORIO, Paulo Léo M. Introdução ao uso do MATLAB e da "TOOLBOX" de Sinais. DEE/PUC-RIO, 2002.

PEDRO. Apostila de Acústica Elétrica em Áudio: Ondas Estacionárias. 2005.

RODRIGUES, Diego da Silva. Redução de Ruído Aditivo: Uma Visão Uniformizada. Universidade Federal do Rio de Janeiro - Escola Politécnica - Departamento de Eletrônica e de Computação. 2008.

SILVA, Samuel da. Projeto de Filtros Digitais. Centro de Engenharias e Ciências Exatas. Universidade Estadual do Oeste do Paraná - Foz do Iguaçu, 2011.

VALLE, Sólon do. Manual Prático de Acústica. 1 ed. Rio de Janeiro: Música & Tecnologia, 2006.

ANEXOS

Distorção

```
/** @file pa_fuzz.c
        @ingroup test_src
    @brief Distort input like a fuzz box.
        @author Phil Burk  http://www.softsynth.com
*/
/*
 * $Id: pa_fuzz.c 1368 2008-03-01 00:38:27Z rossb $
 *
 * This program uses the PortAudio Portable Audio Library.
 * For more information see: http://www.portaudio.com
 * Copyright (c) 1999-2000 Ross Bencina and Phil Burk
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files
 * (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
FOR
 * ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
 */

/*
 * The text above constitutes the entire PortAudio license; however,
 * the PortAudio community also makes the following non-binding requests:
 *
 * Any person wishing to distribute modifications to the Software is
 * requested to send the modifications to the original developer so that
 * they can be incorporated into the canonical version. It is also
 * requested that these non-binding requests be included along with the
 * license above.
```

```
 */

#include <stdio.h>
#include <math.h>
#include "portaudio.h"
/*
** Note that many of the older ISA sound cards on PCs do NOT support
** full duplex audio (simultaneous record and playback).
** And some only support full duplex at lower sample rates.
*/
#define SAMPLE_RATE         (44100)
#define PA_SAMPLE_TYPE      paFloat32
#define FRAMES_PER_BUFFER   (64)

typedef float SAMPLE;

float CubicAmplifier( float input );
static int fuzzCallback( const void *inputBuffer, void *outputBuffer,
                         unsigned long framesPerBuffer,
                         const PaStreamCallbackTimeInfo* timeInfo,
                         PaStreamCallbackFlags statusFlags,
                         void *userData );

/* Non-linear amplifier with soft distortion curve. */
float CubicAmplifier( float input )
{
    float output, temp;
    if( input < 0.0 )
    {
        temp = input + 1.0f;
        output = (temp * temp * temp) - 1.0f;
    }
    else
    {
        temp = input - 1.0f;
        output = (temp * temp * temp) + 1.0f;
    }

    return output;
}
#define FUZZ(x) CubicAmplifier(CubicAmplifier(CubicAmplifier(CubicAmplifier(x))))

static int gNumNoInputs = 0;
/* This routine will be called by the PortAudio engine when audio is needed.
** It may be called at interrupt level on some machines so don't do anything
** that could mess up the system like calling malloc() or free().
*/
static int fuzzCallback( const void *inputBuffer, void *outputBuffer,
                         unsigned long framesPerBuffer,
                         const PaStreamCallbackTimeInfo* timeInfo,
```

```
                    PaStreamCallbackFlags statusFlags,
                    void *userData )
{
    SAMPLE *out = (SAMPLE*)outputBuffer;
    const SAMPLE *in = (const SAMPLE*)inputBuffer;
    unsigned int i;
    (void) timeInfo; /* Prevent unused variable warnings. */
    (void) statusFlags;
    (void) userData;

    if( inputBuffer == NULL )
    {
        for( i=0; i<framesPerBuffer; i++ )
        {
            *out++ = 0;  /* left - silent */
            *out++ = 0;  /* right - silent */
        }
        gNumNoInputs += 1;
    }
    else
    {
        for( i=0; i<framesPerBuffer; i++ )
        {
            *out++ = FUZZ(*in++);  /* left - distorted */
            *out++ = *in++;          /* right - clean */
        }
    }

    return paContinue;
}

/*******************************************************************/
int main(void);
int main(void)
{
    PaStreamParameters inputParameters, outputParameters;
    PaStream *stream;
    PaError err;

    err = Pa_Initialize();
    if( err != paNoError ) goto error;

    inputParameters.device = Pa_GetDefaultInputDevice(); /* default input device */
    if (inputParameters.device == paNoDevice) {
      fprintf(stderr,"Error: No default input device.\n");
      goto error;
    }
    inputParameters.channelCount = 2;       /* stereo input */
    inputParameters.sampleFormat = PA_SAMPLE_TYPE;
```

```
    inputParameters.suggestedLatency = Pa_GetDeviceInfo( inputParameters.device
)->defaultLowInputLatency;
    inputParameters.hostApiSpecificStreamInfo = NULL;

    outputParameters.device = Pa_GetDefaultOutputDevice(); /* default output device
*/
    if (outputParameters.device == paNoDevice) {
      fprintf(stderr,"Error: No default output device.\n");
      goto error;
    }
    outputParameters.channelCount = 2;       /* stereo output */
    outputParameters.sampleFormat = PA_SAMPLE_TYPE;
    outputParameters.suggestedLatency = Pa_GetDeviceInfo(
outputParameters.device )->defaultLowOutputLatency;
    outputParameters.hostApiSpecificStreamInfo = NULL;

    err = Pa_OpenStream(
              &stream,
              &inputParameters,
              &outputParameters,
              SAMPLE_RATE,
              FRAMES_PER_BUFFER,
              0, /* paClipOff, */  /* we won't output out of range samples so don't bother
clipping them */
              fuzzCallback,
              NULL );
    if( err != paNoError ) goto error;

    err = Pa_StartStream( stream );
    if( err != paNoError ) goto error;

    printf("Hit ENTER to stop program.\n");
    getchar();
    err = Pa_CloseStream( stream );
    if( err != paNoError ) goto error;

    printf("Finished. gNumNoInputs = %d\n", gNumNoInputs );
    Pa_Terminate();
    return 0;

error:
    Pa_Terminate();
    fprintf( stderr, "An error occured while using the portaudio stream\n" );
    fprintf( stderr, "Error number: %d\n", err );
    fprintf( stderr, "Error message: %s\n", Pa_GetErrorText( err ) );
    return -1;
}
```

Filtro FIR

```
/*
	Windowed Sinc FIR Generator
	Bob Maling (BobM.DSP@gmail.com)
	Contributed to musicdsp.org Source Code Archive
	Last Updated: April 12, 2005

	Usage:
		Lowpass:	wsfirLP(h, N, WINDOW, fc)
		Highpass:	wsfirHP(h, N, WINDOW, fc)
		Bandpass:	wsfirBP(h, N, WINDOW, fc1, fc2)
		Bandstop:	wsfirBS(h, N, WINDOW, fc1, fc2)

	where:
		h (double[N]):		filter coefficients will be written to this array
		N (int):		number of taps
		WINDOW (int):		Window (W_BLACKMAN, W_HANNING, or W_HAMMING)
		fc (double):	cutoff (0 < fc < 0.5, fc = f/fs)
					--> for fs=48kHz and cutoff f=12kHz, fc = 12k/48k => 0.25

		fc1 (double): low cutoff (0 < fc < 0.5, fc = f/fs)
		fc2 (double): high cutoff (0 < fc < 0.5, fc = f/fs)


	Windows included here are Blackman, Hanning, and Hamming. Other windows	can be
	added by doing the following:
		1. "Window type constants" section: Define a global constant for the new window.
		2. wsfirLP() function: Add the new window as a case in the switch() statement.
		3. Create the function for the window

		  For windows with a design parameter, such as Kaiser, some modification
		  will be needed for each function in order to pass the parameter.
*/
#ifndef WSFIR_H
#define WSFIR_H

#include <math.h>

// Function prototypes
void wsfirLP(double h[], const int &N, const int &WINDOW, const double &fc);
void wsfirHP(double h[], const int &N, const int &WINDOW, const double &fc);
void wsfirBS(double h[], const int &N, const int &WINDOW, const double &fc1, const double &fc2);
```

```
void wsfirBP(double h[], const int &N, const int &WINDOW, const double &fc1, const
double &fc2);
void genSinc(double sinc[], const int &N, const double &fc);
void wBlackman(double w[], const int &N);
void wHanning(double w[], const int &N);
void wHamming(double w[], const int &N);

// Window type contstants
const int W_BLACKMAN = 1;
const int W_HANNING = 2;
const int W_HAMMING = 3;

// Generate lowpass filter
//
// This is done by generating a sinc function and then windowing it
void wsfirLP(double h[],              // h[] will be written with the filter coefficients
                        const int &N,                 // size of the filter (number of taps)
                        const int &WINDOW,       // window function (W_BLACKMAN,
W_HANNING, etc.)
                        const double &fc)   // cutoff frequency
{
        int i;
        double *w = new double[N];                 // window function
   double *sinc = new double[N]; // sinc function

        // 1. Generate Sinc function
        genSinc(sinc, N, fc);

        // 2. Generate Window function
        switch (WINDOW) {
                case W_BLACKMAN:       // W_BLACKMAN
                        wBlackman(w, N);
                        break;
                case W_HANNING:                // W_HANNING
                        wHanning(w, N);
                        break;
                case W_HAMMING:                // W_HAMMING
                        wHamming(w, N);
                        break;
                default:
                        break;
        }

        // 3. Make lowpass filter
        for (i = 0; i < N; i++) {
                h[i] = sinc[i] * w[i];
        }

        // Delete dynamic storage
        delete []w;
```

```
	delete []sinc;

	return;
}

// Generate highpass filter
//
// This is done by generating a lowpass filter and then spectrally inverting it
void wsfirHP(double h[],		// h[] will be written with the filter coefficients
			const int &N,			// size of the filter
			const int &WINDOW,	// window function (W_BLACKMAN,
W_HANNING, etc.)
			const double &fc)	// cutoff frequency
{
	int i;

	// 1. Generate lowpass filter
	wsfirLP(h, N, WINDOW, fc);

	// 2. Spectrally invert (negate all samples and add 1 to center sample) lowpass
filter
	// = delta[n-((N-1)/2)] - h[n], in the time domain
	for (i = 0; i < N; i++) {
		h[i] *= -1.0;	// = 0 - h[i]
	}
	h[(N-1)/2] += 1.0;	// = 1 - h[(N-1)/2]

	return;
}

// Generate bandstop filter
//
// This is done by generating a lowpass and highpass filter and adding them
void wsfirBS(double h[],		// h[] will be written with the filter taps
			const int &N,			// size of the filter
			const int &WINDOW,	// window function (W_BLACKMAN,
W_HANNING, etc.)
			const double &fc1, // low cutoff frequency
			const double &fc2) // high cutoff frequency
{
	int i;
	double *h1 = new double[N];
	double *h2 = new double[N];

	// 1. Generate lowpass filter at first (low) cutoff frequency
	wsfirLP(h1, N, WINDOW, fc1);

	// 2. Generate highpass filter at second (high) cutoff frequency
	wsfirHP(h2, N, WINDOW, fc2);
```

```
	// 3. Add the 2 filters together
	for (i = 0; i < N; i++) {
		h[i] = h1[i] + h2[i];
	}

	// Delete dynamic memory
	delete []h1;
	delete []h2;

	return;
}

// Generate bandpass filter
//
// This is done by generating a bandstop filter and spectrally inverting it
void wsfirBP(double h[],		// h[] will be written with the filter taps
			const int &N,			// size of the filter
			const int &WINDOW,	// window function (W_BLACKMAN,
W_HANNING, etc.)
			const double &fc1, // low cutoff frequency
			const double &fc2) // high cutoff frequency
{
	int i;

	// 1. Generate bandstop filter
	wsfirBS(h, N, WINDOW, fc1, fc2);

	// 2. Spectrally invert bandstop (negate all, and add 1 to center sample)
	for (i = 0; i < N; i++) {
		h[i] *= -1.0;	// = 0 - h[i]
	}
	h[(N-1)/2] += 1.0;	// = 1 - h[(N-1)/2]

	return;
}

// Generate sinc function - used for making lowpass filter
void genSinc(double sinc[],		// sinc[] will be written with the sinc function
			const int &N,			// size (number of taps)
			const double &fc)	// cutoff frequency
{
	int i;
	const double M = N-1;
	double n;

	// Constants
	const double PI = 3.14159265358979323846;

	// Generate sinc delayed by (N-1)/2
	for (i = 0; i < N; i++) {
```

```
                if (i == M/2.0) {
                        sinc[i] = 2.0 * fc;
                }
                else {
                        n = (double)i - M/2.0;
                        sinc[i] = sin(2.0*PI*fc*n) / (PI*n);
                }
        }

        return;
}

// Generate window function (Blackman)
void wBlackman(double w[],                  // w[] will be written with the Blackman
window
                        const int &N)       // size of the window
{
        int i;
        const double M = N-1;
        const double PI = 3.14159265358979323846;

        for (i = 0; i < N; i++) {
                w[i] = 0.42 - (0.5 * cos(2.0*PI*(double)i/M)) +
(0.08*cos(4.0*PI*(double)i/M));
        }

        return;
}

// Generate window function (Hanning)
void wHanning(double w[],          // w[] will be written with the Hanning window
                        const int &N)               // size of the window
{
        int i;
        const double M = N-1;
        const double PI = 3.14159265358979323846;

        for (i = 0; i < N; i++) {
                w[i] = 0.5 * (1.0 - cos(2.0*PI*(double)i/M));
        }

        return;
}

// Generate window function (Hamming)
void wHamming(double w[],                  // w[] will be written with the Hamming
window
                        const int &N)               // size of the window
{
        int i;
```

```
	const double M = N-1;
	const double PI = 3.14159265358979323846;

	for (i = 0; i < N; i++) {
		w[i] = 0.54 - (0.46*cos(2.0*PI*(double)i/M));
	}

	return;
}

#endif
```