UNIVERSIDADE FEDERAL DO ABC
GRADUAÇÃO EM ENGENHARIA DA INFORMAÇÃO

EDUARDO DA SILVA CRUZ

PLATAFORMA DE PROGRAMAÇÃO PARA PROCESSAMENTO DE ÁUDIO EM TEMPO REAL

SANTO ANDRÉ - SP
2015

EDUARDO DA SILVA CRUZ

PLATAFORMA DE PROGRAMAÇÃO PARA PROCESSAMENTO DE ÁUDIO EM TEMPO REAL

Trabalho apresentado ao Curso de Graduação Universidade Federal do ABC, como requisito parcial para obtenção do grau em Engenharia da Informação

Orientador: Prof. Dr. Claudio José Bordin Júnior

Santo André – SP
2015

Ficha Catalográfica

Cruz, Eduardo da Silva
Plataforma de Programação Para Processamento de Áudio em Tempo Real
Santo André, SP: UFABC, 2009.

Eduardo da Silva Cruz

PLATAFORMA DE PROGRAMAÇÃO PARA PROCESSAMENTO DE ÁUDIO EM TEMPO REAL

Essa dissertação foi julgada e aprovada para a obtenção do grau em Engenharia da Informação no curso de Graduação em Engenharia da Informação da Universidade Federal do ABC

Santo André, 18 de Agosto de 2015

_____

Prof. Dr. Ricardo Suyama
Coordenador do Curso

BANCA EXAMINADORA

_____
Prof. Dr. Claudio José Bordin Júnior
Orientador

_____
Prof. Dr. Irineu Antunes Júnior
UFABC

_____
Prof. Dr. Marcelo Bender Perotoni
UFABC

A minha mãe Lourdes (in memoriam)
e meu pai Tarciso.

Agradecimentos

Universidade Federal do ABC.
Ao orientador Prof. Doutor Claudio Bordin, pela dedicação e acompanhamento pontual e competente.

Aos professores do Curso de Engenharia da Informação

A todos os que direta ou indiretamente contribuíram para a minha formação.

“Quem inaugura a negação dos homens não são os que tiveram a sua humanidade negada, mas as que a negaram, negando também a sua.”

Paulo Freire

Resumo

Desenvolveu-se um programa para tratamento de áudio em tempo real para a plataforma PC com o sistema operacional Microsoft Windows que permite processar sinais de áudio captados pela entrada de som do PC e aplicar filtros FIR ou algoritmos de redução de ruído baseados em modificação de coeficientes transformados. O programa foi escrito na linguagem C++, utilizando o compilador Visual Studio 2013. A interface gráfica do programa foi desenvolvida utilizando Windows Forms, e a manipulação de áudio em tempo real empregou a API de código aberto PortAudio. Através de uma metodologia de mensuração baseada em realimentação, verificou-se que o programa desenvolvido introduz uma latência da ordem de 60ms sem o uso de qualquer hardware especial.

Abstract

This work describes a real time audio treatment program for the PC platform using the Microsoft Windows operating system. The program processes audio signals captured through a PC's standard audio input, and applies FIR filters or a denoising algorithm based on modifications of coefficients in a transformed domain. The program was written in the C++ language for the Visual Studio 2013 compiler. The program's graphical interface was developed using Windows Forms, and the audio real time handling employed the PortAudio open source API. We verified via a feedback based measurement technology that the developed program's processing latency can be as low as 60ms, without using any specialized hardware.

SUMÁRIO

1. INTRODUÇÃO

O processamento e tratamento de sinais [1] é uma importante área da engenharia e tem como objetivo capturar, manipular, tratar e emitir sinais de origem analógica ou digital. Destacam-se, em especial, os sinais analógicos temporais, ou seja, funções que são continuas e bem definidas quando representas no tempo e que são encontrados comumente em diversas áreas da natureza dentre elas o som.

O som é a propagação de uma frente de compressão mecânica ou onda mecânica [2]; é uma onda longitudinal, que se propaga de forma circuncêntrica, apenas em meios materiais (que têm massa e elasticidade), como os sólidos, líquidos ou gasosos.

Os sons naturais são, na sua maior parte, combinações de sinais, mas um som puro monotônico, representado por uma senoide pura, possui uma taxa de oscilação ou frequência que se mede em hertz (Hz) e uma amplitude que se mede em decibéis. Os sons audíveis pelo ouvido humano têm uma frequência entre 20 Hz e 20.000 Hz. Abaixo e acima desta faixa estão infrassom e ultrassom, respectivamente.

Os seres humanos e vários animais percebem sons com o sentido da audição, com seus dois ouvidos, o que permite estimar a distância e posição da fonte sonora: a chamada audição estereofônica. O áudio é uma representação do som, geralmente se utiliza a representação matemática [2] conforme demonstrado na Figura 1.

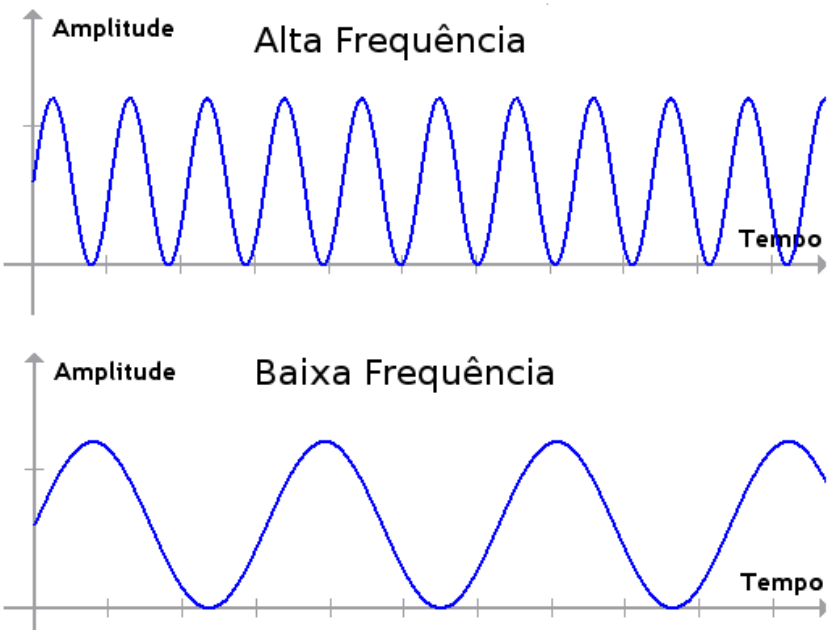

Figura 1 Esquema representando dois sinais sonoros de diferentes frequências

Devido às limitações físicas, estrutura e arquitetura dos equipamentos digitais não é possível representar e tratar um sinal analógico (continuo no tempo) em um dispositivo eletrônico sem que haja perda de informação. Para que um sinal analógico seja processado em meio digital é necessário realizar alguns procedimentos para a captação, processamento, armazenamento e reprodução posterior ou em tempo real. Após a captação do som por um equipamento apropriado como, por exemplo, o microfone, um dos primeiros processos na digitalização é realizar o processo de amostragem do sinal analógico.

A amostragem [1] é o processo de discretização temporal de um sinal contínuo. É importante observar que sinais digitais possuem valores pré-determinados e portanto são discretos (descontínuos) no tempo. Amostragem pode ser definida como o processo de medição instantânea de valores de um sinal analógico em intervalos regulares. O intervalo entre as amostras é determinado por um pulso de sincronismo e a sua frequência é chamada de taxa de amostragem.

A amostragem pode ser feita por diferentes funções no espaço de tempo, ou de qualquer outra dimensão, e são obtidos resultados semelhantes em duas ou mais dimensões. O teorema da amostragem de Nyquist–Shannon [1] estabelece a necessidade de amostragem à taxa igual ou superior a duas vezes a maior frequência do sinal amostrado para que seja possível recuperar o sinal original.

A reconstrução do sinal amostrado no sinal continuo é feita por algoritmos de interpolação. A fórmula de interpolação Whittaker-Shannoné matematicamente equivalente a um ideal Filtro Passa Baixa (FPB) anti-aliasing, cuja entrada é uma sequência de funções delta de Dirac que são modulados (multiplicados) pelos valores de amostra [1]. Quando o intervalo de tempo entre as amostras adjacentes é uma constante (T), a sequência de funções delta é chamada um pente de Dirac [1]. Matematicamente, o modulado pente Dirac é equivalente ao produto da função de pente de Dirac com $s(t)$ [3]. Essa abstração puramente matemática é muitas vezes referida como amostragem por impulso ou amostragem impulsiva. A representação de amostragem do sinal é mostrada na Figura 2.

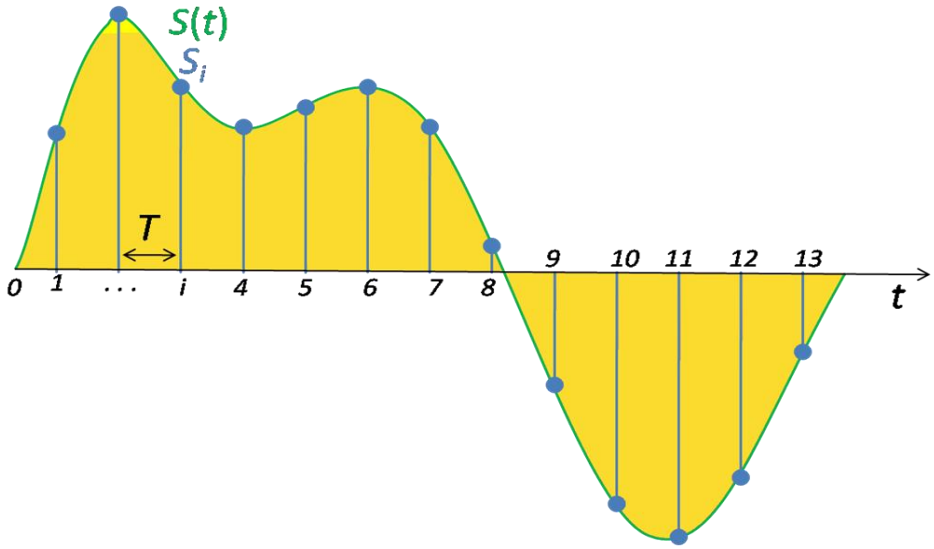

Figura 2 Representação de amostragem do sinal. O sinal contínuo está representada com uma linha de cor verde, enquanto as amostras discretas são indicados pelas linhas verticais azuis.

Sem um filtro *anti-aliasing*, as frequências superiores à frequência de Nyquist influenciariam as amostras de uma forma que processo de interpolação seria prejudicado resultando em um sinal incorreto [1].

Na prática, o sinal contínuo é amostrado usando um conversor analógico-digital (ADC), um dispositivo com várias limitações físicas. Isso resulta em desvios na reconstrução do sinal gerando distorção. Vários tipos de distorções e erros podem ocorrer, tais como: *Aliasing* (sobreposição), Erro de Abertura, *Jitter*, Ruído, *Slew Rate*, Erro de Quantização e outras distorções não lineares [1].

Com a possibilidade de amostragem e captura do sinal e do advento do computador que permitiu a representação e a manipulação rápida de símbolos discretos utilizando a representação discreta (com perda de informação) dos sinais analógicos, deu-se origem à área de processamento de sinais digitais. Muitas das ferramentas do domínio analógico estão presentes no domínio digital com as devidas adaptações e restrições.

Após a amostragem é possível iniciar o processamento do sinal propriamente dito para que o mesmo seja armazenado e reproduzido posteriormente ou então para que seja processado e reproduzido imediatamente (processamento em tempo real) conforme a necessidade [4].

1.1 Processamento em tempo real

O termo *tempo real* [4] se refere à restrição de tempo para aplicações críticas cujo tempo necessário para que o processamento e tratamento necessário, incluindo seu retorno, seja satisfatório e quase imperceptível. Entendemos que o processamento de sinais é viável em tempo real quando existe uma limitação de atraso constante e aceitável entre a entrada e a saída do sistema, permitindo em teoria uma produção de um fluxo de saída ininterrupto. O processamento em tempo real deve garantir resposta dentro de rigorosas restrições de tempo ou prazos. Respostas em tempo real são geralmente da ordem de milissegundos, e às vezes microssegundos dependendo de alguns fatores do cenário em que está sendo executado o processamento.

Para que ocorra o processamento do áudio é necessário contar com uma plataforma física (*hardware*) composta por processador e unidades de entrada e saída e também por um aplicativo (*software*) que seja capaz de coordenar o hardware para que suas entradas e saídas produzam o resultado esperado, dentre esses *software* podemos destacar o sistema operacional. Sistema operacional é um programa ou um conjunto de programas cuja função é gerenciar os recursos do sistema (definir qual programa recebe atenção do processador, gerenciar memória, criar um sistema de arquivos, etc.), fornecendo uma interface entre o computador e o usuário.

Um *sistema em tempo real* é um capaz de controlar um ambiente de recepção de dados, processá-los e devolver os resultados de forma rápida o suficientemente para afetar o ambiente naquele mesmo momento, ou seja, um processamento e retorno sem uma demora perceptível [4]. *Software* em tempo real pode ser usado, por exemplo, em sistemas operacionais de tempo real, redes em tempo real ou sistemas de missão critica, proporcionando estruturas essenciais para a construção de uma aplicação com as características necessárias para o suporte ao tempo real [4].

Um sistema operacional de tempo real [5], como o QNX ou algumas versões especificas do Linux, tem a característica fundamental [4] a sua coerência sobre a quantidade de tempo que leva a aceitar e completar tarefas, a variabilidade de *jitter*, mínimas latência de interrupção e mínima latência troca de *thread*. Esse tipo de sistema operacional é mais valorizado para a rapidez ou previsivelmente de como pode responder do que pela quantidade de trabalho que pode realizar em um determinado período de tempo. Vale ressaltar que sistemas operacionais de tempo real facilitam a concepção de um sistema em tempo real, mas não garante que o resultado final seja um sistema de tempo real, para tal é necessário que o programa nele implementado tenha sido corretamente desenvolvido [4].

Em relação a sistemas em tempo real, se utiliza o termo *plataforma acessível* [4] para plataformas que não tenham sido projetadas exclusivamente para o processamento de áudio, mas que também possam ser utilizadas para este fim levando em consideração o custo, a complexidade, o acesso e a possibilidade de alteração do código fonte, como por exemplo, a plataforma de *hardware* PC, *smartphones* ou *tablets*. Os Sistemas Operacionais predominantes são Microsoft Windows e os sistemas baseados em Unix

(Linux, Android, MacOS X, iOS), e as principais arquiteturas são as baseadas nos processadores x86 e ARM.

Sistemas operacionais como o Microsoft Windows ou as distribuições mais comuns do Linux, que são amplamente utilizados na plataforma de computação pessoal e dispositivos móveis, possuem suporte a processamento multitarefas, mas *não são* sistemas de tempo real [4]. No entanto, é possível *contornar* a ausência nativa ao suporte a sistemas em tempo real, tanto no Linux quanto no Microsoft Windows, através do uso de interfaces de programação (API, do inglês *Application Programming Interface*) adequadas [4]. Vale salientar, porém, que os programas desenvolvidos com essas API's não operam verdadeiramente em tempo real; eles simulam esse comportamento através do uso e do gerenciamento de *buffers* e outros recursos para manutenção de sincronismo providos pelo sistema operacional ou certas interfaces de mais baixo nível, como os *drivers* ASIO [6].

1.2 API's para processamento de áudio

Estão disponíveis na internet diversos conjuntos de Interface de Programação de Aplicativos chamadas de API's (*Application Programming Interface*) [7] para o processamento de áudio em tempo *quase* real. Essas API's se distinguem pela linguagem em que são desenvolvidas e pelos compiladores e sistemas operacionais suportados [8].

Dentre as versões que se pode mencionar, podemos destacar o *PortAudio*. O *PortAudio* [9] é uma biblioteca multiplataforma de fonte aberta (*Open Source*) que fornece API's para a gravação e ou reprodução de som usando dois paradigmas distintos: o de *callback functions* (funções de retorno) e o de funções bloqueantes. O *PortAudio* oferece suporte a diversos sistemas operacionais, incluindo o Windows, Mac OS X e Linux, e faz parte do projeto *PortMusic*, que tem como objetivo fornecer um conjunto de bibliotecas independentes de plataforma para aplicações de música. O código do *PortAudio* está escrito na linguagem de programação C, mas possui interfaces (*wrappers*) nas linguagens C++, suportando os compiladores GNU (gcc, MinGW) e o Visual .NET.

2 OBJETIVO

De uma forma geral o sistema operacional e suas API apenas fornecem métodos de acesso às entradas e saídas gerenciadas pelo sistema, mas não possuem métodos mais avançados para o tratamento e processamento de áudio. No caso do sistema operacional Microsoft Windows caso se tenha interesse em uma aplicação que faça algum tipo de processamento de áudio mais avançado é necessário adquirir algum programa adicional para essa tarefa ou então desenvolver uma nova aplicação (ou seja, um novo programa de computador) que atenda seus objetivos. O objetivo desse trabalho, portanto, é cobrir esta lacuna, desenvolvendo um novo aplicativo para o tratamento de áudio captado em tempo real que não necessite de licença de *software*.

3 DESENVOLVIMENTO

3.1 Algoritmos Implementados

Neste trabalho, implementou-se um código de programação em C/C++ que processa áudio estéreo em tempo real na plataforma Windows seguindo as praticas de desenvolvimento de software em cascata [10], aplicando três algoritmos: i) filtro FIR com *buffer* linear (Seção 3.1.1) e ii) um filtro FIR utilizando *buffers* circulares (Seção 3.1.2). Os coeficientes dos filtros FIR são calculados através do método de janelamento (Seção 3.1.3). Implementou-se também o iii) algoritmo de *denoising* através de modificação de coeficientes transformados (Seção 3.1.4).

3.1.1 Filtro FIR com buffer linear

Em processamento de sinais, o filtro de resposta ao impulso finita (*finite impulse response – FIR*) [1] é um filtro cuja resposta ao impulso, ou resposta a qualquer entrada de comprimento finito é de finita duração. A resposta de impulso, isto é, a saída em

resposta a um delta de Kronecker [11] de entrada de um filtro FIR de tempo discreto de ordem ***N*** dura exatamente ***N + 1*** amostras (a partir de primeiro elemento diferente de zero até o último elemento diferente de zero), antes de acrescentar o zero.

Para um filtro FIR causal de tempo discreto de ordem ***N***, cada valor de saída é a sequência de uma soma ponderada dos valores de entrada:

$$y[n] = \ b_0 x[n] + \ b_1 x[n-1] + \cdots + \ b_N x[n-N] = \ \sum_{i=0}^{N} b_i x[n-i] \quad (1)$$

em que *x[n]* é a entrada do sinal, *y[n]* é a saída do sinal, *N* é a ordem do filtro e b_i é o valor da resposta ao impulso para os instantes de tempo ***i*** entre $0 \leq i \leq$ N (coeficientes do filtro).

O cálculo da soma descrita na Equação (1) é realizado através do *produto interno* de dois vetores armazenados na memória, $B = [b_0\ b_1 ...\ b_N]^T$, o vetor de coeficientes, e ***X[n]*** = [*x[n] x[n-1] ... x[n-N]*]^T. Enquanto o vetor de coeficientes *B* é estático, o vetor ***X[n]*** que armazena o sinal deve ser atualizado a cada amostra recebida.

O esquema de *buffer* linear atualiza *X[n]* da forma trivial, porém computacionalmente complexa: a cada nova amostra *x[n]* recebida, todas os elementos do vetor *X[n]* são deslocados de uma posição a baixo e amostra mais atual, *x[n]*, é copiada na primeira posição do vetor. Isto envolve N-1 operações de cópia de elementos da memória a cada amostra de entrada processada.

3.1.2 Filtros FIR com Buffer Circular

O esquema de *buffer* circular (Figura 3) diminui a complexidade computacional dos filtros FIR eliminando a necessidade de se realizar operações de deslocamento de elementos da memória.

Para este fim, se utiliza de um esquema de endereçamento indireto que induz uma mapeamento circular. Isto é conseguido criando-se um ponteiro (*First*, na Figura 3), que contém o índice do elemento mais antigo (i.e., de menor índice) do *buffer*.

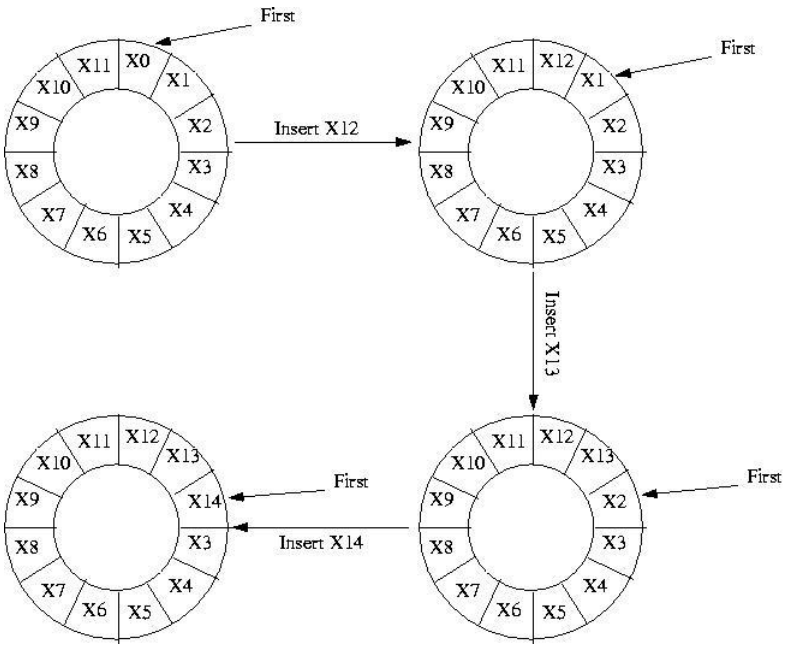

Figura 3: Ilustração do funcionamento de um *buffer* circular (Fonte: https://www.ee.iitb.ac.in/uma/~rajeev81/mathematics/node10.html).

Para se atualizar o *buffer*, a amostra atual, *x[n]*, é então copiada na posição indicada por *First*, e o valor deste índice é incrementado circularmente: acrescenta-se 1; se o valor resultante exceder *N*, subtrai-se *N*, de modo que o índice esteja sempre no intervalo de 0 a *N*.

A desvantagem do método de *buffer* circular é complicar ligeiramente o algoritmo para o cálculo do produto interno, que deve incluir a lógica de atualização circular de índices. Embora pareça trivial, isto tem o efeito de impedir a otimização do código pelo compilador C++, podendo fazer com que o código resultante seja mais lento que o correspondente ao filtro que utiliza arquitetura linear [12].

3.1.3 Cálculo de Coeficientes do através do Método de Janelamento

Foram implementados filtros FIR passa-baixas e passa-altas de ordem variável, cujos coeficientes foram calculados através do método de janelamento [1]. Utilizou-se a

janela de Blackman, que apresenta o menor nível de lóbulo lateral entre as janelas mais usuais [1].

Seja um filtro passa-baixas ideal com fase linear:

$$H_d(e^{jw}) = \begin{cases} e^{-j\omega n_d}, & |\omega| \leq \omega_c \\ 0, & \omega_c < |\omega| \leq \pi \end{cases} \quad (2)$$

Onde:

ω é a frequência

n_d é o atraso

ω_c é a frequência de corte

A correspondente resposta impulsiva ideal é:

$$h_d[n] = \frac{sin\omega_c(n-n_d)}{\pi(n-n_d)} \quad -\infty < n < \infty \quad (3)$$

Nota-se que a resposta impulsiva tem duração infinita e não-causal. Uma solução para isso é truncar a resposta impulsiva, tomando N amostras:

$$h_d[n] = \begin{cases} h_d[n], & 0 \leq \mathrm{n} \leq \mathrm{M} = \mathrm{N} - 1 \\ 0, & caso\ contrário \end{cases} \quad (4)$$

Isso equivale multiplicar a resposta impulsiva ideal $h_d[n]$ por uma janela de duração finita $w[n]$:

$$h[n] = h_d[n].w[n] \quad (5)$$

Onde no caso de um simples truncamento, $w[n]$ é uma janela retangular:

$$w[n] = \begin{cases} 1, & 0 \leq \mathrm{n} \leq \mathrm{M} = \mathrm{N} - 1 \\ 0, & caso\ contrário \end{cases} \quad (6)$$

3.1.4 Algoritmo de Redução de Ruído (*Denoising Audio Signal*)

Implementou-se o algoritmo de redução de ruído descrito em [13], que opera através da modificação de coeficientes transformados.

O algoritmo foi implementado utilizando-se uma transformada de cossenos discretos (DCT - vide Apêndice II) com sobreposição. Assim, a cada $N/2$ amostras recebidas do sinal ruidoso de entrada $x[n]$, é formado um bloco de N amostras $\{x[m]\}_{n\text{-}N/2 \leq m \leq n+N/2-1}$, contendo as $N/2$ amostras mais recentes $\{x[m]\}_{n \leq m \leq n+N/2-1}$ e as $N/2$ amostras anteriores $\{x[m]\}_{n\text{-}N/2 \leq m < n}$ (Figura 4). É calculada então a DCT do bloco de N amostras, dando origem a N coeficientes X_k , $0 \leq k \leq N$. Utilizou-se uma janela (*window*) retangular.

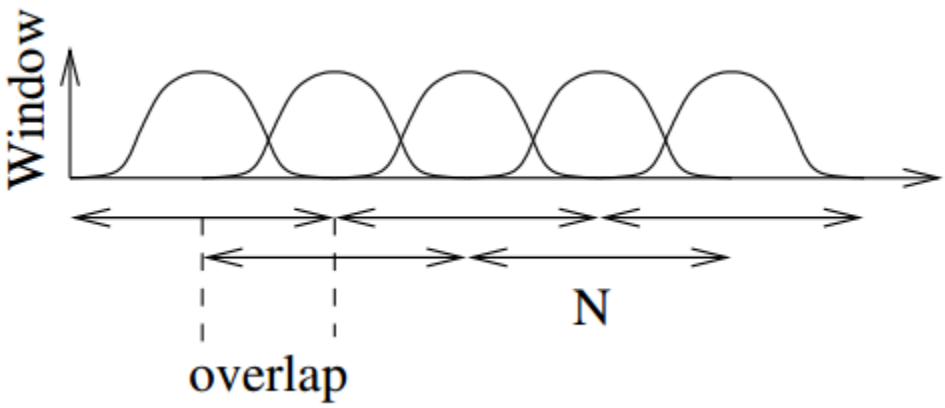

Figura 4: Transformada com sobreposição (overlap). Na imagem a palavra Window pod ser traduzida como Janena. No eixo vertical representasse a amplitude. (Fonte: http://dsp.stackexchange.com/questions/19311/stft-why-overlapping-the-window).

De acordo com o método proposto em [13], os coeficientes transformados X_k são modificados *individualmente* através de funções de limiar, gerando os coeficientes

$Y_k = \rho_{t_1,t_2}(X_k)$. Foram implementadas dois funções de limiar propostas em [13], as funções *HardHard* e *SoftSoft*.

A função *HardHard* é definida, para o argumento *X*, como:

$$P_{t_1,t_2}^{(HH)}(X) = \begin{cases} X, & |X| > t_2 \\ 0, & t_1 \leq |X| \leq t_2 \\ X, & |X| < t_1 \end{cases} \quad (7)$$

em que t_1 é o valor do limiar inferior e t_2 o do limiar superior.

A função *SoftSoft*, por sua vez, é definida

$$P_{t_1,t_2}^{(SS)}(X) = \begin{cases} sign(X)(|X| - \Delta t), & |X| > t_2 \\ sign(X)t_1, & t_1 \leq |X| \leq t_2 \\ X, & |X| < t_1 \end{cases} \quad (8)$$

em que $\Delta t = t_2 - t_1$ e *sign(.)* denota a função sinal[1].

.

Diferentemente de [13], por simplicidade, os valores dos limiares foram ajustados *independentemente* em cada bloco, da seguinte forma: a partir dos parâmetros t_1' e t_2', contidos no intervalo [*0 1*], definidos pelo usuário, calculou-se $t_i = t_i' \, max\{|X_k|\}$, para *i=1,2*, ou seja, os limiares são determinados em proporção ao coeficiente de maior magnitude no domínio transformado.

Cada bloco de N amostras Y_k é então anti-transformado, gerando o bloco *y[l]*, $0 \leq l \leq N$. O sinal processado é então calculado através da sobreposição:

$$\tilde{x}[n - N + l] = \frac{1}{2}(y[l] + y_{ANT}[l + N/2]). \quad (9)$$

em que $0 \leq l < N/2$ e $y_{ANT}[l]$ denota a antitransformada do bloco anterior.

[1] A função sign(x) de um número real X é definida como:

$$sign(x) := \begin{cases} -1, & se\ x < 0 \\ 0, & se\ x = 0 \\ 1, & se\ x > 0 \end{cases}$$

4 Medidas de Desempenho

Analisaram-se algumas plataformas disponíveis no mercado levando em conta não só o baixo custo de aquisição e facilidade de obtenção (em comparação com soluções profissionais ou especializadas), como também outros aspectos relevantes, dentre eles a licença de utilização e quão atual ou obsoleta era a plataforma.

Após a analise optou-se por utilizar a plataforma PC, pois, apesar de não ser nativamente de tempo real, ela é de fácil acesso, possui alta disponibilidade no mercado, é de fácil manipulação e programação e tem capacidade de processamento elevada comparada a outras plataformas como smartphones ou microcontroladores [14].

Para os testes e simulações iniciais utilizaram-se dois computadores com as seguintes configurações de hardware:

- Processador: Intel i5-2410M / Intel i7-4770K
- Memória RAM: 4Gb / 8 Gb
- Dispositivo de áudio: Realtek High Definition Audio
- Dispositivo de microfone comum com conector P2
- Duas caixas de som para computador com conector P2

O software utilizado foi

- Sistema Operacional Windows 8 pro x64 / Windows 10 home x64
- Microsoft Visual Studio Express 2013
- API de áudio *PortAudio* [9]
- Bibliotecas Kiss FFT [15]
- Audacity [16]

4.1 Mensuração do atraso

Um dos principais fatores que comprometem uma aplicação de tempo real é o atraso entre o áudio gerado pela saída do dispositivo (que nos testes foram as caixas de som do computador) e a captação do áudio na entrada (que nos testes foram um microfone convencional para PC). Esse atraso é gerado principalmente pelos processos de conversão do sinal analógico para digital, processamento do sinal de áudio já em formato digital com a aplicação do filtro e parâmetros escolhidos e por fim a conversão do sinal digital para analógico. Devido ao fato desses processos dependerem tanto do *hardware* como do *software* isso faz com que dependendo da situação obtenham-se diferentes desempenhos, gerando atrasos maiores ou menores. Por isso todos os testes foram realizados em um mesmo equipamento. Para medir o atraso de forma efetiva usou-se um comportamento similar ao efeito de reverberação. A reverberação é um efeito físico gerado pelo som, sendo a reflexão múltipla de uma frequência resultando na persistência de um som depois de ter sido extinta sua emissão por uma fonte. Esse efeito pode ser gerado por reflexões construtivas do som em um ambiente. Para os testes realizados colocamos o computador em um ambiente silencioso, conectou-se e instalou-se o microfone o mais próximo possível a frente da saída do autofalante conforme a Figura 5.

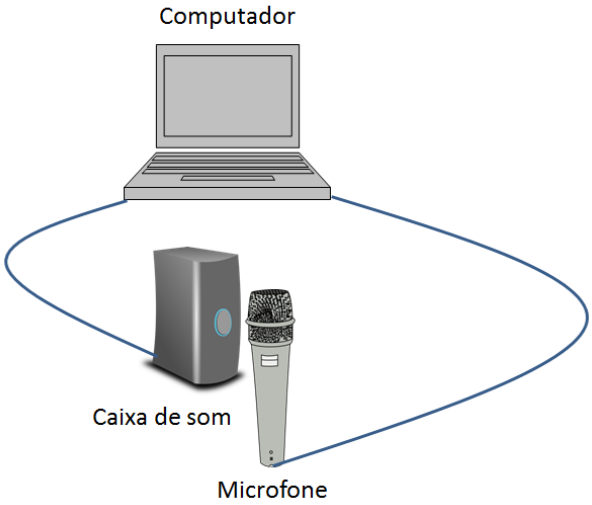

Figura 5 – Esquema físico do ambiente de testes

Dessa forma criou-se uma realimentação positiva do som capitado e um sistema instável que reproduzira de forma repetida com algum intervalo de tempo qualquer áudio capitado inicialmente pelo microfone. Simulou-se então uma função Delta de Dirac ou função impulso com um som de estalo mais alto e mais breve possível e analisou-se o intervalo que esse som se repetia com software Audacity [6] a gravação da saída de áudio. Interrompeu-se a gravação antes que não fosse possível mais verificar quando o áudio se repetia devido a amplificação e distorção gerado pelo sistema.

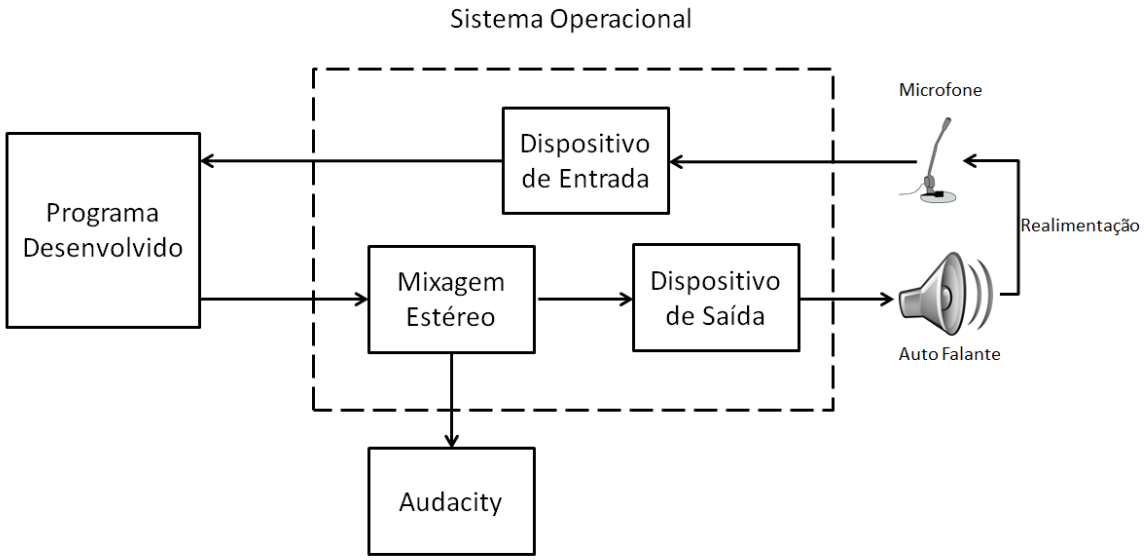

Figura 6 – Modelo simplificado do fluxo do sinal de áudio pelo ambiente e a realimentação. A latência total é estimada como o atraso entre o sinal original e a sua cópia atrasada produzida pela realimentação autofalante-microfone

O atraso é estimado graficamente, analisando-se as formas de onda gravadas pelo audacity.

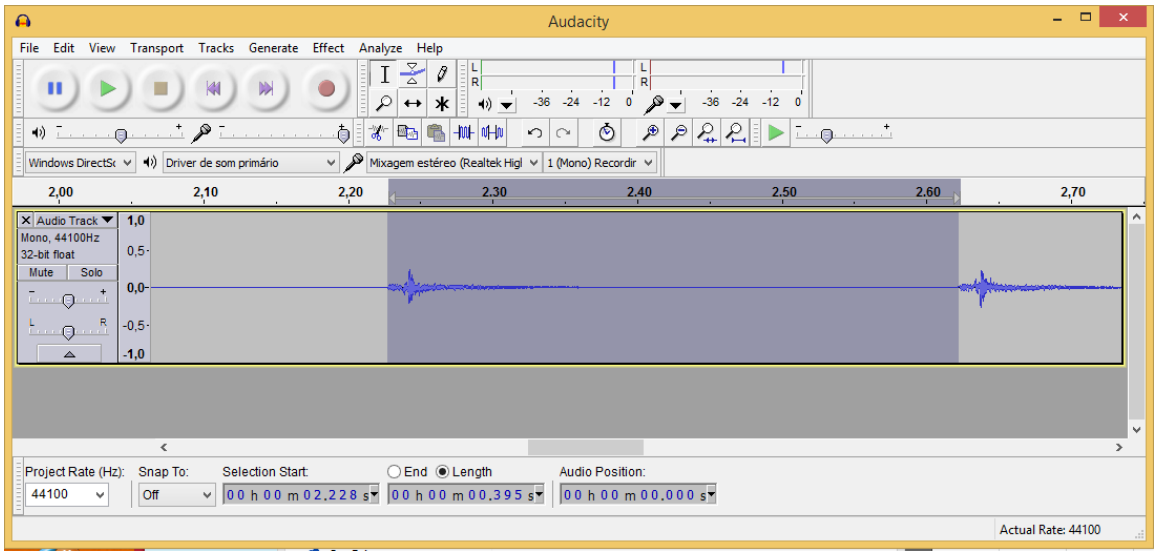

Figura 7 - Medição do atraso com o sistema de realimentação microfone e caixa de som

Todos os tempos mensurados foram divididos por dois para achar a media simples do atraso gerado pela realimentação

4.2 Mensuração do processamento e memória

Durante o início do desenvolvimento da aplicação foram realizados testes para verificar o consumo de processamento e memória para aplicação conforme o Apêndice II.

Para a análise final de desempenho de processamento e consumo de memória analisou-se a aplicação desenvolvida e compilada sendo analisados somente três cenários distintos somente pela opção *Enable Enhanced Instruction Set*, sendo esses definidos em: *No set*, *Streaming SIMD Extensions 2(/arch:SSE2)* e *Advanced Vector Extensions* (/arch:AVX) e mantendo as demais configurações conforme a Figura 8 e a Figura 9.

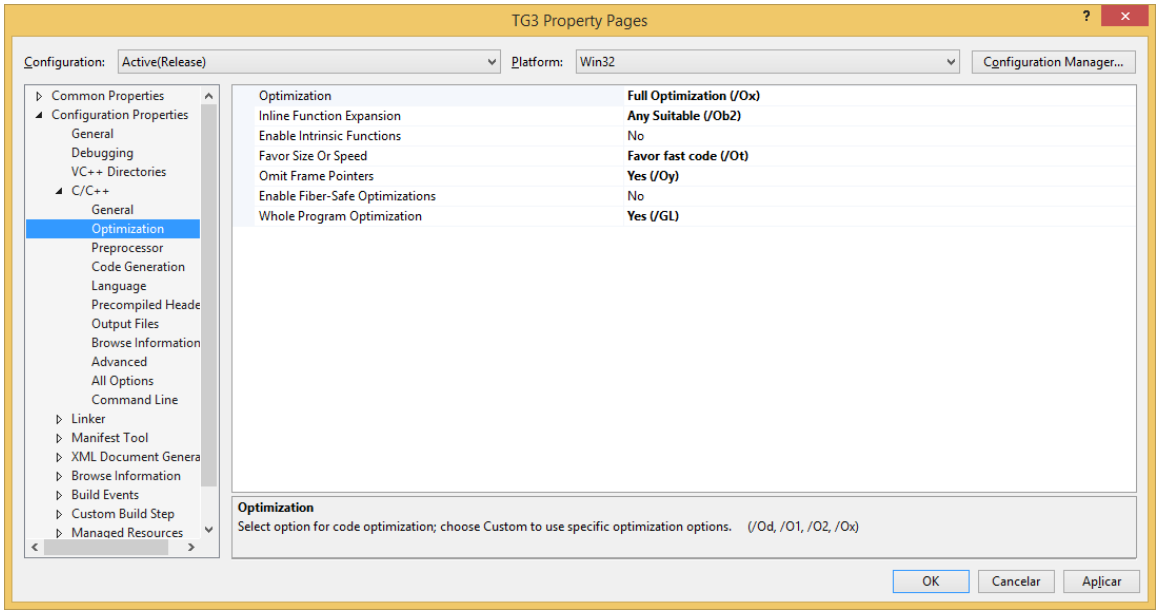

Figura 8 – Parâmetros de configuração do Visual Studio

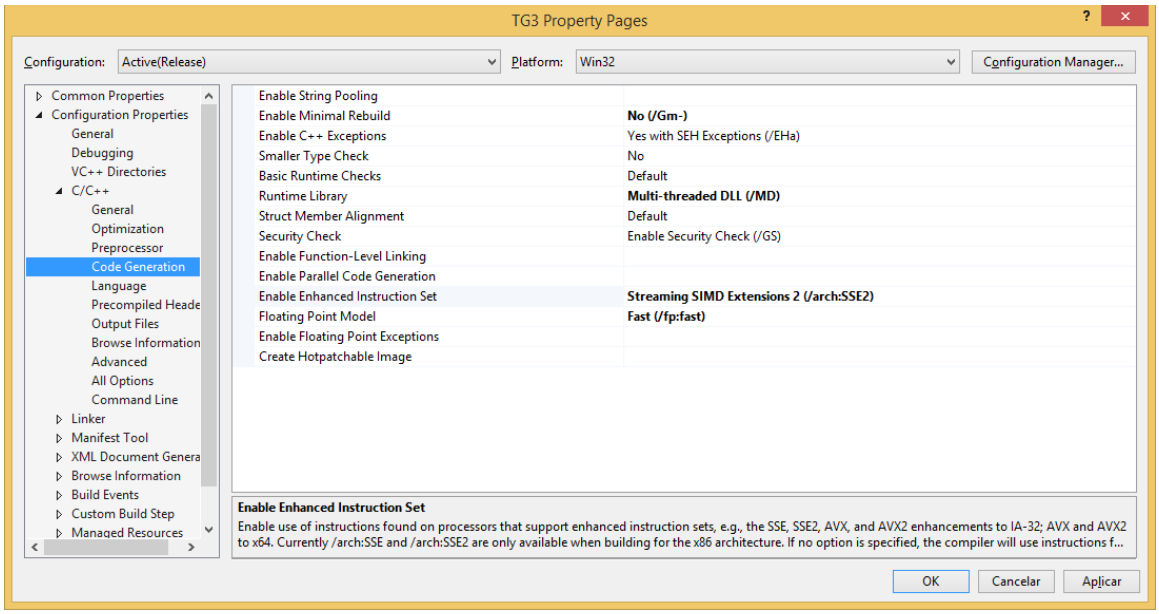

Figura 9 – Parâmetros de configuração do Visual Studio

4.3 Implementação da aplicação

A aplicação foi desenvolvida em linguagem C/C++ no Microsoft Visual Studio, o código criado está no Apêndice III.

Foram acrescentadas ao projeto criado Microsoft Visual Studio as bibliotecas do *PortAudio* versão na versão estável v19 (arquivo pa_stable_v19_20140130.tgz) [9] com o objetivo de utilizar suas APIs de leitura e gravação de áudio e a biblioteca de código aberto *Kiss FFT* [15] que possibilitou o cálculo das DCT's necessárias para o algoritmo de *denoising*.

Para permitir que o algoritmo de tratamento de áudio fosse controlado a partir da interface gráfica que foi desenvolvida em *Windows Forms* utilizando a linguagem C++/Cli, foi necessário utilizar um esquema de *multithreading* [17] cooperativo através de objetos da classe *BackgroundWorker*.

5 RESULTADOS

5.1 Interface gráfica

Com o Microsoft Visual Studio elaborou-se a seguinte interface gráfica da Figura 10:

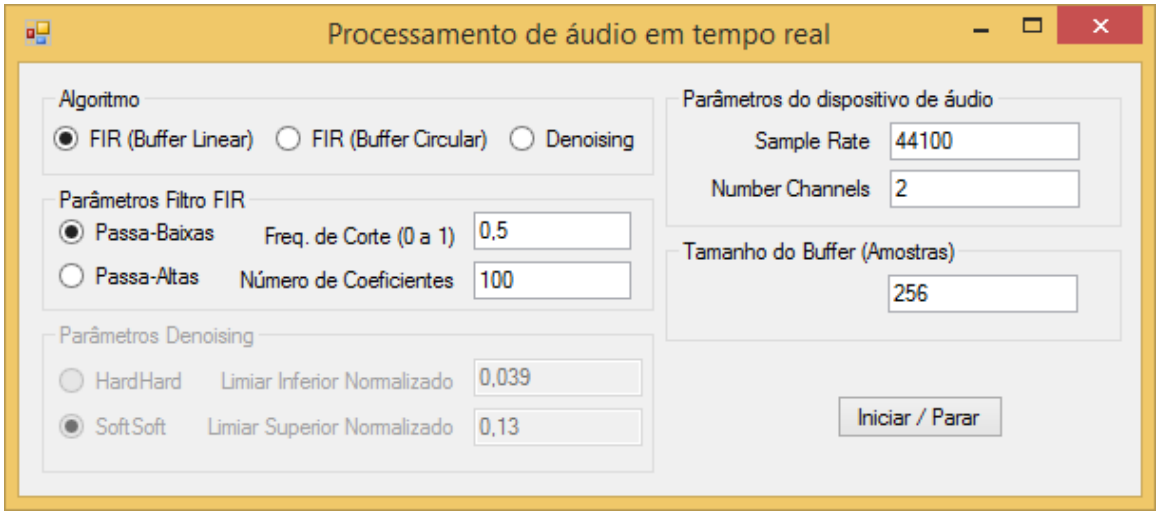

Figura 10: Interface gráfica desenvolvida para a aplicação

Através dessa interface é possível iniciar e parar a captura do áudio pela aplicação com o botão “Iniciar / Parar”. Se a captação não estiver em funcionamento ao pressionar esse botão se dará início a captura do áudio pela entrada do microfone com os parâmetros que estiverem definidos nas demais opções da aplicação. Caso a captação de áudio esteja em funcionamento e o botão “Iniciar / Parar” seja pressionado, a captação será interrompida.

É possível escolher qual será o efeito aplicado selecionando uma das três opções de algoritmo: FIR (*Buffer* Linear), FIR (*Buffer* Circular) e *Denoising*.

Caso seja escolhido o algoritmo FIR (*Buffer* Linear) ou FIR (*Buffer* Circular), e ainda é possível escolher se o filtro será Passa-Baixas ou Passa-Altas e seus parâmetros de frequência de corte entre 0 a 1 e o numero de coeficientes entre 3 e o máximo (depende da capacidade do hardware em utilização)

Caso seja selecionado o algoritmo *Denoising* é possível escolher entre *HardHard* e *SoftSoft* com os parâmetros de limite inferior e superior.

Além disso, é possível escolher o tamanho do buffer e os parâmetros do dispositivo de áudio como numero de canais (*Number Channels*) e a taxa amostragem de sinal (*Sample Rate*)

5.2 Mensuração do atraso

Em seguida utilizou-se a versão do programa com interface gráfica, utilização de threads e os parâmetros de configuração especificados nas figuras 9 e 10 e utilizando o filtro FIR (*Buffer* Linear) com as configurações conforme a Figura 11 apenas alterando os valores de tamanho de *Buffer*:

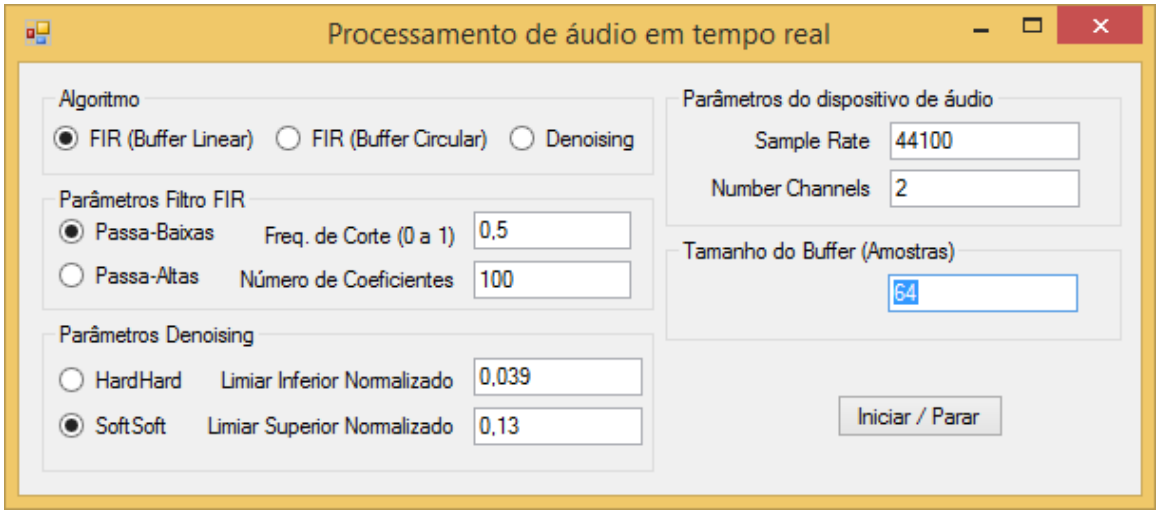

Figura 11

Mensurou-se o atraso e anotou-se a media simples na Tabela 1:

Tabela 1: Latência total de propagação do sinal, da aquisição à reprodução, utilizando o programa compilado para diversos conjuntos de instruções e utilizando diferentes valores tamanhos de *buffer*. O filtro FIR implementado produz um atraso de 49,5 amostras (1,1ms à taxa de amostragem de 44.100Hz). Os valores em fontes normais se referem a testes num computador com processador Intel i5-2410M enquanto os em negrito com processador Core i7-4770K.

	Atraso em ms		
Tamanho do Buffer	x86	sse2	avx
64	53/**44**	57/**52**	52/**49**
256	58/**51**	59/**48**	60/**48**
1024	86/**77**	83/**75**	91/**78**

Estimou-se também a capacidade máxima de processamento do programa, avaliada como o número máximo de coeficientes do filtro FIR que não causa perda de amostras, detectadas através do aparecimento de "cliques" audíveis no sinal de saída, em função do conjunto de instruções utilizado e do tamanho do *buffer* com as mesmas configurações e anotaram-se os valores na tabela.

Tabela 2: Número máximo de coeficientes do filtro FIR com *buffer* linear. Os valores em fontes normais se referem a testes num computador com processador Intel i5-2410M enquanto os em negrito com processador Intel Core i7-4770K.

Tamanho do Buffer	x86	sse2	avx
64	15000/**16000**	15000/**16000**	15000/**16000**
256	15000/**16000**	15000/**16000**	15000/**16000**
1024	15000/**16000**	15000/**16000**	15000/**16000**

Como via de regra, observou-se que os "cliques" passam a ocorrer quando o programa utiliza 100% do *tempo de processamento* de um dos núcleos do processador. Vale ressaltar que a utilização tempo de CPU pelo programa *não* é uma função linear da ordem do filtro: verificamos experimentalmente que a utilização é muito pequena até um determinado valor de ordem, crescendo rapidamente a partir deste. Creditamos este efeito ao esgotamento da memória cache do processador: quando a memória utilizada pelo filtro excede o tamanho do cache L2 do processador, a CPU tende a passar a maior parte parada aguardando a leitura de dados de outras regiões da memória, fenômeno que é conhecido como *cache misses* [18].

Como se pode notar observando os dados da Tabela 5, os *cache misses* são o principal fator de limitação de capacidade do algoritmo empregando *buffer* linear. O conjunto de instruções utilizado, bem como o tamanho do *buffer* do PortAudio, não fizeram diferença.

Na Tabela 3, repetiu-se a mesma análise, utilizando a implementação do filtro FIR com buffer circular.

Tabela 3: Número máximo de coeficientes do filtro FIR com buffer circular. Os valores em fontes normais se referem a testes num computador com processador Intel i5-2410M enquanto os em negrito com processador Core i7-4770K

Tamanho do Buffer	x86	sse2	avx
64	1900/**2000**	1900/**2000**	3800/**4000**
256	1900/**2000**	1900/**2000**	5200/**5500**
1024	5500/**6000**	2800/**3000**	5500/**6000**

Como se pode observar naTabela 3, o desempenho da implementação com *buffer* circular é pior, pois o numero máximo de coefiecientes é menor. Neste caso, o esgotamento da capacidade lógica do processador é o principal fator de limitação de desempenho, uma vez que o uso de instruções que processam ou não múltiplos argumentos produziu desempenhos distintos. Em particular, o conjunto de instruções AVX [19], que permite processar múltiplas operações aritméticas em paralelo, levou a um desempenho consistentemente melhor. Além disso, ao se aumentar o tamanho do *buffer* do PortAudio, o desempenho tende a melhorar. Isto ocorre porque o uso de *buffers* maiores torna o algoritmo de tratamento de áudio mais eficiente, uma vez que o número de chamadas de funções, que é proporcional ao *número* de *buffers* tratados, diminui. O efeito colateral de se aumentar o tamanho dos *buffers*, *porém*, é o aumento da latência, como visto anteriormente.

6 CONCLUSÕES

O objetivo de desenvolver um novo aplicativo para o tratamento de áudio captado em tempo real que não necessite de licença de *software* se mostrou viável.

Observou-se nas seçoes de desenvolvimento e medidas de desempenho que, utilizando a API *Portaudio* e um PC de configuração relativamente modesta, é possível executar em tempo real um filtro FIR com alguns milhares de coeficientes operando à taxa de 44.100 Hz com dois canais incorrendo em gasto de tempo processamento e de memória relativamente baixos Curiosamente, o uso de estruturas baseadas em *buffers* circulares não gerou ganhos de desempenho, muito provavelmente devido à maior complexidade do código, que dificultou a sua otimização pelo compilador.

Os atrasos observados na Tabela 1 ao se utilizar a API WASAPI são viáveis para uma aplicação de tempo real, desde que não sejam utilizados *buffers* de tamanho superior a 1.000 amostras. Para esse caso, verificou-se subjetivamente que os atrasos causados pelo conjunto formado pela aplicação desenvolvida, sistema operacional e hardware são pouco perceptíveis. Vale ressaltar que conforme visto nas Tabelas 2 e 3 há um compromisso na escolha do tamanho dos *buffers*: se estes forem muitos pequenos, aumenta-se a probabilidade de ocorrerem estouros (*overruns*) ou esvaziamentos (*underruns*) de *buffers* devido a atrasos aleatórios produzidos pelo sistema operacional. Em ambos os casos, a perda de sincronismo resultante se faz sentir através da ocorrência de "cliques" audíveis.

Como sugestão de aprimoramento desse trabalho sugere-se desenvolver essa aplicação para outras plataformas como *Linux* ou *Smartphones*.

7 BIBLIOGRAFIA

[1] A. V. Oppenheim e . W. Schafer, Discrete-Time Signal Processing, Pearson Education Ltd., 2013.

[2] R. C. Boulanger e V. Lazzarini, The Audio Programming Book, MIT Press, 2011.

[3] Wikipédia, "Pente de Dirac," Wikipédia A enciclopédia livre, 25 08 2015. [Online]. Available: https://pt.wikipedia.org/wiki/Pente_de_Dirac. [Acesso em 25 08 2015].

[4] R. Shyamasundar, Real Time Programming: Languages, Specification and Verification, World Scientific, 2010.

[5] Wikipedia, "List of real-time operating systems," [Online]. Available: http://en.wikipedia.org/wiki/List_of_real-time_operating_systems. [Acesso em 01 09 2014].

[6] Audacity Team, "ASIO Audio Interface," [Online]. Available: http://wiki.audacityteam.org/wiki/ASIO_Audio_Interface. [Acesso em 01 09 2014].

[7] Microsoft Corp., "Core Audio APIs," [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/dd370802(v=vs.85).aspx. [Acesso em 01 09 2014].

[8] W. a. e. livre, "Linguagem de programação," Wikipédia, [Online]. Available: https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o. [Acesso em 07 08 2015].

[9] PortAudio community, [Online]. Available: http://www.portaudio.com/. [Acesso em 01 09 2014].

[10] Wikipedia, "Processo de desenvolvimento de software," Wikipedia, [Online]. Available: https://pt.wikipedia.org/wiki/Processo_de_desenvolvimento_de_software. [Acesso em 07 08 2015].

[11] "Kronecker delta," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Kronecker_delta. [Acesso em 30 04 2015].

[12] A. Alexandrescu, Modern C++ Design, Addison-Wesley, 2001.

[13] I. A. JR, "REDUÇÃO DE RUÍDO EM SINAIS DE VOZ USANDO CURVAS ESPECIALIZADAS DE MODIFICAÇÃO DOS COEFICIENTES DA TRANSFORMADA EM CO-SENO," Escola Politécnica da Universidade de São Pualo, São Paulo, 2006.

[14] A. J. Bianchi, Processamento de áudio em tempo real em plataformas computacionais de alta disponibilidade e baixo custo., São Paulo: Dissertação de Mestrado. Instituto de Matemática e Estatística da Universidade de São Paulo, 2013.

[15] Source Forge, "Kiss FFT," [Online]. Available: http://sourceforge.net/projects/kissfft/. [Acesso em 07 08 2015].

[16] "Audacity Software," Audacity Team, [Online]. Available:

http://web.audacityteam.org/. [Acesso em 30 04 2015].

[17] Wikipédia a enciclopédia livre, “Thread (ciência da computação),” [Online]. Available: https://pt.wikipedia.org/wiki/Thread_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o). [Acesso em 07 08 2015].

[18] Wikipédia, “CPU Cache,” Wikipédia - A enciclopédia livre, [Online]. Available: https://pt.wikipedia.org/wiki/CPU_cache. [Acesso em 25 08 2015].

[19] Wikipedia, “Advanced Vector Extensions,” Wikipedia, the free encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Advanced_Vector_Extensions. [Acesso em 25 08 2015].

[20] “About WASAPI (Windows),” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/dd371455(v=vs.85).aspx. [Acesso em 30 04 2015].

APÊNDICE

Apêndice I

Testes de latência preliminares para escolha das API's de áudio

Durante o desenvolvimento do aplicativo na linguagem C/C++ foram realizados vários testes para verificação de atraso entre a entrada do som no microfone e a saída nos dispositivos de auto falante para mensurar quais as técnicas, procedimentos de compilação e API's que proporcionavam melhores no melhores desempenhos de latência, processamento e memória utilizados pelo aplicativo. Inicialmente na versão do software iniciou-se os testes com os seguintes parâmetros fixos, sem utilizar threads e sem interface gráfica (funcionando apenas em *prompt* de comando) anotou-se a média simples dos resultados obtidos Tabela 4 e com esses dados gerou-se o gráfico da Figura 12

Tabela 4 - Tempo obtido para a execução da aplicação com um filtro FIR de 1000 coeficientes em função do comprimento do *buffer*

	Tempo de atraso (em milissegundos)	
Frames por *buffer*	Utilizando WASAPI	Utilizando WINMME
4	45	226
100	54	238
500	62	261
1000	93	247
1500	111	257
10000	498	377

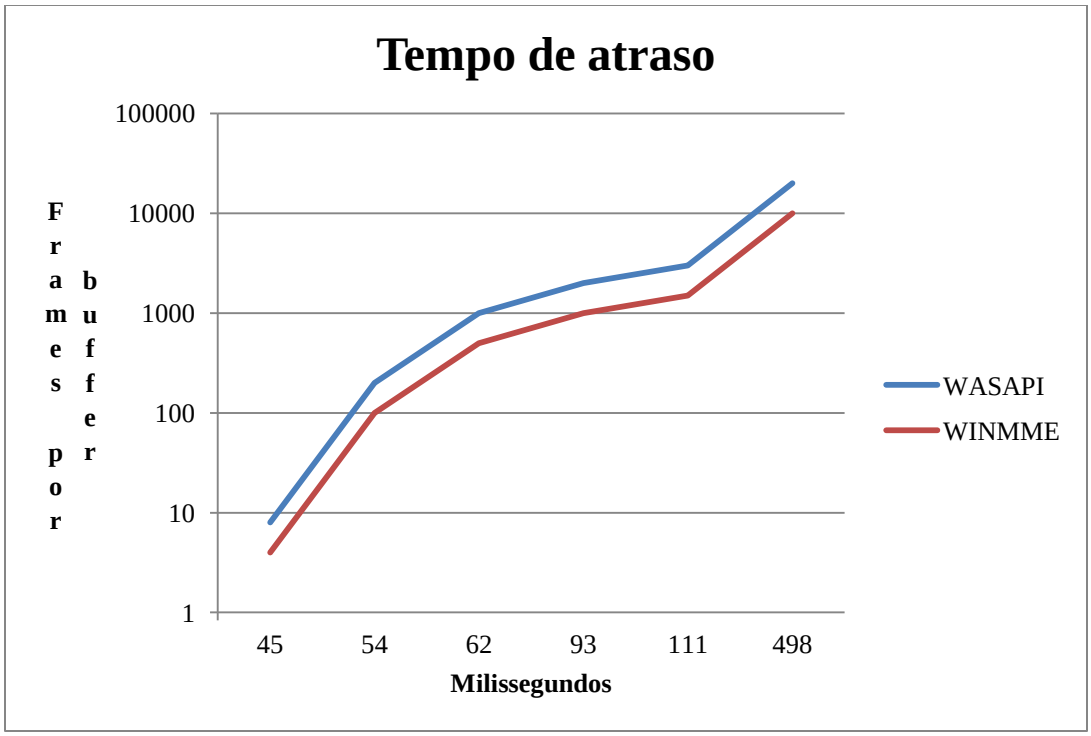

Figura 12 - Tempo obtido para a execução da aplicação com um filtro FIR de 1000 coeficientes em função do comprimento do *buffer*

Com esses resultados observou-se que a API WASAPI permitiu utilizar um maior numero de frames por *buffer* com um atraso menor.

Em seguida executou-se a aplicação com diferentes opções de otimização e desempenho.

Tabela 5 – Consumo médio percentual de processador para a execução da aplicação com um filtro FIR de 1000 coeficientes

	Consumo do processador (%)	
	FIR com *buffer* linear	FIR com *buffer* circular
Sem otimização e sem uso de instruções avançadas	5	7
Sem otimização e com instruções avançadas	4	5
Com otimização e sem instruções avançadas	3	5
Com otimização e instruções avançadas	0	4

Tabela 6 - Consumo médio de memória RAM para a execução da aplicação com um filtro FIR de 1000 coeficientes

	Consumo de memória RAM (Kb)	
	FIR com *buffer* linear	FIR com *buffer* circular
Sem otimização sem uso de instruções avançadas	952	920
Sem otimização e com instruções avançadas	904	896
Com otimização e sem instruções avançadas	900	900
Com otimização e instruções avançadas	1000	896

Testou-se também o desempenho da aplicação com a interface de programação de aplicações nativa do Microsoft Windows a WASAPI [20] e WINMME que permitem que aplicativos criados para a plataforma gerenciem o fluxo de dados de áudio entre o aplicativo e um dispositivo áudio.

Para a realização dos testes de atraso e latência causados pelo conjunto sistema operacional, hardware e aplicação desenvolvida foi implementado na aplicação um filtro FIR com 1000 coeficientes. Para a análise de desempenho de processamento e consumo de memória inicial analisou-se com o gerenciador de tarefas do Windows o processo referente a aplicação desenvolvida e compilada com diferentes configurações de otimização. Foram testadas as seguintes configurações:

- Sem otimização

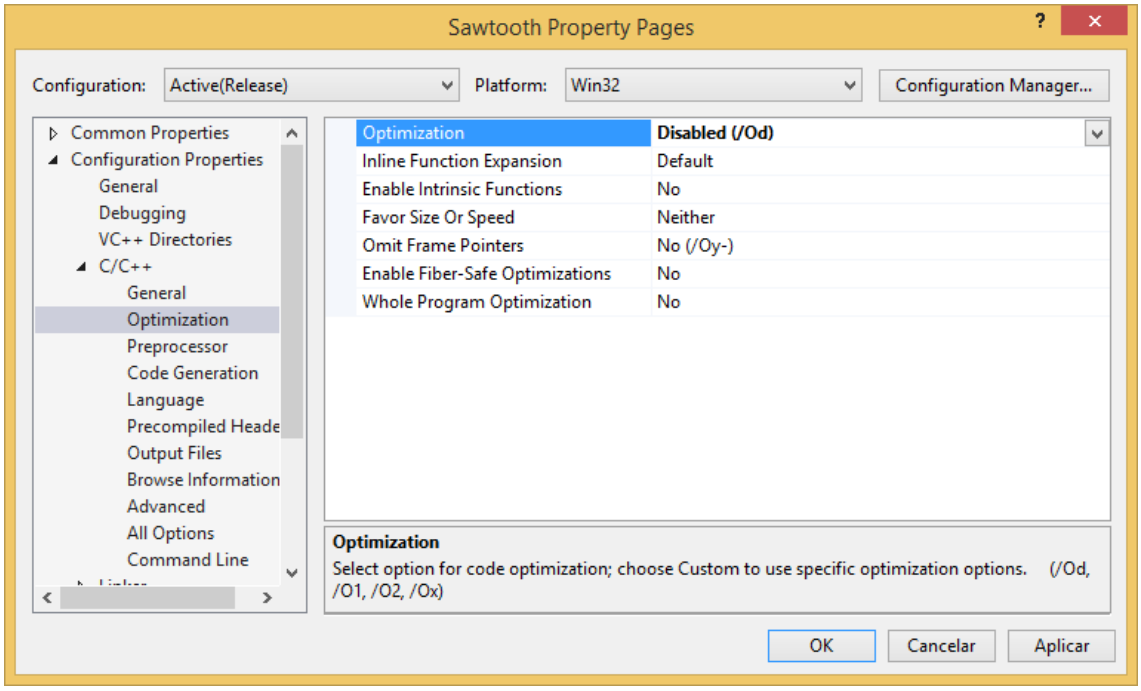

Figura 13 – Parâmetros de configuração do Visual Studio

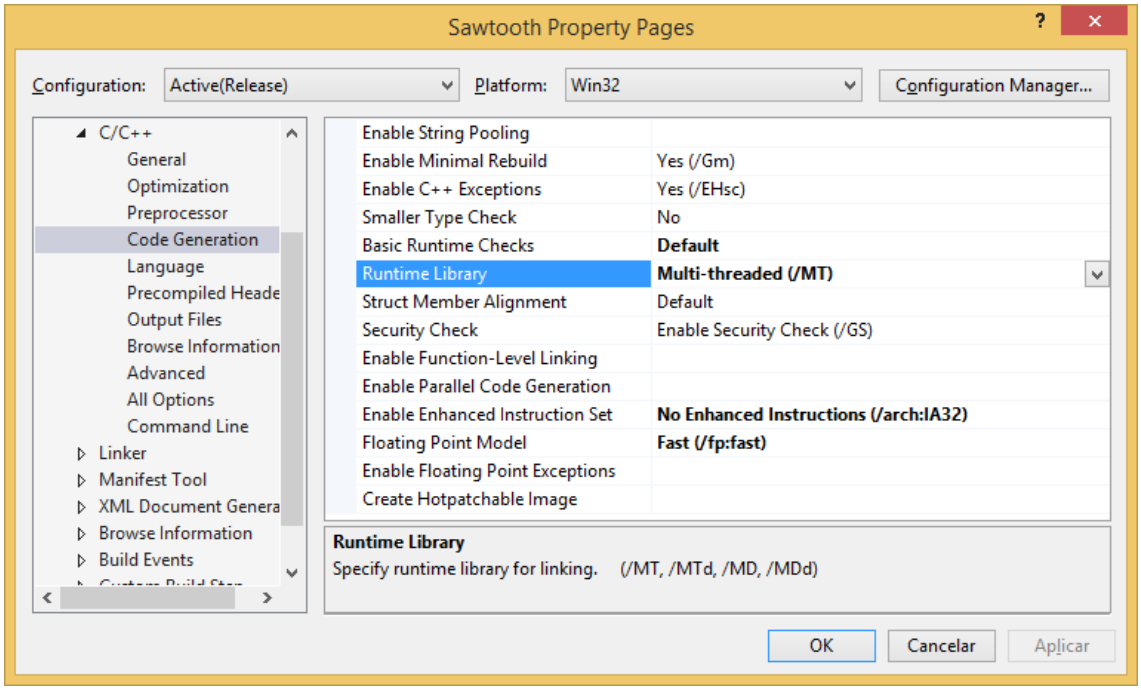

Figura 14 –– Parâmetros de configuração do Visual Studio

- Com otimização

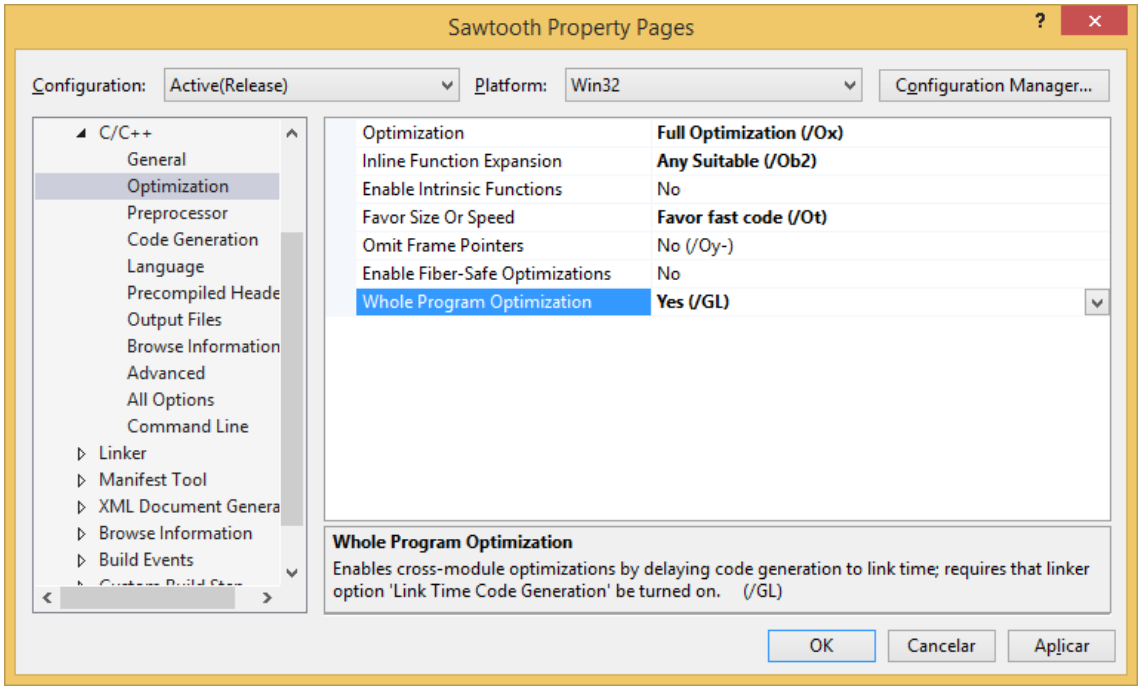

Figura 15 – Parâmetros de configuração do Visual Studio

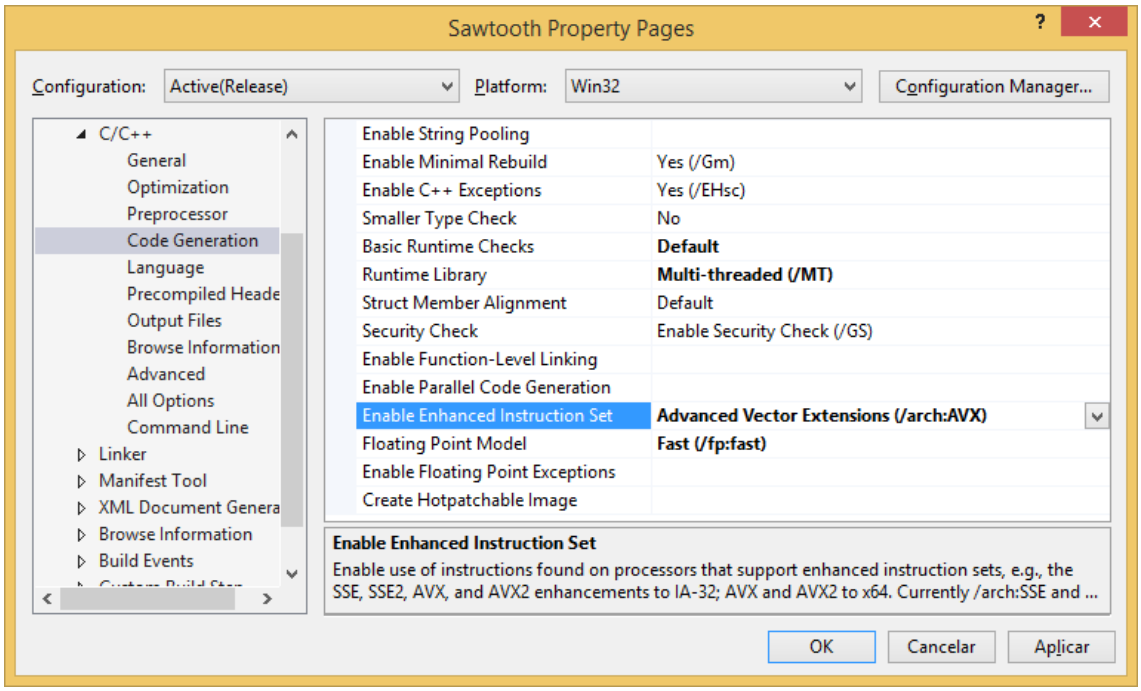

Figura 16 – Parâmetros de configuração do Visual Studio

- E a combinação entre ambas

Após esses testes iniciais concluiu-se que os parâmetros que ofereciam menor tempo de latência com consumo de processamento e memória baixos foram:

- API WASAPI
- Full optimization /OX
- Favor fast code /Ot
- Multi-treaded /Md

E com base nesses parâmetros (que são os mesmos listados nas Figura 8 e Figura 9) prosseguimos com os ajustes finos descritos na metodologia e resultados

Apêndice II

A transformada de cossenos discreta

A transformada de cosseno discreta (DCT) expressa uma sequência finita de pontos de dados em termos de uma soma de funções cosseno em diferentes frequências de oscilação. As DCT's são importantes para inúmeras aplicações em ciência e engenharia, como por exemplo na compressão com perdas de áudio em arquivos MP3 ou em imagens JPEG onde pequenas componentes de alta frequência podem ser descartadas. O uso de funções cosseno ao invés de funções seno é crítico para a compressão, pois menos funções cosseno são necessárias para aproximar um sinal típico, considerando que para equações diferenciais os cossenos expressam uma escolha particular de condições de contorno.

Formalmente, a transformada de cosseno discreto é uma função linear, que pode ser invertida $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ Onde $\mathbb{R}$ representa o conjunto dos números reais.

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)\left(k + \frac{1}{2}\right)\right] \qquad k = 0, \ \ldots, \ N-1$$

No programa que desenvolvemos, a transformada de cosseno discreto foi calculada utilizando-se o algoritmo baseado na FFT (*Fast Fourier Transform*) implementado pelo MATLAB nas funções *dct.m* e *idct.m*. O código em C++ está listado no Apêndice III, no arquivo *fir.cpp*.

Apêndice III

Código desenvolvido

Arquivo: MyForm.h

```
#include <stdbool.h>
#include "portaudio.h"
#include "pa_types.h"
#include <math.h>
#include <vector>
#include "fir.h"

#define PA_SAMPLE_TYPE  paInt24
#define SAMPLE_SIZE (3)

#pragma once

namespace TG3 {
	using namespace std;
	using namespace System;
	using namespace System::Windows::Forms;

	/// <summary>
	/// Summary for MyForm
	/// </summary>
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:

		int ORD = 100;
		float freq_corte;
		int SAMPLE_RATE;
		int FRAMES_PER_BUFFER;
		int NUM_CHANNELS;
		int which_algorithm = 3;
		bool is_low_pass = true;
		bool is_hard = false;
		float lower_thr;
		float upper_thr;

		MyForm(void)
		{
			InitializeComponent();
		}

	protected:
		~MyForm()
		{
			if (components)
			{
				delete components;
			}
		}

	private: System::Windows::Forms::Button^  button1;
```

```
        private: System::ComponentModel::BackgroundWorker^  backgroundWorker1;
        private: System::Windows::Forms::TextBox^  txtSampleRate;
        private: System::Windows::Forms::Label^  label1;
        private: System::Windows::Forms::Label^  label3;
        private: System::Windows::Forms::TextBox^  txtChannels;
        private: System::Windows::Forms::GroupBox^  groupBoxParametros;
        private: System::Windows::Forms::RadioButton^  radioButton1;
        private: System::Windows::Forms::RadioButton^  radioButton2;
        private: System::Windows::Forms::RadioButton^  radioButton3;
        private: System::Windows::Forms::GroupBox^  groupBoxFiltro;
        private: System::Windows::Forms::GroupBox^  groupBoxFIR;
        private: System::Windows::Forms::TextBox^  textInferior;
        private: System::Windows::Forms::Label^  label5;
        private: System::Windows::Forms::Label^  label6;
        private: System::Windows::Forms::TextBox^  textSuperior;
        private: System::Windows::Forms::GroupBox^  groupBoxDenoise;
        private: System::Windows::Forms::RadioButton^  radioButton5;
        private: System::Windows::Forms::RadioButton^  radioButton4;
        private: System::Windows::Forms::Label^  label7;
        private: System::Windows::Forms::Label^  label4;
        private: System::Windows::Forms::TextBox^  textBox2;
        private: System::Windows::Forms::TextBox^  textBox1;
        private: System::Windows::Forms::RadioButton^  radioButton7;
        private: System::Windows::Forms::RadioButton^  radioButton6;
        private: System::Windows::Forms::GroupBox^  groupBox1;
        private: System::Windows::Forms::TextBox^  txtBuffer;

        protected:

        private:
                /// <summary>
                /// Required designer variable.
                /// </summary>
                System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                void InitializeComponent(void)
                {
                        this->button1 = (gcnew System::Windows::Forms::Button());
                        this->backgroundWorker1 = (gcnew
System::ComponentModel::BackgroundWorker());
                        this->txtSampleRate = (gcnew System::Windows::Forms::TextBox());
                        this->label1 = (gcnew System::Windows::Forms::Label());
                        this->label3 = (gcnew System::Windows::Forms::Label());
                        this->txtChannels = (gcnew System::Windows::Forms::TextBox());
                        this->groupBoxParametros = (gcnew System::Windows::Forms::GroupBox());
                        this->radioButton1 = (gcnew System::Windows::Forms::RadioButton());
                        this->radioButton2 = (gcnew System::Windows::Forms::RadioButton());
                        this->radioButton3 = (gcnew System::Windows::Forms::RadioButton());
                        this->groupBoxFiltro = (gcnew System::Windows::Forms::GroupBox());
                        this->groupBoxFIR = (gcnew System::Windows::Forms::GroupBox());
                        this->label7 = (gcnew System::Windows::Forms::Label());
                        this->label4 = (gcnew System::Windows::Forms::Label());
                        this->textBox2 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox1 = (gcnew System::Windows::Forms::TextBox());
```

```
                this->radioButton7 = (gcnew System::Windows::Forms::RadioButton());
                this->radioButton6 = (gcnew System::Windows::Forms::RadioButton());
                this->textInferior = (gcnew System::Windows::Forms::TextBox());
                this->label5 = (gcnew System::Windows::Forms::Label());
                this->label6 = (gcnew System::Windows::Forms::Label());
                this->textSuperior = (gcnew System::Windows::Forms::TextBox());
                this->groupBoxDenoise = (gcnew System::Windows::Forms::GroupBox());
                this->radioButton5 = (gcnew System::Windows::Forms::RadioButton());
                this->radioButton4 = (gcnew System::Windows::Forms::RadioButton());
                this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
                this->txtBuffer = (gcnew System::Windows::Forms::TextBox());
                this->groupBoxParametros->SuspendLayout();
                this->groupBoxFiltro->SuspendLayout();
                this->groupBoxFIR->SuspendLayout();
                this->groupBoxDenoise->SuspendLayout();
                this->groupBox1->SuspendLayout();
                this->SuspendLayout();
                //
                // button1
                //
                this->button1->Location = System::Drawing::Point(430, 174);
                this->button1->Name = L"button1";
                this->button1->Size = System::Drawing::Size(88, 23);
                this->button1->TabIndex = 0;
                this->button1->Text = L"Iniciar / Parar";
                this->button1->UseVisualStyleBackColor = true;
                this->button1->Click += gcnew System::EventHandler(this,
&MyForm::Click_Iniciar);
                //
                // backgroundWorker1
                //
                this->backgroundWorker1->WorkerReportsProgress = true;
                this->backgroundWorker1->WorkerSupportsCancellation = true;
                this->backgroundWorker1->DoWork += gcnew
System::ComponentModel::DoWorkEventHandler(this, &MyForm::backgroundWorker1_DoWork);
                this->backgroundWorker1->RunWorkerCompleted += gcnew
System::ComponentModel::RunWorkerCompletedEventHandler(this,
&MyForm::backgroundWorker1_RunWorkerCompleted);
                //
                // txtSampleRate
                //
                this->txtSampleRate->Location = System::Drawing::Point(118, 19);
                this->txtSampleRate->Name = L"txtSampleRate";
                this->txtSampleRate->Size = System::Drawing::Size(100, 20);
                this->txtSampleRate->TabIndex = 1;
                this->txtSampleRate->Text = L"44100";
                //
                // label1
                //
                this->label1->AutoSize = true;
                this->label1->Location = System::Drawing::Point(44, 22);
                this->label1->Name = L"label1";
                this->label1->Size = System::Drawing::Size(68, 13);
                this->label1->TabIndex = 2;
                this->label1->Text = L"Sample Rate";
                //
                // label3
                //
                this->label3->AutoSize = true;
```

```
                this->label3->Location = System::Drawing::Point(21, 47);
                this->label3->Name = L"label3";
                this->label3->Size = System::Drawing::Size(91, 13);
                this->label3->TabIndex = 6;
                this->label3->Text = L"Number Channels";
                //
                // txtChannels
                //
                this->txtChannels->Location = System::Drawing::Point(118, 44);
                this->txtChannels->Name = L"txtChannels";
                this->txtChannels->Size = System::Drawing::Size(100, 20);
                this->txtChannels->TabIndex = 5;
                this->txtChannels->Text = L"2";
                //
                // groupBoxParametros
                //
                this->groupBoxParametros->Controls->Add(this->label1);
                this->groupBoxParametros->Controls->Add(this->label3);
                this->groupBoxParametros->Controls->Add(this->txtSampleRate);
                this->groupBoxParametros->Controls->Add(this->txtChannels);
                this->groupBoxParametros->Location = System::Drawing::Point(340, 12);
                this->groupBoxParametros->Name = L"groupBoxParametros";
                this->groupBoxParametros->Size = System::Drawing::Size(240, 74);
                this->groupBoxParametros->TabIndex = 7;
                this->groupBoxParametros->TabStop = false;
                this->groupBoxParametros->Text = L"Parâmetros do dispositivo de áudio";
                //
                // radioButton1
                //
                this->radioButton1->AutoSize = true;
                this->radioButton1->Location = System::Drawing::Point(6, 19);
                this->radioButton1->Name = L"radioButton1";
                this->radioButton1->Size = System::Drawing::Size(111, 17);
                this->radioButton1->TabIndex = 8;
                this->radioButton1->Text = L"FIR (Buffer Linear)";
                this->radioButton1->UseVisualStyleBackColor = true;
                this->radioButton1->CheckedChanged += gcnew System::EventHandler(this,
&MyForm::radioButton1_CheckedChanged);
                //
                // radioButton2
                //
                this->radioButton2->AutoSize = true;
                this->radioButton2->Location = System::Drawing::Point(123, 19);
                this->radioButton2->Name = L"radioButton2";
                this->radioButton2->Size = System::Drawing::Size(117, 17);
                this->radioButton2->TabIndex = 9;
                this->radioButton2->Text = L"FIR (Buffer Circular)";
                this->radioButton2->UseVisualStyleBackColor = true;
                this->radioButton2->CheckedChanged += gcnew System::EventHandler(this,
&MyForm::radioButton2_CheckedChanged);
                //
                // radioButton3
                //
                this->radioButton3->AutoSize = true;
                this->radioButton3->Checked = true;
                this->radioButton3->Location = System::Drawing::Point(246, 19);
                this->radioButton3->Name = L"radioButton3";
                this->radioButton3->Size = System::Drawing::Size(72, 17);
                this->radioButton3->TabIndex = 10;
```

```
                this->radioButton3->TabStop = true;
                this->radioButton3->Text = L"Denoising";
                this->radioButton3->UseVisualStyleBackColor = true;
                this->radioButton3->CheckedChanged += gcnew System::EventHandler(this,
&MyForm::radioButton3_CheckedChanged);
                //
                // groupBoxFiltro
                //
                this->groupBoxFiltro->Controls->Add(this->radioButton1);
                this->groupBoxFiltro->Controls->Add(this->radioButton3);
                this->groupBoxFiltro->Controls->Add(this->radioButton2);
                this->groupBoxFiltro->Location = System::Drawing::Point(12, 12);
                this->groupBoxFiltro->Name = L"groupBoxFiltro";
                this->groupBoxFiltro->Size = System::Drawing::Size(322, 47);
                this->groupBoxFiltro->TabIndex = 11;
                this->groupBoxFiltro->TabStop = false;
                this->groupBoxFiltro->Text = L"Algoritmo";
                //
                // groupBoxFIR
                //
                this->groupBoxFIR->Controls->Add(this->label7);
                this->groupBoxFIR->Controls->Add(this->label4);
                this->groupBoxFIR->Controls->Add(this->textBox2);
                this->groupBoxFIR->Controls->Add(this->textBox1);
                this->groupBoxFIR->Controls->Add(this->radioButton7);
                this->groupBoxFIR->Controls->Add(this->radioButton6);
                this->groupBoxFIR->Enabled = false;
                this->groupBoxFIR->Location = System::Drawing::Point(12, 65);
                this->groupBoxFIR->Name = L"groupBoxFIR";
                this->groupBoxFIR->Size = System::Drawing::Size(322, 65);
                this->groupBoxFIR->TabIndex = 13;
                this->groupBoxFIR->TabStop = false;
                this->groupBoxFIR->Text = L"Parâmetros Filtro FIR";
                //
                // label7
                //
                this->label7->AutoSize = true;
                this->label7->Location = System::Drawing::Point(101, 42);
                this->label7->Name = L"label7";
                this->label7->Size = System::Drawing::Size(120, 13);
                this->label7->TabIndex = 19;
                this->label7->Text = L"Número de Coeficientes";
                //
                // label4
                //
                this->label4->AutoSize = true;
                this->label4->Location = System::Drawing::Point(114, 19);
                this->label4->Name = L"label4";
                this->label4->Size = System::Drawing::Size(107, 13);
                this->label4->TabIndex = 18;
                this->label4->Text = L"Freq. de Corte (0 a 1)";
                //
                // textBox2
                //
                this->textBox2->Location = System::Drawing::Point(227, 39);
                this->textBox2->Name = L"textBox2";
                this->textBox2->Size = System::Drawing::Size(83, 20);
                this->textBox2->TabIndex = 17;
                this->textBox2->Text = L"100";
```

```
                    //
                    // textBox1
                    //
                    this->textBox1->Location = System::Drawing::Point(227, 13);
                    this->textBox1->Name = L"textBox1";
                    this->textBox1->Size = System::Drawing::Size(83, 20);
                    this->textBox1->TabIndex = 16;
                    this->textBox1->Text = L"0,5";
                    //
                    // radioButton7
                    //
                    this->radioButton7->AutoSize = true;
                    this->radioButton7->Location = System::Drawing::Point(9, 38);
                    this->radioButton7->Name = L"radioButton7";
                    this->radioButton7->Size = System::Drawing::Size(80, 17);
                    this->radioButton7->TabIndex = 15;
                    this->radioButton7->Text = L"Passa-Altas";
                    this->radioButton7->UseVisualStyleBackColor = true;
                    this->radioButton7->CheckedChanged += gcnew System::EventHandler(this,
&MyForm::radioButton7_CheckedChanged);
                    //
                    // radioButton6
                    //
                    this->radioButton6->AutoSize = true;
                    this->radioButton6->Checked = true;
                    this->radioButton6->Location = System::Drawing::Point(9, 15);
                    this->radioButton6->Name = L"radioButton6";
                    this->radioButton6->Size = System::Drawing::Size(88, 17);
                    this->radioButton6->TabIndex = 14;
                    this->radioButton6->TabStop = true;
                    this->radioButton6->Text = L"Passa-Baixas";
                    this->radioButton6->UseVisualStyleBackColor = true;
                    this->radioButton6->CheckedChanged += gcnew System::EventHandler(this,
&MyForm::radioButton6_CheckedChanged);
                    //
                    // textInferior
                    //
                    this->textInferior->Location = System::Drawing::Point(227, 19);
                    this->textInferior->Name = L"textInferior";
                    this->textInferior->Size = System::Drawing::Size(89, 20);
                    this->textInferior->TabIndex = 14;
                    this->textInferior->Text = L"0,039";
                    //
                    // label5
                    //
                    this->label5->AutoSize = true;
                    this->label5->Location = System::Drawing::Point(91, 24);
                    this->label5->Name = L"label5";
                    this->label5->Size = System::Drawing::Size(130, 13);
                    this->label5->TabIndex = 15;
                    this->label5->Text = L"Limiar Inferior Normalizado";
                    //
                    // label6
                    //
                    this->label6->AutoSize = true;
                    this->label6->Location = System::Drawing::Point(84, 48);
                    this->label6->Name = L"label6";
                    this->label6->Size = System::Drawing::Size(137, 13);
                    this->label6->TabIndex = 17;
```

```
                this->label6->Text = L"Limiar Superior Normalizado";
                //
                // textSuperior
                //
                this->textSuperior->Location = System::Drawing::Point(227, 45);
                this->textSuperior->Name = L"textSuperior";
                this->textSuperior->Size = System::Drawing::Size(89, 20);
                this->textSuperior->TabIndex = 16;
                this->textSuperior->Text = L"0,13";
                //
                // groupBoxDenoise
                //
                this->groupBoxDenoise->Controls->Add(this->radioButton5);
                this->groupBoxDenoise->Controls->Add(this->radioButton4);
                this->groupBoxDenoise->Controls->Add(this->label5);
                this->groupBoxDenoise->Controls->Add(this->label6);
                this->groupBoxDenoise->Controls->Add(this->textInferior);
                this->groupBoxDenoise->Controls->Add(this->textSuperior);
                this->groupBoxDenoise->Location = System::Drawing::Point(12, 136);
                this->groupBoxDenoise->Name = L"groupBoxDenoise";
                this->groupBoxDenoise->Size = System::Drawing::Size(322, 79);
                this->groupBoxDenoise->TabIndex = 18;
                this->groupBoxDenoise->TabStop = false;
                this->groupBoxDenoise->Text = L"Parâmetros Denoising";
                //
                // radioButton5
                //
                this->radioButton5->AutoSize = true;
                this->radioButton5->Checked = true;
                this->radioButton5->Location = System::Drawing::Point(9, 46);
                this->radioButton5->Name = L"radioButton5";
                this->radioButton5->Size = System::Drawing::Size(63, 17);
                this->radioButton5->TabIndex = 19;
                this->radioButton5->TabStop = true;
                this->radioButton5->Text = L"SoftSoft";
                this->radioButton5->UseVisualStyleBackColor = true;
                this->radioButton5->CheckedChanged += gcnew System::EventHandler(this,
&MyForm::radioButton5_CheckedChanged);
                //
                // radioButton4
                //
                this->radioButton4->AutoSize = true;
                this->radioButton4->Location = System::Drawing::Point(9, 22);
                this->radioButton4->Name = L"radioButton4";
                this->radioButton4->Size = System::Drawing::Size(71, 17);
                this->radioButton4->TabIndex = 18;
                this->radioButton4->Text = L"HardHard";
                this->radioButton4->UseVisualStyleBackColor = true;
                this->radioButton4->CheckedChanged += gcnew System::EventHandler(this,
&MyForm::radioButton4_CheckedChanged);
                //
                // groupBox1
                //
                this->groupBox1->Controls->Add(this->txtBuffer);
                this->groupBox1->Location = System::Drawing::Point(340, 92);
                this->groupBox1->Name = L"groupBox1";
                this->groupBox1->Size = System::Drawing::Size(240, 54);
                this->groupBox1->TabIndex = 21;
                this->groupBox1->TabStop = false;
```

```
                this->groupBox1->Text = L"Tamanho do Buffer (Amostras)";
                //
                // txtBuffer
                //
                this->txtBuffer->Location = System::Drawing::Point(117, 19);
                this->txtBuffer->Name = L"txtBuffer";
                this->txtBuffer->Size = System::Drawing::Size(100, 20);
                this->txtBuffer->TabIndex = 21;
                this->txtBuffer->Text = L"256";
                //
                // MyForm
                //
                this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                this->ClientSize = System::Drawing::Size(590, 227);
                this->Controls->Add(this->groupBox1);
                this->Controls->Add(this->groupBoxDenoise);
                this->Controls->Add(this->groupBoxFIR);
                this->Controls->Add(this->groupBoxFiltro);
                this->Controls->Add(this->groupBoxParametros);
                this->Controls->Add(this->button1);
                this->Name = L"MyForm";
                this->Text = L"Processamento de áudio em tempo real";
                this->groupBoxParametros->ResumeLayout(false);
                this->groupBoxParametros->PerformLayout();
                this->groupBoxFiltro->ResumeLayout(false);
                this->groupBoxFiltro->PerformLayout();
                this->groupBoxFIR->ResumeLayout(false);
                this->groupBoxFIR->PerformLayout();
                this->groupBoxDenoise->ResumeLayout(false);
                this->groupBoxDenoise->PerformLayout();
                this->groupBox1->ResumeLayout(false);
                this->groupBox1->PerformLayout();
                this->ResumeLayout(false);

        }
#pragma endregion

    private: System::Void Click_Iniciar(System::Object^, System::EventArgs^) {

        bool ready_to_start = true;
        bool frames_per_buffer_is_even = true;
        bool valid_order = true;
        bool valid_thr = true;
        bool valid_freq = true;

        if (which_algorithm == 3)
        {
            try
            {
                lower_thr = (float)Double::Parse(textInferior->Text);
                if (lower_thr<0.0 || lower_thr>1.0)
                {
                    ready_to_start = false;
                    valid_thr = false;
                }
            }
            catch (Exception^)
            {
```

```
                ready_to_start = false;
        }

        try
        {
                upper_thr = (float)Double::Parse(textSuperior->Text);
                if (upper_thr<0.0 || upper_thr>1.0)
                {
                        ready_to_start = false;
                        valid_thr = false;
                }
        }
        catch (Exception^)
        {
                ready_to_start = false;
        }
}
else{
        try
        {
                freq_corte = (float)Double::Parse(textBox1->Text);
                if (freq_corte < 0.0 || freq_corte > 1.0)
                {
                        valid_freq = false;
                        ready_to_start = false;
                }
        }
        catch (Exception^)
        {
                ready_to_start = false;
        }

        try
        {
                ORD = Int32::Parse(textBox2->Text);
                if (ORD < 1)
                {
                        valid_order = false;
                        ready_to_start = false;
                }
        }
        catch (Exception^)
        {
                ready_to_start = false;
        }

}

try
{
        SAMPLE_RATE = Int32::Parse(txtSampleRate->Text);
}
catch (Exception^)
{
        ready_to_start = false;
}

try{
        FRAMES_PER_BUFFER = Int32::Parse(txtBuffer->Text);
```

```
                        if (FRAMES_PER_BUFFER % 2 == 1)// Se for ímpar
                        {
                                ready_to_start = false;
                                frames_per_buffer_is_even = false;
                        }
                }
                catch (Exception^)
                {
                        ready_to_start = false;
                }

                try{ NUM_CHANNELS = Int32::Parse(txtChannels->Text); }
                catch (Exception^)
                {
                        ready_to_start = false;
                }

                if (this->backgroundWorker1->IsBusy)
                        backgroundWorker1->CancelAsync();
                else
                        if (ready_to_start)
                        {
                                groupBoxFiltro->Enabled = false;
                                groupBoxDenoise->Enabled = false;
                                groupBoxFIR->Enabled = false;
                                groupBoxParametros->Enabled = false;
                                groupBox1->Enabled = false;
                                backgroundWorker1->RunWorkerAsync(1);
                        }
                        else
                        {
                                if (frames_per_buffer_is_even == true && valid_order == true &&
valid_thr == true && valid_freq == true)
                                        MessageBox::Show("Parâmetros Inválidos. Para Windows em
Inglês, use ponto como separador decimal; caso contrário, use vírgula.");
                                else if (frames_per_buffer_is_even == false)
                                        MessageBox::Show("O tamanho do buffer deve ser par.");
                                else if (valid_order == false)
                                        MessageBox::Show("O número de coeficientes do filtro deve ser
maior ou igual a 1.");
                                else if (valid_thr == false)
                                        MessageBox::Show("Os limiares devem estar entre 0 e 1.");
                                else if (valid_freq == false)
                                        MessageBox::Show("A frequência de corte do filtro deve estar
entre 0 e 1.");

                        }
        }


        private: System::Void backgroundWorker1_DoWork(System::Object^,
System::ComponentModel::DoWorkEventArgs^ e)
        {
                PaStreamParameters inputParameters, outputParameters;
                PaStream *stream = NULL;
                PaError err;

                int BLOCK = 2 * FRAMES_PER_BUFFER;// Não pode ser mudado. Sorry.
```

```
                vector<unsigned char>sampleBlock(FRAMES_PER_BUFFER * NUM_CHANNELS *
SAMPLE_SIZE, 0);
                vector<float> ch1(FRAMES_PER_BUFFER);
                vector<float> ch2(FRAMES_PER_BUFFER);
                vector<float> ch1out(FRAMES_PER_BUFFER);
                vector<float> ch2out(FRAMES_PER_BUFFER);

                vector<float> coef;
                vector<float> mem1(ORD);
                vector<float> mem2(ORD);
                int buffer_offset;

                vector<float> e1(BLOCK);
                vector<float> e2(BLOCK);
                vector<float> s1_past(BLOCK);
                vector<float> s2_past(BLOCK);
                vector<float> s1(BLOCK);
                vector<float> s2(BLOCK);// Blocos de saída (pós DCT)

                // Inicialização do Port Audio
                err = Pa_Initialize();
                if (err != paNoError) goto error;

                try{
                        inputParameters.device = Pa_GetDefaultInputDevice(); /* default input device */
                        inputParameters.channelCount = NUM_CHANNELS;
                        inputParameters.sampleFormat = PA_SAMPLE_TYPE;
                        inputParameters.suggestedLatency = Pa_GetDeviceInfo(inputParameters.device)-
>defaultHighInputLatency;
                        inputParameters.hostApiSpecificStreamInfo = NULL;

                        outputParameters.device = Pa_GetDefaultOutputDevice(); /* default output device */
                        outputParameters.channelCount = NUM_CHANNELS;
                        outputParameters.sampleFormat = PA_SAMPLE_TYPE;
                        outputParameters.suggestedLatency = Pa_GetDeviceInfo(outputParameters.device)-
>defaultHighOutputLatency;
                        outputParameters.hostApiSpecificStreamInfo = NULL;
                }
                catch (NullReferenceException^) { goto error; }

                /* -- setup -- */
                err = Pa_OpenStream(
                        &stream,
                        &inputParameters,
                        &outputParameters,
                        SAMPLE_RATE,
                        FRAMES_PER_BUFFER,
                        //paClipOff,
                        (PaStreamFlags)0x00000000, // paClipOn
                        NULL, /* no callback, use blocking API */
                        NULL); /* no callback, so no callback userData */
                if (err != paNoError) goto error;

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


        //=================================================================
                // Inicialização dos Algoritmos
```

```
                if (which_algorithm == 1 || which_algorithm == 2)
                {
                        coef = projeto_FIR(freq_corte,ORD,is_low_pass);
                        filtro_FIR(ch1, ch2, ch1out, ch2out, coef, mem1, mem2, true,
FRAMES_PER_BUFFER, ORD);
                        buffer_offset = filtro_FIRC(ch1, ch2, ch1out, ch2out, coef, mem1, mem2,
buffer_offset, true, FRAMES_PER_BUFFER, ORD);
                }
                else
                {
                        denoise(ch1, ch2, ch1out, ch2out, e1, e2, s1, s2, s1_past, s2_past, lower_thr,
upper_thr, is_hard, true, FRAMES_PER_BUFFER);
                }


        //=====================================================================
                // Laço Principal
                while (true)
                {
                        if (backgroundWorker1->CancellationPending) break;

                        err = Pa_ReadStream(stream, &sampleBlock[0], FRAMES_PER_BUFFER);
                        if (err) goto xrun;


        Int24tofloat(sampleBlock,ch1,ch2,FRAMES_PER_BUFFER,SAMPLE_SIZE*NUM_CHANNELS);

                        switch (which_algorithm)
                        {
                        case 1:
                                // FILTRO FIR

        filtro_FIR(ch1,ch2,ch1out,ch2out,coef,mem1,mem2,false,FRAMES_PER_BUFFER,ORD);
                                break;
                        case 2:
                                buffer_offset =
filtro_FIRC(ch1,ch2,ch1out,ch2out,coef,mem1,mem2,buffer_offset,false,FRAMES_PER_BUFFER,ORD);
                                break;
                        case 3:
                                // Denoise

        denoise(ch1,ch2,ch1out,ch2out,e1,e2,s1,s2,s1_past,s2_past,lower_thr,upper_thr,is_hard,false,FRAMES
_PER_BUFFER);
                        }

                        // SAIDA DO SINAL
                        floattoInt24(sampleBlock, ch1out, ch2out, FRAMES_PER_BUFFER,
SAMPLE_SIZE*NUM_CHANNELS);

                        err = Pa_WriteStream(stream, &sampleBlock[0], FRAMES_PER_BUFFER);
                        if (err) goto xrun;
                }
                // Fim do laço principal

        //=====================================================================

                // O programa só chega aqui se ocorrer um "break"
                err = Pa_StopStream(stream);
                if (err != paNoError) goto error;
```

```
                Pa_Terminate();
                e->Cancel = true;
                return;

        xrun:
                if (stream)
                {
                        Pa_AbortStream(stream);
                        Pa_CloseStream(stream);
                }
                Pa_Terminate();
                MessageBox::Show("Ocorreu um estouro ou esvaziamento de buffer no laço principal.");
                e->Cancel = true;
                return;

        error:
                if (stream)
                {
                        Pa_AbortStream(stream);
                        Pa_CloseStream(stream);
                }
                Pa_Terminate();
                MessageBox::Show("O Portaudio retornou o erro número: " + err + "\n\n" + "Verifique se há
um dispositivo de gravação de áudio e se as configurações dele (taxa de amostragem, bits por canal, número de
canais) coincidem com as fornecidas.");
                e->Cancel = true;
                return;
        }


        private: System::Void radioButton3_CheckedChanged(System::Object^, System::EventArgs^) {
                groupBoxFIR->Enabled = false;
                groupBoxDenoise->Enabled = true;
                which_algorithm = 3;
        }

        private: System::Void radioButton1_CheckedChanged(System::Object^, System::EventArgs^) {
                groupBoxFIR->Enabled = true;
                groupBoxDenoise->Enabled = false;
                which_algorithm = 1;
        }

        private: System::Void radioButton2_CheckedChanged(System::Object^, System::EventArgs^) {
                groupBoxFIR->Enabled = true;
                groupBoxDenoise->Enabled = false;
                which_algorithm = 2;
        }

        private: System::Void radioButton4_CheckedChanged(System::Object^, System::EventArgs^) {
                is_hard = true;
        }

        private: System::Void radioButton5_CheckedChanged(System::Object^, System::EventArgs^) {
                is_hard = false;
        }

        private: System::Void radioButton6_CheckedChanged(System::Object^, System::EventArgs^ ) {
                is_low_pass = true;
        }
```

```
        private: System::Void radioButton7_CheckedChanged(System::Object^, System::EventArgs^) {
                    is_low_pass = false;
        }

        private: System::Void backgroundWorker1_RunWorkerCompleted(System::Object^,
System::ComponentModel::RunWorkerCompletedEventArgs^) {
                    groupBoxFiltro->Enabled = true;
                    groupBoxDenoise->Enabled = true;
                    groupBoxFIR->Enabled = true;
                    groupBoxParametros->Enabled = true;
                    groupBox1->Enabled = true;
        }
};
}
```

Arquivo: fir.cpp

```
#include "..\..\kiss_fft130\kiss_fft.h"
#include <math.h>
#include "pa_types.h"
#include <vector>
#include <numeric>
#include <functional>
#include "fir.h"

using namespace std;
//---------------------------------------------------------------------------------------
vector<float> projeto_FIR(float freq_corte,int ORD,bool is_low_pass)
{
        vector<float> coef(ORD);
        vector<float> window(ORD);

        // Cálculo dos coeficientes do filtro FIR
        if (ORD == 1) window[0] = 1.0;

        if (ORD == 2) { window[0] = 1.0; window[1] = 1.0; }

        // Calcula os coeficiente das janela de Blackman, que só são bem definidos para ORD>2
        if (ORD > 2)
                for (int k = 0; k < ORD; k++)
                        window[k] = 0.42659f - 0.49656f * cos(2.0f*PI*k / (ORD - 1)) + 0.076849f *
cos(4.0f*PI*k / (ORD - 1));

        float DLY = 0.5f * (ORD - 1);

        // Passa-altas: a freq. de transição do passa-baixas é 1 - freq. de corte
        if (is_low_pass == false) freq_corte = 1.0f - freq_corte;

        for (int k = 0; k < ORD; k++)
        {
                if (k != DLY)
                        coef[k] = window[k] * sin(PI*freq_corte*(k - DLY)) / (PI*(k - DLY));
                else
                        coef[k] = window[k] * freq_corte;
        }

        if (is_low_pass == false)// Passa-altas: a resp. é igual à do passa-baixas, multiplicada por (-1)^k
```

```
                    for (int k = 0; k < ORD; k++) if (k % 2 == 1) coef[k] = -coef[k];

          return coef;
}

//---------------------------------------------------------------------------
void Int24tofloat(vector<unsigned char> & sampleBlock, vector<float> & ch1, vector<float> & ch2,int
FRAMES_PER_BUFFER,int STEP)
{
          unsigned char *src = &sampleBlock[0];
          PaInt32 temp;

          // Estrutura do Buffer Int24
          // [ (byte 0 (menos sig.) - canal esquerdo) (byte 1 - esq.) (byte 2 - esq.) (byte 0 - direito) (byte 1 - dir.)
(byte 2- dir.)
          for (int kk = 0; kk < FRAMES_PER_BUFFER; kk++)
          {
                    // Canal direito
                    temp = (((PaInt32)src[3]) << 8);
                    temp = temp | (((PaInt32)src[4]) << 16);
                    temp = temp | (((PaInt32)src[5]) << 24);
                    ch1[kk] = ((float)temp) * const_1_div_2147483648_;// a constante é para que o valor em
ponto flutuante fique em [-1 1]

                    // Canal esquerdo
                    temp = (((PaInt32)src[0]) << 8);
                    temp = temp | (((PaInt32)src[1]) << 16);
                    temp = temp | (((PaInt32)src[2]) << 24);
                    ch2[kk] = ((float)temp) * const_1_div_2147483648_;

                    src = src + STEP;
          }
}

//---------------------------------------------------------------------------
void floattoInt24(vector<unsigned char> & sampleBlock, vector<float> & ch1out, vector<float> & ch2out, int
FRAMES_PER_BUFFER, int STEP)
{
          unsigned char *src = &sampleBlock[0];
          PaInt32 temp;

          // Estrutura do Buffer Int24
          // [ (byte 0 (menos sig.) - canal esquerdo) (byte 1 - esq.) (byte 2 - esq.) (byte 0 - direito) (byte 1 - dir.)
(byte 2- dir.)
          for (int kk = 0; kk < FRAMES_PER_BUFFER; kk++)
          {
                    // Essas posições do buffer correspondem ao canal direito
                    temp = (PaInt32)(2147483647.0 *ch1out[kk]);// inverso da constante aplicada na conversão
para float
                    src[3] = (unsigned char)(temp >> 8);
                    src[4] = (unsigned char)(temp >> 16);
                    src[5] = (unsigned char)(temp >> 24);

                    // Canal esquerdo
                    temp = (PaInt32)(2147483647.0 * ch2out[kk]);
                    src[0] = (unsigned char)(temp >> 8);
                    src[1] = (unsigned char)(temp >> 16);
                    src[2] = (unsigned char)(temp >> 24);
```

```
                src = src + STEP;
        }
}
//---------------------------------------------------------------------------------------------
void filtro_FIR(vector<float> & ch1, vector<float> & ch2, vector<float> & ch1out, vector<float> & ch2out,
vector<float> & coef, vector<float> & mem1, vector<float> & mem2, bool inicializa, int
FRAMES_PER_BUFFER, int ORD)
{
        // Inicialização dos vetores mem1 e mem2
        if (inicializa) for (int k = 0; k < ORD; k++) { mem1[k] = 0; mem2[k] = 0; }

        // Recebemos FRAMES_PER_BUFFER amostras por chamada
        for (int vf = 0; vf < FRAMES_PER_BUFFER; vf++)
        {
                // desloca os elementos do buffer para baixo
                //for (int k = ORD - 1; k >= 1; k--)
                //{
                //      mem1[k] = mem1[k - 1];
                //      mem2[k] = mem2[k - 1];
                //}
                // alimenta o topo do buffer
                //mem1[0] = ch1[vf]; // faz o papel da amostra que chegou agora
                //mem2[0] = ch2[vf];

                // Joga fora o último elemento
                mem1.pop_back();
                mem2.pop_back();

                // Alimenta o topo do buffer
                mem1.insert(mem1.begin(), ch1[vf]);
                mem2.insert(mem2.begin(), ch2[vf]);

                ch1out[vf] = 0.0;
                ch2out[vf] = 0.0;
                for (int k = 0; k < ORD; k++)// produto interno
                {
                        ch1out[vf] = ch1out[vf] + mem1[k] * coef[k];
                        ch2out[vf] = ch2out[vf] + mem2[k] * coef[k];
                }

                // Produto interno
                //ch1out[vf] = inner_product(mem1.begin(), mem1.end(), coef.begin(), 0.0f);
                //ch2out[vf] = inner_product(mem2.begin(), mem2.end(), coef.begin(), 0.0f);
        }

}

//---------------------------------------------------------------------------
int filtro_FIRC(vector<float> & ch1, vector<float> & ch2, vector<float> & ch1out, vector<float> & ch2out,
vector<float> & coef, vector<float> & mem1, vector<float> & mem2, int buffer_offset,bool inicializa, int
FRAMES_PER_BUFFER, int ORD)
{
        if (inicializa)
        {
                buffer_offset = 0; // posição do ponteiro dentro do buffer circular
                for (int k = 0; k < ORD; k++) { mem1[k] = 0; mem2[k] = 0; }
        }
```

```
		// Recebemos FRAMES_PER_BUFFER amostras por chamada
		for (int vf = 0; vf < FRAMES_PER_BUFFER; vf++)
		{
			// alimenta o topo do buffer
			mem1[buffer_offset] = ch1[vf]; // faz o papel da amostra que chegou agora
			mem2[buffer_offset] = ch2[vf];

			ch1out[vf] = 0.0f;
			ch2out[vf] = 0.0f;

			int indx = buffer_offset;

			for (int k = 0; k < ORD; k++)// produto interno
			{
				ch1out[vf] = ch1out[vf] + mem1[indx%ORD] * coef[k];
				ch2out[vf] = ch2out[vf] + mem2[indx%ORD] * coef[k];
				indx++;
			}

			(buffer_offset)--;
			(buffer_offset) = (buffer_offset) < 0 ? (buffer_offset) += ORD : (buffer_offset);

		}

		return buffer_offset;
}
//---------------------------------------------------------------------
vector<float> dct(vector<float> & in)
{
		int BLOCK = in.size();
		vector<float> out(BLOCK);
		kiss_fft_cfg cfg = kiss_fft_alloc(2 * BLOCK, 0, 0, 0);
		vector<kiss_fft_cpx> cx_in(2 * BLOCK);
		vector<kiss_fft_cpx> cx_out(2 * BLOCK);

		for (int k = 0; k < BLOCK; k++) { cx_in[k].r = in[k]; cx_in[k + BLOCK].r = in[BLOCK - 1 - k];
cx_in[k].i = 0; cx_in[k + BLOCK].i = 0; }

		kiss_fft(cfg, &cx_in[0],&cx_out[0]);

		float incr = PI / (2 * BLOCK);
		float prop = sqrt(1.0f / (2* BLOCK));

		for (int k = 0; k < BLOCK; k++)
			out[k] = prop * (cos(incr*k)*cx_out[k].r + sin(incr*k)*cx_out[k].i);

		out[0] = out[0] / sqrt(2.0f);

		free(cfg);
		return out;
}
//---------------------------------------------------------------------
vector<float> idct(vector<float> & in)
{
		int BLOCK = in.size();
		vector<float> out(BLOCK);
		kiss_fft_cfg cfg = kiss_fft_alloc(BLOCK, 1, 0, 0);
		vector<kiss_fft_cpx> cx_in(2 * BLOCK);
		vector<kiss_fft_cpx> cx_out(2 * BLOCK);
```

```
	float incr = PI / (2 * BLOCK), prop = sqrt(2.0f / BLOCK);

	for (int k = 0; k < BLOCK; k++)
	{
		cx_in[k].r = prop * cos(k*incr) * in[k];
		cx_in[k].i = prop * sin(k*incr) * in[k];
	}

	cx_in[0].r = cx_in[0].r / sqrt(2.0f);
	cx_in[0].i = cx_in[0].i / sqrt(2.0f);

	kiss_fft(cfg, &cx_in[0], &cx_out[0]);

	for (int k = 0; k < BLOCK / 2; k++){ out[2 * k] = cx_out[k].r;  out[2 * k + 1] = cx_out[BLOCK - k - 1].r; }

	free(cfg);
	return out;
}
//---------------------------------------------------------------------------
void denoise(vector<float> & ch1, vector<float> & ch2, vector<float> & ch1out, vector <float> & ch2out,vector<float> & e1, vector<float> & e2, vector<float> & s1, vector <float> & s2, vector<float> & s1_past, vector<float> & s2_past,float lower_thr, float upper_thr, bool is_hard, bool inicializa, int FRAMES_PER_BUFFER)
{
	int BLOCK = 2 * FRAMES_PER_BUFFER;
	vector<float> T1(BLOCK);
	vector<float> T2(BLOCK);

	if (inicializa) for (int k = 0; k < BLOCK; k++) { e1[k] = 0; e2[k] = 0; s1[k] = 0; s2[k] = 0; s1_past[k] = 0; s2_past[k] = 0; }

	// Hipoteses - a) BLOCK = 2 FRAMES_PER_BUFFER b) DESLOCAMENTO = FRAMES_PER_BUFFER

	// Desloca os buffers de FRAMES_PER_BUFFER
	// [1 ORD] -> [FRAMES_PER_BUFFER +1 ORD (1 FRAMES_PER_BUFFER)]
	for (int k = 0; k < BLOCK - FRAMES_PER_BUFFER; k++)// desloca os elementos do buffer para baixo
	{
		e1[k] = e1[k + FRAMES_PER_BUFFER];
		e2[k] = e2[k + FRAMES_PER_BUFFER];
	}
	// Escreve o sinal recebido nas últimas FRAMES_PER_BUFFER posições
	for (int k = 0; k < FRAMES_PER_BUFFER; k++)
	{
		e1[k + BLOCK - FRAMES_PER_BUFFER] = ch1[k];
		e2[k + BLOCK - FRAMES_PER_BUFFER] = ch2[k];
	}

	// Pass-through
	//for (int k = 0; k < BLOCK; k++) { s1[k] = e1[k]; s2[k] = e2[k]; }

	//-----------------------------------------------------
	// Denoising
	T1=dct(e1);
	T2=dct(e2);
```

```
float abs_T1, sign_T1, abs_T2, sign_T2;

// Calcula o máximo valor absoluto dentro de cada bloco
float max_T1 = 0.0, max_T2 = 0.0;
for (int k = 0; k < BLOCK; k++)
{
	abs_T1 = abs(T1[k]);
	abs_T2 = abs(T2[k]);
	if (abs_T1 > max_T1) max_T1 = abs_T1;
	if (abs_T2 > max_T2) max_T2 = abs_T2;
}

float upper1=0.0, upper2=0.0, lower1=0.0, lower2=0.0;

// Normaliza os limiares
if (max_T1 > 0.0)
{
	upper1 = upper_thr * max_T1;
	lower1 = lower_thr * max_T1;
}

if (max_T2 > 0.0)
{
	upper2 = upper_thr * max_T2;
	lower2 = lower_thr * max_T2;
}

if (is_hard)
	for (int k = 0; k < BLOCK; k++)
	{
		// Fórmula HardHard - Tese Irineu Antunes Jr. - p. 42
		abs_T1 = abs(T1[k]);
		abs_T2 = abs(T2[k]);
		if (abs_T1 >= lower1 && abs_T1 <= upper1) T1[k] = 0.0;
		if (abs_T2 >= lower2 && abs_T2 <= upper2) T2[k] = 0.0;
	}
else
	for (int k = 0; k < BLOCK; k++)
	{
		abs_T1 = abs(T1[k]);
		abs_T2 = abs(T2[k]);
		sign_T1 = T1[k]>0.0 ? 1.0f : -1.0f;
		sign_T2 = T2[k]>0.0 ? 1.0f : -1.0f;

		// Formula SoftSoft - Tese Irineu Antunes Jr. - p. 43
		if (abs_T1 >= lower1 && abs_T1 <= upper1)
			T1[k] = sign_T1*lower1;
		else if (abs_T1 > upper1)
			T1[k] = sign_T1 * (abs_T1 - (upper1 - lower1));

		if (abs_T2 >= lower2 && abs_T2 <= upper2)
			T2[k] = sign_T2*lower2;
		else if (abs_T2 > upper2)
			T2[k] = sign_T2 * (abs_T2 - (upper2 - lower2));
	}

s1=idct(T1);
s2=idct(T2);
//-------------------------------------------------------
```

```
        // Alimenta o buffer de saída
        for (int k = 0; k < FRAMES_PER_BUFFER; k++)
        {
                ch1out[k] = 0.5f*(s1_past[k + BLOCK - FRAMES_PER_BUFFER] + s1[k]);
                ch2out[k] = 0.5f*(s2_past[k + BLOCK - FRAMES_PER_BUFFER] + s2[k]);
        }


        // s1_past <- s1
        for (int k = 0; k < BLOCK; k++)
        {
                s1_past[k] = s1[k];
                s2_past[k] = s2[k];
        }
}
/*********************************************************************/
```

Arquivo: fir.h

```
#include <stdbool.h>
#include <vector>
using namespace std;

static const float PI = 3.14159265358979323846264338327950;
static const float const_1_div_2147483648_ = 1.0 / 2147483648.0;

vector<float> projeto_FIR(float freq_corte, int ORD, bool is_low_pass);

void Int24tofloat(vector<unsigned char> & sampleBlock, vector<float> & ch1, vector<float> & ch2,
        int FRAMES_PER_BUFFER,int STEP);

void floattoInt24(vector<unsigned char> & sampleBlock, vector<float> & ch1out, vector<float> & ch2out,
        int FRAMES_PER_BUFFER, int STEP);

void filtro_FIR(vector<float> & ch1, vector<float> & ch2, vector<float> & ch1out, vector<float> & ch2out,
        vector<float> & coef, vector<float> & mem1, vector<float> & mem2, bool inicializa, int
FRAMES_PER_BUFFER, int ORD);

int filtro_FIRC(vector<float> & ch1, vector<float> & ch2, vector<float> & ch1out, vector<float> & ch2out,
        vector<float> & coef, vector<float> & mem1, vector<float> & mem2, int buffer_offset, bool
inicializa,
        int FRAMES_PER_BUFFER, int ORD);

void denoise(vector<float> & ch1, vector<float> & ch2, vector<float> & ch1out, vector <float> & ch2out,
        vector<float> & e1, vector<float> & e2, vector<float> & s1, vector <float> & s2,
        vector<float> & s1_past, vector<float> & s2_past,
        float lower_thr, float upper_thr, bool is_hard, bool inicializa, int FRAMES_PER_BUFFER);
```