UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UMA FERRAMENTA DE GERAÇÃO DE EFEITOS SONOROS PARA INSTRUMENTOS MUSICAIS

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

TARCÍSIO LUÍS TAMANINI

BLUMENAU, DEZEMBRO/2000

2000/2-52

PROTÓTIPO DE UMA FERRAMENTA DE GERAÇÃO DE EFEITOS SONOROS PARA INSTRUMENTOS MUSICAIS

TARCÍSIO LUÍS TAMANINI

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Dalton Solano dos Reis — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Dalton Solano dos Reis

_____

Prof. Miguel Wisintainer

_____

Prof. Paulo Cesar Rodacki Gomes

DEDICATÓRIA

Dedico este trabalho a minha família,
em especial meus pais que sempre me incentivaram e me
apoiaram mesmo nos momentos mais difíceis.

AGRADECIMENTOS

Agradeço a todos os meus familiares principalmente a meus pais Luiz e Fêde Tamanini, pessoas na qual amo e apostaram na minha vitória.

Também a todos meus professores, que me transmitiram seu conhecimento e suas experiências a qual sou muito grato e em especial ao professor Dalton Solano dos Reis, meu orientador e amigo.

E a todos meus colegas e amigos que passaram comigo cada momento destes quatro anos e meio.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

O presente trabalho consiste em um estudo para tratamento de ondas sonoras digitais. As ondas sonoras são obtidas a partir de placas de som de microcomputadores.

O trabalho também consiste no desenvolvimento de uma ferramenta de software para adicionar, em tempo real, efeitos tais como digital *delay*, *reverb* e outros, ao sinal de áudio. O software pode ser utilizado com o sinal de áudio proveniente de microfones ou qualquer outro instrumento musical elétrico, como a guitarra, por exemplo.

ABSTRACT

The present work consists on the study digital sound waves treatment. The sound waves are obtained from a personal computer's sound board.

The work also consists on the development of a software tool that adds effects, such as reverb, digital delay, and others, in real time to the audio signal obtained from the sound board's audio input port. The software tool can be used with the signal from a microfone or any electric musical instrument, such as the eletric guitar.

1 INTRODUÇÃO

1.1 ORIGEM DA MÚSICA ELETRÔNICA

De acordo com [RAT1995], a música está entre as mais populares das artes, adquirindo ritmos e estilos nas mais diversas sociedades, desde as cantigas de roda até sofisticados estilos musicais como o *jazz*, *blues*, *folk*, *rock*, entre outros. Entre as várias formas de se produzir música encontram-se os instrumentos musicais, participando assiduamente de eventos culturais e sociais, apresentando ritmos e melodias das mais diversas formas. Partindo-se do pressuposto "populares", encontram-se o violão acústico, posteriormente o violão elétrico e a guitarra elétrica.

Segundo [MOO1994], o início da música eletrônica se deu por volta de 1906 por um instrumento inventado por Thaddeus Cahill. O instrumento inventado por Cahill produzia sons ligando a saída de bancos de dínamos ao alto-falante, ou cápsula receptora de um telefone. Após a experiência de Cahill, surgiu o órgão construído pelos físicos Coupleux-Givelet na década de 1930, usando uma válvula osciladora para cada nota da escala, utilizando desta forma centenas de válvulas.

Ainda, segundo [MOO1994], por volta de 1970 a Bell Labs desenvolveu um modo de amostrar digitalmente o som, tornando o que é um evento fundamentalmente analógico e contínuo em um sinal digital e discreto, representado por uma série de sinais binários (1 e 0), que podem ser armazenados, manipulados e reproduzidos.

Por volta dos anos 80, surgem então as placas de som, por exemplo, a *Sound Blaster* que disponibilizam uma série de recursos aos microcomputadores. Através delas foi possível utilizar recursos de áudio tais como: manipulação de sons, gravação e reprodução, ouvir CDs de áudio, animações, etc... utilizando estes recursos em diversos segmentos da computação.

Segundo [MOO1994], todas as placas *Sound Blaster* apresentam três funções básicas:

a) podem amostrar entradas de som, digitalizando e armazenando-o em um arquivo para reprodução ou modificação;

b) podem sintetizar o som de instrumentos musicais e a voz humana, através de componentes dedicados;

c) podem controlar instrumentos musicais compatíveis com a interface *MIDI*[1].

1.2 ARQUIVOS DE SOM

A tarefa de converter o som em uma forma que possa ser armazenada em um computador é realizado por um circuito processador de sinais digitais (DSP[2]).

Quando um som é digitalizado, o *software* que realiza esta operação comanda o ADC[3] localizado na placa de som. O ADC faz sucessivas leituras do sinal de áudio, convertendo essas leituras em valores digitais que podem ser armazenados em arquivos. Como ocorre com qualquer outro tipo de arquivo, é necessário que exista uma padronização no formato dos arquivos sonoros, para que possam ser acessados por diversos programas diferentes.

Segundo [MOO1994], o padrão *MIDI* usa tecnologia musical para codificar o som em informações binárias, que são transferidos por meio de uma linha física (cabo *MIDI*) de um equipamento para outro.

Para transferir as informações através do cabo *MIDI*, o instrumento codifica essas informações sob a forma de números binários, que são enviados, serialmente, e sem a utilização do ADC. A transmissão ocorre a uma velocidade de 31250 *bits* por segundo, e normalmente é unidirecional (o transmissor não recebe nenhuma resposta do receptor). O instrumento receptor (escravo) recebe cada um dos *bytes* e a partir deles monta novamente a informação ([RAT1995]).

Os formatos mais usados para representar arquivos sonoros são o WAV e o VOC. Os arquivos *VOC* eram muito usados pelos programas que acompanhavam as primeiras placas de som, podendo hoje ainda serem encontrados nas placas de som mais modernas.

---

[1] MIDI = Musical Instrument Digital Interface

[2] DSP = Digital Signal Processing

[3] ADC = Conversor Analógico Digital

Além desses formatos, podemos encontrar outros que são usados por programas isolados, como o SND, o AU, o MP3 e o MIDI, este, que é completamente diferente dos arquivos WAV e VOC. Arquivos WAV e VOC contém amostras digitalizadas de sons reais que são convertidos para sons através do DAC[4]. Já os arquivos MIDI contém códigos que representam instrumentos musicais, notas musicais e a duração das notas, e, a partir desses códigos os sons são gerados através do sintetizador de áudio e são gravados em *ROM*, ([VAS1995]).

Na intenção de suprir a necessidade da grande maioria da sociedade de gerar efeitos sobre um som original, principalmente instrumentalistas, surgiram as mais variadas marcas e modelos de equipamentos, denominados de “pedaleiras” e/ou “bancos de efeitos”.

Hoje esta variedade de marcas e modelos, disponíveis no mercado, partem de modelos que integram recursos de *hardware* que acionam circuitos eletrônicos também controlados por *hardware*, formando um componente totalmente dedicado à reproduzir o som com o efeito selecionado.

A idéia é interessante e pode tornar-se muito útil àqueles que necessitam de recursos de tratamento de som, usufruindo dos recursos da placa de som, das mais variadas marcas, já disponíveis hoje na maioria dos computadores.

1.3 OBJETIVO

Os principais objetivos deste trabalho são:

- criar um ambiente com recursos que permitam capturar e reproduzir o som vindo de um instrumento musical ou microfone conectado à placa de som e que permita manipular estes sons de acordo com as necessidades de cada usuário;
- permitir, sobre esta manipulação, adicionar efeitos em “tempo real” tais como *delay*, *reverb*, *phaser* e *flanger*;
- utilizar recursos DSP para o tratamento dos sinais.

[SAF1] Comentário: Talvez isto seja eliminado..... ou siga como extenção

[4] DAC = Conversor Digital Analógico

1.4 ORGANIZAÇÃO DO TEXTO

No capítulo 2, o trabalho aborda sinais de áudio e a representação desses sinais.

No capítulo 3, mostra alguns tipos de sons digitalizados, formatos de arquivos, evolução dos formatos de arquivamento, envolvendo compactação e formas de reprodução. Mostra também alguns ambientes tratadores de som, características, efeitos aplicáveis, formas de manipulação e tipos de arquivos de som que possam ser tratados.

O capítulo 4, apresenta técnicas de processamento do som e a descrição de alguns efeitos.

O capítulo 5 apresenta o desenvolvimento do protótipo e seu funcionamento.

O capítulo 6 traz os resultados finais obtidos através de testes aplicados, bem como suas limitações.

2 SINAIS DE ÁUDIO

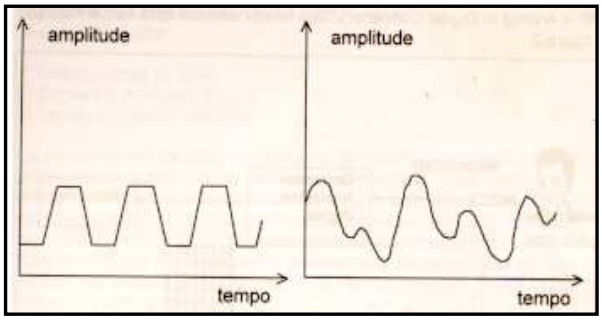
Já houve um tempo em que os microcomputadores eram completamente “mudos”. Não possuíam alto-falante, sendo assim incapazes de emitir qualquer tipo de som. Isso ocorreu muito com os antigos micros de 8 *bits*. Alguns micros possuíam alto-falantes e eram uma verdadeira sensação, podiam emitir diversos tipos de sons, desde simples “beeps” até músicas, mas não tinham filtros nem amplificadores.

Apesar de serem sonorizados, esses micros tinham dificuldade para reproduzir sons mais elaborados, como a voz humana ou instrumentos musicais. Todo o som era formado por “beeps”, sendo alguns mais agudos e outros mais graves. Essa é uma limitação devido ao uso de sinais digitais.

Os sinais elétricos digitais consistem em ondas retangulares, com um valor fixo (0 ou 1), e apenas com a freqüência variável. Para apresentar a voz humana, os instrumentos musicais e diversos sons existentes na natureza, é necessário usar o mesmo tipo de sinal elétrico que é utilizado nos aparelhos de áudio, como microfones, rádios e amplificadores. Esse tipo de sinal é chamado de analógico, e pode variar em freqüência e também em amplitude ([VAS1995]).

A Figura 1, apresentada em [VAS1995], expõe gráficos que representam os sinais elétricos digitais e analógicos.

Figura 1 – Representação de sinais



Sinal Digital **Sinal Analógico**

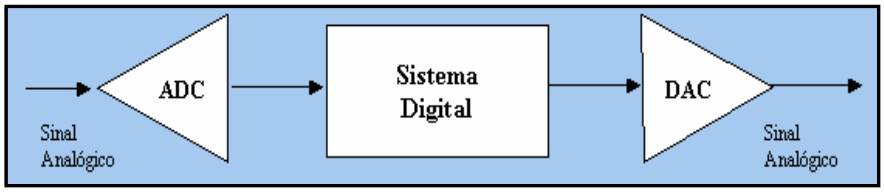
2.1 REPRESENTAÇÃO DIGITAL DO SOM

Os dispositivos e sistemas analógicos representam o sinal sonoro, que é um sinal de pressão mecânica, por um sinal magnético ou elétrico, de amplitude proporcional à amplitude do sinal acústico original ([FIL2000]). Amplificadores e gravadores de fita cassete convencional são exemplos de dispositivos analógicos. Já nos dispositivos e sistemas eletrônicos digitais os sinais são representados por seqüência de números.

Ainda segundo [FIL2000], um microcomputador de 8 *bits*, possuía um *chip* especial chamado de "sintetizador". Este *chip* era capaz de gerar, através dos seus circuitos analógicos, diversos tipos de sons semelhantes aos encontrados na natureza. Este sintetizador, podia gerar sons parecidos com os de certos instrumentos musicais e efeitos sonoros diversos, como as explosões e tiros usados nos jogos. Apesar de gerar sons muito melhores que os obtidos usando apenas o alto-falante, esse *chip* ainda não tornava o computador capaz de gravar e reproduzir sons da natureza, como instrumentos musicais com maior realismo e nem a voz humana.

Segundo [VAS1995], para gravar a voz humana com precisão através do computador, existe uma certa dificuldade técnica. A voz humana, ao ser captada por um microfone, é um sinal elétrico analógico, e os dados armazenados no computador são digitais. Portanto, para que a voz humana seja armazenada na memória do computador, precisa antes ser digitalizada, através do circuito eletrônico ADC. Da mesma forma, para que os dados digitais armazenados pelo computador possam ser transformados em sons analógicos, é necessário utilizar um circuito eletrônico DAC, conforme Figura 2 ilustrada por [FIL2000].

Figura 2 – Digitalização do som



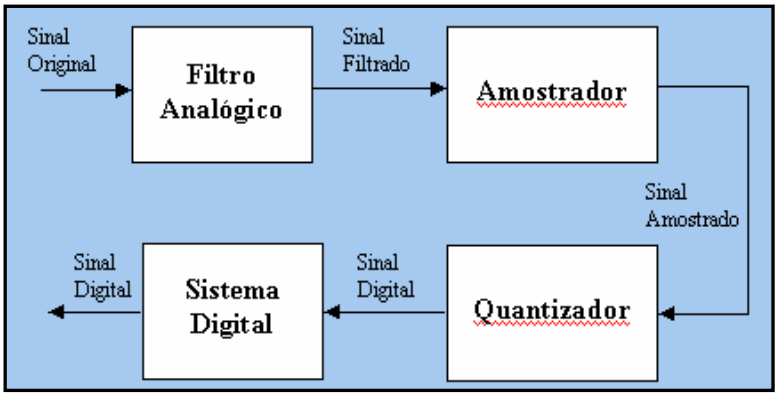
Quando um sinal de áudio é digitalizado, convertido em *bytes*, dizemos que está ocorrendo uma digitalização. A digitalização, por sua vez, consiste em fazer sucessivas amostras do sinal analógico. A cada amostra, o ADC gera um número inteiro que é

proporcional à amplitude lida. Para que se possa ter uma representação digital bem precisa do sinal analógico, é preciso que sejam lidas milhares de amostras a cada segundo. A Figura 3 mostra o processo de digitalização do som.

Segundo [FIL2000], pode-se distinguir as seguintes etapas para os processos de digitalização:

a) **Filtragem**: por meio de um filtro analógico de entrada, faz-se uma limitação da faixa de freqüências existentes no sinal. Freqüências acima da freqüência de corte do filtro são eliminadas, para evitar a ocorrência da pseudonímia (seção 2.1.2) ;
b) **Amostragem**: por meio de um amostrador, faz-se a conversão do sinal analógico contínuo em uma seqüência de pulsos. A amplitude de cada pulso representará uma amostra de som;
c) **Quantização**: faz-se a conversão dos pulsos para números binários, via ADC. As amostras de som são convertidas em números;
d) **Arquivos de áudio:** são formados pelas seqüências de amostras de som.

Figura 3 – O processo de digitalização do som



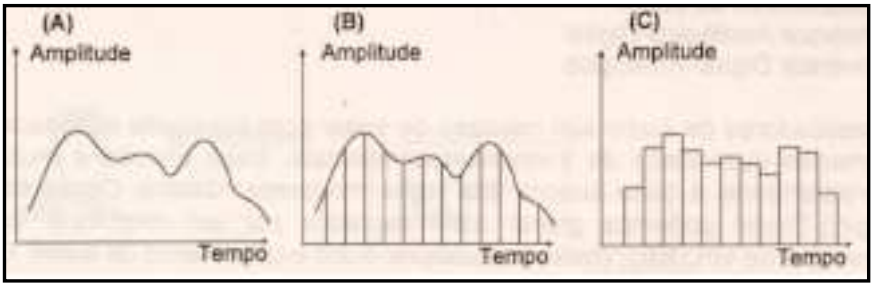
2.1.1 A AMOSTRAGEM

Para efetuar esta etapa, o sinal analógico deve passar por um circuito amostrador. A saída do amostrador é uma seqüência de pulsos elétricos (trem de pulsos) de freqüência constante. A altura de cada pulso corresponde a uma “fotografia” do sinal analógico no instante da amostragem. A freqüência com que as “fotos” são tiradas, ou o número de amostras lidas a cada segundo é chamada de **taxa de amostragem**. Uma taxa de amostragem

de 10 Khz, por exemplo, indica que são realizadas 10000 amostras por segundo. A cada segundo são portanto gerados 10000 valores digitais que representam os valores analógicos do sinal de áudio ([FIL2000]) e ([VAS1995]).

A Figura 4 ilustrada por [VAS1995] mostra o processo de digitalização de um sinal analógico usando uma taxa de amostragem de 4 KHz. Sendo realizadas 4000 amostras por segundo, o conversor A/D realiza uma amostra a cada 0,25 ms pelo conversor D/A, resultam em um sinal como mostrado no item C da Figura 4.

Figura 4 – Digitalização a 4 KHz

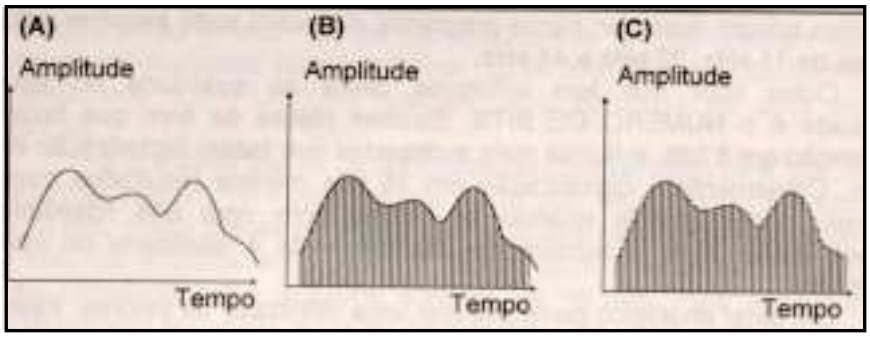


Sinal analógico original Amostragem em 4 Khz Sinal digitalizado

O sinal indicado no item C é parecido com o sinal original em (A). Quando a taxa de amostragem é muito lenta, existe uma grande distorção no processo de digitalização. Se o sinal original for uma música, o resultado mostrado no item C será correspondente a mesma música, mas extremamente distorcida, como se fosse gerada por uma estação de rádio mal sintonizada.

A forma utilizada para evitar a distorção do sinal digitalizado é usar uma taxa de amostragem mais elevada. Na Figura 5 podemos observar o mesmo sinal analógico da Figura 4, mas desta vez sendo digitalizado com uma taxa de amostragem de 10 KHz. Observe como o sinal digitalizado no item C da Figura 5 é muito mais parecido com o sinal analógico original, se compararmos com os resultados obtidos na Figura 4.

Figura 5 – Digitalização a 10 KHz



Sinal analógico original Amostragem em 10 Khz Sinal digitalizado

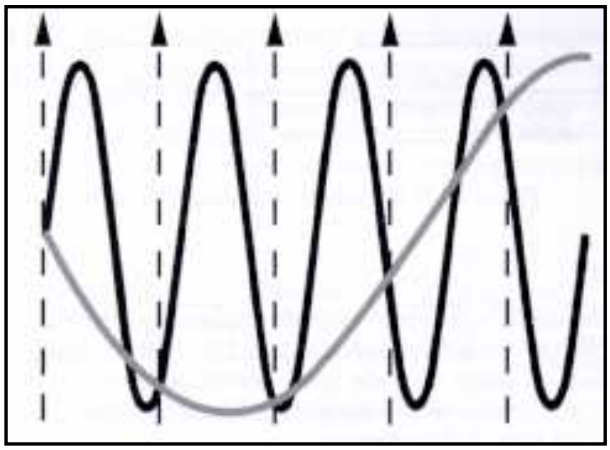
Comparando os resultados obtidos nas Figura 4 e 5, concluímos que quanto maior é a taxa de amostragem, maior será a perfeição do sinal digitalizado, se comparado ao sinal original. Mas uma taxa de amostragem exageradamente alta não chega a trazer melhorias adicionais. Uma amostragem em 100 KHz não resulta em qualidade sonora melhor que o obtido com 44 KHz e gera arquivos sonoros muito grandes, conforme será visto na próxima seção.

2.1.2 PSEUDONÍMIA

Segundo [VAS1995] e [FIL2000], as taxas de amostragem muito baixas produzem distorção no sinal de áudio, enquanto que as taxas muito altas resultam em alto custo e desperdício de espaço de armazenamento em disco. Para definir qual a taxa de amostragem ideal é utilizado o "*Critério de Nyquist*": *Se um sinal analógico está limitado a uma freqüência* ***f****, a taxa de amostragem que permite sua fiel reconstituição é* ***2f.*** Ou seja, um sinal periódico digitalizado só poderá ser reconstituído corretamente se a taxa de amostragem tiver um valor mínimo, sendo que, este valor dever ser superior ao dobro da freqüência da componente de mais alta freqüência que esteja presente no sinal original.

Quando um sinal contém componentes superiores à freqüência de Nyquist, acontece o fenômeno da pseudonímia (*aliasing*). A Figura 6 mostra a formação de pseudônimo.

Figura 6 – Formação de pseudônimo



As freqüências acima da freqüência de Nyquist são convertidas, no processo de reconstituição, em uma freqüência mais baixa, o pseudônimo ou *alias* ([FIL2000]).

2.1.3 FILTRAGEM ANTIPSEUDONÍMIA

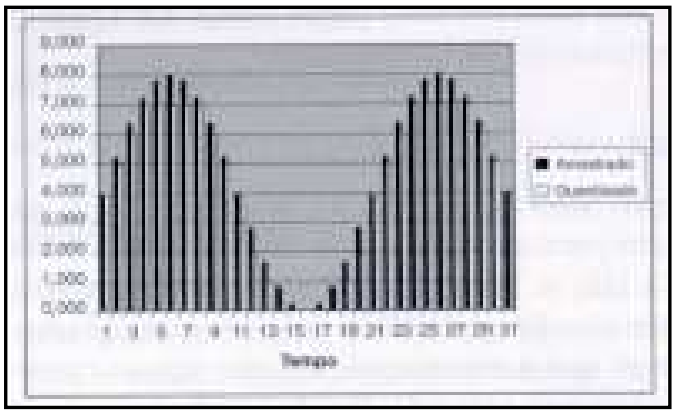
De acordo com [FIL2000] e [VAS1995], para evitar o dobramento, é preciso inserir antes do amostrador um flltro analógico passa-baixa que corte as freqüências acima da freqüência de Nyquist. Como a audição humana pode ir até 20 KHz, uma taxa de amostragem mínima para a digitalização de alta fidelidade seria de 40 KHz. Na prática, como os filtros reais não são perfeitos, é preciso usar taxas ainda mais altas. O padrão CD, por exemplo, trabalha com uma taxa de 44.100 Hz, e o padrão DAT chega a 48 KHz. A voz humana, tem quase toda sua potência sonora formada por freqüências inferiores a 4 KHz. Portanto, para digitalizar a voz, é indicado usar uma freqüência de amostragem de 8 KHz. Na digitalização do som telefônico, uma taxa de amostragem de 8 Khz é suficiente, já que as normas telefônicas impõem que o sinal de voz seja filtrado para uma faixa limitada a cerca de 3.500 Hz. A limitação das faixas acarreta enormes economias para o sistema telefônico, permitindo que as redes acomodem uma quantidade de canais de voz muito maior do que se o som conversasse a alta fidelidade.

2.1.4 QUANTIZAÇÃO

A digitalização exige uma segunda etapa de passagem para a forma discreta. Cada amostra passa por um ADC, que produz um número binário que representa a amplitude da amostra ([FIL2000]).

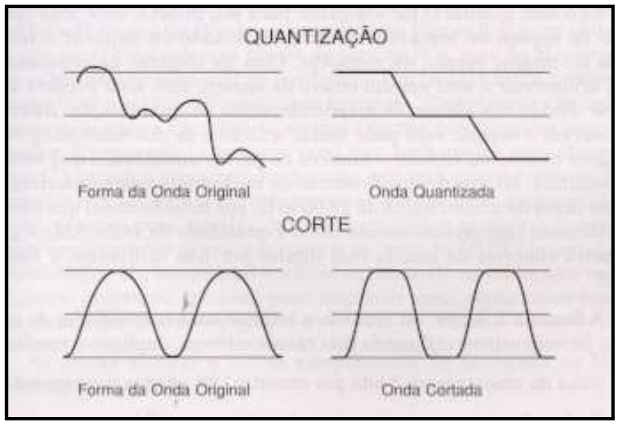
Supõe-se aqui que o ADC pode produzir 3 *bits* (8 números diferentes) na saída. As alturas dos pulsos são truncadas, de modo que a seqüência passe a usar oito alturas de pulsos diferentes. Foi aplicada uma mudança de escala para facilitar a comparação entre os pulsos antes da quantização (barras escuras) e depois da quantização (barras claras) da Figura 7.

Figura 7 – A Quantização



A Figura 8, mostra uma onda quantizada e uma onda cortada.

Figura 8 - Quantização e Cortes



Segundo [VAS1995], suponha que um sinal analógico possa assumir valores de 0 até 1 Volt. Entre *0* e *1* Volt, este sinal pode ter infinitos valores possíveis de voltagem. Por outro lado, uma amostragem em 8 *bits* só permite representar 256 valores possíveis. Consideramos a tensão limitada entre *0* e *1,024* Volts. Se dividir o intervalo de *0* a *1,024* em 256 partes iguais, cada parte terá o valor de *0,004* Volt. Será possível então representar de forma digital valores conforme a Tabela 1.

Tabela 1 – Representação de valores de forma digital

Analógico	Digital	Analógico	Digital	Analógico	Digital
0,000	0	0,100	25	...	...
0,004	1	0,104	26	0,988	247
0,008	2	0,108	27	0,992	248
0,012	3	0,112	28	0,996	249
0,016	4	0,116	29	1,000	250
0,020	5	0,120	30	1,004	251
0,024	6	0,124	31	1,008	252
0,028	7	0,128	32	1,012	253
0,032	8	0,132	33	1,016	254
...	...	...	...	1,020	255

Uma tensão elétrica de, por exemplo, *0,120* Volts pode ser representada com exatidão. Pela Tabela , vemos que o valor digital correspondente a *0,120* Volts é o número 30. Já uma tensão de *0,121* Volts não pode ser representada com precisão. Os dois valores mais próximos representados pelo conversor A/D são *0,120* e *0,124*, através dos números 30 e 31, respectivamente. Nesse caso, o conversor faz uma espécie de "arredondamento" ao escolher 30, que é o valor mais próximo.

O efeito sonoro resultante dos arredondamentos feitos nos conversores de 8 *bits* é uma espécie de "chiado", um som equivalente ao de uma estação de rádio ligeiramente fora de sintonia.

Segundo [FIL2000], como o conversor possui um número limitado de *bits*, ele introduzirá um erro de quantização. Este erro se traduzirá auditivamente em um ruído, ouvido na reprodução do sinal reconstituído. Entretanto, se usarmos 16 *bits* por amostragem, como se faz no padrão CD, a relação sinal-ruído será de 2^{16} (ou 96 dB, expressa na forma logarítmica).

Um conversor A/D de 16 *bits* apresenta resultados muito melhores. Também ocorrem erros de arredondamento, mas esses erros são 256 vezes menores que os existentes nas

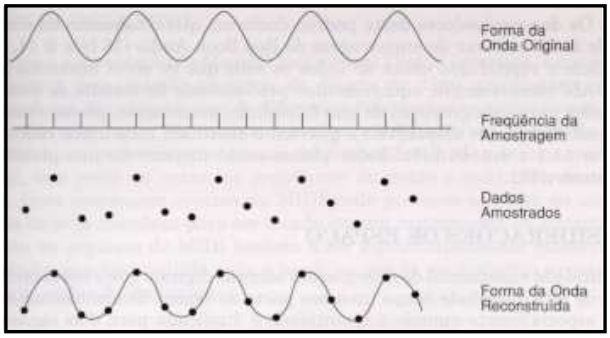
conversões em 8 *bits*. O “chiado” resultante da conversão é, para todos os efeitos práticos, inaudível.

2.1.5 RECONSTRUÇÃO

Na saída de um sistema digital de som, as amostras são retiradas de um dispositivo digital de armazenamento e enviadas para os DAC’s. Para manter o fluxo de reprodução em tempo real, é preciso empregar discos de tecnologia rápida, e os arquivos de som devem ser localizados em trilhas adjacentes. Em alguns sistemas, isto pode requerer restrições à liberdade de atribuição de espaço físico nos discos. Pode ser preciso, portanto, contornar a gerência normal de arquivos de sistemas operacionais como o MS-DOS e o UNIX. Segundo [FIL2000], uma técnica usada é a criação de discos virtuais desfragmentados e dedicados à gravação e à reprodução de arquivos de áudio.

Ao sair do DAC, o sinal de som apresentará uma forma de escada, resultante da representação digital. Esse novo erro pode ser audível como ruído, e deve ser eliminado por mais um filtro passa-baixa. A Figura 9 apresenta a reconstrução da forma da onda original.

Figura 9 - Reconstrução da forma da onda original



Dispositivos capazes de selecionar determinadas freqüências e rejeitar outras são chamados de filtros, e são eles:

- Filtros passa-baixa cortam freqüências acima de um certo valor;
- Filtros passa-alta cortam freqüências abaixo de um certo valor;
- Filtros passa-faixa aceitam apenas freqüências situadas entre certos limites;
- Filtros rejeita-faixas rejeitam apenas freqüências situadas entre certos limites.

3 SONS DIGITALIZADOS

Segundo [VAU1994], arquivos de áudio são arquivos de computador que armazenam dados de áudio digital. Quando utilizamos um formato de áudio qualquer para gravar som, simplesmente gravamos a saída do *mixer*, que são os controles de equalização do som (graves, médios, agudos, volume, etc...), bem como os canais de entrada e saída (I/O) da placa de som, entre eles:

- *Line In (I)* : linha de entrada do sinal de áudio, que pode ser utilizada para a conexão de um aparelho de som para a captura de sons provenientes do mesmo;
- *MIC (I)*: conexão para microfone e/ou instrumento musical elétrico;
- *Line Out (O)* : linha de saída do sinal de áudio, podendo ser conectado a outros equipamentos antes da reprodução, como amplificadores, equalizadores, etc..

Dependendo do formato usado (WAV, VOC, etc.) podemos ter diversas taxas de amostragem, resolução de *bits* e canais de áudio disponíveis. Por exemplo, um arquivo gravado usando a mesma qualidade de um CD possui uma taxa de amostragem de 44.1 KHz, uma resolução de 16 *bits*, e som estereofônico.

Arquivos de áudio são, geralmente, arquivos muito grandes, que podem ocupar enormes quantidades de espaço em disco. Usando a qualidade de CD, produz-se 180 Kb por segundo, ou seja, 10 *megabytes* por minuto. Para minimizar este problema, alguns formatos aceitam que seus dados de áudio digital sejam comprimidos, ou seja, o arquivo terá um tamanho menor. As vantagens são claras: o arquivo irá ocupar menos espaço em disco, e é mais transportável, através de disquetes, ou via *Internet*.

O som pode ser digitalizado a partir de um microfone, de um sintetizador, de gravações de fitas existentes, de programas de rádio e televisão ao vivo, de CDs populares, entre outros. Na verdade, é possível digitalizar sons a partir de qualquer fonte, natural ou pré-gravada.

Segundo [VAU1994], o som digitalizado é um som “sampleado”. A cada enésima fração de um segundo, uma amostra do som é capturada e armazenada como informações digitais em *bits* e *bytes*. A freqüência com que as amostras são capturadas determina a taxa de amostragem, e a quantidade de informações armazenadas a cada amostragem determina o

tamanho da amostra. Quanto maior a freqüência de amostragem e mais dados armazenar em cada amostra, maior será a resolução e a qualidade do som capturado quando ele for executado.

As três freqüências de amostragem mais usadas na multimídia são 44,1 KHz, 22,05 KHz e 11,025 KHz. Os tamanhos das amostragens são de 8 ou 16 *bits*.

3.1 FORMATOS DE ARQUIVOS

Dentre os mais variados formatos de arquivos de áudio, padronizados ou proprietários, destacam-se alguns que são os mais utilizados.

3.1.1 WAVE

O formato de arquivo wave (.WAV) suporta uma variedade de resoluções de *bit*, taxas de amostragem, e canais de áudio. Este formato é muito popular em plataformas IBM PC, e é extensamente utilizado em programas profissionais que processam áudio digital.

Um arquivo wave, é uma coleção de vários tipos diferentes de "*chunk's*". Há um formato exigido, "*chunk de fmt***"** que é a parte que contém parâmetros importantes que descrevem o *waveform*, como sua taxa de amostra, largura de amostra, canais, etc.. e o "*chunk de dados"* que contém o *waveform* atual dos dados, que também é requerido. Todos os outros *chunk's* são opcionais, entre eles, há uma lista de parâmetros de instrumentos, ([GON2000]) e ([MAT2000]).

A especificação de wave suporta vários algoritmos de compressão diferentes. A entrada da etiqueta no formato *chunk de fmt* indica o tipo de compressão usado. Um valor de 1 indica Modulação de Código de Pulsação (PCM), que são técnicas de compactação. Valores diferentes de 1, indicam alguma forma de compressão.

O formato wave é um padrão que foi mexido por muitos programadores que não coordenaram corretamente a adição de rotinas ao produto final (ao contrário do padrão AIFF, por exemplo, que foi projetado em sua maior parte por um grupo pequeno, coordenado).

O resultado final é que há muitos trechos *chunk's* de informação que podem ser encontrados em um arquivo wave, e muitos deles duplicam a mesma informação encontrada em outros trechos. Mas de uma maneira desnecessariamente diferente, simplesmente porque muitos programadores tomaram liberdade adicionando suas próprias contribuições ao formato wave, sem corretamente chegar a um consenso sobre o que todos precisariam.

3.1.2 IFF (INTERCHANGE FILE FORMAT)

Foi projetado pela Electronic Arts, que merece crédito por deixar a vida mais simples tanto para programadores quanto para usuários finais. Criou o formato IFF e liberou a documentação para o mesmo, bem como o código fonte em linguagem C para ler e escrever arquivos do tipo IFF, que na verdade é o formato de intercâmbio entre arquivos. Assim, tornou simples para os programadores escreverem formatos de arquivos "extensíveis" e "retro-compatíveis". IFF também ajuda a desenvolvedores escreverem programas que lêem facilmente arquivos de dados criados com *softwares* compatíveis com IFF de outros fabricantes, mesmo que não exista nenhuma relação de negócios entre eles.

IFF ajuda a minimizar problemas como novas versões de um programa em particular que tem problemas em ler dados produzidos por versões mais antigas, ou precisem de um novo formato sempre que uma nova versão precise armazenar informações adicionais. Também encoraja formatos padronizados que não estejam amarrados a um produto em particular. Isto tudo é bom para os usuários finais, porque significa que seus dados não ficarão presos a algum formato proprietário, e não possam ser usados com uma grande variedade de *hardware* e *software* ([GON2000]).

3.1.3 MP3

Segundo [DIG2000], o MPEG *Layer* 3 teve como origem o MPEG, ou Moving Picture Experts Group. Este grupo, que trabalha sob direção da International Standards Organization (ISO) e da International Electro-Technical Commission (IEC), foi estabelecido em janeiro de 1988 com objetivo de definir um padrão para compactação de vídeo, para ser usado tanto na Internet quanto na TV digital. Dentro do padrão MPEG, foi definido uma parte para vídeo e uma parte para áudio. A taxa de compactação é variável, para que possamos adequá-la à nossa

necessidade, jogando com a relação qualidade/tamanho. Esta taxa de compactação pode chegar a um ponto tal que é viável a transmissão pela Internet.

A parte de áudio possui três padrões: MPEG Audio *Layer* 1, 2 e 3, sendo o *Layer* 3 o algoritmo mais complexo e de melhor qualidade. Cada um deles é compatível com os anteriores. Por exemplo: um *software* que entende *Layer* 3, entende também *Layer* 1 e *Layer* 2.

Os arquivos de computador com informação MPEG Audio *Layer* 3 são chamados MP3. A taxa de compactação é definida por uma variável: o "bitrate", ou número de *bits* por segundo. Com bitrate de 128 Kbps (kilobits por segundo) tem-se um arquivo com qualidade praticamente igual à de CD, com resposta de freqüência de 20Hz a 20KHz. Uma música de 4 minutos estéreo, compactada a 128 Kbps, ocupa apenas 4 *megabytes*, comparada a de um WAV que é mais ou menos 60 *megabytes*.

A qualidade dos arquivos MP3 é espantosamente boa. A tecnologia de compressão, semelhante à do Minidisc, funciona segundo o princípio de *perceptual audio coding*. Apenas ouvidos treinados notam uma diferença mínima para a qualidade de CD.

3.1.4 REALAUDIO

Este formato, projetado pela empresa Progressive Networks, é um formato bastante padronizado e sem dúvida o mais comum na Internet (provavelmente por ter sido o primeiro), apesar do formato MP3 estar quase tão comum quanto ele. Até a versão 3.0, RealAudio requeria um servidor próprio, mas essa limitação não existe mais. A qualidade sonora das compressões para altas larguras de banda são excelentes.

É verdade que um arquivo MPEG de taxas de 96 *kbit* ou 112 *kbit* realmente soam melhor que o RealAudio de 80 *kbit*, mas a diferença não é tão dramática. Em alguns casos, o arquivo *RealAudio* pode soar ainda melhor. A compressão RealAudio é feita em tempo real, mesmo para larguras de banda mais altas. Além disso, arquivos RealAudio são suportados na maioria das plataformas, e o *player* é gratuito.

3.1.5 AIFF (AUDIO INTERCHANGE FILE FORMAT)

Este formato suporta uma variedade de resoluções de *bit*, taxas de amostragem, e canais de áudio. Este formato é muito popular em plataformas *Apple* (seria o equivalente ao Wave para PCs em termos de popularidade), e é extensamente utilizado em programas profissionais que processam áudio digital nesta plataforma.

3.1.6 MIDI

O formato MIDI padrão (*Standard MIDI File*, ou SMF) é um formato de arquivos especificamente projetado para armazenar os dados que um seqüenciador (*software* ou *hardware*) grava e reproduz.

Este formato armazena as mensagens MIDI padrão (*bytes* de status, com os *bytes* de dados apropriados) mais um "*timestamp*" para cada mensagem (isto é, uma série de *bytes* que representam quantos pulsos de relógio aguardar antes de "reproduzir" o evento). O formato permite salvar informações sobre tempo, número de pulsos em resolução de semínima (ou resolução expressa em divisões de segundo, chamada *configuração SMPTE*[5]), duração e tom da música, e nomes de trilhas e padrões. Pode armazenar múltiplos padrões e trilhas, de forma que qualquer aplicativo pode preservar estas estruturas quando carregar o arquivo, ([GON2000]).

Uma *trilha* usualmente é análoga à uma parte musical, como a parte de um trumpete. Um *padrão* seria análogo à todas as partes musicais (Trumpete, Piano, Bateria etc.) para uma canção, ou trecho de uma canção.

O formato foi projetado para ser genérico, de forma que qualquer seqüenciador poderia ler ou escrever tal arquivo, sem perder os dados mais importantes. E flexível o suficiente para um determinado aplicativo armazenar seus dados próprios ("extras") de forma que outro aplicativo poderia carregá-lo sem problemas, e poderia ignorar com segurança esses dados extras que não precisasse. O formato de arquivos MIDI como uma versão musical de um arquivo texto *ASCII* (exceto que o arquivo MIDI contém dados binários também), e os

---

[5] *Society of Motion Picture and Television Engineers*

vários seqüenciadores como editores de texto capazes de ler tal arquivo. Mas, ao contrário do *ASCII*, o formato de arquivos MIDI salvam dados em *blocos* (grupos de *bytes* precedidos por um identificador e tamanho) que podem ser analisados, carregados, saltados etc. Assim, pode ser facilmente estendido para incluir informações proprietárias de um programa.

Por exemplo, talvez um programa queira salvar um "*byte* sinalizador" que indica se o usuário ligou o clique de um metrônomo audível. O programa pode colocar este *byte* sinalizador em um arquivo MIDI de tal forma que outro aplicativo possa saltar este *byte* sem ter que entender para que ele serve. No futuro, o formato de arquivos MIDI também poderá ser estendido para incluir novos "blocos oficiais" que todos os seqüenciadores possam carregar e usar. Isto pode ser feito de modo que arquivos de dados antigos não se tornem obsoletos (isto é, o formato é projetado para ser extensível e portável).

Uma das características mais interessantes dos arquivos MIDI é o seu tamanho reduzido. Apenas alguns poucos *bytes* são necessários para definir uma nota tocada por um instrumento musical. Por exemplo, um arquivo MIDI com 2 minutos de música pode ter entre 10 Kb e 100 Kb de tamanho, onde essa mesma música apresentada em um arquivo WAV, digitalizada com 8 *bits* a 10 Khz terá um tamanho de 1.2 *Mb*.

3.2 AMBIENTES TRATADORES

São programas que permitem realizar operações simples de processamento no domínio do tempo sobre arquivos de áudio.

Muitos são os *softwares* disponíveis para edição de som, como Cakewalk, Sound Forge, WavLab, Wave Studio, Samplitude, entre outros. Dois destes foram analisados e relatados, são eles o WaveStudio da Creative e o Sound Forge da Sonic Foundry.

3.2.1 WAVE STUDIO DA SOUND BLASTER

O WaveStudio da Creative Technology Ltd. é um *software* de edição de arquivos WAV baseado em *Windows*. Trabalha também com arquivos nos formatos VOC. Caracteriza-se por atualizações de tela de carregamento rápido, permite múltiplas fontes de gravação, controle de *mixer* e controle de CD *player*.

O painel superior mostra um detalhe de um arquivo de áudio cuja visão global é dada no painel inferior, conforme Figura 10. Já a Figura 11, mostra a tela de abertura de arquivos de som.

Figura 10 – Tela principal do WaveStudio

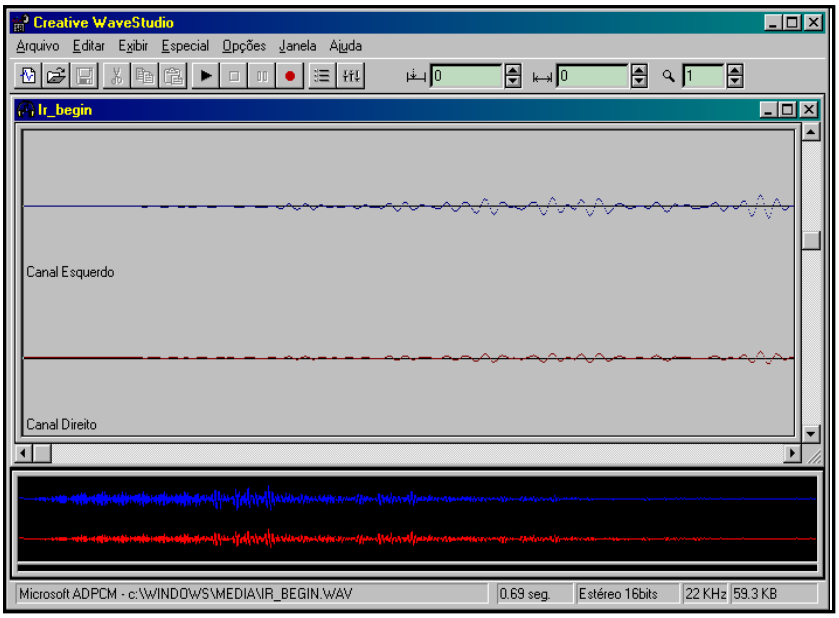
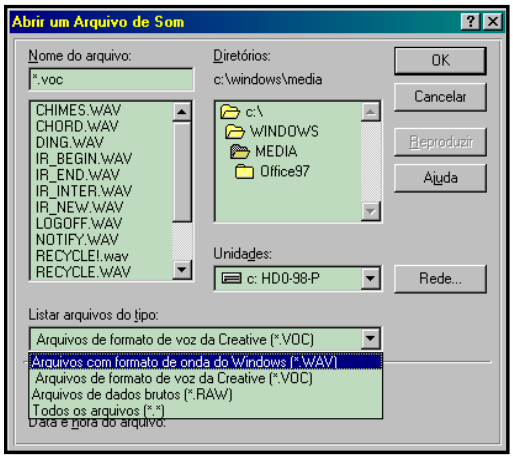


Figura 11 – Tela de abertura de arquivos de som

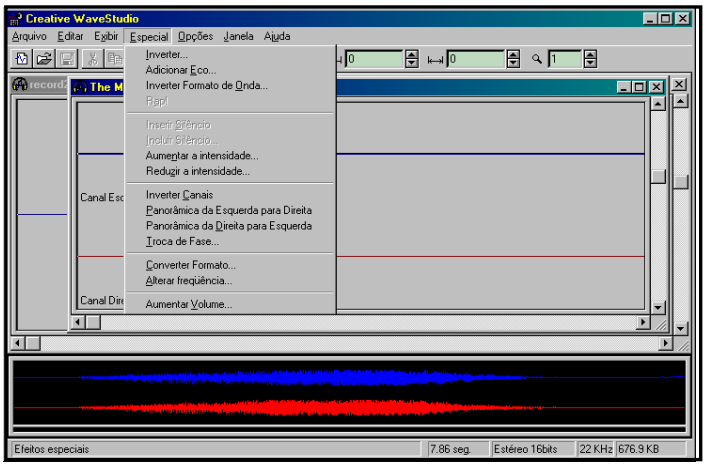


Possibilita a edição de vários arquivos individuais numa mesma tela principal, onde a cada arquivo a ser criado ou aberto para edição gera uma sub-tela com a visualização dos

sinais de áudio digitais e analógicos pertencentes a este arquivo, com opção de "*zoom*" nos dados da onda.

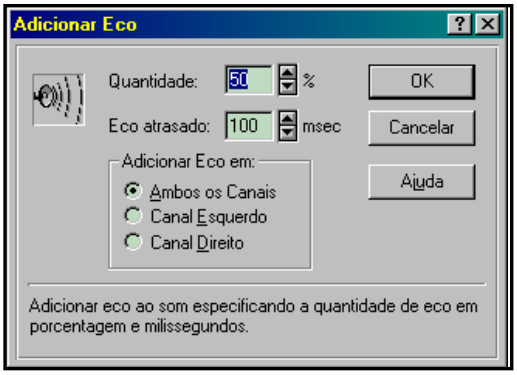
Os recursos de edição incluem cortar, colar, excluir, copiar e marcar partes de um arquivo de som, possibilitando o filtro e o tratamento deste som. Inclui à título de "Especial" como efeito, o *eco*, que possibilita também a inversão do formato da onda, aumentar e reduzir a intensidade, alteração da freqüência, aumentar volume, entre outros, mostrados na Figura 12.

Figura 12 – Opção para tratamento do arquivo de som



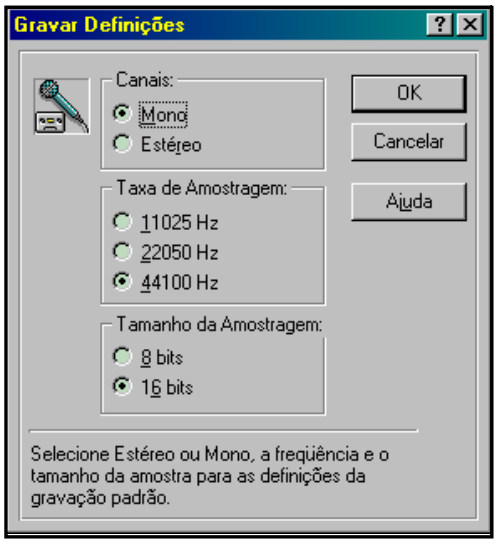
Para a aplicação do *eco*, o wavestudio disponibiliza a quantidade em percentual de taxa de *eco* e o atraso em milissegundos a ser aplicado, podendo ainda escolher o canal de aplicação em caso de som estéreo, como mostra a Figura 13.

Figura 13 – Tela de definição de *eco*



Controles que possibilitam que o usuário possa definir suas preferências de gravação como tipo de canal (mono/estéreo), taxa de amostragem de 11025 Hz, 22050 Hz e 44100 Hz e tamanho da amostragem de 8 ou 16 *bits*, conforme Figura 14.

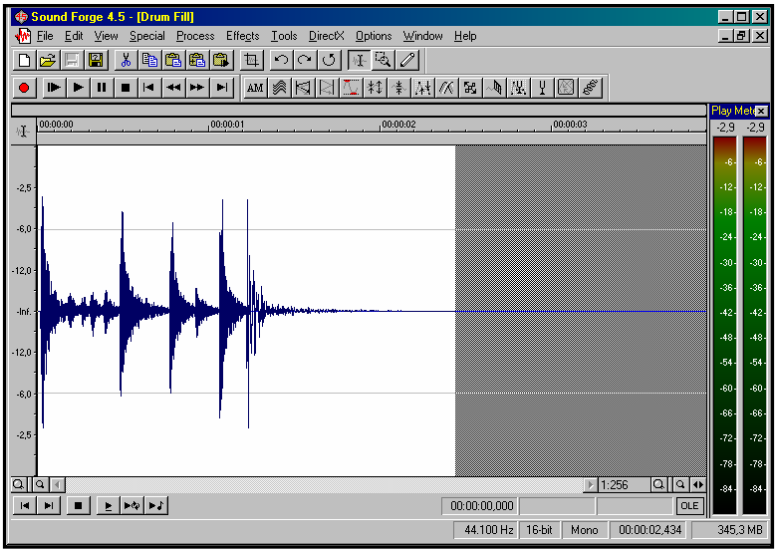
Figura 14 – Tela de definição de gravação



3.2.2 SOUND FORGE

O Sound Forge da Sonic Foundry, segundo [FOU2000], teve introdução desenvolvida por músicos, editores de som, desenhistas de multimídia, desenhistas de jogo, e vários outros, entre eles, profissionais da música e indústrias de som. Estão incluídas características que são específicas a aplicações diferentes, trabalhando com arquivos WAV como padrão, mas trabalha ainda com uma maior variedade de formatos de arquivos se compararmos com o WaveStudio. Inclui tratamento de arquivos MIDI e arquivos de vídeo no formato AVI. A Figura 15, mostra a tela principal do Sound Forge.

Figura 15 – Tela principal do Sound Forge



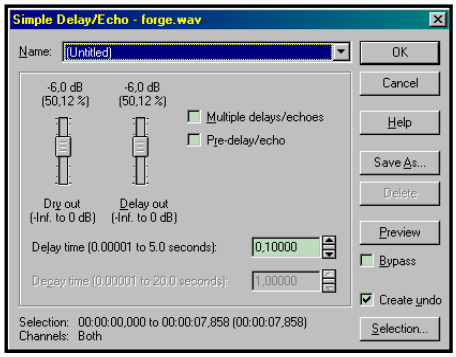
Em sua tela principal, o Sound Forge apresenta várias opções, incluindo edição do arquivo, podendo aplicar efeitos a partes deste arquivo, especialidades como compactação de tempo do áudio, processos de limitação do volume do áudio através de suas ondas, *zoom* de tempo, entre outras.

A nível de efeitos aplicáveis, ele oferece uma variedade muito grande, entre eles o *delay*/*eco*, *reverb*, distorção, *chorus*, *noise gate*. Possibilita aplicar o efeito a apenas um determinado canal no caso de um áudio estéreo. No caso do *eco*, tem recursos a mais como:

- *delay/eco* simples, cria cópias do som original;
- *eco* pré-fixado, cria *ecos* ouvidos antes do som original;
- *delay/eco* múltiplo, significa que em vez de só um *eco*, será reproduzido múltiplos *ecos* que se deterioram do som original. O tempo de decadência determina quanto tempo leva para estes *ecos* diminuírem.

A Figura 16, mostra a tela de aplicação de *eco* do Sound Forge.

Figura 16 – Tela de aplicação de *eco*



O usuário também pode definir suas preferências de gravação como tipo de canal (mono/estéreo), taxa de amostragem de 22050 Hz e tamanho da amostragem de 8 ou 16 *bits*.

É considerado um dos melhores *softwares* para edição de áudio, sendo utilizado por uma grande parte dos profissionais e estúdios de gravação.

4 TÉCNICAS DE PROCESSAMENTO DIGITAL DE SOM

Segundo [DIG2000], para o processamento de som, o computador deve possuir interfaces, dispositivos e programas para processamento de áudio digital. Neles, são criados os materiais sonoros utilizados em títulos multimídia, assim como efeitos especiais e sons sintéticos que serão armazenados e reproduzidos em fitas e CDs de áudio. O espaço em disco rígido deve ser suficiente para acomodar a quantidade desejada de amostras de som. Quanto mais rápido o disco, maior a possibilidade de trabalhar em tempo real com múltiplas trilhas de som, e de utilizar arquivos de áudio com maior taxa de amostragem.

4.1 O DOMÍNIO DE TEMPO

De acordo com [FIL2000], a representação gráfica mais simples dos sinais de som é a representação do domínio do tempo. O tempo é representado no eixo horizontal, e a intensidade ou amplitude do sinal é representada no eixo vertical, freqüentemente medida em decibéis em relação a um valor de referência convencionado.

Conforme a Figura 15, já mostrada anteriormente, onde é apresentada a tela principal do Sound Forge, é dada uma representação no domínio do tempo de um pequeno trecho de bateria.

4.2 O DOMÍNIO DE FREQÜÊNCIAS

Segundo [FIL2000], o matemático francês Jean-Baptiste Joseph Fourier, qualquer função periódica pode ser representada como uma soma de senóides de amplitudes e fases adequadas. Estas amplitudes e fases podem ser obtidas a partir do sinal original por uma operação matemática que é conhecida como a transformação de Fourier. Cada senóide resultante é chamada de uma componente do sinal original.

A lista das amplitudes e fases das componentes é a transformada de Fourier do sinal, muitas vezes também chamada de espectro do sinal. Os gráficos que representam o espectro mostram a amplitude e a fase contra a freqüência.

Ainda segundo [FIL2000], parte da diferença dos timbres dos instrumentos é explicada pelas diferenças de espectro. Sons que apresentam maior proporção de componentes de alta freqüência são tidos como mais “brilhantes”. Em sons complexos, a componente de freqüência mais baixa é normalmente percebida como a altura musical.

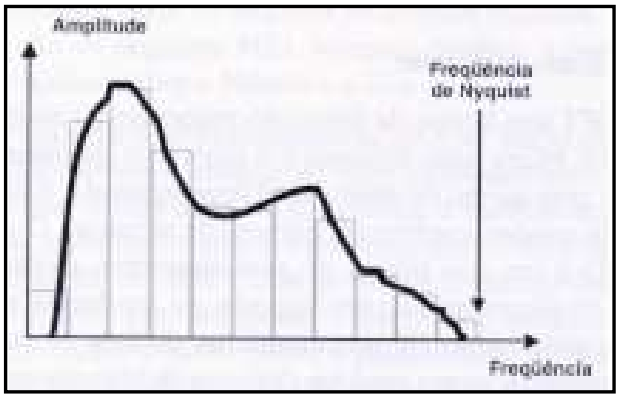
Em um espectro ideal, haveria uma única linha correspondente ao pico, e o resto do espectro seria zero. No espectro real, isto não acontece por causa de erros de aproximação no processo de cálculo, mas a forma do espectro é suficiente para identificar quais são as freqüências presentes, que são representadas pelos picos.

O Processamento no domínio da freqüência são operações que requerem a análise de seqüências de amostras de som. Abrange operações que não dependem do processamento de amostras isoladas, mas requerem o tratamento de seqüências de amostras, como um todo. Suas aplicações incluem:

- a filtragem digital, que tem como aplicação a recuperação de gravações;
- ajustes de duração e altura de amostras de som, usados tanto para fins de sincronização e correção de gravações como para a produção de efeitos especiais de áudio;
- várias técnicas de síntese musical;
- identificação e reconhecimento de voz.

De acordo com [FIL2000], o processamento digital no domínio de freqüência requer o cálculo de uma versão digital de transformação de Fourier (DFT) e sua inversa (IDFT). A passagem da versão contínua para a versão discreta da transformada de Fourier requer sempre uma aproximação, semelhante à que acontece nos processos de amostragem e quantização do sinal. A transformação discreta de Fourier pode ser vista como o resultado da amostragem e quantização do espectro do sinal, conforme Figura 17.

Figura 17 – Discretização do espectro



Um ponto importante é que o sinal de som deve ser dividido em segmentos com ou sem superposição, que são chamados de *janelas* ou *quadros*. A seqüência de amostras de cada quadro deve ser combinada com uma função de enquadramento (*windowing*). Esta função opera como uma envoltória que começa e termina no nível zero, ou próximo dele. Funções de enquadramento normalmente usadas tem formas de retângulo, triângulo e sino. Isso assegura que o quadro comece e termine próximo do silêncio, evitando uma série de problemas técnicos que distorceriam o espectro.

O algoritmo de DFT tem tempo de execução proporcional ao quadrado do número de amostras. Muito mais eficiente é o algoritmo de transformada rápida de Fourier (FFT), cujo tempo de execução proporcional é dada pelo Quadro 1, onde ***n*** é o comprimento do quadro, medido em número de amostras. O algoritmo FFT e seu inverso IFFT, é um dos pilares do processamento digital de sinais, e é freqüentemente implementado com suporte de *hardware* em arquiteturas orientadas para multimídia. Os DSP's são processadores especializados em executar de forma rápida as operações requeridas pela FFT.

Quadro 1 – Tempo de execução proporcional da FFT

"n X log n"

4.3 O DSP

Segundo [FRE1981], o DSP é uma CPU especial para processamento de sinais digitais. Fornece seqüências de instruções extra-rápidas, tais como rotação, adição, multiplicação e adição, comumente usadas em aplicativos de processamento de sinais com muitos cálculos matemáticos.

Uma categoria de técnicas que analisa sinais de origens, tais como som, satélites meteorológicos e monitores de abalos sísmicos. Os sinais são convertidos para dados digitais e analisados com o uso de diversos algoritmos tais como o da transformada rápida de Fourier (FFT), ([ELE1998]).

Nas placas de som, os *chips* DSP são usados para comprimir e descomprimir formatos de áudio, bem como para auxiliar na gravação, reprodução e síntese de voz. Outras utilidades de áudio para os *chips* DSP são:

- técnicas usadas para filtragem, simulação da acústica de alguns ambientes como amplificadores estéreo, os quais são programados para simular platéias de teatro e efeitos de cinema para teatros domésticos;

- a audição de música e outros efeitos especiais que podem ser usados para restaurar ou melhorar o som gravado originalmente.

Segundo [MOO1994] e [SHA1995], o DSP pode converter qualquer som que possa ser capturado por um microfone ou pelos seus conectores *Line-in* e *Cd-in* e gravá-lo digitalmente. Quando se quer reproduzir o som, as amostras são convertidas de volta para sinais de áudio analógico pelo DSP.

O mesmo DSP é usado também para gerar sinais elétricos especiais que são entendidos por instrumentos compatíveis com a interface MIDI.

Aumentam consideravelmente os recursos das interfaces de som, permitindo que muitas funções de processamento sejam realizadas na própria placa, significando ganhos de desempenho.

O Quadro 2, nos mostra como manipular os recursos de leitura e escrita através do DSP específico da *Sound Blaster* 16 *bits*.

Quadro 2 – Algoritmo em Assembly para ler e escrever através do DSP

```
{Função para leitura através do DSP da placa de som Sound Blaster 16 bits}
{O algoritmo lê os sinais digitais capturados do canal de entrada através do DSP}
Function DSPRead : Byte; Assembler;
Asm
  Mov   DX,DataReadyPort  /* variável do tipo word para armazenamento dos sinais
@Loop:
  In    AL,DX        /* captura dos sinais de entrada através do registradores DX e AL
  Test  AL,1 Shl 7   /* insere zeros à direita
  Jz    @Loop
  Mov   DX,ReadDSPPort /* variável do tipo word para armazenamento dos sinais
  In    AL,DX
End;

{Função para escrita através do DSP da placa de som Sound Blaster 16 bits}
{O algoritmo escreve os sinais digitais no canal de saída através do DSP}
Procedure DSPWrite (Data : Byte); Assembler;
Asm
  Mov   DX,WriteDSPPort /* variável do tipo word para armazenamento dos sinais
@Loop:
  In    AL,DX
  Test  AL,1 Shl 7    /* insere zeros à direita
  Jnz   @Loop
  Mov   AL,Data
  Out   DX,AL        /* gera a saída dos sinais através dos registradores DX e AL
End;
```

4.4 EFEITOS SONOROS

A descrição dos efeitos a seguir foram extraídas de documentos da [SWI2000].

4.4.1 REVERB

Reverb é usado para simular a acústica de diferentes ambientes. É a reflexão do som que é ouvido em um quarto com superfícies duras, onde saltos de som ao redor do quarto são reproduzidos durante algum tempo depois do som originalmente gerado.

Este efeito exige muito poder de computação para poder ser bem reproduzido. Com níveis e harmonia variados com o passar do tempo. *Reverb's* normalmente oferecem uma escolha de algoritmos diferentes para simular ambientes diferentes classificados segundo o tamanho e corredores.

4.4.2 DELAY

Delay é um efeito de *eco* que joga de novo o que tocou para uma ou mais vezes depois de um período de tempo. É algo como os *ecos* que se poderia ouvir gritando contra uma parede de canhão. Este tempo é medido em milisegundos (ms).

O Anexo A mostra um algoritmo que implementa o efeito *Delay*.

Neste algoritmo estão definidas macros, conforme mostra quadro 3, que pré-definem parâmetros como:

- Tamanho de *buffer* a ser utilizado para a armazenagem dos sinais de áudio;
- Número de blocos para a geração do efeito D*elay*, bem como o tamanho destes blocos;
- Taxa de amostragem, entre outros.

Quadro 3 – Macros para definição de parâmetros do efeito *delay*

```
#define BFSZ      4096          /* tamanho do buffer */
double Buf[BFSZ];
//float (far *Buf)[BFSZ];
#define BLOCK_LEN    256 /* tamanho do bloco (2 blocos/DMA buffer) */
#define BLOCK_DELAY   10  /* número de blocos para o delay */
#define RATE       22050   /* taxa de amostragem */
#define NA           0.0
#define MAXIMUM_DELAY   30000 /* tamanho máximo do buffer de Delay */
```

Além destas macros, o algoritmo faz chamada à um arquivo externo (anexo B) com várias macros, incluindo tratamento de operacionalização via teclado e mouse.

Cria também variáveis do tipo ponteiro para armazenar os *buffers* dos sinais de áudio, conforme quadro 4.

Quadro 4 – Variáveis do tipo ponteiro para armazenar sinais de áudio

```
signed short (far *tempbuf_16)[BLOCK_DELAY][BLOCK_LEN];
signed char  (far *tempbuf_8)[BLOCK_DELAY][BLOCK_LEN];
signed short (far *Delay_Memory)[MAXIMUM_DELAY];
```

O algoritmo trata os sinais de áudio de forma dinâmica através destes ponteiros. Cria, além destes ponteiros, uma estrutura de dados, conforme quadro 5, para a definição de valores que irão definir as configurações para a definição do tipo de *Delay* a ser gerado, como:

- Tempo de realimentação do sinal;
- Tempo de *Delay*, entre outros.

Quadro 5 – Estrutura para definição de valores e o tipo de *Delay* a ser gerado

```
struct program {
    char (*name)[];
    double dry_mix; /* seqüência do mixer */  /* -2<= <=2 */
    double wet_mix; /* -2<= <=2 */
    double feedback; /* realimentação */ /* -1<= <=1 */
    double delay; /* tempo de delay */ /* 0< <=6 */
} program;
```

O quadro 6, apresenta algumas configurações pré-definidas, bem como a descrição do tipo de *Delay* que irá gerar, e possíveis de serem aplicadas.

Quadro 6 – Configurações pré-definidas de alguns tipos de *Delay*

```
struct program programs[] = {
/* Nome              Dry      Wet    Realimentação      Delay */
{ “Eco Simples",    0.999,  0.999,       0.0,           0.20},
{ "Ecos longos",    0.999,  0.999,       0.7,           0.50},
{ "Inversão de Eco",0.999,  0.999,      -0.7,           0.80}
};
```

Incorpora funções de:

- Armazenamento dos sinais de entrada em formato 16 bits, conforme mostra o quadro 7. Neste mesmo quadro, tem-se também a rotina de conversão dos dados de 16 bits para 8 bits, com a finalidade de minimizar o uso de memória e não ser necessário para a geração dos blocos de *Delay*, sendo novamente convertidos para 16 bits somente para reprodução do som final. É aplicado, nestes sinais, mais realimentação no sinal para gerar maior tempo de vida na reprodução do som.

Quadro 7 – Armazenamento dos sinais de entrada

```
 void far inthandler_16(void)
   {
     int i;
 /* Copia dados dos buffers de entrada para os buffers temporáros em 16
    bits */
_fmemcpy((*tempbuf_16)[curinbuf], blockptr_16[curblock_16], 2*BLOCK_LEN);

 /* Conversão dos dados de 16 para 8 bits */
     for (i = 0; i < BLOCK_LEN; i++)
     {
         /* leitura da entrada */
         data = (*tempbuf_16)[curinbuf][i];
         outval = (*Delay_Memory)[delay_count];
         /* insere mais realimentação */
         temp = (long)((inval = (double)data) + outval *
                ]programs[cp].feedback);
         if(temp > 32767.0)
             temp = 32767;
         else if(temp < -32768.0)
             temp = -32768;
         (*Delay_Memory)[delay_count] = (int)temp;
         delay_count = (delay_count + 1) % delay_length;
         /* escreve a saída */
         (*tempbuf_8)[curinbuf][i] = ((signed char)(data >> 8));
   }
    /* altera os blocos temporários de entrada */
     curinbuf = (curinbuf+1) % BLOCK_DELAY;
   }
```

- Reprodução dos sinais armazenados em 8 bits, onde estes sinais, conforme quadro 8, são transferidos para os *buffers* de saída. Tem-se ainda nesta função, a alteração dos blocos de *Delay*, bem como o cálculo da amplitude, para gerar o tempo de retardo e a reprodução do efeito *Delay*.

Quadro 8 – Alteração dos blocos de saída e cálculo da amplitude

```
 void far inthandler_8(void)
/* Copia dados em 8-bit dos buffers temporários p/ o buffer de saída*/
 _fmemcpy(blockptr_8[curblock_8], (*tempbuf_8)[curoutbuf], BLOCK_LEN);
    /* alteração do tempo do bloco de saída */
     curoutbuf = (curoutbuf+1) % BLOCK_DELAY;
   }
 int avg_vol(short far *buf16)
  /* Cálculo da amplitude */
   {
     int i;
     long sum = 0;
     for (i = 0; i < BLOCK_LEN; i++)
       sum += abs(buf16[i]);
     return sum / BLOCK_LEN;
   }
```

- Inicialização de captura e reprodução dos sinais de áudio, de acordo com o quadro 9, verificando-se também a taxa de amostragem destes sinais para filtrar sinais abaixo da taxa permitida conforme parâmetro inicial que é 22.050 Mhz.

Quadro 9 – Inicialização de captura e reprodução dos sinais de áudio

```
void start_sound (void)
{
/* busca dos parâmetros */
    delay_length = (unsigned int)(programs[cp].delay * (double)RATE);
    if(delay_length > MAXIMUM_DELAY) {
        printf("Taxa de amostragem muito baixa.\n");
        exit(1);
    }
    start_output8();
    start_input16();
}
```

4.4.3 PHASER

Phasers usam um baixo oscilador de freqüência interno para automaticamente o movimento ser inserido de cima para baixo na resposta de freqüência e o espectro de freqüência.

Phasing trabalha misturando o sinal original com o som que é fase trocada em cima da freqüência do espectro.

4.4.4 FLANGER

Flangers, misturam um sinal atrasado e variado, aproximadamente de 5 a 15 ms, do som original, produzindo uma série de entradas na resposta de freqüência do sinal. A diferença importante entre *flanger* e *phaser* é que um *flanger* produz um número grande de sinais, isso é musicalmente relacionado, enquanto um *phaser* produz um número pequeno de sinais, sendo espalhado uniformemente pelo espectro de freqüência.

5 DESENVOLVIMENTO DO PROTÓTIPO

Com base nos conceitos apresentados nos capítulos anteriores, e nos materiais adquiridos, tornou-se possível o desenvolvimento do protótipo de uma ferramenta para geração de efeitos sonoros para instrumentos musicais. Neste capítulo, será abordado a especificação , implementação e o funcionamento do protótipo.

5.1 ESPECIFICAÇÃO DO PROTÓTIPO

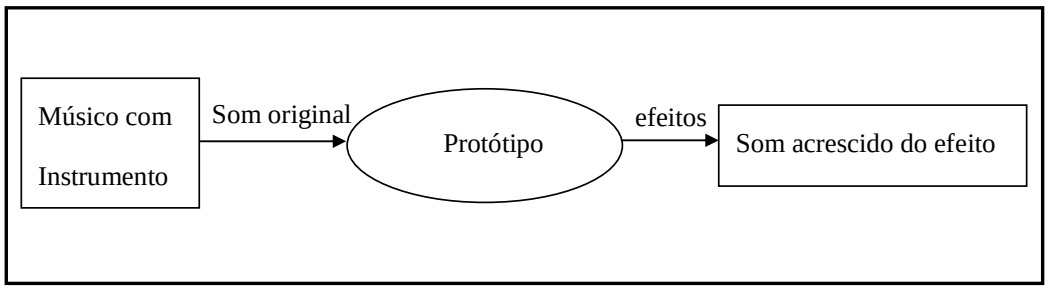
Segundo [MEL1990], para o desenvolvimento de sistemas, a prototipação representa uma boa solução para a maioria dos problemas.

A metodologia de prototipação de sistemas utilizada neste protótipo é a *prototipação evolutiva*. Conforme [MEL1990] e [CAN1997], na prototipação evolutiva, o produto final será o próprio sistema, na sua forma mais aperfeiçoada. A prototipação evolutiva é usada na identificação gradual do problema e na construção de modelos concretos, adaptados e corrigidos a medida que o usuário e o analista vão conhecendo a realidade e a solução do problema.

5.1.1 DIAGRAMA DE CONTEXTO

O Diagrama de Contexto é uma representação gráfica do sistema como um todo e os seus relacionamentos. Na Figura 18, tem-se como escopo deste protótipo, o músico com instrumento musical gerando um som puro e original, a geração dos efeitos e a saída do som acrescido do efeito selecionado.

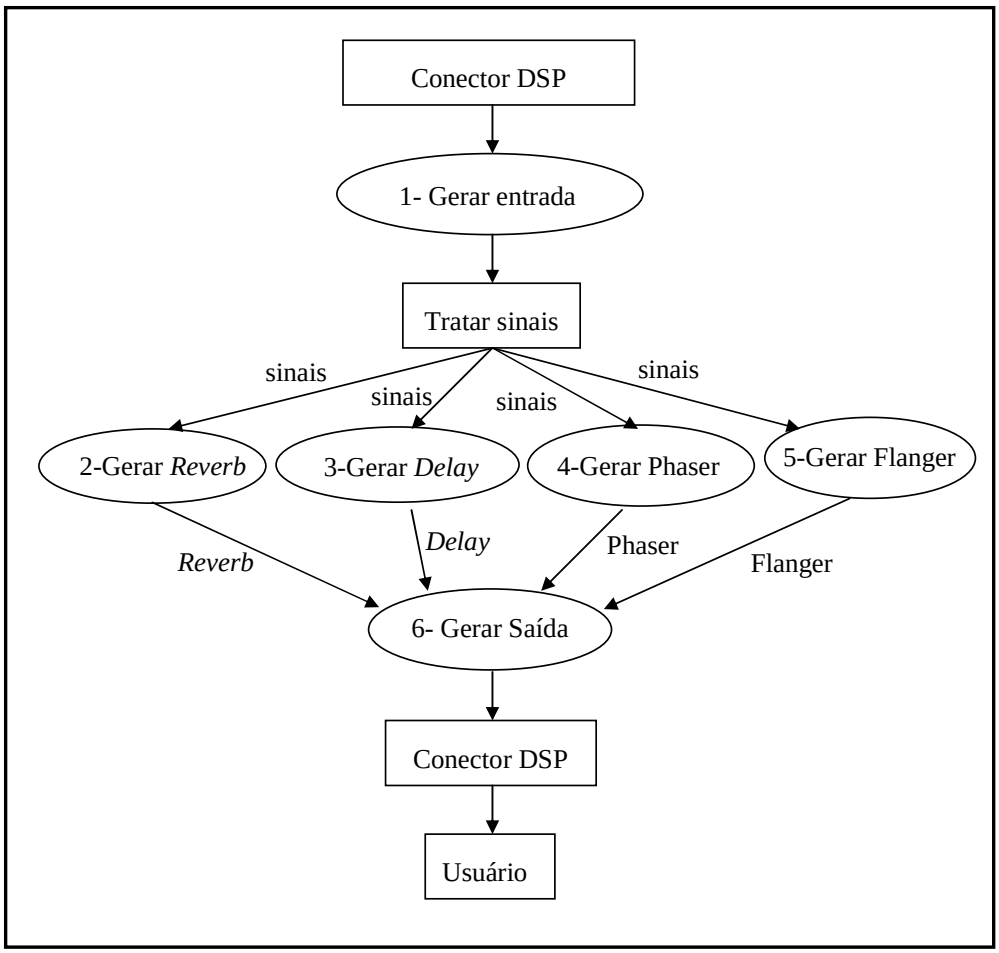
Figura 18 – Diagrama de Contexto



5.1.2 DIAGRAMA DE FLUXO DE DADOS

Na Figura 19, encontra-se o Diagrama de Fluxo de Dados de nível 1, que descreve o fluxo dos sinais sonoros e as transformações que são aplicadas à medida que o som é gerado.

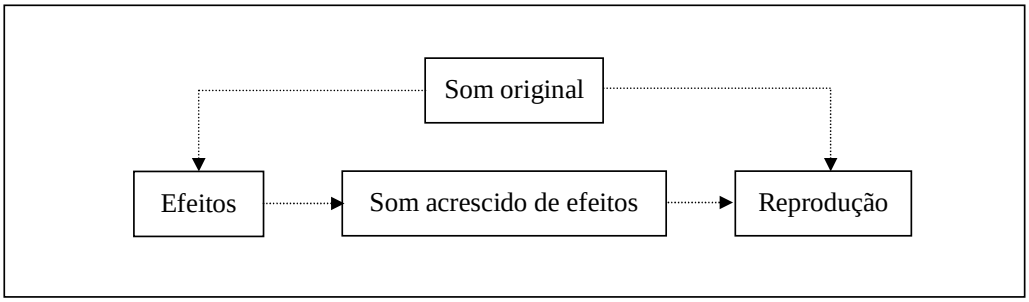
Figura 19 – Diagrama de Fluxo de Dados



5.1.3 MER

Na Figura 20 encontra-se o Modelo de Entidade e Relacionamento

Figura 20 – Modelo de Entidade e Relacionamento



5.1.4 ESTRUTURA DO PROTÓTIPO

O protótipo segue uma programação estruturada baseada em listas lineares, tomando como pressuposto os sinais de entrada.

A captura dos sinais digitais da entrada *MIC* da placa de som é feita através do DSP da própria placa. O DSP da placa se encarrega de converter os sinais (CAD).

Os sinais digitais de som quando armazenados são inseridos em uma lista denominada de *fila*, utilizando alocação dinâmica, uma vez que posições de memória são alocadas (ou desalocadas) na medida em que são necessárias.

5.2 IMPLEMENTAÇÃO DO PROTÓTIPO

Ao estudar as formas de acesso e de tratamento dos sinais gerados pelas placas de som, este protótipo teve sua implementação no ambiente *Delphi* 5.0 da *Inprise Corporation* por disponibilizar os recursos necessários para a realização deste trabalho.

Para tanto foram utilizados, além de componentes comuns do *Delphi*, componentes de multimída da empresa *SWIFT Software* [SWI2000], componentes estes que possibilitam o

tratamento dos sinais sonoros, de forma que conseguem reproduzir estes sinais, acrescidos de efeitos.

5.2.1 APRESENTAÇÃO DOS COMPONENTES MULTIMÍDIA

Considerando que os sinais que transitam pela placa de som são capturados em formato digital, necessitam ser de alguma forma, armazenados.

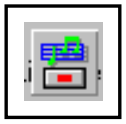
Para armazenar estes sinais, os componentes multimídia adotam o formato WAV (seção 3.1.1), e todo o conteúdo destes sinais são armazenados em áreas de memória para poder reproduzir o som original acrescido do efeito selecionado em um tempo aceitável.

No total foram utilizados nove tipos de componentes de multimídia da [SWI2000], os quais estão descritos abaixo.

5.2.1.1 COMPONENTE MMWAVEIN

O componente `MMWaveIn` tem como função, estabelecer um canal de comunicação com a placa de som através do *driver* da placa. Ele lê e interpreta os sinais capturados pela entrada *MIC* da placa e os armazena em *buffers* de dados na área de memória em formato WAV. A Figura 21, mostra o *ícone* do componente `MMWaveIn`.

Figura 21 - Componente `MMWaveIn`



Dentre as propriedades principais do componente, pode-se descrever:

- BitLength: define o tamanho de amostragem utilizada na captura dos sinais de entrada e saída, podendo ser a 8 ou a 16 *bits*;
- BufferSize: define o tamanho da área de memória alocada para a guarda dos sinais antes de serem redirecionados para o *buffer* do componente `MMRingBuffer`;

- SampleRate: define a taxa de amostragem medida em HZ.

5.2.1.2 COMPONENTE MMRINGBUFFER

O componente `MMRingBuffer` tem como função, garantir a reprodução dos sinais capturados de maneira contínua, gerando uma alimentação circular nos sinais de áudio.

Considerando que há um tempo gasto pela CPU para processar os *buffers* de sinais de áudio armazenados, é necessário que se tenha um *buffer* intermediário entre os sinais puros e os sinais que estão sendo processados pelos algoritmos dos componentes de efeitos, evitando trechos mudos de som. A Figura 22, mostra o *ícone* do componente `MMRingBuffer`.

Figura 22 - Componente `MMRingBuffer`

Dentre as propriedades principais do componente, pode-se descrever:

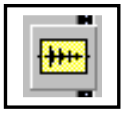
- BufferSize: define o tamanho da área de memória alocada para a guarda dos sinais antes de serem redirecionados para o *buffer* do componente subseqüente, que será o componente responsável por gerar o efeito selecionado;

- NumBuffers: pode-se definir a quantidade de *buffers* usados, neste caso a área total se dá pelo *BufferSize X NumBuffers*.

5.2.1.3 COMPONENTE MMREVERB

O componente `MMReverb` tem como função, gerar os efeitos *reverb* e *delay*. Ele lê os sinais armazenados no `MMRingBuffer` e os processa. Após alterados estes sinais, o `MMReverb` direciona-os para o *buffer* do componente `MMWaveOut` para a reprodução do som com o efeito. A Figura 23, mostra o *ícone* do componente `MMReverb`.

Figura 23 - Componente `MMReverb`



Este componente, apesar de se chamar `MMReverb`, pode gerar também o efeito *delay*, necessitando simplesmente alterar parâmetros como:

- ganho do sinal de entrada;
- ganho do sinal de saída;
- taxa de reflexão;
- valor de realimentação do sinal que está sendo gerado;
- ativar e desativar filtro.

5.2.1.4 COMPONENTE MMPHASESHIFT

O componente `MMPhaseShift` tem como função, gerar o efeito *phaser*. Ele lê os sinais armazenados no `MMRingBuffer` e os processa. Após alterados estes sinais, o `MMPhaseShift` direciona-os para o *buffer* do componente `MMWaveOut` para a reprodução do som com o efeito *phaser*. A Figura 24, mostra o *ícone* do componente `MMPhaseShift`.

Figura 24 - Componente `MMPhaseShift`

5.2.1.5 COMPONENTE MMFLANGER

O componente `MMFlanger` tem como função, gerar o efeito *flanger*. Ele lê os sinais armazenados no `MMRingBuffer` e os processa. Após alterados estes sinais, o `MMFlanger` direciona-os para o *buffer* do componente `MMWaveOut` para a reprodução do som com o efeito *flanger*. A Figura 25, mostra o *ícone* do componente `MMFlanger`.

Figura 25 - Componente `MMFlanger`

5.2.1.6 COMPONENTE MMWAVEOUT

O componente `MMWaveOut` tem a função inversa do `MMWaveIn`. Seu *buffer* é alimentado pelos componentes de geração de efeitos, `MMReverb`, `MMPhaseShift` e `MMFlanger`.

O `MMWaveOut` reproduz os sinais armazenados em seu *buffer* no formato WAVE, sinais estes que podem ou não terem sido alterados pelos componentes de geração de efeitos, (condição de ligar ou desligar o efeito), e os devolve para a placa de som, estabelecendo um canal de comunicação com a placa de som através do *driver* da placa. A Figura 26, mostra o *ícone* do componente `MMWaveOut`.

Figura 26 - Componente `MMWaveOut`

Dentre as propriedades principais do componente, pode-se descrever:

- BufferSize: define o tamanho da área de memória alocada para a guarda dos sinais antes de serem redirecionados para o *driver* da placa de som para ser reproduzido;

- NumBuffers: pode-se definir a quantidade de *buffers* usados, neste caso a área total se dá pelo *BufferSize X NumBuffers*.

5.2.1.7 COMPONENTE MMCONNECTOR

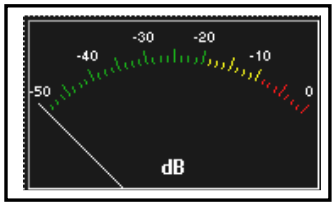
O componente `MMConnector` tem como função, receber os sinais do `MMWaveIn` ou `MMWaveOut` e direcioná-los para os componentes `MMMeter` e `MMOscope` para a representação gráfica. Caso se queira representar graficamente os sinais de entrada puros, pode-se usar somente o `MMWaveIn` ligado com o `MMConnector` e direcioná-lo para os componentes `MMMeter` e `MMOscope`. A Figura 27, mostra o *ícone* do componente `MMConnector`.

Figura 27 - Componente `MMConnector`

5.2.1.8 COMPONENTE MMMETER

O componente `MMMeter` tem como função, receber os sinais do `MMConnector` e representá-los graficamente. Ele gera uma representação com medidas em decibéis dos sinais de som que estão sendo reproduzidos. A Figura 28, mostra o *ícone* do componente `MMMeter`.

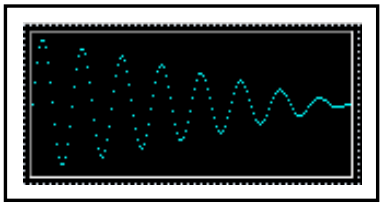
Figura 28 - Componente `MMMeter`



5.2.1.9 COMPONENTE MMOSCOPE

O componente `MMOscope` também tem como função, receber os sinais do `MMConnector` e representá-los graficamente. Ele gera gráficos senoidais gerados pelos sinais de som que estão sendo reproduzidos. A Figura 29, mostra o *ícone* do componente `MMOscope`.

Figura 29 - Componente `MMOscope`



De acordo com as descrições anteriores dos componentes, os efeitos são obtidos através das seqüências de ligações observadas na Figura 30, onde tem-se respectivamente (cada linha) os efeitos de: *reverb*, *delay*, *phaser* e *flanger*.

Figura 30 – Interligação dos componentes

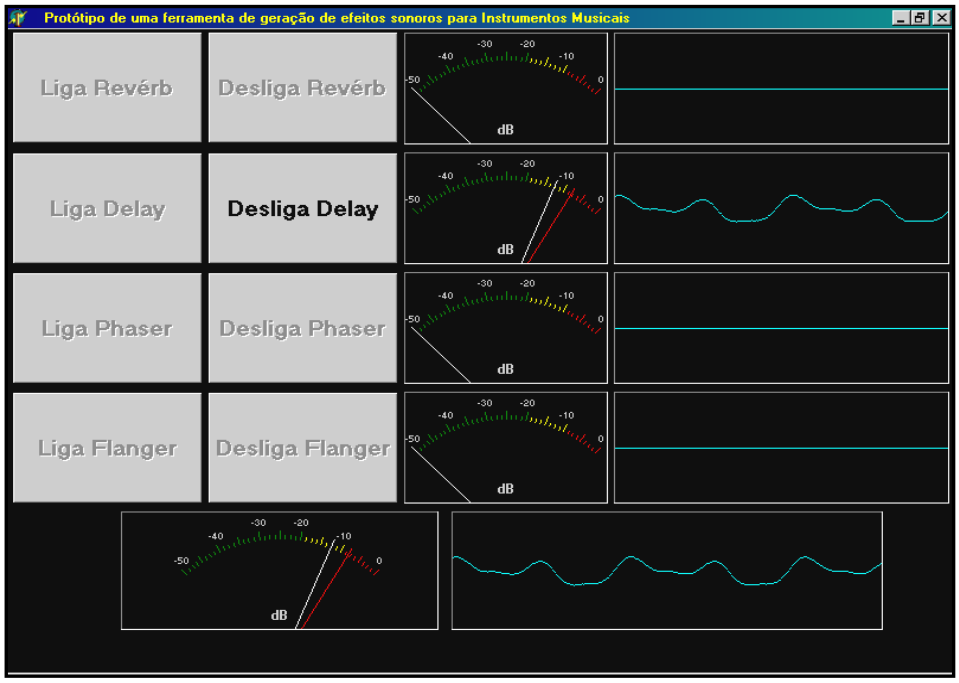
5.3 FUNCIONAMENTO DO PROTÓTIPO

Neste ponto será descrito o funcionamento do protótipo, bem como suas características e funcionalidade.

O protótipo não tem por finalidade gerar arquivos de som para serem armazenados em arquivos e posteriormente serem editados, como fazem alguns *softwares* de edição de som como o WAVE Studio e o Sound Forge (seção 3.2), mas sim gerar o som acrescido do efeito em “tempo real”, ou seja, o músico está tocando uma guitarra elétrica que está conectada na entrada *MIC* da placa de som, e escuta o som que está sendo emitido pela guitarra acrescido do efeito que ele selecionou quase simultaneamente.

A Figura 31, apresenta a tela do protótipo.

Figura 31 - Tela do Protótipo



O protótipo possui uma única tela principal que está dividida em três áreas:

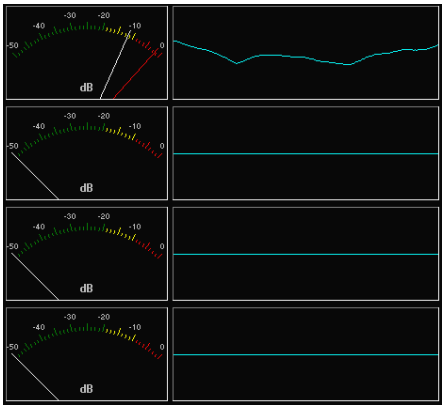
- área 1: área de seleção (à esquerda) - composta pelos botões de liga e desliga;
- área 2: área de representação dos sinais acrescidos de efeito (à direita);
- área 3: área de representação dos sinais puros (base);

Na área 1, estão disponíveis os botões utilizados para ligar e desligar o efeito. Estão disponíveis os efeitos *reverb*, *delay*, *phaser* e *flanger*, conforme Figura 32.

Figura 32 - Área 1: seleção de efeitos

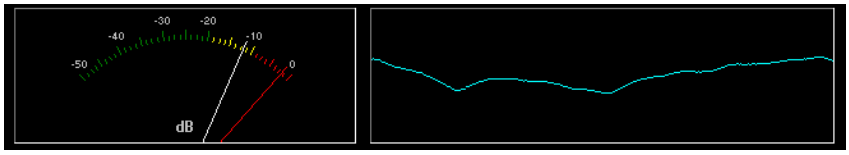
Na área 2, mostra respectivamente o gráfico dos sinais de som quando um efeito é ligado. São geradas amostras dos sinais pelo `MMMeter` (seção 5.2.1.8) e pelo `MMOscope` (seção 5.2.1.9), conforme Figura 33.

Figura 33 - Área 2: área de representação dos sinais acrescidos de efeito



Na área 3, o `MMMeter` e o `MMOscope` mostram graficamente os sinais de som puro, conforme Figura 34.

Figura 34 - Área 3: área de representação dos sinais puros



6 RESULTADOS FINAIS

6.1 TESTES APLICADOS

Primeiro foram aplicados testes com um microfone conectado na entrada *MIC* da placa de som. Nestes testes, foram pronunciadas várias frases para a observação da reprodução do som. Observou-se que há um atraso na reprodução do sinal quando é ligado algum efeito, este atraso se dá pelo tempo de processamento para gerar o efeito e montagem dos *buffers* dos sinais.

Como segundo teste, foi utilizada uma guitarra elétrica também conectada na entrada *MIC* da placa de som. Foram tocados alguns acordes, e observou-se que o atraso foi menor se comparado com o atraso com o uso do microfone.

6.2 CONSIDERAÇÕES FINAIS

O protótipo alcançou os objetivos principais. Adiciona efeitos e os reproduz de modo satisfatório. Dos efeitos gerados através da placa de som, pôde-se comprovar que é possível utilizar o computador para o uso com instrumentos musicais para a geração de efeitos, através dos resultados dos testes aplicados.

Através do estudo realizado, verificou-se que muitos fatores envolvem o tratamento dos sinais de áudio. Deve-se considerar desde a captura de um sinal de áudio puro, tratar estes sinais e reproduzi-los, acrescidos de efeitos de forma que o tempo de reprodução para isto seja satisfatório.

O que se verificou durante o desenvolvimento deste trabalho é que a manipulação dos sinais de áudio através das placas de som, das mais variadas marcas e modelos, é algo muito pouco exposto pelos fabricantes e pelas pessoas que atuam na área, sendo importante considerá-la durante a elaboração de um projeto.

6.3 LIMITAÇÕES

Como limitação o protótipo apresentou o uso exclusivo dos recursos da placa de som quando ativado algum dos efeitos disponíveis, deixando estes recursos totalmente dedicados ao aplicativo (protótipo).

O mesmo acontece quando um *software* de áudio, estilo Winamp da NullSoft Inc. que se estiver sendo utilizado, este estará utilizando os recursos da placa de som, fazendo com que o protótipo consiga ser executado mas incapaz de acionar alguns dos efeitos.

6.4 EXTENSÕES

Como extensões deste protótipo, sugere-se:

- melhorias na reprodução para uso em tempo real;
- maiores testes em diferentes equipamentos;
- uso de outros algoritmos de tratamento das ondas sonoras para a sua reprodução em tempo mais satisfatório;
- implementação de outros efeitos como: vários tipos de distorção, *wah-wah*, etc.;
- construção de um DSP acoplado a um pedal de guitarra para uso exclusivo do recurso;
- síntese de instrumentos;
- análise de sinais sonoros.

ANEXO A: ALGORITMO DE IMPLEMENTAÇÃO DO EFEITO DELAY

```
#ifndef __LARGE__
#error DELAY1 must be compiled in the LARGE model
#endif

#define BFSZ      4096                    /* tamanho do buffer */
double Buf[BFSZ];
//float (far *Buf)[BFSZ];

#define BLOCK_LEN      256            /* tamanho do bloco (2 blocos/DMA buffer) */
#define BLOCK_DELAY     10            /* número de blocos para o delay */
#define RATE          22050           /* taxa de amostragem                */
#define NA             0.0
#define MAXIMUM_DELAY   30000    /* tamanho máximo do buffer de Delay */

#include <alloc.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <mem.h>
#include <stdio.h>
#include <stdlib.h>

#include "..\sbplugin.h"                /* ANEXO B

int curinbuf  = BLOCK_DELAY-1;
int curoutbuf = 0;

signed short (far *tempbuf_16)[BLOCK_DELAY][BLOCK_LEN];
signed char  (far *tempbuf_8)[BLOCK_DELAY][BLOCK_LEN];
signed short (far *Delay_Memory)[MAXIMUM_DELAY];

typedef union bw {
   unsigned char b[2];
   int w;
} bw;
typedef union wl {
   unsigned int w[2];
   long l;
} wl;

struct program {
   char (*name)[];
   double dry_mix;          /* seqüência do mixer          */   /* -2<= <=2 */
   double wet_mix;                                              /* -2<= <=2 */
   double feedback;        /* realimentação */                  /* -1<= <=1 */
   double delay;           /* tempo de delay */                   /* 0< <=6 */
} program;

char (*programname)[]=" Delay (algorithm 1) Presets ";
#define NPROGS (sizeof(programs) / sizeof(struct program))
int i,j;
char Back=8<<4;

char cp = 0;
struct program programs[] = {
   /* Nome                  Dry       Wet       Realimentação     Delay */
   { "Eco Simples",          0.999,     0.999,      0.0,              0.20},
```

```
    { "Ecos longos",          0.999,     0.999,      0.7,                0.50},
    { "Inversão de Eco",      0.999,     0.999,     -0.7,                0.80}
};

/* variáveis intermediárias */
unsigned int delay_count=0,delay_length;
double inval,outval;
long temp;
int data;

void far inthandler_16(void)

  {
    int i;

   /* Copia dados dos buffers de entrada para os buffers temporáros em 16 bits */
    _fmemcpy((*tempbuf_16)[curinbuf], blockptr_16[curblock_16], 2*BLOCK_LEN);

   /* Conversão dos dados de 16 para 8 bits */
    for (i = 0; i < BLOCK_LEN; i++)
    {
        /* leitura da entrada */
        data = (*tempbuf_16)[curinbuf][i];

        outval = (*Delay_Memory)[delay_count];
        /* insere mais realimentação */
        temp = (long)((inval = (double)data) + outval * programs[cp].feedback);
        if(temp > 32767.0)
            temp = 32767;
        else if(temp < -32768.0)
            temp = -32768;
        (*Delay_Memory)[delay_count] = (int)temp;
        delay_count = (delay_count + 1) % delay_length;


        /* escreve a saída */
        (*tempbuf_8)[curinbuf][i] = ((signed char)(data >> 8));
    }
   /* altera os blocos temporários de entrada */
    curinbuf = (curinbuf+1) % BLOCK_DELAY;
  }

void far inthandler_8(void)

   /* Copia dados em 8-bit dos buffers temporários para o buffer de saída */
    _fmemcpy(blockptr_8[curblock_8], (*tempbuf_8)[curoutbuf], BLOCK_LEN);
   /* alteração do tempo do bloco de saída */
    curoutbuf = (curoutbuf+1) % BLOCK_DELAY;
  }

int avg_vol(short far *buf16)
 /* Cálculo da amplitude */
  {
    int i;
    long sum = 0;
```

```
    for (i = 0; i < BLOCK_LEN; i++)
      sum += abs(buf16[i]);

    return sum / BLOCK_LEN;
  }

void start_sound (void)
{
    /* busca dos parâmetros */
    delay_length = (unsigned int)(programs[cp].delay * (double)RATE);
    if(delay_length > MAXIMUM_DELAY) {
        printf("Taxa de amostragem muito baixa.\n");
        exit(1);
    }

    start_output8();
    start_input16();
}


void main(void)
  {
    if (!(init_sb(RATE)==1))
    {
        printf("Sound Blaster 16\n");
        exit(EXIT_FAILURE);
    }

    set_blocklength(BLOCK_LEN);
    get_buffers();

    install_handler_16(inthandler_16);
    install_handler_8(inthandler_8);

   /* Limpa e aloca todos os buffers*/
    tempbuf_16 = farmalloc(sizeof(*tempbuf_16));
    tempbuf_8  = farmalloc(sizeof(*tempbuf_8));

    Delay_Memory  = farmalloc(sizeof(*Delay_Memory));

    if ((tempbuf_16==NULL) || (tempbuf_8==NULL) ||
        (Delay_Memory==NULL))
    {
        printf("memória insuficiente para alocar buffers rs\n");
        exit(EXIT_FAILURE);                    /* término do programa por falta de memória */
    }

    _fmemset((*tempbuf_16), 0x00, sizeof(*tempbuf_16));
    _fmemset((*tempbuf_8),  0x00, sizeof(*tempbuf_8));
    _fmemset((*Delay_Memory),  0x00, sizeof(*Delay_Memory));

    do
     {
       if (KeyDown[UpScan]) if (cp>0)
       {
         cp -= 1;
         delay(100);
         stop_sound();
         free_buffers();
         get_buffers();
         start_sound();
```

```
    } else;
    else
    if (KeyDown[DownScan]) if (cp<((NPROGS>21)?21:NPROGS)-1)
    {
      cp += 1;
      delay(100);
      stop_sound();
      free_buffers();
      get_buffers();
      start_sound();
    }
   }
  while (!WasDown[EscScan] && !error);
  SetOldKeyInt();
  stop_sound();
  if (error)
  {
     printf("\a       ********** ERRO ********** \n");
     error = 0;
  }
  farfree(Delay_Memory);
  farfree(tempbuf_8);
  farfree(tempbuf_16);
  free_buffers();
  shutdown_sb();
}
```

ANEXO B: ARQUIVO DE PARÂMETROS SBPLUGIN.H

```
#define TRUE  1
#define FALSE 0

#define  EscScan         0x01 //Escape
#define  Key1Scan        0x02 //1
#define  Key2Scan        0x03 //2
#define  Key3Scan        0x04 //3
#define  Key4Scan        0x05 //4
#define  Key5Scan        0x06 //5
#define  Key6Scan        0x07 //6
#define  Key7Scan        0x08 //7
#define  Key8Scan        0x09 //8
#define  Key9Scan        0x0A //9
#define  Key0Scan        0x0B //0
#define  MinusScan       0x0C //-
#define  PlusScan        0x0D //+
#define  BackScan        0x0E //Back Space
#define  TabScan         0x0F //Tab
#define  QScan           0x10 //Q
#define  WScan           0x11 //W
#define  EScan           0x12 //E
#define  RScan           0x13 //R
#define  TScan           0x14 //T
#define  YScan           0x15 //Y
#define  UScan           0x16 //U
#define  IScan           0x17 //I
#define  OScan           0x18 //O
#define  PScan           0x19 //P
#define  LQScan          0x1A //[
#define  RQScan          0x1B //]
#define  EnterScan       0x1C //Enter
#define  CtrlScan        0x1D //Ctrl
#define  AScan           0x1E //A
#define  SScan           0x1F //S
#define  DScan           0x20 //D
#define  FScan           0x21 //F
#define  GScan           0x22 //G
#define  HScan           0x23 //H
#define  JScan           0x24 //J
#define  KScan           0x25 //K
#define  LScan           0x26 //L
#define  PointCommaScan  0x27 //;
#define  LApostrofScan   0x28 //'
#define  RApostrofScan   0x29 //`
#define  LShiftScan      0x2A //LShift
#define  BackSlashScan   0x2B //\
#define  ZScan           0x2C //Z
#define  XScan           0x2D //X
#define  CScan           0x2E //C
#define  VScan           0x2F //V
#define  BScan           0x30 //B
#define  NScan           0x31 //N
#define  MScan           0x32 //M
#define  CommaScan       0x33 //,
#define  PointScan       0x34 //.
#define  SlashScan       0x35 ///
#define  RShiftScan      0x36 //RShift
#define  PrtScan         0x37 //Print Screen
```

```
#define  AltScan         0x38 //Alt
#define  SpaceScan       0x39 //Space
#define  CapsScan        0x3A //Caps Lock
#define  F1Scan          0x3B //F1
#define  F2Scan          0x3C //F2
#define  F3Scan          0x3D //F3
#define  F4Scan          0x3E //F4
#define  F5Scan          0x3F //F5
#define  F6Scan          0x40 //F6
#define  F7Scan          0x41 //F7
#define  F8Scan          0x42 //F8
#define  F9Scan          0x43 //F9
#define  F10Scan         0x44 //F10
#define  NumScan         0x45 //Num Lock
#define  ScrollScan      0x46 //Scroll Lock
#define  HomeScan        0x47 //Home
#define  UpScan          0x48 //Up
#define  PgUpScan        0x49 //Page Up
#define  GrMinusScan     0x4A //Grey -
#define  LeftScan        0x4B //Left
#define  GrMiddleLScan   0x4C //Grey 5
#define  RightScan       0x4D //Right
#define  GrPlusScan      0x4E //Grey +
#define  EndScan         0x4F //End
#define  DownScan        0x50 //Down
#define  PgDnScan        0x51 //Page Down
#define  InsScan         0x52 //Insert
#define  DelScan         0x53 //Delete
#define  F11Scan         0x57 //F11
#define  F12Scan         0x58 //F12

typedef unsigned char BYTE;

extern int base_io;
extern int irq;
extern int dma_8;
extern int dma_16;

extern int majv,minv;

extern int  init_sb(int rate);
extern void shutdown_sb(void);

extern void write_mixer(BYTE reg, BYTE value);
extern BYTE read_mixer(BYTE reg);

extern void set_blocklength(int block_length);

extern void get_buffers(void);
extern void free_buffers(void);

extern void install_handler_8(void (far *handler)());
extern void install_handler_16(void (far *handler)());

extern void start_input16(void);
extern void start_output8(void);

extern void stop_sound(void);

extern volatile long intcount;
extern volatile long intcount_8;
extern volatile long intcount_16;

extern volatile int error;
```

```
extern int curblock_8;
extern int curblock_16;

extern unsigned char far *blockptr_8[2];
extern unsigned char far *buf_8;

extern signed short far *blockptr_16[2];
extern signed short far *buf_16;

extern InitNewKeyInt (void);
extern SetOldKeyInt (void);
extern ClearWasDownArray (void);

extern char KeyDown[128];
extern char WasDown[128];

extern char mouse_installed (void);
extern int get_mouse_x (void);
extern int get_mouse_y (void);
extern char get_mouse_buttons (void);
extern void set_limits (int mnh, int mxh, int mnv, int mxv);
extern void set_mouse_position (int x, int y);
```

Referências bibliográficas

[CAN1997] CANTÚ, Marco. **Dominando o delphi 3: "A Bíblia".** São Paulo, 1997.

[DIG2000] DIGITAL, Audio. **Audio News.** 2000. Endereço Eletrônico: http://www.audiodigital.com.br Data da consulta: 30/09/2000.

[ELE1998] ELECTRONICS, Elecktor. **The Electronics & Computer Magazine**. 1998.

[FIL2000] FILHO, Wilson de Pádua Paula. **Multimídia – conceitos e aplicações**. Rio de Janeiro: Editora LTC, 2000.

[FOU2000] FOUNDRY, Sonic. **Sound Forge.** 2000. Endereço Eletrônico: http://www.sonicfoundry.com Data da consulta: 08/10/2000.

[FRE1981] FREEDMAN, Alan. **Dicionário de Informática.** São Paulo : Makron Books, 1981.

[GON2000] GONTIJO, Christian Haagensen. **Textos e Arte, Sítio MIDI**. 2000. Endereço Eletrônico: http://start.at/MIDIga Data da consulta: 15/10/2000.

[MAT2000] MATRIZ, ART**.** 2000. Endereço Eletrônico: http://www.lightlink.com Data da consulta: 28/10/2000.

[MEL1990] MELENDEZ FILHO, Rubem. **Prototipação de sistemas de informações: fundamentos, técnicas e metodologias**. Rio de Janeiro : Livros Técnicos e Científicos, 1990.

[MOO1994] MOORE, Martin L. **SoundBlaster: o livro definitivo.** Rio de Janeiro: Campus, 1994.

[RAT1995] RATTON, Miguel. **Criação de música e sons no computador.** Rio de Janeiro: Campus, 1995.

[SHA1995] SHATZ, Phil. **Modelando sonhos em CD – *Trilhas da imaginação***. Rio de Janeiro : Axcel Books do Brasil, 1995. Páginas 27/29

[SWI2000] SWIFT, SOFTWARE **Multimídia Tools.** 2000. Endereço Eletrônico: http://www.swiftsoft.de Data da consulta: 25/10/2000.

[VAS1995] VASCONCELOS, Laércio. **Introdução à Multimídia.** Rio de Janeiro : LVC, 1995.

[VAU1994] VAUGHAN, Tay. **Multimídia na prática.** São Paulo : Makron Books, 1994.