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

Criação e manipulação de áudio 3D em tempo real utilizando unidades de processamento gráfico (GPU)

Diego Augusto Rodrigues Gomes

Brasília
2012

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

Criação e manipulação de áudio 3D em tempo real utilizando unidades de processamento gráfico (GPU)

Diego Augusto Rodrigues Gomes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Pedro de Azevedo Berger

Brasília
2012

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

Coordenador: Prof.ª Dr.ª Myléne Christine Queiroz de Farias

Banca examinadora composta por:

Prof. Dr. Pedro de Azevedo Berger (Orientador) — CIC/UnB
Prof. Dr. Alexandre Zaghetto — Cic/UnB
Prof. Dr. Bruno Luiggi Macchiavello Espinoza — Cic/UnB

CIP — Catalogação Internacional na Publicação

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

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

Criação e manipulação de áudio 3D em tempo real utilizando unidades de processamento gráfico (GPU)

Diego Augusto Rodrigues Gomes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof. Dr. Pedro de Azevedo Berger (Orientador)
CIC/UnB

Prof. Dr. Alexandre Zaghetto
Cic/UnB

Prof. Dr. Bruno Luiggi Macchiavello Espinoza
Cic/UnB

Prof.ª Dr.ª Myléne Christine Queiroz de Farias
Coordenador do Mestrado em Informática

Brasília, 04 de Junho de 2012

Dedicatória

Ao meu pai, mãe, irmãos, namorada, amigos e mestres.

Agradecimentos

Gratidão. Tenho muito que agradecer às inúmeras pessoas que me incentivaram e que me deram forças para seguir com esse trabalho. Com toda certeza, suas palavras de apoio tiveram um peso enorme.

Em primeiro lugar, agradeço humildemente à Deus pela força para concluir mais essa etapa.

Agradeço à minha família que sempre me deu apoio para alcançar meus sonhos. Em especial à minha mãe, guerreira. Ao meu pai, de quem sinto muita saudade e que com certeza está olhando para mim e ajudando a clarear meus caminhos. Aos meus irmãos Alexandre e Eduardo que, independentemente de suas esquisitices (shuuuu), estão sempre ao meu lado. Vanilda, sempre com aquela paciência espartana mas também sempre presente. Antônia, parceira de todas as tardes. Tios, tias, primos e primas... Amo vocês!

Um sonoroso obrigado à minha amiga, namorada e cúmplice, Rosana, pelo total apoio e compreensão.

Meus sinceros agradecimentos aos meus amigos do "elite": valeu, galera!!

Ao pessoal das bandas - Sacramento, Essência, DF76, Flor de Severino, Fogo de Palha, "Pães do céu", Na Hora Sai - muito obrigado!

Meu agradecimento ao professor Pedro Berger que primeiramente acreditou e teve confiança de que eu seria capaz de desenvolver este trabalho. Muito obrigado, Pedro, por me incentivar a buscar aqueles resultados que eu não conseguia ver, mas que com seu "*feeling*", estavam lá. Obrigado por clarear as coisas e ver o trabalho com outras perspectivas. Agradeço também pela ajuda do professor Julio Cesar Boscher Torres, por ter tirado várias dúvidas e fornecido o material inicial no desenvolvimento dessa pesquisa. Também sou grato à todos do Departamento de Ciência da Computação da UnB que, de maneira direta ou não, fizeram parte da minha formação.

Muito obrigado aos meus amigos e colegas da computação que, muitas vezes, fizeram sugestões para melhoria do trabalho. Obrigado em especial a alguns amigos que acompanharam essa etapa: Aman Rathie, Daniel Sundfeld e Fabrício Buzeto. Muito obrigado, por compartilhar comigo praticamente todos esses anos de estudo e pelas dicas e sugestões. Agradeço também ao meu amigo Carlos Costa pelo incentivo nessa pesquisa.

To my foreign friends Marwa, Salma, Shereen, Lee, Dana, Mike: thanks a lot!
Sou grato a todas as pessoas que estiveram comigo nessa etapa.

Resumo

O uso crescente de unidades de processamento gráfico, *Graphics Processing Unit*s (GPUs), no desenvolvimento de aplicações de propósito geral nos permite criar softwares e sistemas capazes de processar grandes volumes de dados paralelamente. Isso faz com que aplicações, que antes imaginávamos inviáveis no sentido de sua utilização, possam ser construídas. No contexto de aplicações que demandam alto poder de processamento, encontra-se o processo de criação de áudio tridimensional em tempo real e os sistemas de auralização. Tais sistemas têm o objetivo de simular virtualmente o posicionamento de fontes sonoras por meio de funções matemáticas responsáveis por aplicar os parâmetros de direcionalidade em um bloco de sinal de áudio, transmitindo-nos a sensação de posicionamento no campo sonoro ao nosso redor. Essa capacidade demanda a utilização de procedimentos de interpolação capazes de estimar as funções matemáticas de posições pelas quais tais valores não são conhecidos, a partir de funções vizinhas previamente conhecidas. A possibilidade de otimização desses métodos de interpolação e consequentemente de sistemas que se utilizam destes métodos, abre caminhos para novas experiências sonoras no campo do entretenimento e da realidade virtual. Para tornar essa otimização possível, experimenta-se o uso de dispositivos eficientes e acessíveis capazes de processar um volume massivo de dados em paralelo.

Este trabalho mantém o foco no desenvolvimento de uma biblioteca capaz de realizar a interpolação de HRTFs, as funções de transferência relacionadas à cabeça responsáveis por transmitir a percepção de direcionalidade de uma fonte sonora, e a síntese de áudio tridimensional em tempo real por meio da utilização de hardware gráfico para o processamento massivo de dados em paralelo e tomando como base o algoritmo de interpolação no domínio da transformada wavelet desenvolvido em trabalhos anteriores. Além disso, com a otimização desse método de interpolação, apresenta-se um programa capaz de sintetizar em tempo real o áudio resultante da aplicação dessas funções a partir de dados referentes ao posicionamento espacial da fonte sonora fornecidos por um usuário. Também como parte do escopo desse trabalho, a implementação com técnicas de processamento paralelo com base no uso de múltiplas threads é comparada com a implementação utilizando GPU.

Palavras-chave: Auralização, Interpolação, som 3D, HRTF, *Head-related Impulse Response* (HRIR), *Compute Unified Device Architecture* (CUDA), GPU

Abstract

The increasing use of graphics processing units (GPUs) in the development of general-purpose applications allows us to develop software and systems capable of processing large amount of data in parallel. This means that applications once thought impractical, in the sense of its use, can be constructed with the use of these powerful and emerging processing units. In the context of applications that are heavy in terms of processing, there is the process of creating three-dimensional audio in real time and the auralization systems. Such systems are designed to virtually simulate the placement of sound sources by means of mathematical functions responsible for applying the parameters of directionality in a block of audio signal, giving us the sense of position in the sound field around the listener. This capability requires the use of interpolation procedures able to estimate the mathematical functions of positions, for which such values are not known, from neighboring functions previously known. The possibility of optimization of these methods of interpolation, and as a consequence the systems that use these methods, brings new sound experiences in the field of entertainment and virtual reality. To make this optimization possible, we experiment to use efficient and affordable devices capable of handling a massive volume of data in parallel.

This work aims to focus on the development of a library to interpolate the HRTFs, the mathematical functions responsible for transmitting the perception of directionality of a sound source, and to synthesize 3D audio in real-time by using graphic hardware for massive and parallel data processing based on the interpolation algorithm in wavelet domain developed in previous works. In addition, with the optimization of this method of interpolation, as a result of using graphics processing units, a program was created to synthesize real-time audio resulting from the application of these transfer functions from data on the spatial positioning of the source sound provided by a user. Also as part of the scope of this work, we compare the implementation using multiple threads techniques with the implementation using GPU.

Keywords: Auralization, Interpolation, 3D sound, HRTF, HRIR, CUDA, GPU

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Funções

Lista de Abreviaturas e Siglas

2D bidimensional. 5

3D tridimensional. 5, 22, 47

ALU *Arithmetic Logic Unit*. 41

ANSI *American National Standards Insitute*. 63

API *Application Programming Interface*. 43, 63

C linguagem de programação C. ix, xiii, xiv, 3, 4, 44, 50, 58, 59, 63, 66, 69–72, 89–94

CPU *Central Processing Unit*. xii–xiv, 35, 41, 43–45, 47, 48, 59, 66, 67, 69, 70, 76, 89

CUDA *Compute Unified Device Architecture*. ix, xii–xiv, 3, 4, 43–47, 49, 50, 67, 69–72, 89–95

DFT *Discrete Fourier Transform*. 20, 82

FFT *Fast Fourier Transform*. 19, 20, 22, 47, 58, 65, 67, 69, 82, 84

FFTW *Fastest Fourier Transform in the West*. 65, 67

FIR *Finite Impulse Response*. 20–22, 26, 37, 38

GFLOPS *Giga Floating point Operations Per Second*. 41

GPGPU *General-Purpose Computing on Graphics Processing Unit*. 47

GPU *Graphics Processing Unit*. vi, vii, ix, xiv, 3, 4, 35, 41, 43–45, 47–50, 67, 69, 70, 73, 74, 76

HRIR *Head-related Impulse Response*. xii, xiii, 2, 26, 27, 29–31, 33, 34, 37, 40, 51, 53, 54, 56, 59–62, 66–69, 72–74, 76, 84

HRTF *Head-related Transfer Function*. vi, vii, ix, xii, 2–4, 14, 15, 22, 26, 27, 29–35, 40, 49–51, 56, 58, 60, 61, 63–67, 69, 72, 73

IID *Interaural Intensity Difference*. 14

IIR *Infinite Impulse Response*. 20, 34

ILD *Interaural Level Difference*. 13, 14, 34

IPTF *Interpositional Transfer Function*. 31, 33

IRCAM *Institut de Recherche et Coordination Acoustique/Musique*. 27

ITD *Interaural Time Difference*. 13, 14, 34

ITU-R *International Telecommunication Union - Radiocommunication sector*. 73, 74

KEMAR *Knowles Electronics Mannequin for Acoustics Research*. 27, 29

libSDL *Simple DirectMedia Layer*. 65

MATLAB Linguagem de computação técnica *MATrix LABoratory*. ix, 4, 50, 51, 58, 59, 62, 69, 72

MIT *Massachusetts Institute of Technology*. 27, 30, 65

MVC *Model-View-Controller*, Modelo-Visão-Controlador. 63

ODG *Objective Difference Grade*. xiii, 73–75

PEAQ *Perceptual Evaluation of Audio Quality*. 72–74

SDG *Subjective Difference Grade*. 73

SFRS *Spacial Frequency Response Surface*. 33

WAVE *WAVEform audio format*. 63

WHRTF HRTF criada pelo método de interpolação no domínio da transformada wavelet. xiii, xiv, 66, 69, 70, 72, 76, 89, 94, 95

Capítulo 1

Introdução

1.1 Motivação

Ir a uma sessão de cinema hoje em dia pode parecer algo simples e corriqueiro, entretanto frequentemente não questionamos as inúmeras tecnologias envolvidas na reprodução da imagem e do áudio. Compramos nossos ingressos e compartilhamos com outros a experiência de estar praticamente dentro do filme, atuando como meros observadores. A tela grande, o ambiente escuro de forma que não enxerguemos com nitidez aquilo que está ao redor e a intensidade do som cria uma sensação de que estamos imersos na história que está sendo contada. Cada vez mais, filmes que utilizam tecnologia de visualização em três dimensões são apresentados nas salas de cinema. Percebe-se que essa evolução não ocorre apenas nessas salas: ela caminha para dentro das casas. Como exemplo, grandes fabricantes de televisores já disponibilizam aparelhos capazes de reproduzir imagens em três dimensões.

A atenção dada aos sistemas de realidade virtual vem aumentando devido à evolução tecnológica, que permite a criação de aplicações que se utilizam da percepção espacial e auditiva de forma a obter maior integração entre o ser humano e esse tipo de aplicação (jogos eletrônicos, simuladores de voos, sistemas de som tridimensional, dentre outros) [1].

Em trabalhos anteriores [2], os autores criaram o conceito de *3D Internet*, que nada mais é que uma espécie de evolução da Internet a qual conhecemos, de forma a torná-la mais versátil, interativa e usável. Tal artigo motiva a pesquisa em diversas áreas como redes, segurança e computação distribuída, visando incrementar a natureza interativa da Web 2.0 com o desenvolvimento de aplicações com tecnologia tridimensional (3D) e de novos paradigmas para a mesa de trabalho dentro do computador. A ideia principal é criar uma interface totalmente imersiva onde o usuário possa navegar pelos conteúdos da Web de uma maneira mais interativa e real.

Alguns estudos mais recentes [3] mostram que as tecnologias de visualização 3D na representação da realidade já estão maduras o suficiente para serem mescladas com aplicações da Internet, como é o caso do Second Life [4] e World of Warcraft [5]. Essa tendência está alterando progressivamente a maneira pela qual as pessoas irão interagir com a Internet do futuro. Outros estudos [6] já investigam ambientes virtuais tridimensionais para uma rede de sensores totalmente imersivos, visando o desenvolvimento de um conceito de interação homem-rede.

Para acompanhar essa evolução nos sistemas de visualização e interação tridimensional e tendo como fontes de inspiração e motivação áudios publicados na Internet como, por exemplo, o áudio conhecido como *Virtual Barbershop* [7], é preciso desenvolver também sistemas de realidade virtual acústica que transmitam fielmente a sensação auditiva de espacialidade e auralização do som. Entretanto tais sistemas requerem uma considerável complexidade computacional tanto em aplicações pré-processadas quanto em aplicações de tempo real. Com base nisso, métodos de processamento mais elaborados e eficientes são estudados para atender a demanda de sistemas de auralização principalmente no campo do entretenimento. Além disso, mecanismos eficientes de auralização também podem e são aplicados em ferramentas de simulação acústica de salas. Tais aplicações também irão se beneficiar do projeto desenvolvido a partir da pesquisa realizada nesse trabalho.

Na tentativa de recriar artificialmente ambientes espaciais de áudio, pode-se seguir duas abordagens diferentes. A primeira, intimamente relacionada a área de psicoacústica, explora os testes subjetivos de audição para examinar a capacidade do sistema auditivo humano na identificação de posições. Já a segunda, relacionada à disciplina de processamento de sinais, preocupa-se com a implementação de métodos otimizados para a criação de áudio espacial [8]. Os sistemas de auralização que utilizam fones de ouvido, conhecidos também como biaurais, conforme será apresentado no Capítulo 2, exploram a aplicação de filtros digitais sobre o sinal de áudio para simular um certo posicionamento no espaço ao redor de um ouvinte. Esses filtros, conhecidos como funções de transferência relativas à cabeça (HRTFs) ou respostas a impulso relativas à cabeça (HRIRs), descrevem como o sistema auditivo processa sons de localizações particulares. Em outras palavras, a imposição de uma HRTF sobre fontes sonoras não espacializadas, isto é, que não possuem características de posicionamento ou efeitos de reverberação, também conhecidas como fontes sonoras "secas", adiciona as propriedades espaciais descritas por essa HRTF [8]. Em suma, as HRTFs (ou suas HRIRs equivalentes) descrevem como uma onda sonora de uma determinada posição atinge a membrana do tímpano de uma pessoa. Cada posição ao redor de um ouvinte é representada por um único par de HRTFs (para as orelhas esquerda e direita).

Para se obter os valores de HRIRs são necessários muitos recursos tais como tempo e equipamentos especializados. Além disso, no artigo [9] os autores mencionam a dificuldade em mensurar empiricamente os valores de HRIRs. O tempo, o custo e as possíveis armadilhas encontradas no processo de medição de HRTFs contribuem para o desejo de se ter um banco de dados generalizado. Devido a essas limitações, apenas algumas amostras de HRIRs no espaço tridimensional são tomadas [10]. Em virtude deste fato e também com o crescimento das pesquisas sobre HRTFs, surgiu a necessidade de se investigar métodos de interpolação capazes de encontrar as HRIRs de posições não conhecidas a partir de posições já sabidas. Na literatura foram encontrados trabalhos envolvendo sistemas biaurais e estes tendem a buscar maneiras eficientes de se representar HRTFs minimizando o processamento e o espaço de armazenamento mas muitas vezes com a perda de certa parte das informações [8]. Este trabalho trata as HRTFs em seu tamanho original de modo a evitar a perda de dados conforme mencionado anteriormente.

O presente trabalho conserva seu escopo no contexto da realidade virtual acústica, mantendo o foco na evolução e otimização de um mecanismo de interpolação de funções de transferência relativas à cabeça no domínio da transformada wavelet por meio do uso de placas gráficas (GPU) para o processamento paralelo de dados e aceleração do algoritmo,

criando a sensação de espacialidade do som e objetivando a melhoria da performance de execução. Associada à implementação desse método de interpolação, está a criação de um mecanismo eficiente de posicionamento tridimensional sonoro que irá possibilitar ao usuário posicionar e avaliar o sinal resultante em tempo real. Para tanto, faz-se necessário compreender os fundamentos da espacialização, auralização, interpolação dessas funções de transferência relativas à cabeça e o processamento em paralelo por meio do uso de unidades gráficas de processamento. Assim, é possível identificar de que forma um sistema pode contemplar os requisitos necessários para tal. Além disso, avaliações de performance são realizadas no intuito de identificar as melhorias obtidas ao se utilizar GPU. Também, avaliações subjetivas com base na audição dos resultados são realizadas para a verificação do método de interpolação implementado. Para complementar os testes subjetivos, são feitos também, testes automatizados com uma ferramenta capaz de obter um valor objetivo para estimar o grau de diferenciação entre dois sinais. Esse valor dá uma ideia do quanto o sinal foi prejudicado após a aplicação dos filtros utilizados no desenvolvimento desse trabalho.

1.2 Objetivos

O objetivo dessa dissertação é implementar um sistema de auralização, com desempenho em tempo real, de sinais de áudio digitais utilizando para isso um método de interpolação de HRTFs que faz o uso de unidades de processamento gráfico. Mais especificamente, pretende-se:

1. Construir uma biblioteca em C que realiza a interpolação de HRTFs e a síntese de áudio tridimensional em tempo real, utilizando o algoritmo de interpolação de HRTFs no domínio da transformada wavelet, que é um algoritmo recente e que apresenta alguma possibilidade de paralelismo.

2. Estudar e comparar técnicas de processamento paralelo para melhorar o desempenho da biblioteca supracitada com o uso de múltiplas threads e unidades de processamento gráfico (GPU) que, neste caso, está limitado ao uso da arquitetura de computação paralela CUDA da fabricante NVIDIA. O uso da GPU é interessante pois é um recurso que está disponível na maioria dos computadores modernos e que pode estar geralmente ocioso. Portanto, sempre que possível, vale a pena utilizá-lo para o processamento de dados.

3. Implementar um aplicativo que realiza o posicionamento de um objeto sonoro em qualquer posição no campo espacial tridimensional, utilizando, para isso, a biblioteca desenvolvida. Tal aplicativo deve ser eficiente e intuitivo no propósito de mixagem de áudio, possibilitando o posicionamento de uma fonte sonora em pontos virtuais ao redor de um ouvinte.

Por fim, pretende-se disponibilizar a biblioteca e o aplicativo desenvolvido como um projeto de código aberto (*open source*) para que a comunidade interessada no assunto possa desenvolvê-lo e usufruir de seus benefícios. Essa biblioteca, além de poder ser aplicada aos sistemas biaurais, também poderá ser utilizada no contexto da compressão de áudio espacializado. Além disso, com esse trabalho, procura-se estimular a pesquisa

na área de processamento de áudio em GPU haja vista que, na pesquisa feita sobre a bibliografia relacionada ao assunto, percebe-se que esse tema é pouco discutido e não há uma biblioteca de código aberto semelhante capaz de realizar a tarefa proposta.

Como consequência desse trabalho, as ideias aqui desenvolvidas poderão ser aplicadas na construção de sistemas de mixagem tridimensional profissionais ou até mesmo a criação de módulos que se integrem a produtos já existentes no mercado.

1.3 Estrutura do Documento

Nos capítulos que se seguem, os conceitos que envolvem a reprodução e a percepção de sons no espaço tridimensional são apresentados e explicados com o intuito de auxiliar no entendimento desse projeto. Além disso, também são apresentados capítulos que discutem como o projeto foi implementado, os resultados alcançados e as conclusões obtidas.

O Capítulo 2, entitulado como "Sistemas de Espacialização e Auralização", apresenta uma introdução aos mecanismos de espacialização e auralização do som. Neste capítulo abordam-se algumas características do espaço sonoro monaural, estéreo e multicanal, assunto este relacionado à sensação de espacialidade e envolvimento do som. Também explora-se o conceito de auralização, referindo-se a campos sonoros gerados a partir de modelos físicos ou matemáticos cuja percepção de direcionalidade deve coresponder às expectativas declaradas pelo modelo utilizado.

Já o capítulo seguinte, Capítulo 3 entitulado como "Síntese Sonora Tridimensional", apresenta alguns conceitos básicos de processamento de sinais e os conceitos referentes à síntese de áudio biaural. Nesse capítulo, também são apresentados o princípio da auralização e algumas técnicas relacionadas à síntese de áudio biaural em tempo real.

O próximo capítulo, Capítulo 4 cujo título é "Funções de Transferência Relacionadas à Cabeça", introduz o conceito de funções de transferência relativas à cabeça, como essas funções são medidas e alguns métodos de interpolação. Nesse capítulo, o método de interpolação escolhido, conhecido como "interpolação de HRTFs no domínio da transformada wavelet", é apresentado detalhadamente tendo como base os trabalhos desenvolvidos em [1, 11, 12, 13, 14].

O Capítulo 5, "Computação Paralela com o uso de Unidades de Processamento Gráfico", introduz os conceitos principais de unidades de processamento gráfico (GPU) e como essas unidades são utilizadas no desenvolvimento de aplicações que demandam alto poder de processamento. Além disso, neste capítulo, são apresentados alguns trabalhos que utilizam a GPU para processamento de áudio.

No Capítulo 6, "Proposta e Implementação", é apresentada a parte técnica deste trabalho. As etapas no processo de melhoria do algoritmo de interpolação de HRTFs no domínio da transformada wavelet com processamento em GPU e a implementação de um software para verificação e validação dessas melhorias aplicadas em amostras reais de áudio são apresentadas em detalhes, bem como alguns algoritmos que compõem esse processo. Nesse capítulo, o ambiente utilizado para desenvolvimento é apresentado bem como as três etapas de implementação desse projeto referentes à cada ambiente utilizado para desenvolvimento: MATLAB, C e CUDA. Os resultados obtidos nessas três etapas são apresentados e discutidos também nesse mesmo capítulo.

Enfim, o Capítulo 7, "Conclusão e Trabalhos Futuros", sintetiza o que foi desenvolvido nesse trabalho, as conclusões obtidas e os trabalhos que poderão ser desenvolvidos poste-

riormente a partir deste. Também mostra algumas limitações da implementação e mostra que os objetivos propostos foram alcançados com êxito.

Capítulo 2

Sistemas de Espacialização e Auralização

Os sistemas de áudio bidimensional (2D) e tridimensional (3D) podem ser diferenciados em dois tipos principais: os sistemas para espacialização e os sistemas de auralização. O primeiro, sugere uma sensação de espacialização do som, gerando campos sonoros envolventes. Já o segundo, é mais preciso ao proporcionar a percepção de diretividade e distância, reproduzindo mais fielmente as características acústicas do ambiente. Esse capítulo tem o intuito de apresentar as diferenças entre eles e mostrar que trata-se basicamente de um processo evolutivo nas formas de reprodução de áudio, embora ambos co-existam e são utilizados em diferentes aplicações.

Na Seção 2.1, serão abordados os aspectos teóricos e algumas características do espaço sonoro monaural, estéreo e multicanal que envolvem o conceito de espacialização do som. Na Seção 2.2, o conceito de auralização será explicado, incluindo os mecanismos de posicionamento por meio das funções de transferência relacionadas à cabeça.

2.1 Espacialização

Com o desenvolvimento da tecnologia, a espacialização e a auralização do som tornaram-se mecanismos importantes no processo de composição de obras musicais. A movimentação do som sobre o espaço permite criar efeitos que provocam diferentes sensações entre os ouvintes. Cada vez mais, sistemas que oferecem maior realidade visual e auditiva são apresentados às pessoas e a manipulação de sons digitais no computador e num espaço virtual tem tomado a atenção do público em geral [15].

Em contraste com a grande evolução nas interfaces gráficas para usuário e com o desenvolvimento de ferramentas para manipulação de informações visuais, a evolução dos mecanismos de áudio por computador não caminha tão rapidamente. Isso se deve ao fato de a visão ser o sentido de percepção dominante para a maioria das pessoas. Com isso, é o principal meio para obter informações. Acredita-se também que os sistemas de manipulação de áudio foram deixados em segundo plano pois não eram necessários para poder operar um computador [15]. Além disso, em seu início, era praticamente de uso exclusivo por pesquisadores da área de computação musical e psicoacústica [15]. Após a aceitação e padronização dos sons digitais e a diminuição do preço dessa tecnologia, um

número maior de pessoas passou a conhecê-la e utilizá-la. É inevitável então a evolução de interfaces para a manipulação de áudio digital em computadores pessoais.

A espacialização trata de formas de reprodução sonora de um efeito de envolvimento sem se importar tanto com a localização precisa dos objetos sonoros [16]. Também, não é tão importante a proximidade dos objetos sonoros em relação aos ouvintes. Esse tipo de sistema de áudio é utilizado amplamente em sistemas comerciais voltados para o entretenimento tais como filmes e jogos, sendo menos explorado na reprodução de campos sonoros bidimensionais e tridimensionais em que a diretividade é primordial para a aplicação.

Ao falar de espacialização, é preciso primeiramente conhecer os mecanismos de reprodução sonora que fazem parte do conhecimento humano e como eles evoluiram até o estado em que se encontram atualmente. Nesta seção, verifica-se que a evolução nos meios de reprodução de áudio está intimamente ligada à evolução das formas de entretenimento. É interessante entender como o conhecimento desenvolvido por Thomas Edison e seus contemporâneos no final do século XIX possa ter evoluído para tecnologias tão complexas e que, ao mesmo tempo, ainda são capazes de proporcionar experiências sonoras inéditas aos ouvintes.

A história dos mecanismos de reprodução sonora inicia-se no século XIX [16]. Os primeiros dispositivos de registro e reprodução, inventados aproximadamente na década de 1880, baseavam-se nos princípios de transdução acústico-mecânicos com apenas um único alto-falante para a saída de áudio. Com a invenção da estereofonia, vários experimentos foram realizados e culminaram mais uma vez na evolução dessa tecnologia para um arranjo com quatro alto-falantes. Esse novo arranjo, detalhado pelo trabalho desenvolvido em [17], mostra que para um sistema de som *surround* funcionar eficientemente, ele deve ser capaz de capturar todas as nuances do som reverberante e reproduzi-las uniformemente ao redor do ouvinte.

Os efeitos de espacialização mais populares são o *panning* horizontal e a reverberação. O *panning* horizontal se caracteriza pelo movimento lateral do som de um alto-falante a outro em um plano horizontal. A reverberação consiste em adicionar densidade e profundidade ao som por meio de ecos, fazendo-o parecer estar localizado em um ambiente maior do que o local em que ele realmente está [18].

Serão apresentados nessa seção alguns conceitos relacionados às formas de reprodução sonora culminando em sistemas de espacialização multicanais. Inicia-se com um breve histórico sobre o surgimento dos sistemas monaurais (Seção 2.1.1), seguindo para a estereofonia (Seção 2.1.2) e os sistemas multicanais (Seção 2.1.3).

2.1.1 Sistemas Monaurais

Em meados da década de 1880, os mecanismos de registro e reprodução sonora utilizavam cilíndros de cera pelos quais percorria uma agulha capaz de gravar informações sobre ele e posteriormente ler essas informações. Além disso, para criar um alto-falante, era preciso um pedaço de papelão em formato de funil sobre um orifício estreito preso a um pedaço de papel impermeável. Assim, criava-se uma membrana vibratória que estava ligada a uma agulha. Dessa forma, quando alguém falava ou cantava em frente ao funil, as vibrações provocadas sobre o papelão eram transmitidas à agulha e, consequentemente, à superfície coberta com cera. Esta superfície girava lentamente até o seu limite. Após ter a gravação marcada sobre a cera, uma camada de verniz era aplicada sobre ela para enrijecer

o cilíndro e para que a marca criada pudesse ser lida de alguma maneira sem prejudicar a marcação original. Ao percorrer novamente este caminho, a agulha transmitia a vibração originada do cilíndro ao papelão, reproduzindo o som gravado anteriormente [19].

Os dispositivos conhecidos como fonógrafos (vide Figura 2.1), criado por Thomas A. Edison, e sua evolução, o gramofone, proporcionaram um grande impacto na indústria da música no início do século XX. Novas experiências musicais foram possíveis e esse mecanismo permitiu também o fortalecimento do compartilhamento de experiências culturais populares [20]. O gramofone (vide Figura 2.2), criado pelo alemão Emile Berliner, diferentemente do fonógrafo de Edison, utilizava um disco plano e a gravação, ao invés de ser em hélice (como nos cilíndros), era formado por uma espiral. Na década de 1930, o disco tornou-se um padrão de gravação na indústria da música [21].

Figura 2.1: Fonógrafo. Fonte: *Gramophone, Film, Typewriter* [19]

Uma característica comum a esses dois aparelhos é que eles reproduzem o áudio por apenas um alto-falante. O sinal lido do cilindro (ou disco), por uma agulha, é transmitido a um alto-falante apenas, caracterizando uma fonte sonora pontual. Atualmente, em sistemas monaurais, a utilização de um só alto-falante pode fornecer alguma informação espacial mas não permite identificar e mapear a diretividade e posicionamento das fontes sonoras da gravação original. Apesar de ser possível reproduzir sinais de instrumentos musicais com grande fidelidade, tanto em seu espectro de frequências quanto em amplitude, não somos capazes de reproduzir a forma de irradiação natural do instrumento. É preciso utilizar uma outra maneira de reprodução que possa adicionar os indicadores de diretividade [16].

Figura 2.2: Gramofone. Fonte: *Gramophone, Film, Typewriter* [19]

2.1.2 Estereofonia

Alguns testes permitiram identificar que com a utilização de dois canais de áudio, dois alto-falantes, obtia-se maior realismo espacial sonoro do que quando usava-se apenas um alto-falante. Ao utilizar um sistema de reprodução de áudio composto por dois canais de sons monaurais, tem-se o que é chamado de estereofonia.

O primeiro registro de testes com sons estereofônicos data de 1881 e foi realizado por Clément Ader. Ele utilizou pares de linhas de telefone para transmitir a Ópera de Paris. Entretanto, o maior contribuidor à estereofonia foi Alan Dower Blumlein que, em 1931, patenteou os elementos essenciais dos sistemas de sons estéreo utilizados até hoje [22]. Somente após a Segunda Guerra Mundial, a partir da década de 50, quando o sistema estéreo foi introduzido comercialmente no mercado, mais experimentos foram realizados [17] e foi possível estudar como essa tecnologia poderia atuar no processo de posicionamento de uma fonte sonora, produzindo efeitos sonoros tridimensionais e melhorando a sensação de espacialidade [16]. De acordo com esse mesmo trabalho, a criação de efeitos tridimensionais e a exploração de movimentos que o sistema estéreo oferecia, permitiu que a indústria da música utilizasse esse recurso com grande competência. Um exemplo disso é o álbum "The Wall" do grupo de rock progressivo britânico Pink Floyd.

Para a correta implementação de ilusões espaciais por meio de sons estereofônicos, é preciso entender os mecanismos de senso de direção presente nos humanos. A Figura 2.3 apresenta uma representação de uma cabeça humana "ouvindo" uma fonte sonora localizada à esquerda de sua fronte.

Percebe-se que o sinal percorre caminho maior para atingir o ouvido direito (R) e, portanto, esse ouvido irá receber o sinal em um instante de tempo diferente que o ouvido esquerdo (L) o recebe. Em baixas frequências, isso caracteriza o chamado *phase shift* ou deslocamento de fase e está representado pela Figura 2.4.

Em frequências mais altas, em torno de 10kHz, o espaçamento entre as orelhas tem o tamanho de muitos comprimentos de onda e produz uma confusão com as diferentes fases em que um sinal pode estar (vide Figura 2.5). Isso causa ambiguidades no intervalo de tempo entre os dois sinais.

Tons puros são difíceis de serem localizados pois são formados por uma forma de onda senoidal. Assim, um ciclo é praticamente idêntico ao próximo ciclo, permitindo situações

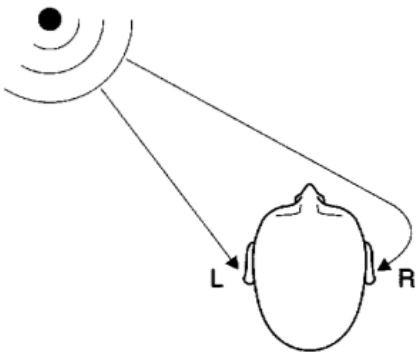

Figura 2.3: Representação do sinal recebido nas duas orelhas (L, esquerda e R, direita), evidenciando a distância que as separa. Fonte: *The Art of Sound Reproduction* [22]

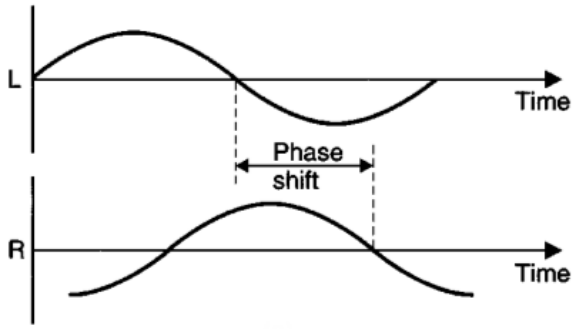

Figura 2.4: Deslocamento de fase provocado pelo espaçamento entre os ouvidos. Fonte: *The Art of Sound Reproduction* [22]

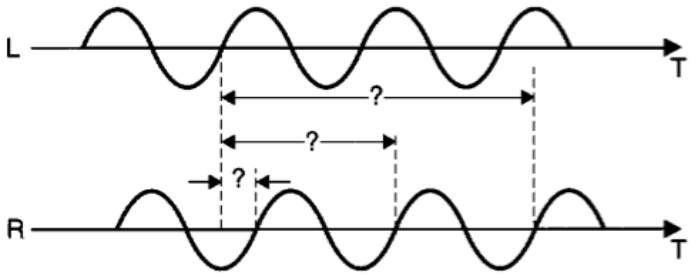

Figura 2.5: Ambiguidade entre fases provocada pelo espaçamento entre os ouvidos. Fonte: *The Art of Sound Reproduction* [22]

de ambiguidade conforme mencionado anteriormente.

Outro fator que influencia na direcionalidade em sistemas estereofônicos é o sombreamento acústico causado pela cabeça. Esse sombreamento sugere que em frequências acima de 1500 Hz o sinal que atinge o ouvido mais distante da fonte sonora é menos intenso que o sinal que atinge o ouvido mais próximo da fonte sonora devido ao sombreamento que a cabeça causa. Dessa forma, tomando como exemplo a Figura 2.6, o sinal que atinge a orelha direita é atenuado pois não é difratado ao redor da cabeça. Já para sinais com frequência abaixo de 1500 Hz, a onda sonora irá difratar ao redor da cabeça de maneira que não ocorram diferenças significativas na intensidade do sinal que atinge os dois ouvidos [22].

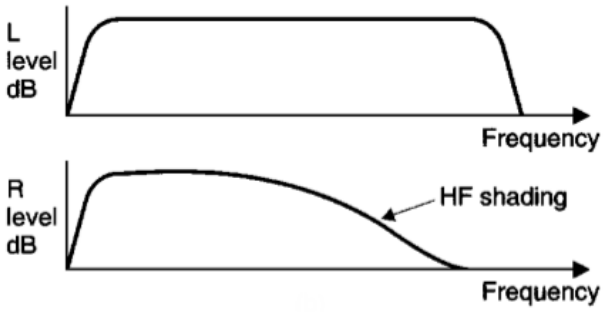

Figura 2.6: Sombreamento causado pela cabeça em frequências altas (*high frequencies*) acima de 1500 Hz. Fonte: *The Art of Sound Reproduction* [22]

A Figura 2.7 mostra como funciona o efeito estereofônico e seu arranjo ótimo, onde os alto-falantes e o ouvinte encontram-se em vértices de um triângulo equilátero. Mostra também que esse mecanismo é capaz de ponderar a intensidade do sinal relativa entre os alto-falantes. Isso é proporcionado por um dispositivo chamado potenciômetro panorâmico. Esse dispositivo produz sinais de saída iguais quando está posicionado ao centro. Caso seja posicionado à esquerda ou à direita, a saída correspondente irá aumentar e a outra diminuirá sua intensidade, movendo o som para o lado escolhido.

2.1.3 Sistemas multicanais - som *surround*

No intuito de criar uma representação espacial do áudio em áreas de reprodução maiores, o esquema de 5.1 canais foi proposto para aplicações sonoras mais complexas, geralmente filmes e TV. Nesse esquema, além dos alto-falantes direito e esquerdo à frente do ouvinte, um terceiro alto-falante central é adicionado e geralmente reproduz áudio de diálogos ou de cenas que estão centralizadas na tela. Conforme é apresentado na Figura 2.8, outros dois alto-falantes (esquerdo e direito) são adicionados atrás da área de audição, possibilitando reproduzir objetos sonoros em qualquer direção [22].

De acordo com o trabalho desenvolvido por Regis Faria [16], o som *surround* (envolvente) consiste em expandir a imagem espacial do áudio monaural ou estéreo para duas ou três dimensões. Esta ideia sugere que o ouvinte faça parte de um campo sonoro que o envolve, o que é bastante interessante em salas de cinema, teatro, *home-theaters* e outras aplicações.

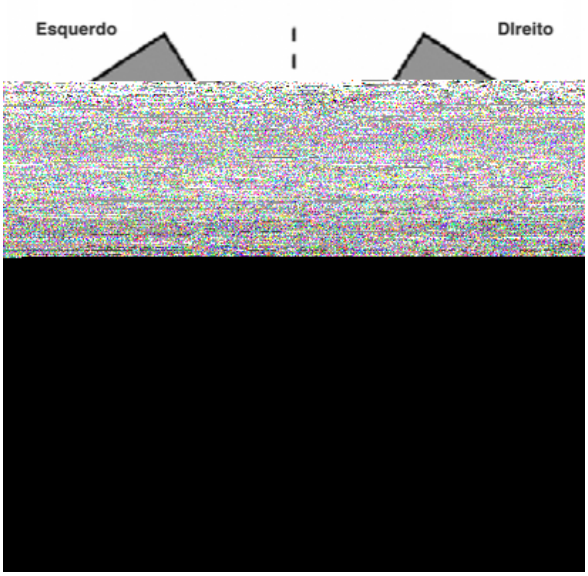

Figura 2.7: Efeito estereofônico. Fonte: *Auralização em Ambientes Audiovisuais Imersivos* [16]

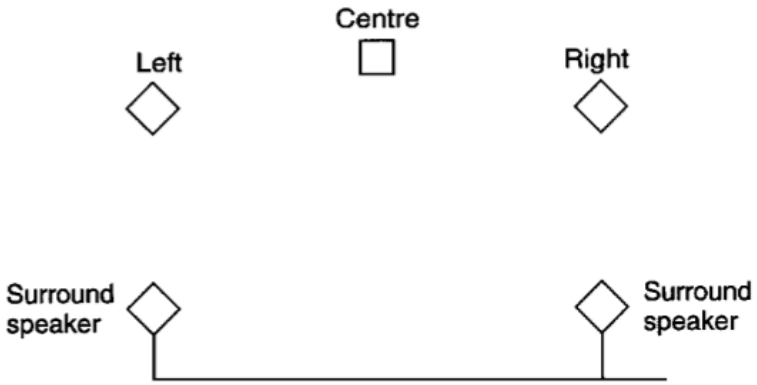

Figura 2.8: Arranjo dos alto-falantes no esquema Dolby surround 5.1, com 5 canais e um *subwoofer*. Fonte: *The Art of Sound Reproduction* [22]

Este sistema vem evoluindo constantemente com a adição de mais canais. Até então já existem arranjos de 3, 4, 5, 6 ou até mesmo 7 ou 8 canais. Este sistema se tornou popular pois, além de serem úteis para criar efeitos de envolvimento e criar imagens sonoras em planos bidimensionais e tridimensionais, foi padronizado.

2.2 Auralização

A auralização refere-se a campos sonoros gerados a partir de um modelo físico ou matemático coerente com o que é esperado na propagação do som em um determinado ambiente acústico. A palavra "auralização" é análoga à palavra "visualização". Enquanto a segunda está relacionada ao sentido da visão, a primeira relaciona-se com a audição. É o termo utilizado para descrever a geração, processamento e reprodução de sons perceptíveis e o seu resultado, tendo como contexto algum problema acústico, sala, edifício ou qualquer outro produto industrial [23].

Segundo o livro *Auralization: Fundamentals of Acoustic, Modelling, Simulation, Algorithms and Acoustical Virtual Reality* [23]:

> Auralização é a técnica de criação de arquivos de som audíveis a partir de informações numéricas simuladas, medidas ou sintetizadas.

O conceito da auralização foi criado em meados de 1929 e estava relacionado à modelagem acústica de campos sonoros em salas. Inicialmente tentou-se processar sinais a partir de modelos projetados em laboratório tais que alguém pudesse ouvir a acústica de uma sala baseada nesse modelo criado aritificialmente. Apenas no começo da década de 90 que a palavra "auralização" foi apresentada. A partir desse instante, foram feitas muitas melhorias nos algoritmos de processamento biaural e das técnicas de reprodução. Atualmente, as técnicas de auralização são amplamente utilizadas em diversos campos tais como: realidade virtual acústica, acústica de salas e veículos [23].

Diversos métodos de auralização permitem a síntese de áudio espacial ou tridimensional. Entretanto esses métodos são classificados em duas categorias principais que dependem do meio de renderização e reprodução utilizados. São elas: biaural e multicanal. As técnicas de auralização biaurais são caracterizadas por utilizarem dois canais (direito e esquerdo) direcionados a cada um dos ouvidos. Já as técnicas de auralização por múltiplos canais, como o próprio nome diz, são caracterizadas pela renderização de sinais para matrizes de alto-falantes, o que a torna adequada para a simulação de ambientes acústicos reais para mais de uma pessoa e que não necessitam utilizar fones de ouvido [16]. A seguir serão apresentadas algumas características desses métodos de auralização.

2.2.1 Biaural

O método biaural, conforme mencionado anteriormente, é caracterizado pela geração de dois canais de som, um para cada ouvido. Esses dois canais contém informações tais que o aparelho auditivo consegue identificar a direção de uma fonte sonora.

A característica de auralização do som pode ser percebida por seres humanos pois estes são capazes de localizar uma fonte sonora devido a variações no sinal que é percebido. Essas variações são causadas por reflexões e difrações do som na cabeça, tronco (principalmente ombros) e ouvido externo. Isto se deve também, em grande parte, à anatomia

de sua cabeça e ao posicionamento dos aparelhos auditivos: os ouvidos. Da mesma forma que percebe-se o mundo ao redor em três dimensões, devido ao espaçamento dos órgãos que permite enxergar, os olhos, o sinal recebido por cada um dos ouvidos dá a sensação de localização de uma determinada fonte sonora no espaço tridimensional. É a chamada audição biauricular, também conhecida como biaural. Nos dois casos, dos olhos e ouvidos, o cérebro processa as informações vindas desses receptores e proporciona a sensação de profundidade tanto visual quanto auditiva.

Há muito se estuda a influência destes parâmetros no mecanismo de localização de fontes sonoras. No final do século XIX, o físico britânico Lord Rayleigh constatou que a cabeça causava um sombreamento acústico em sinais com frequência acima de um certo valor. Ele verificou que esta sombra provoca uma diferença nos níveis de pressão sonora interaural e determinou que este era o parâmetro básico para a localização lateral de sons de alta frequência. No caso de sinais de frequências mais baixas, Raylegh demonstrou que os seres humanos são sensíveis a diferenças de fases de um sinal. Desta forma, a diferença de tempo interaural também passou a ser um parâmetro utilizado para a localização lateral do som. Estes estudos ficaram conhecidos como *Teoria Duplex* [24] que será apresentada com mais detalhes nessa mesma seção.

O processo de localização de fontes sonoras depende de três parâmetros principais [18]:

- **Azimute:** ângulo horizontal entre a fonte sonora e o centro da cabeça.
- **Elevação:** ângulo vertical entre a fonte sonora e o centro da cabeça [25].
- **Distância:** distância entre a fonte sonora e a cabeça quando a fonte está estática e também chamada de **Velocidade** quando a fonte sonora está em movimento.

Basicamente, percebe-se o posicionamento de uma fonte sonora no espaço tridimensional ao redor devido à diferença de tempo e de intensidade que o sinal atinge cada um dos ouvidos. Essa diferença de tempo é explicada, em termos simples, pelo fato do som se propagar a uma determinada velocidade em algum meio[1] e percorrer distâncias distintas até atingir a entrada de cada canal auditivo. Assim, o intervalo de tempo percorrido pelo som a partir da fonte sonora até os ouvidos pode não ser igual para cada ouvido devido às suas diferenças de posição. Caso a fonte sonora esteja em uma posição cuja a distância para o ouvido direito e esquerdo sejam iguais, então é provável que o intervalo de tempo que o som leva para atingir os canais auditivos sejam iguais também. Esta característica é conhecida como diferença de tempo interaural, do inglês *Interaural Time Difference* (ITD) [1, 24]. Outra propriedade que age na percepção da localização de uma fonte sonora é a diferença de nível interaural, do inglês *Interaural Level Difference* (ILD) [1, 24], e está associada ao nível de pressão sonora, ou seja, à intensidade com que o som atinge cada um dos ouvidos. Além disso existem outros fatores que afetam na percepção do som como, por exemplo, as variações do sinal causadas por reflexões e difrações deste na cabeça, tronco (principalmente nos ombros) e ouvido externo do receptor.

As variações de azimute podem ser verificadas por meio da ILD e ITD. Já as variações na distância podem ser verificadas por meio da perda de sinais de alta frequência e de detalhes do som com o aumento da distância entre o ouvinte e a fonte sonora quando esta for estática. No caso da fonte sonora estar em movimento, verifica-se a velocidade e as mudanças de frequência com base nos conceitos do efeito Doppler [18].

[1] Considere o ar como meio de propagação padrão adotado neste trabalho.

A elevação pode ser calculada por meio da verificação das alterações no espectro causadas pelas reflexões e difrações do som na cabeça, tronco e ouvidos externos do ouvinte. Isto pode ser feito por meio da filtragem do sinal de entrada (filtragem esta configurada de acordo com a posição que deseja-se simular). A resposta de frequência obtida como resultado da aplicação do filtro que simula as mudanças no espectro causada pelos fenômenos de reflexão e difração do som é chamada de HRTF, função de transferência relacionada à cabeça [18], que será detalhada no Capítulo 4. A síntese de som tridimensional que utiliza esse mecanismo usa técnicas de filtragem linear em que pares de funções de transferência (para os canais direito e esquerdo), que reproduzem as características auditivas de uma fonte sonora localizada em uma determinada posição do espaço tridimensional ao redor do ouvinte, são aplicados sobre o sinal de áudio que será filtrado.

Teoria Duplex

De acordo com o estudo realizado por Lord Rayleigh [26, 27], a *Teoria Duplex* é um modelo utilizado para estimar a localização de uma fonte sonora tendo como base dois parâmetros biaurais: a diferença de tempo interaural (ITD) e a diferença de intensidade interaural (ILD ou *Interaural Intensity Difference* (IID)), também chamada de diferença de nível de pressão sonora interaural (ILD).

Conforme mencionado anteriormente, a ITD e ILD são parâmetros importantes na percepção do som em uma localização específica do espaço no plano horizontal, ou seja, no plano onde é possível distinguir o som à direita e à esquerda. Neste modelo, mediante a emissão de tons puros com frequência abaixo de 1500 Hz, a diferença de fase do sinal sonoro nas duas orelhas (ITD) é proporcional ao deslocamento lateral da fonte sonora [26].

Com uma frequência próxima de 1500 Hz, o comprimento de onda do sinal é aproximadamente do mesmo tamanho do diâmetro da cabeça (cerca de 23 cm. Vide fórmula 2.1). Desta forma, é possível que o valor da ITD corresponda a mais de um único azimute, o que é indesejável pois permite a presença de resultados ambíguos. Para frequências acima deste valor, a cabeça provoca uma sombra acústica no ouvido mais distante da fonte sonora de forma que a energia que este ouvido capta seja menor que a energia recebida pelo ouvido mais próximo da origem. Assim, é interessante calcular, para frequências acima de 1500 Hz, a diferença de nível sonoro interaural (diferença de intensidade do som) [26].

$$\lambda = \frac{c}{f} \left| \begin{array}{ll} \lambda & \text{comprimento da onda sonora} \\ \text{c} & \text{velocidade do som no ar (343 m/s a } 20^{\text{o}}\text{C)} \\ \text{f} & \text{frequência da onda} \end{array} \right. \tag{2.1}$$

Apesar de sua simplicidade, a teoria duplex não é suficiente para localizar corretamente pontos no espaço tridimensional pois ela é capaz de identificar apenas o deslocamento horizontal, o azimute. Assim, não é possível identificar variações na elevação e na distância de uma fonte sonora.

De acordo com esta teoria, existe uma situação chamada "cone de confusão" em que todos os pontos dentro de um cone de vértice na entrada do canal auditivo compartilham os mesmos valores de ITD e ILD. Logo, eles são indistinguíveis.

No plano mediano (vertical) a localização se dá por meio das alterações no espectro provocadas pelas reflexões e difrações do sinal sonoro no tronco, cabeça e orelhas [26] pelas quais as respostas de frequência são representadas pelas HRTFs.

2.2.2 Sistemas multicanais

Este trabalho mantém o foco apenas no sistema de auralização biaural por meio de HRTFs. Para fins informativos e, tomando como referências a dissertação de Regis Faria [16] e o livro [23], esta seção lista alguns métodos de auralização multicanais.

Sistema Vetorial de Panorama por Amplitude (SVPA)

Do inglês, *Vector-based Amplitude Panning*, é um método que não limita o uso de alto-falantes e que permite o deslocamento de fontes sonoras virtuais e seu posicionamento no espaço. Utiliza a técnica de panorama por amplitude em uma combinação de alto-falantes em localizações arbitrárias. Vários sistemas de realidade virtual acústica utilizam esse mecanismo por ser simples em termos teóricos, pelo baixo custo técnico e pela simplicidade no posicionamento dos alto-falantes [16, 23].

Ambisonics

O Ambisonics é um método criado na década de 70 e trata-se de um padrão de gravação de quatro canais. Os canais geralmente representam esquerda-direita (X), frente-trás (Y), em cima-em baixo (Z) e um canal mono (W) onidirecional que capta a região esférica ao seu redor. Os quatro canais representam a decomposição em harmônicos esféricos. Esse método tem origem na busca por uma técnica que permita gravar um campo sonoro tridimensional e sua posterior reprodução [16, 23].

Wave Field Synthesis (WFS)

A síntese de campos de onda surgiu por volta de 1988. Ela permite a geração de campos sonoros bidimensionais e tridimensionais que preserva as propriedades temporais e espaciais dentro de uma área cercada por uma matriz de alto-falantes. Os algoritmos matemáticos básicos são um tipo de transformação espacial de Fourier entre os domínios do espaço e o número da onda. Assim, campos de onda complexos são decompostos em ondas elementares [16, 23].

Capítulo 3

Síntese Sonora Tridimensional

Quando se estuda sobre métodos de auralização, o objetivo de tal estudo, grande parte das vezes, está relacionado ao desenvolvimento de algum sistema que permita avaliar subjetivamente o resultado da síntese de áudio biaural. Para entender como funciona a síntese sonora tridimensional é necessário primeiramente compreender os princípios básicos de processamento de sinais para auralização. O livro [23] listado na bibliografia utilizada nesse trabalho é a referência base no desenvolvimento desse capítulo.

3.1 O Princípio da Auralização

No Capítulo 2, o conceito de auralização foi apresentado de maneira abstrata conforme se segue:

> Auralização é a técnica de criação de arquivos de som audíveis a partir de informações numéricas simuladas, medidas ou sintetizadas.

Em termos técnicos, o princípio da auralização inicia-se com a definição e descrição de uma fonte sonora ou um sinal criado ou gravado. Geralmente esse sinal primário é anecóico e deve estar disponível, por exemplo, em escala de amplitude ou em unidades de pressão (intensidade) sonora e então, alimenta um caminho de transmissão. O resultado dessa transmissão pode ser considerado perceptível e pronto para reprodução. Desse modo, por meio de ferramentas de processamento de sinais, a auralização é efetuada com a utilização de funções de transferência mensuradas ou simuladas, interpretadas como filtros. Nesse contexto, a operação de convolução (ver Seção 3.2.6 para mais detalhes) é a base para o processamento e análise de sinais e está relacionada a sistemas lineares que não variam no tempo (*Linear Time-invariant Systems* - Sistemas LTI). A Figura 3.1 é uma representação visual do princípio da auralização.

3.2 Fundamentos de Processamento de Sinais

O estudo dos sistemas de auralização e síntese sonora tridimensional invariavelmente exige o conhecimento de conceitos da área de processamento de sinais digitais. Esta seção tem o objetivo de introduzir estes conceitos de maneira a facilitar o entendimento da aplicação de filtros digitais sobre sinais de áudio.

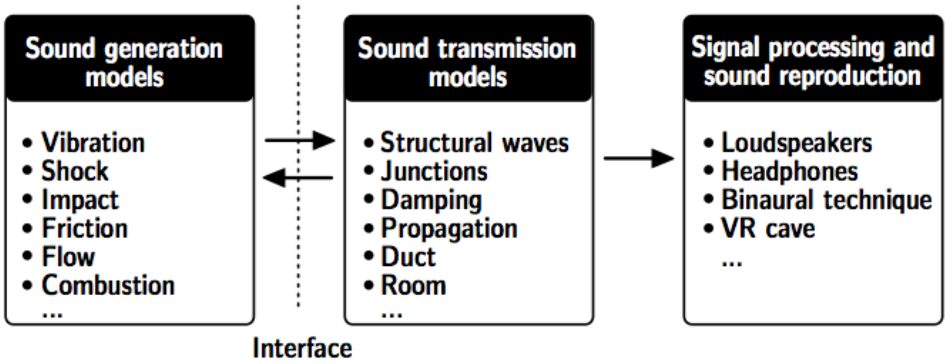

Figura 3.1: Princípio da auralização. Fonte: *Auralization: Fundamentals of Acoustic, Modelling, Simulation, Algorithms and Acoustical Virtual Reality.* [23]

A Seção 3.2.1 mostra a definição de sistemas lineares, conceito fortemente relacionado com o propósito desse capítulo, e algumas de suas propriedades. Já a Seção 3.2.2 apresenta o conceito de resposta impulsiva, também chamada de resposta a impulso. Essas respostas a impulsos são de grande relevância pois é com um par específico delas que se consegue sintetizar áudio com diretividade perceptível. A Seção 3.2.3 apresenta as informações essenciais para o entendimento do conceito de funções de transferência. Como será visto posteriormente, as estruturas que carregam as informações de diretividade percebidas por um indivíduo são um tipo específico de funções de transferência. Por fim, a Seção 3.2.7 apresenta os conceitos básicos sobre filtros digitais.

3.2.1 Sistemas Lineares

Um sistema é um conjunto de conceitos e componentes organizados de tal forma a executar uma ou mais tarefas. Na caracterização destes sistemas, as relações entre entrada e saída são abordadas sem tomar como essencial a natureza física do fenômeno envolvido [28]. Em resumo, é qualquer processo que produz um sinal de saída em resposta a algum sinal de entrada [29].

A Figura 3.2 exemplifica este conceito: a partir de magnitudes providas por uma entrada u, o comportamento do sistema é afetado, modelando o sinal de saída y [30]. A propagação de ondas sonoras pode ser caracterizada como sendo parte de um tipo comum de sistemas [29]: os sistemas lineares.

Figura 3.2: Diagrama de blocos representando um sistema.

Sistemas lineares são sistemas que devem obedecer a duas propriedades matemáticas:

- **Homogeneidade**: significa que uma mudança da amplitude do sinal de entrada $x[n]$ deve provocar uma alteração correspondente na amplitude do sinal de saída $y[n]$. Se $x[n]$ resulta em $y[n]$, então $kx[n]$ deve resultar em $ky[n]$.
- **Aditividade**: tendo dois sinais de entrada $x_1[n]$ e $x_2[n]$ e suas respectivas saídas $y_1[n]$ e $y_2[n]$, o sinal de entrada $x_1[n] + x_2[n]$ deve resultar na saída $y_1[n] + y_2[n]$ para toda entrada possível.

Existe uma terceira propriedade não obrigatória para a linearidade mas que deve ser considerada quando um sistema linear for analisado: **invariância no deslocamento**. Esta propriedade diz que um deslocamento no sinal de entrada resulta em um deslocamento idêntico no sinal de saída. Logo, para qualquer sinal de entrada $x[n]$ que resulta no sinal de saída $y[n]$, então $x[n + s]$ resulta em $y[n + s]$ para uma constante s qualquer [29].

3.2.2 Resposta Impulsiva

Uma resposta a um impulso, do inglês *Impulse Response* (IR), é o sinal de saída de um sistema quando o sinal de entrada é um impulso [29]. Uma função de impulso unitário digital é definida pelo livro de referência [31] como:

$$\delta(n) = \left\{ \begin{array}{ll} 1 & n = 0 \\ 0 & n \neq 0 \end{array} \right\} \tag{3.1}$$

3.2.3 Funções de Transferência

Uma função de transferência é uma representação matemática da relação entre a entrada e saída de um sistema linear (3.2.1).

No livro *Dynamic Modeling and Control of Engineering Systems* [28], os autores afirmam que uma função de transferência $T(s)$ de um sistema linear pode ser encontrada por meio da razão entre a transformada de Laplace (detalhes na Seção 3.2.4) do sinal de saída $y(t)$ e a transformada de Laplace do sinal de entrada $u(t)$ quando estes sinais são funções contínuas no tempo.

$$T(s) = \frac{L\{y(t)\}}{L\{u(t)\}} = \frac{Y(s)}{U(s)} \tag{3.2}$$

Em caso de sistemas discretos, o que geralmente é utilizado em aplicações computacionais, esta função é descrita utilizando a transformada Z conforme equação abaixo:

$$T(z) = \frac{Y(z)}{U(z)} \tag{3.3}$$

3.2.4 Transformada de Laplace

No apêndice 2 do livro [28], os autores afirmam que a transformada de Laplace é um mapeamento entre o domínio do tempo e o domínio da variável complexa s. Este mapeamento é definido pela equação (3.4) abaixo:

$$F(s) = L\{f(t)\} = \int_0^\infty f(t)e^{-st}dt \tag{3.4}$$

onde $s = \sigma + j\omega$ e $f(t)$ é uma função contínua no domínio do tempo igual a 0 para $t < 0$. Além disso, para que exista a transformada de Laplace de $f(t)$, é necessário que a integral definida na equação (3.4) também exista. Logo, deve existir números reais A e b tais que $|f(t)| < Ae^{bt}$.

Este tipo de transformada é geralmente utilizado na resolução de equações diferenciais lineares. Aplicando-se esta transformada, as equações diferenciais envolvendo a variável de tempo t são transformadas em uma equação algébrica no domínio da variável complexa s. Após o cálculo desta equação, que é mais simples de ser resolvida por números complexos do que pelas equações diferenciais, aplica-se a transformada de Laplace inversa no resultado para obtê-lo no domínio do tempo t novamente.

3.2.5 Transformada Z

A transformada Z é uma ferramenta importante na descrição e análise de sistemas digitais. Dada uma função $x[n]$, a transformada Z $X(z)$ ou $Z(x[n])$ desta função é definida como:

$$\begin{aligned} X(z) &= Z(x[n]) \\ &= \sum_{n=0}^{\infty} x[n]z^{-n} \\ &= x[0]z^{-0} + x[1]z^{-1} + x[2]z^{-2} + \ldots \end{aligned} \tag{3.5}$$

onde z é a variável complexa [31]. Nesta equação, todos os valores de z que fazem com que o somatório exista, formam uma região de convergência baseada na sequência $x[n]$ no domínio desta transformada. Outros valores de z que estão fora desta região de convergência provocam uma divergência no somatório. A transformada Z também apresenta as seguintes propriedades:

Tabela 3.1: Propriedades da transformada Z

Propriedade	**Domínio do Tempo**	**Transformada Z**
Linearidade	$ax_1(n) + bx_2(n)$	$aZ(x_1(n)) + bZ(x_2(n))$
Deslocamento	$x(n-m)$	$z^{-m}X(z)$
Convolução linear	$x_1(n) * x_2(n) = \sum_{k=0}^{\infty} x_1(n-k)x_2(k)$	$X_1(z)X_2(z)$

3.2.6 Convolução

A operação de convolução é uma das mais importantes técnicas no campo de processamento de sinais digitais. É uma operação matemática bem como adição e multiplicação, mas que, ao invés de tomar dois números para produzir um terceiro, toma dois sinais para produzir um terceiro sinal. Em sistemas lineares, é utilizada para descrever o relaciona-

mento entre três sinais: o sinal de entrada $x[n]$, a resposta impulsiva $h[n]$ e o sinal de saída $y[n]$.

O sinal de entrada, combinado com a resposta impulsiva é igual ao sinal de saída conforme a equação a seguir:

$$x[n] * h[n] = y[n] \tag{3.6}$$

Convolução Discreta (Simples)

Na referência [31] o autor define a convolução como sendo o seguinte:

$$\begin{aligned} y[n] &= \sum_{k=-\infty}^{\infty} h[k]x[n-k] \\ &= \ldots + h[-1]x[n+1] + h[0]x[n] + h[1]x[n-1] + h[2]x[n-2] + \ldots \end{aligned} \tag{3.7}$$

Neste caso, o sinal de entrada $x[k]$ e o filtro $h[k]$ são armazenados em vetores como sequências temporais. Também, $h[k]$ e $x[k]$ podem ser invertidos de posição sem que o resultado da operação se altere. Esse processo requer $M \cdot N$ multiplicações em ponto flutuante, sendo M o tamanho do sinal de entrada e N o número de coeficientes que compõem o filtro FIR, cujos detalhes serão apresentados na Seção 3.2.7.

Em termos práticos, a operação de convolução pode ser utilizada para impor as características de um sinal sobre outro [8].

Convolução via Transformada de Fourier

A operação de convolução no domínio do tempo pode ser um processo custoso [8]. A convolução via transformada de Fourier utiliza o princípio de que a multiplicação no domínio da frequência corresponde à convolução no domínio do tempo. Assim, o sinal de entrada é transformado no seu correspondente no domínio da frequência e multiplicado pela representação no domínio da frequência do filtro ao qual também foi aplicada a transformada de Fourier. Aplica-se sobre esse resultado a transformada inversa de Fourier para que seja obtido o mesmo resultado representado no domínio do tempo. Segundo o livro *Numerical Recipes in C - The art of Scientific Computing* [32], capítulo 13, onde o autor menciona o uso da transformada rápida de Fourier, do inglês *Fast Fourier Transform* (FFT), para convolução de sequências pequenas de um sinal s com um filtro f qualquer, deve-se primeiro fazer o *padding* (adição de zeros) da menor sequência (geralmente o filtro) para que as duas sequências tenham o mesmo tamanho. A partir daí aplica-se a FFT em s e f, multiplica-se o resultado das duas transformadas elemento a elemento, e aplica-se a FFT inversa sobre esse produto. O resultado dessa operação é o filtro f aplicado ao sinal s.

Utilizar o algoritmo da FFT para o cálculo da transformada discreta de Fourier, do inglês *Discrete Fourier Transform* (DFT), pode tornar a operação de convolução no domínio da frequência mais rápida que a mesma operação feita diretamente no domínio do tempo. Em ambos os casos o resultado final é o mesmo, apenas o número de operações matemáticas se altera devido ao uso de um algoritmo mais eficiente [29].

3.2.7 Filtros digitais

Filtros digitais são utilizados no pós e pré processamento de sinais. Servem como filtros passa-baixa, passa-alta e passa-banda para a separação de sinais que foram combinados e para a restauração de sinais que, por algum motivo, sofreram distorções. No contexto da auralização e reprodução sonora, eles servem de base para a filtragem, convolução e para ajustes de efeitos de áudio. São projetados a partir de combinações de componentes de adição, multiplicação e atraso e são divididos em dois grupos principais: filtros de resposta a impulso infinito, *Infinite Impulse Response* (IIR); e filtros de resposta a impulso finito, *Finite Impulse Response* (FIR).

Os conceitos de filtros digitais são interessantes, úteis e aplicáveis a diversos tipos de mecanismos de auralização. Não há preferência por um ou outro tipo de filtro. Isso depende da aplicação e da implementação do software em questão.

Filtros IIR

Também chamados de filtros recursivos [29], os filtros IIR são caracterizados por proverem uma forma eficiente de alcançar longas respostas a impulsos sem precisar executar uma operação longa de convolução. Além disso, sua resposta impulsiva decai exponencialmente. Esse tipo de filtro realiza a convolução de um sinal com um filtro muito longo, embora apenas alguns coeficientes desse filtro estejam envolvidos na operação.

Filtros FIR

Nesse tipo de filtro, a resposta impulsiva é de tamanho finito. São estáveis pois sua saída depende somente dos dados de entrada. A saída de um sistema linear que não varia no tempo é a convolução do sinal de entrada com os coeficientes do filtro.

Segundo informações do livro [31], um filtro FIR pode ser especificado pelo relacionamento de entrada/saída abaixo:

$$\begin{aligned} y(n) &= \sum_{i=0}^{K} b_i x(n-i) \\ &= b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + \ldots + b_K x(n-K) \end{aligned} \tag{3.8}$$

onde b_i são os coeficientes do filtro FIR e $K+1$ é o tamanho deste filtro. Ao aplicar a transformada Z (detalhes na Seção 3.2.5) em ambos os lados da equação (3.8) e após a divisão por $X(z)$ nos dois lados, obtém-se:

$$\begin{aligned} Y(z) &= b_0 X(z) + b_1 z^{-1} X(z) + \ldots + b_K z^{-K} X(z) \\ &= (b_0 + b_1 z^{-1} + \ldots + b_K z^{-K}) X(z) \\ \frac{Y(z)}{X(z)} &= (b_0 + b_1 z^{-1} + \ldots + b_K z^{-K}) \\ &= H(z) \end{aligned} \tag{3.9}$$

onde $H(z)$ é a função de transferência que representa o filtro FIR.

3.3 Síntese de Áudio Biaural

A síntese de áudio biaural é a chave para muitas técnicas de auralização e sistemas de realidade virtual acústica. Para que seja possível sintetizar áudio de maneira biaural, é necessário ter em mãos o sinal (preferencialmente anecóico ou "seco") de uma fonte sonora e as funções de transferência de um sistema. Desse modo, de acordo com a referência [23], o sinal de saída resultante, ou seja, o sinal biaural será obtido pela convolução entre o sinal de entrada e a função de transferência do sistema.

A operação de convolução pode ser realizada de diferentes maneiras tanto no domínio do tempo, por meio da utilização de filtros FIR (válido apenas para sistemas lineares invariantes no tempo), quanto pela utilização da convolução por FFT. Para sistemas que variam no tempo, o sinal deve ser processado em blocos ou janelas que representam trechos aproximados de invariância no tempo. Nesse caso, deve-se cuidar para que no processamento entre os quadros, o *fading* (desvanecimento) seja utilizado para não criar "cliques" nas mudanças desses quadros.

Esse processo de filtragem realizado pela operação de convolução é a forma com que o sinal de áudio é conectado às interpretações e posicionamentos espaciais. A tarefa básica na criação de um sistema de auralização é posicionar a fonte sonora em um determinado ponto do espaço 3D. A Figura 3.3 resume graficamente como um sistema de síntese de áudio biaural deve ser implementado.

Figura 3.3: Blocos de processamento de sinal para auralização. A fonte sonora (*sound source*) é processada por um sistema de processamento digital de sinais de áudio (*audio DSP system*) e é transmitido para a saída em dois canais, representada pelos fones de ouvido (*audio output*). Fonte: *Auralization: Fundamentals of Acoustic, Modelling, Simulation, Algorithms and Acoustical Virtual Reality* [23]

A partir de um conjunto de HRTFs (vide Capítulo 4 para mais detalhes sobre HRTFs), é possível simular qualquer posição de incidência de um som, quando há a convolução de um sinal de áudio mono $s(t)$ com as funções de transferência relativas à cabeça [23] para cada uma das orelhas, conforme equação a seguir:

$$\begin{aligned} p_{right\ ear}(t) &= s(t) * HRTF_{right\ ear} \\ p_{left\ ear}(t) &= s(t) * HRTF_{left\ ear} \end{aligned} \tag{3.10}$$

O arquivo disponibilizado em [33] também apresenta uma maneira simples de sintetizar o som biaural de forma adequada. Deve-se realizar a convolução do sinal de áudio (de preferência anecóico) com o par apropriado de HRTFs e enviar os resultados para seus respectivos canais de saída, esquerdo e direito de um fone de ouvido. Esta é uma forma bem

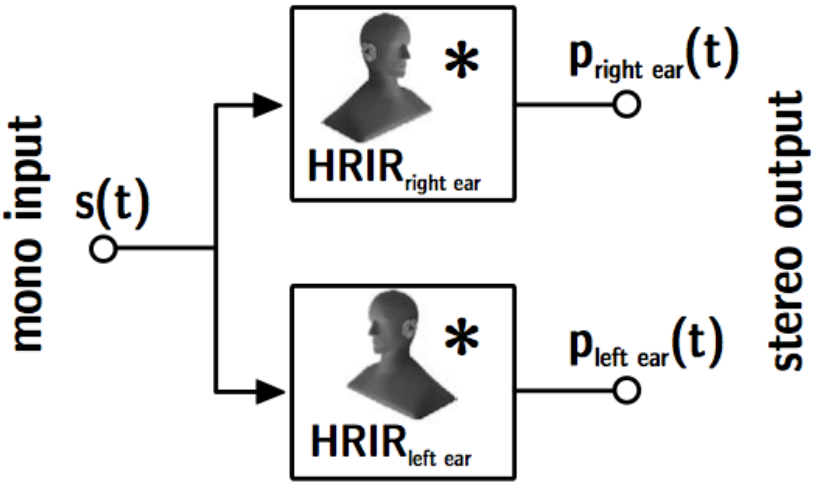

Figura 3.4: Síntese biaural. Convolução do sinal de entrada mono (*mono input*) com o par de HRTFs esquerdo e direito ($HRTF_{left\ ear}$ e $HRTF_{right\ ear}$ respectivamente) e produção da saída em estéreo (*stereo output*). Fonte: *Auralization: Fundamentals of Acoustic, Modelling, Simulation, Algorithms and Acoustical Virtual Reality* [23]

simplista de descrever o processo de síntese sonora biaural e é semelhante ao procedimento apresentado em [23] e representado matematicamente pela equação 3.10 e graficamente pela Figura 3.4. Entretanto, ela oferece apenas uma maneira estática de simular o posicionamento de uma fonte sonora em um determinado ponto no espaço tridimensional. Para se ter a possibilidade de movimentar esse objeto sonoro ao redor do ouvinte, é preciso implementar outro mecanismo de convolução rápida chamado *overlap-save* (vide Seção 3.3.1).

3.3.1 Método *overlap-save*

O método *overlap-save* é um procedimento de convolução em bloco correspondente à implementação da convolução circular de L pontos de uma resposta impulsiva $h[n]$ de P pontos com um segmento $x_r[n]$ de L pontos, identificando a parte da convolução circular que corresponde à convolução linear. Os segmentos resultantes são então agrupados, formando o resultado de saída. Segundo o livro [34], se uma sequência de L pontos é convoluída circularmente com uma sequência de P pontos, onde $P < L$, então os primeiros $(P-1)$ pontos do resultado são incorretos. Já os pontos seguintes são exatamente os mesmos obtidos por uma convolução linear. Segundo essa mesma referência, podemos dividir um sinal $x[n]$ em seções de tamanho L de forma que cada seção de entrada sobrepõe a seção precedente por $(P-1)$ pontos. Define-se as seções como:

$$x_r[n] = x[n + r(L - P + 1) - P + 1], \qquad 0 \leq n \leq N - 1 \tag{3.11}$$

A Figura 3.5 mostra como um sinal $x[n]$ é decomposto por seções de tamanho L. A convolução circular de cada seção com $h[n]$ é denotada por $y_{rp}[n]$. O subscrito p indica que $y_{rp}[n]$ é o resultado da convolução circular com $h[n]$ ainda contendo os pontos que devem ser descartados, conforme apresentados na Figura 3.6.

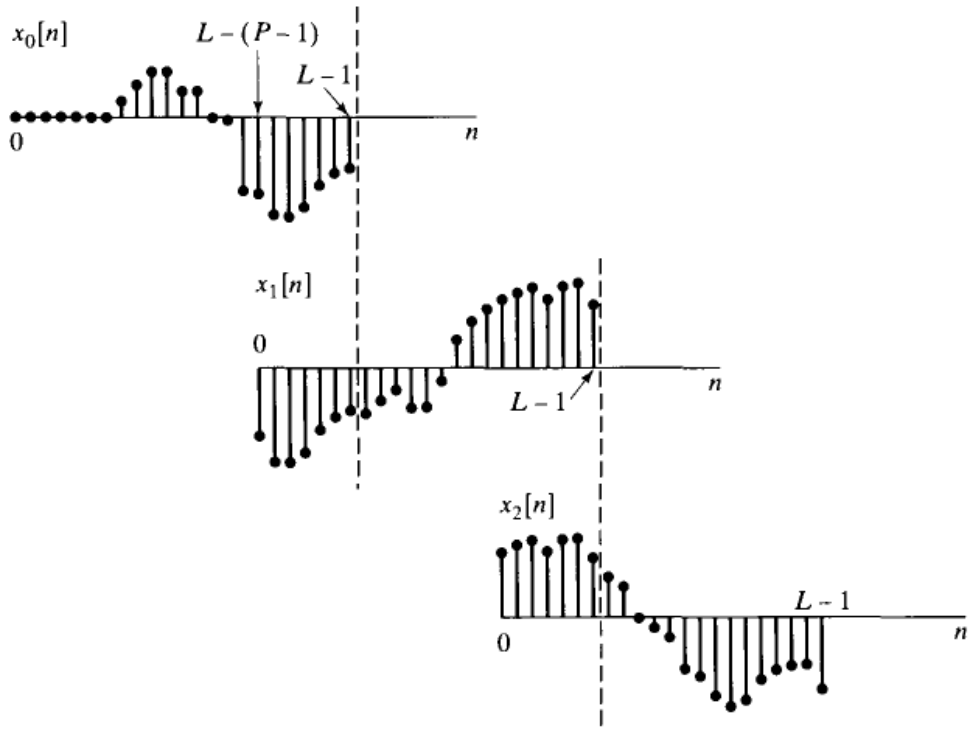

Figura 3.5: Decomposição de $x[n]$ em seções sobrepostas de tamanho L. Fonte: *Discrete-time signal processing* [34].

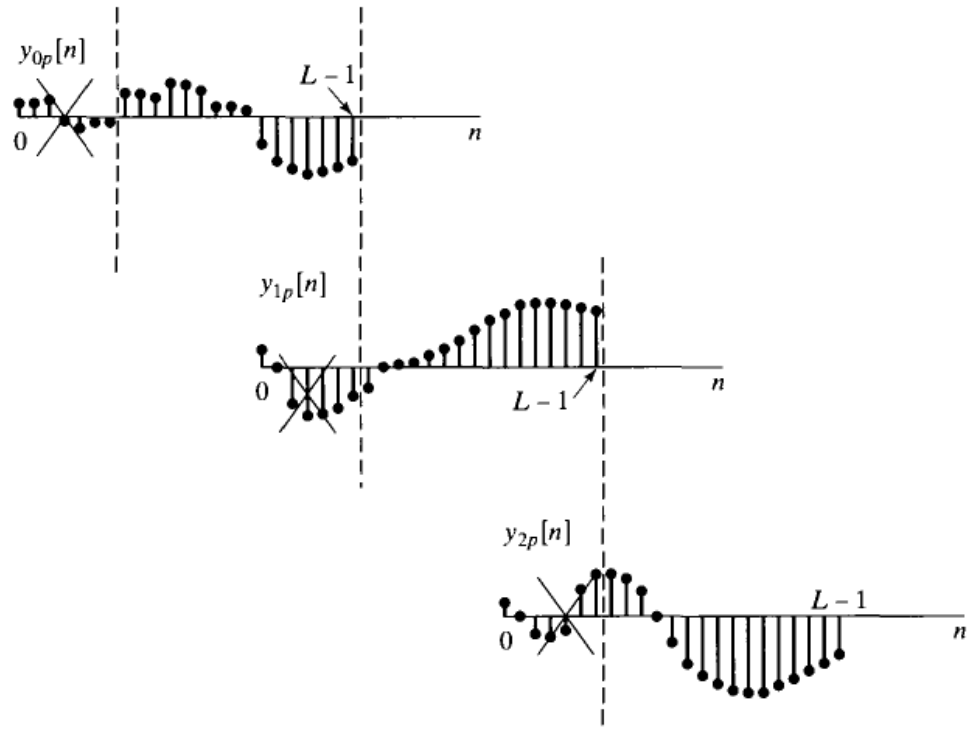

Figura 3.6: Resultado da convolução de cada seção com $h[n]$ com as respectivas partes que são eliminadas. Fonte: *Discrete-time signal processing* [34].

Os pontos que devem ser descartados em cada seção são os pontos referentes ao intervalo $0 \leq n \leq P-2$. O restante das amostras são agrupadas para formar o resultado final.

$$y[n] = \sum_{r=0}^{\infty} y_r[n - r(L-P+1) + P - 1] \tag{3.12}$$

onde

$$y_r[n] = \left\{ \begin{array}{lll} y_{rp}[n] & , & P-1 \leq n \leq N-1 \\ 0 & , & \text{caso contrário} \end{array} \right\} \tag{3.13}$$

Este método é chamado de *overlap-save* pois os segmentos se sobrepõem de certa forma tal que cada seção subsequente consiste de $(L-P+1)$ novos pontos e $(P-1)$ pontos da seção anterior.

3.3.2 Síntese Biaural em Tempo Real

Considerando que os ambientes virtuais que utilizam os sistemas de auralização precisam ser interativos e responder em tempo real às ações de seus usuários, é necessário considerar alguns aspectos da síntese sonora nesse contexto. Algumas condições devem ser levadas em conta para que limites aceitáveis de latência e taxas de atualização sejam estabelecidos.

As mudanças no ambiente promovidas pelo usuário devem ter efeito imediato na sua percepção. A latência deve ser suficientemente pequena para que o ouvinte não se irrite com interrupções e "cliques" no som. Além disso, a taxa de atualização, definida como o tempo em que o sinal de áudio é alterado, recalculado e a saída apresentada ao usuário, deve ser alta o suficiente para garantir a continuidade do som. Em outras palavras, a latência e a taxa de atualização estão associadas, respectivamente, com a capacidade de resposta e a suavidade das transições das funções que são aplicadas ao sinal.

Conforme descrito no livro [23], a convolução em tempo real é um problema especialmente em sistemas de síntese biaural em que muitos canais devem ser processados em paralelo. Em algumas aplicações, as respostas biaurais devem ser alteradas rapidamente sem "cliques" ou transições audíveis. O método de convolução segmentada *overlap-save* apresentado na Seção 3.3.1 é eficiente nesse contexto. Entretanto, depende do tamanho do filtro utilizado.

Capítulo 4

Funções de Transferência Relacionadas à Cabeça

No intuito de definir matematicamente e entender os parâmetros da localização sonora espacial, modelos, medições e simulações são continuamente testados e aprimorados. Tais medições resultam em funções denominadas *Head-related Transfer Functions*, as funções de transferência relacionadas à cabeça, ou simplesmente HRTFs [26].

As HRTFs descrevem as características da filtragem espectral que ocorre no som entre a fonte sonora e o tímpano do ouvinte [35]. Em outras palavras, elas definem como uma onda sonora atinge a entrada do canal auditivo após a reflexão e difração da mesma na cabeça, tronco e no ouvido externo do receptor deste sinal [36]. Por ser dependente de características anatômicas de partes do corpo humano, as HRTFs são individuais, ou seja, cada pessoa possui um par de funções de transferência referentes a um determinado posicionamento entre a fonte sonora e as orelhas [37].

Estas funções são geralmente assumidas como funções lineares e são expressas no domínio da frequência. Por meio da aplicação da transformada inversa de Fourier nesta função, é possível encontrar sua respectiva representação no domínio do tempo como uma resposta a um impulso (*impulse response*). Esta representação no domínio do tempo é chamada de resposta a impulso relacionados à cabeça, HRIR, e pode representar um filtro FIR [38].

Atualmente o comportamento e características das HRTFs são conhecidos e possuem papel fundamental na direcionalidade do som. Utiliza-se o sistema de coordenadas esféricas para referenciar as direções das HRTFs medidas. Essa estrutura em coordenadas esféricas compõe três planos: plano horizontal, plano frontal e plano sagital mediano, conforme apresentado na Figura 4.1. Assim, um ponto P ao redor do ouvinte possui coordenada tal que $P = (r, \theta, \phi)$, onde r é a distância entre a fonte sonora e o centro da cabeça (onde se encontra a origem dos eixos), θ é o ângulo de azimute medido no sentido horário no plano horizontal e ϕ é o ângulo de elevação medido no plano vertical com valores positivos acima do plano horizontal e negativos abaixo deste mesmo plano.

4.1 Medições de HRTFs

As HRTFs (ou suas correspondentes representações no domínio do tempo - HRIRs), são geralmente obtidas por meio de uma técnica muito comum que consiste em medições feitas

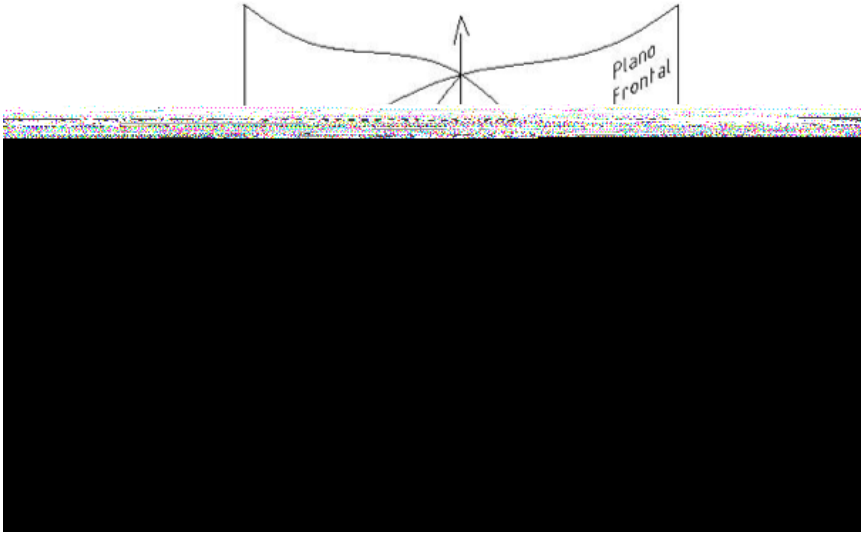

Figura 4.1: Sistema de coordenadas esféricas. Fonte: *Sistema de Auralização Eficiente Utilizando Wavelets* [1]

a partir das respostas a impulsos captadas por microfones posicionados no canal auditivo de uma pessoa (vide Figura 4.2) ou de um manequim *Knowles Electronics Mannequin for Acoustics Research* (KEMAR). O uso deste manequim (ou até mesmo de uma pessoa) é interessante pois permite simular os efeitos de difração e reflexão do som na cabeça e tronco desta pessoa.

O relatório técnico [39], realizado no *Massachusetts Institute of Technology* (MIT) *Media Lab Perceptual Computing*, apresenta uma base de dados com um total de 710 medições em diferentes posições com angulação entre -40° e 90° de elevação e com azimute variando de 0° a 355° em intervalos de 5° (na maioria das elevações). Este relatório informa que as medidas consistem nas respostas a impulsos da orelha esquerda e direita a partir de um alto-falante posicionado a 1,4 metros de distância do manequim, conforme aparato apresentado pelas Figuras 4.3 e 4.4. Neste experimento, a orelha esquerda utilizada no manequim é de formato e tamanho normal. Já a orelha direita possui um tamanho maior. Como este procedimento de medida de HRTFs demanda bastante tempo, recursos e equipamentos, apenas alguns pontos do espaço tridimensional são medidos.

Após submeter o manequim a sons previamente conhecidos, as respostas ao impulso foram obtidas. Nessas medições, para diminuir o tamanho do conjunto de dados sem eliminar informações importantes, decidiu-se descartar as primeiras duzentas amostras de cada resposta a impulso. As 512 amostras seguintes foram consideradas e armazenadas em uma estutura de diretórios cujo nome dos arquivos especificam a localização espacial da HRIR. Outros centros de pesquisa também realizam medições de HRIRs e disponibilizam tais informações para o público que realiza pesquisa nesta área. Como exemplo, o *Listen hrtf database* disponibilizado pelo *Institut de Recherche et Coordination Acoustique/Musique* (IRCAM) [40].

No repositório de HRTFs apresentado em [39], as respostas ao impulso são armazenadas como números inteiros de 16 bits (com sinal), com o bit mais significativo na posição mais baixa. O arquivo contendo as informações da HRTF possui nome seguindo o padrão "XEEeAAAa.dat", onde X representa "L" ou "R" para as respostas a impulsos nas orelhas

Figura 4.2: Montagem do microfone. Fonte: *IRCAM. Listen HRTF Database* [40]

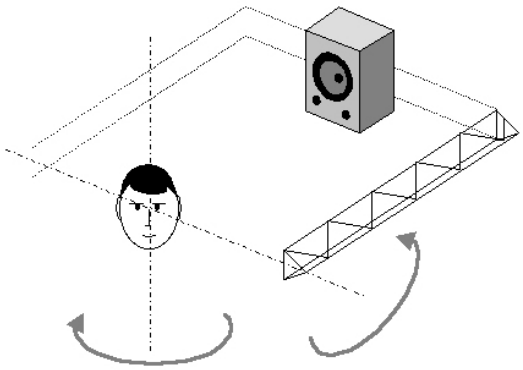

Figura 4.3: Posicionamento da fonte sonora. Fonte: *IRCAM. Listen HRTF Database* [40]

Figura 4.4: Posicionamento da fonte sonora. Fonte: *IRCAM. Listen HRTF Database* [40]

esquerda e direita, EE é o ângulo de elevação em graus (de -40°a 90°) e AAA é o azimute da fonte sonora em graus (de 0°a 355°). Ao utilizar essas medições, aconselha-se selecionar um par de HRTFs simétrico correspondente a uma das orelhas do manequim KEMAR. Por exemplo, utilizar o arquivo "L0e045a.dat" para a orelha esquerda e "L0e315a.dat" para a orelha direita quando a elevação for de 0°e o azimute 45°.

Também neste projeto foi criado um pacote com HRIRs simétricas de tamanho reduzido a partir das respostas a impulsos na orelha esquerda do KEMAR. Cada arquivo contém um par estéreo de respostas de impulsos de 128 amostras, formado por números inteiros de 16 bits dispostos em ordem intercalada em relação ao canal de saída (esquerdo, direito). Cada resposta contendo 128 amostras foi obtida por meio da convolução da HRIR contendo 512 amostras com o filtro inverso de fase mínima do alto-falante utilizado no experimento. A resposta resultante foi obtida a partir da coleta das 128 amostras a partir do índice 26.

No intuito de medir as HRTFs, o artigo [26] executa procedimento semelhante. Microfones são inseridos no ouvido externo do indivíduo e sons de espectro conhecido são executados por um alto-falante localizado em uma posição de azimute θ e elevação ϕ a uma distância pré-determinada da cabeça. Os sons executados podem ser um simples clique ou uma sequência binária pseudo-randômica. Funções de transferência referentes ao aparato de medição, denominadas funções de transferência comuns (*Common Transfer Function* - CTF) são removidas da medição final pois não dizem respeito ao som que deseja-se analisar. O resultado final desta medição é chamado de função de transferência direcional (*Directional Transfer Function* - DTF).

Como o procedimento de medida exige muito tempo, recursos e equipamentos especializados, apenas algumas amostras do espaço tridimensional são tomadas [10]. Em aplicações embarcadas, por exemplo, há também uma limitação em medir e armazenar as HRIRs de todas as posições. Nesses casos, técnicas de interpolação são utilizadas para encontrar HRIRs em posições não medidas a partir de HRIRs vizinhas as quais a medição já foi feita [41]. Tais técnicas são apresentadas com maiores detalhes na Seção 4.2, sobre interpolação de HRTFs.

4.2 Interpolação de HRTFs

No intuito de encontrar HRTFs (ou HRIRs) de posições não medidas, diversos mecanismos de interpolação são estudados e desenvolvidos. Por exemplo: a interpolação direta no domínio da frequência [42] ou no domínio do tempo [43], interpolação espacial [44], dentre outros [41] conforme são apresentados na Seção 4.2.3. Neste trabalho, o foco é dado ao método de interpolação de HRTFs no domínio da transformada wavelet [14] e que está detalhado na Seção 4.3 adiante.

Nesta seção, alguns métodos de interpolação de HRTFs são apresentados e descritos. A Seção 4.2.1 apresenta o método bilinear que caracteriza-se pela escolha dos quatro pontos mais próximos ao ponto P que deseja-se calcular. Já na Seção 4.2.2, o método triangular é apresentado e é semelhante ao bilinear mas utiliza apenas os três pontos mais próximos da HRTF do ponto P que deseja-se calcular.

4.2.1 Método Bilinear

Quando se lida com uma única fonte sonora, uma maneira direta de realizar a interpolação de HRTFs é utilizando o método bilinear. Esse método consiste no cálculo da HRTF de uma determinada posição na esfera de referência ao redor do usuário, tomando como base os quatro pontos mais próximos a esse ponto que deseja-se descobrir e efetuando a média ponderada de suas HRTFs.

O método bilinear pode ser interpretado graficamente por meio da Figura 4.5. Seja θ_{grid} o valor de incremento do azimute e ϕ_{grid} o valor de incremento da elevação. Suponha que um conjunto de HRIRs foram medidas em todos os valores de azimute e elevação seguindo seus respectivos valores de incremento θ_{grid} e ϕ_{grid}. O valor estimado de uma HRIR em uma posição arbitrária (θ, ϕ), conforme mostrado na Figura 4.5 é dado por:

$$\hat{h}(k) = (1 - c_\theta)(1 - c_\phi)h_a(k) + c_\theta(1 - c_\phi)h_b(k) + c_\theta c_\phi h_c(k) + (1 - c_\theta)c_\phi h_d(k) \qquad (4.1)$$

onde $h_a(k)$, $h_b(k)$, $h_c(k)$ e $h_d(k)$ são as HRIRs associadas aos quatro pontos mais próximos do ponto que deseja-se descobrir e os parâmetros c_θ e c_ϕ são definidos como:

$$c_\theta = \frac{C_\theta}{\theta_{grid}} = \frac{\theta \bmod \theta_{grid}}{\theta_{grid}}, \quad c_\phi = \frac{C_\phi}{\phi_{grid}} = \frac{\phi \bmod \phi_{grid}}{\phi_{grid}} \qquad (4.2)$$

em que C_θ e C_ϕ são as posições angulares relativas apresentadas na Figura 4.5 [45, 46].

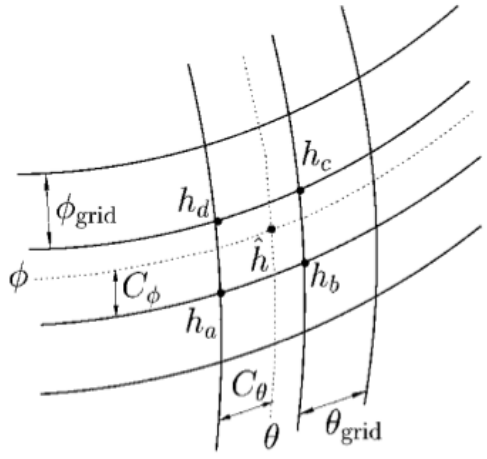

Figura 4.5: Interpretação gráfica do método bilinear. Fonte: *Interpositional Transfer Function for 3D-Sound Generation* [45]

A utilização desse método com o banco de HRIRs disponibilizado pelo MIT [39] não é uma boa opção pois só pode ser aplicada no intervalo de elevação [-20°, 20°] porque o passo de incremento no azimute é igual a 5°. Para elevações abaixo de -20° e acima de 20°, o valor do incremento no azimute (θ_{grid}) varia. Em outras palavras, no intervalo $-20° < \phi < 20°$, o espaçamento entre azimutes é uniforme. Fora desse intervalo esse espaçamento não é uniforme e, portanto, não se aplica ao método bilinear.

Existem dispositivos comerciais que utilizam essa técnica. Segundo Begault [15], o Convolvotron [47] utiliza o método de interpolação com quatro pontos (método bilinear)

para obter a HRTF de uma posição acústica virtual arbitrária. A informação de diretividade consiste de quatro valores de pesos e quatro ponteiros para as HRTFs medidas e armazenadas nesse dispositivo. Esses pesos e ponteiros são utilizados para calcular a HRTF da posição desejada.

4.2.2 Método Triangular

O método triangular, assim como o bilinear (vide Seção 4.2.1), também utiliza a abordagem de ponderações de HRIRs. Entretanto ao invés de quatro pontos, utiliza três posições próximas à posição virtual arbitrária que deseja-se calcular. O método triangular, também conhecido como *IPTF-based*, do inglês *Interpositional Transfer Function* (IPTF), estima a HRIR de um ponto P qualquer dentro de uma região triangular pela ponderação geométrica das HRIRs medidas nos três vértices do triângulo, da seguinte maneira (vide Figura 4.6):

$$h_P(n) = w_A h_A(n) + w_B h_B(n) + w_C h_C(n)$$

onde $h_A(n)$, $h_B(n)$ e $h_C(n)$ são as HRIRs medidas nos pontos A, B e C e w_A, w_B e w_C são

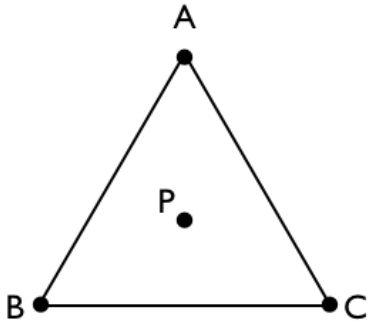

Figura 4.6: Triângulo de interpolação.

os pesos aplicados em cada uma delas e que são calculados da seguinte maneira, conforme Figura 4.7: [14, 45]

$$w_C = \frac{\Delta\phi}{\Delta\phi_{grid}}, \tag{4.3}$$

$$w_B = \frac{\Delta\theta}{\Delta\theta_{grid}}, \tag{4.4}$$

$$w_A + w_B + w_C = 1 \tag{4.5}$$

As distâncias angulares apresentadas na Figura 4.7 são dadas como:

$$\Delta\phi = \phi_P - \phi_A \tag{4.6}$$

$$\Delta\theta = \theta_P - \theta_X \tag{4.7}$$

$$\Delta\theta_A = \theta_P - \theta_A \tag{4.8}$$

$$\Delta\theta_{AC} = \theta_C - \theta_A \tag{4.9}$$

$$\Delta\theta_{grid} = \theta_B - \theta_A \tag{4.10}$$

$$\Delta\phi_{grid} = \phi_C - \phi_A \tag{4.11}$$

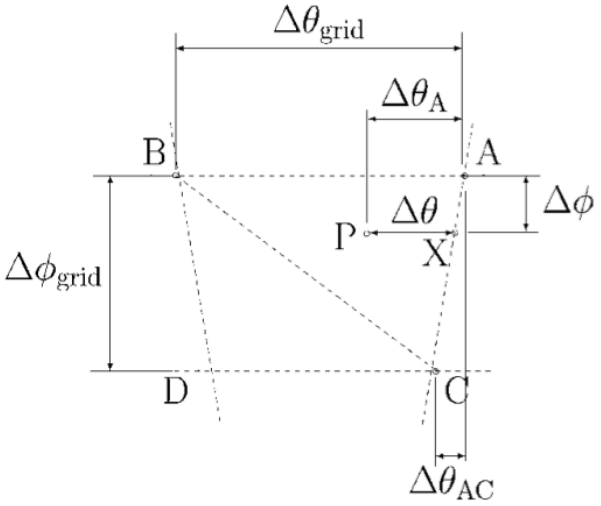

Figura 4.7: Interpretação gráfica do método triangular. Fonte: *Interpositional Transfer Function for 3D-Sound Generation* [45]

Dada uma região triangular escolhida, a HRTF de um ponto P pode ser calculada de dois modos diferentes de acordo com a situação apresentada pela Figura 4.8.

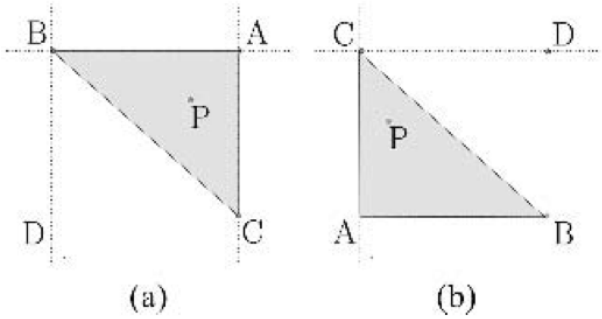

Figura 4.8: Duas posições possíveis de um ponto P em uma região triangular qualquer. Fonte: *Interpositional Transfer Function for 3D-Sound Generation* [45]

Na Figura 4.8, os vértices A e B possuem a mesma elevação. Qualquer ponto P dentro da região sombreada será calculada tomando como base as HRTFs das posições A, B e C.

4.2.3 Outros métodos de interpolação de HRTFs

Com o passar dos anos, técnicas existentes de interpolação de HRTFs vão se desenvolvendo e se aprimorando. Além disso, novos procedimentos são apresentados e geralmente trazem melhorias e avanços aos métodos de interpolação de HRTFs.

Esse trabalho não possui o objetivo de listar os diferentes mecanismos encontrados na literatura e suas particularidades, mas sim, manter documentado que outros métodos existem e que apresentam seus benefícios também. O intuito dessa seção é apenas listar brevemente alguns dos outros métodos de interpolação de HRTFs disponíveis mas que não fazem parte do escopo dessa dissertação. Além disso, é claro, eles serão descritos resumidamente.

Interpolação no domínio da frequência

No artigo [42] os autores comparam quatro modos de interpolação de HRTFs baseado no julgamento de três pessoas. Dentre esses quatro métodos, um trata da interpolação linear simples de HRTFs medidas por meio de mecanismos semelhantes ao apresentado na Seção 4.1.

Apresentado na Seção 4.2.2 anterior, o método triangular pode ser caracterizado como sendo no domínio da frequência pois utiliza as IPTFs [48].

Interpolação no domínio do tempo

O artigo [43] apresenta uma avaliação da capacidade humana de percepção da direcionalidade de fontes sonoras virtuais sintetizadas com a ajuda de respostas a impulsos relativas à cabeça (HRIR), que é a representação das HRTFs no domínio do tempo. A biblioteca de HRIRs utilizada não possui essas respostas a impulsos para todas as posições de uma esfera virtual envolvendo o ouvinte. Para isso, os autores desse artigo expandem essa biblioteca por meio da interpolação linear das HRIRs já existentes. Segundo esse mesmo artigo, a interpolação linear simples é apropriada quando o intervalo entre azimutes é menor que 10°.

Apresentado na Seção 4.2.1 anterior, o método bilinear também pode ser caracterizado como método de interpolação no domínio do tempo.

Interpolação espacial

O trabalho descrito pelo artigo [44] mostra uma outra alternativa aos métodos de interpolação tradicionais. Neste caso, o método de interpolação de HRTFs é derivado de observações feitas a partir das *Spacial Frequency Response Surface*s (SFRSs), as superfícies de respostas espaciais de frequências. Conforme apresentado pelo artigo, essas superfícies provêem uma representação visual imediata e muito compacta das informações de uma HRTF. Eles descrevem quanto de energia cada uma das orelhas recebe em função da localização espacial.

Nesse algoritmo, ao analisar as SFRSs para um determinado valor de frequência, as informações das HRTFs contém características similares que são contínuas e que mudam muito pouco em função da localização espacial. Esse algoritmo de interpolação é uma aplicação imediata e quantitativa das SFRSs e se mostrou convincente ao produzir HRTFs correspondentes à posições espaciais arbitrárias.

Métodos de interpolação mais recentes

Em 2010, foi apresentado um artigo intitulado *Structured IIR Models for HRTF Interpolation* [48] que apresenta um modelo IIR estruturado para representar os filtros HRTF de um banco de dados de HRTFs, adequado para a sua interpolação. Segundo esse artigo, a ideia de representar os filtros HRTF como filtros IIR é de reduzir o esforço computacional necessário para cada simulação. Uma possível aplicação desse método é a criação de instalações sonoras onde o usuário, dispondo de fones de ouvido, caminha ao redor de sensores de posicionamento tais que fazem com que ele receba o som de acordo com sua posição e orientação de sua cabeça.

Em um trabalho ainda mais recente [49], os autores apresentam um método de interpolação que combina as respostas a impulsos da cabeça e da sala em que a fonte sonora se encontra. Essa combinação, denominada de CHRIR foi proposta como uma alternativa ao uso separado da HRIR e RIR (*Room Impulse Response*). Esse artigo apresenta uma técnica de interpolação no domínio da frequência que interpola a diferença de nível interaural (ILD) e a diferença de tempo interaural (ITD) para cada componente de frequência do espectro.

4.3 Interpolação de HRTFs com a Transformada Wavelet

No artigo [14], foi proposto um novo método de interpolação de HRTFs baseado no método Triangular utilizando a transformada wavelet e seguida de filtros esparsos para representar as HRIRs. Neste método, os coeficientes wavelet da HRIR são processados em cada sub-banda m pelos respectivos filtros esparsos $G_m(z)$, conforme a Figura 4.9. Os coeficientes esparsos de cada sub-banda e posição no triângulo de interpolação (vide Figura 4.6) são multiplicados pelos seus respectivos pesos, conforme equação 4.12:

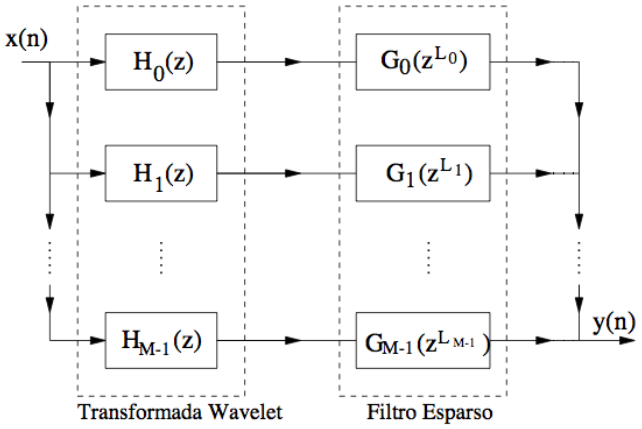

Figura 4.9: Modelagem de HRIRs usando transformada wavelet e filtros esparsos. Fonte: *Interpolação de HRTFs através de Wavelets* [13].

$$G_{P,k}(z) = w_{A,k}G_{A,k}(z) + w_{B,k}G_{B,k}(z) + w_{C,k}G_{C,k}(z) \tag{4.12}$$

onde, k representa a sub-banda wavelet, $G_{P,k}(z)$ são os coeficientes para o k-ésimo sub-filtro no ponto P e $w_{A,k}$, $w_{B,k}$ e $w_{C,k}$ são os pesos dos seus respectivos filtros esparsos $G_{A,k}(z)$, $G_{B,k}(z)$ e $G_{C,k}(z)$.

Para calcular os pesos de cada sub-banda, pode-se utilizar um método de otimização linear através de mínimos quadrados [14]. Isto é feito a partir de posições P conhecidas. Entretanto, ainda assim não é possível calcular os pesos para direções desconhecidas. Para isso, é necessário obter uma função que interpole os pesos de posições desconhecidas a partir dos pesos obtidos para direções já conhecidas [13].

Neste método, dentre outros procedimentos, os autores removem alguns coeficientes de baixa energia visando reduzir o custo computacional. Esta redução no número de coeficientes visa diminuir o volume de dados utilizados nas operações de convolução necessárias tanto no processo de interpolação das HRTFs quanto no processo de síntese do sinal. Entretanto, isto pode ser contornado por meio do uso de mecanismos capazes de processar em tempo real volumes imensos de dados que uma CPU comum não é capaz de processar: as unidades de processamento gráfico (GPUs), que são objeto de estudo deste trabalho.

Com o objetivo de entender de forma mais apurada os detalhes da implementação realizada em [14], seguem alguns conceitos fundamentais no desenvolvimento deste trabalho. São apresentados os bancos de filtros e sua relação com as transformadas wavelet; a maneira como os filtros esparsos são implementados, suas características e a representação polifásica. Também são mostrados aqui os mecanismos utilizados para o cálculo dos pesos associados a cada matriz de coeficientes esparsos.

4.3.1 Bancos de Filtros e as Transformadas Wavelet

Um banco de filtros é um conjunto de filtros digitais que podem ter uma entrada ou saída em comum. Um sistema que possui um banco de filtros de análise (banco de filtros $H_k(z)$) divide um sinal $x(n)$ em M sinais $x_k(n)$ para cada sub-banda k. Um banco de filtros de síntese (banco de filtros $F_k(z)$) combina os sinais das M sub-bandas para produzir um único sinal $\hat{x}(n)$ [50]. A possibilidade de reconstrução perfeita do sinal propiciada pela combinação desses dois bancos de filtros mediante certas condições (biortogonalidade, atraso inserido na reconstrução) permite a criação de aplicações como, por exemplo, compressão de sinais. A reconstrução perfeita será abordada posteriormente.

Em [12], o autor afirma que uma trasformada wavelet discreta pode ser obtida por meio da escolha apropriada, no domínio Z, dos coeficientes dos filtros protótipos $H^0(z)$ (passa-baixas) e $H^1(z)$ (passa-altas) e estruturando-os em árvore da mesma forma que a Figura 4.10. Dessa forma, existe uma relação muito próxima das wavelets com banco de filtros e essa relação está na árvore logarítmica de filtros representada pela Figura 4.10 [51]. As wavelets herdam as propriedades de ortogonalidade e biortogonalidade dos bancos de filtros e, devido ao processo repetido de re-escala, elas decompõem o sinal em sub-bandas de tamanho não uniforme [51]. Assim, a transformada wavelet decompõe o sinal em sub-bandas estreitas para as baixas frequências e largas para as altas frequências. Essa análise aproxima-se do comportamento da percepção auditiva humana.

A estrutura em árvore da Figura 4.10 pode ser representada pelo banco de análise com os filtros $H_m(z)$ e seus correspondentes fatores de decimação L_m conforme apresentado

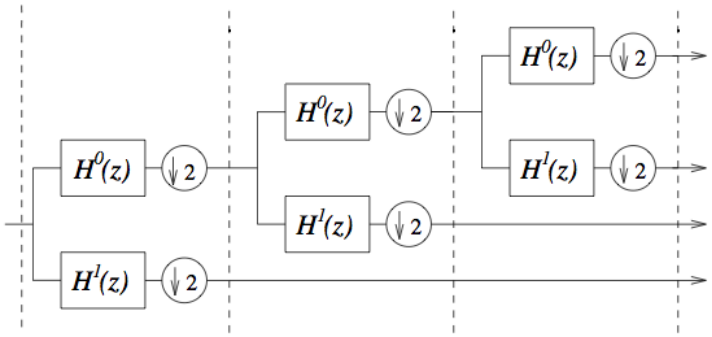

Figura 4.10: Cascateamento dos filtros protótipos $H^0(z)$ (passa-baixa) e $H^1(z)$ (passa-alta). Fonte: *An Efficient Wavelet-Based Model for Auralization* [12].

na Figura 4.11. Nela, os filtros de análise relacionam-se com o filtros protótipos $H^0(z)$ e $H^1(z)$ da seguinte maneira:

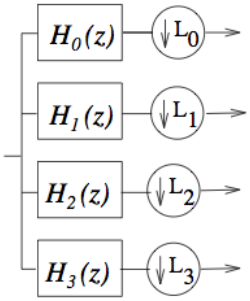

Figura 4.11: Banco de filtros de análise equivalente ao cascateamento dos filtros protótipos apresentado na Figura 4.10. Fonte: *An Efficient Wavelet-Based Model for Auralization* [12].

$$H_0(z) = \prod_{k=0}^{J-1} H^0(z^{2^k}) \tag{4.13}$$

e

$$H_m(z) = H^1(z^{2^{J-m}}) \prod_{k=0}^{J-1-m} H^0(z^{2^k}) \tag{4.14}$$

onde J é o número de estágios da decomposição, $M = J + 1$ é o número de sub-bandas e $m = 1, \ldots, M-1$. L_m é o fator de esparsidade do sub-filtro m e corresponde aos fatores de decimação do banco de filtro de análise $H_m(z)$. L_m é calculado conforme a fórmula 4.15.

$$L_m = \left\{ \begin{array}{ll} 2^J & , m = 0 \\ 2^{J-m+1} & , m = 1, \ldots, J \end{array} \right\} \tag{4.15}$$

Os bancos de filtros de análise e síntese, quando associados, como dito anteriormente, permitem a reconstrução perfeita do sinal original porém com a adição de um atraso l: $\hat{x}(n) = x(n-l)$. Em um banco de filtros de dois canais H_0 (passa-baixas) e H_1 (passa-

altas), as respostas a estes filtros tendem a se sobrepor, gerando o efeito de *aliasing* nos canais, mas também é possível acontecer distorções de amplitude e de fase. Desta forma, para garantir a reconstrução perfeita do sinal, os filtros de síntese F_0 e F_1 devem ser cuidadosamente escolhidos para cancelar os erros e distorções causados pelo banco de filtros de análise. Em outras palavras, os filtros de síntese devem estar associados de alguma maneira aos filtros de análise H_0 e H_1 [51].

Para garantir a reconstrução perfeita de um sinal, as escolhas dos coeficientes dos filtros passa-baixas (H_0) e passa-altas (H_1) devem estar relacionadas. Historicamente os coeficientes do filtro H_0 são escolhidos e os coeficientes do filtro H_1 são obtidos a partir de H_0. Segundo [51], a melhor escolha para tal é a inversão alternada dos coeficientes de H_0. Assim, $H_1(z) = -z^{-N}H_0(-z^{-1})$, que leva à criação de um banco de filtros ortogonal, onde $N+1$ é o número de coeficientes de H_0. É nesse ponto que a família de wavelets Daubechies são utilizadas pois se encaixam neste padrão.

As escolhas dos filtros de síntese devem obedecer as equações de *anti-aliasing* conforme apresentado na equação 4.16.

$$F_0(z) = H_1(-z) \quad e \quad F_1(z) = -H_0(-z) \tag{4.16}$$

Essa equação está relacionada com o fato de que um banco de filtros de dois canais permite a reconstrução perfeita quando as equações 4.17 e 4.18 são satisfeitas. A demonstração para essas equações pode ser encontrada em [51].

$$F_0(z)H_0(z) + F_1(z)H_1(z) = 2z^{-l} \tag{4.17}$$

$$F_0(z)H_0(-z) + F_1(z)H_1(-z) = 0 \tag{4.18}$$

Wavelets

O termo *wavelet* vem do inglês *wave*, que significa onda. Uma onda é definida como uma função oscilante no tempo tal como uma senóide [52]. Uma wavelet é uma "onda pequena" que possui a característica oscilante das ondas mas, além disso, permite a análise simultânea no tempo e na frequência, com base em fundamentos matemáticos flexíveis. No caso das ondas, a análise de Fourier possibilita apenas a análise na frequência. As Figuras 4.12 e 4.13 apresentam graficamente cada um desses termos.

Um sinal $f(t)$ pode ser analizado, descrito ou processado, quando é expresso por uma decomposição linear [52]:

$$f(t) = \sum_l a_l \psi_l(t) \tag{4.19}$$

onde l é um índice inteiro infinito ou não, a_l são os coeficientes de expansão e $\psi_l(t)$ é chamado de conjunto de expansão. Se a expansão é única, o conjunto é chamado de base (*basis*) para uma classe de funções. Se essa base é ortogonal,

$$\langle \psi_k(t), \psi_l(t) \rangle = \int \psi_k(t)\psi_l(t)dt = 0 \qquad k \neq l \tag{4.20}$$

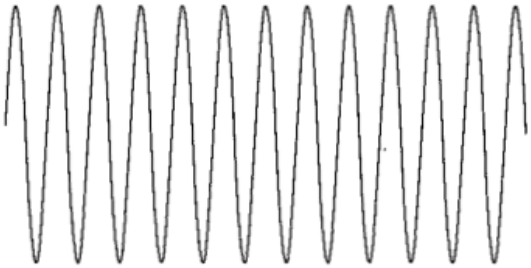

Figura 4.12: Onda senoidal com amplitude constante oscilando no tempo t tal que $-\infty \leq t \leq \infty$ e tendo, portanto, energia infinita. Fonte: *Introduction to Wavelets and Wavelets Transforms* [52].

Figura 4.13: Wavelet contendo sua energia concentrada ao redor de um ponto. Fonte: *Introduction to Wavelets and Wavelets Transforms* [52].

então os coeficientes podem ser calculados pelo produto interno:

$$a_k = \langle f(t), \psi_k(t) \rangle = \int f(t)\psi_k(t)dt \tag{4.21}$$

Para a expansão wavelet, um sistema de dois parâmetros é construído tal que:

$$f(t) = \sum_k \sum_l a_{j,k}\psi_{j,k}(t) \tag{4.22}$$

onde j, k são índices e $\psi_{j,k}(t)$ são as funções de expansão wavelet que geralmente formam uma base ortogonal. O conjunto $a_{j,k}$ são os coeficientes discretos da série de wavelet.

A Figura 4.14 abaixo, semelhante à Figura 4.10, mostra uma estrutura de três escalas wavelet com um banco de filtros de duas bandas. O primeiro estágio divide o espectro em bandas passa-baixas e passa-altas, em outras palavras, os coeficientes de escala e os coeficientes wavelet respectivamente. O segundo estágio divide então o resultado do primeiro passa-baixas em outras duas bandas. Em suma, o primeiro estágio divide o espectro em duas partes iguais. O segundo estágio divide o espectro da parte baixa em dois quartos e, assim, sucessivamente conforme Figura 4.15.

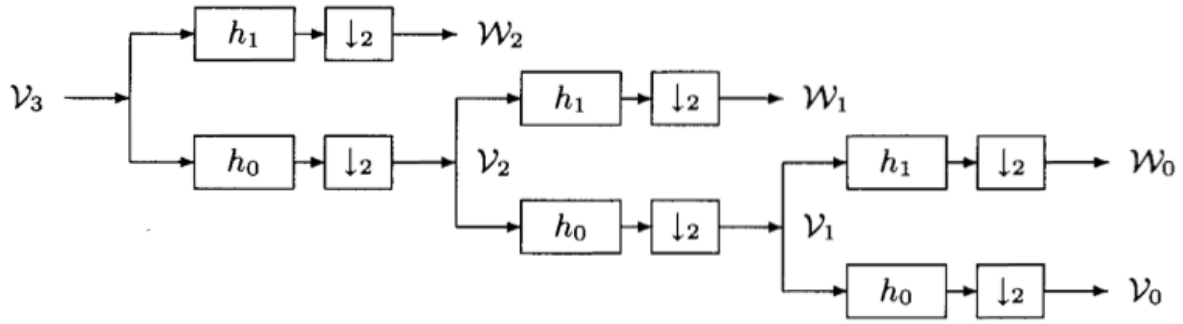

Figura 4.14: Árvore de análise de duas bandas contendo três estágios de escala. Fonte: *Introduction to Wavelets and Wavelets Transforms* [52].

Na Figura 4.15, ω é a frequência de uma senóide em radianos por segundo e $H(\omega)$ é a resposta em frequência de um filtro digital, dado pela *Discrete-time Fourier Transform* (DTFT) de seus coeficientes $h(n)$ conforme equação abaixo:

$$H(\omega) = \sum_{n=-\infty}^{\infty} h(n)e^{i\omega n} \tag{4.23}$$

É interessante notar que a escala musical define as oitavas de maneira similar e os ouvidos respondem às frequências de maneira semelhante à logarítmica conforme apresentado na Figura 4.15.

4.3.2 Filtros Esparsos

Filtros esparsos são filtros FIR em que os coeficientes diferentes de zero são separados por alguns zeros de acordo com um padrão específico. No contexto deste trabalho o fator

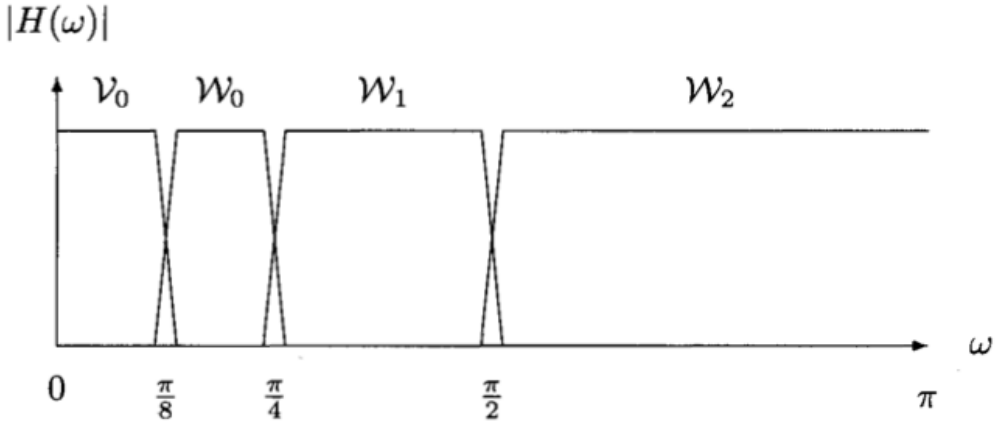

Figura 4.15: Bandas de frequência da árvore de análise da Figura 4.14. Fonte: *Introduction to Wavelets and Wavelets Transforms* [52].

de esparsidade L_m é utilizado, gerando assim, filtros com coeficientes diferentes de zero separados por $L_m - 1$ zeros.

Tais filtros são utilizados para modelar os coeficientes das HRIRs e serem aplicados nas saídas da transformada wavelet conforme apresentado na Figura 4.9. A estrutura mostrada nesta figura, é capaz de modelar qualquer sistema FIR. Como as HRIRs possuem decaimento finito, elas também podem ser consideradas como sistemas FIR [12].

De acordo com [11], considerando um banco de filtros de dois canais e expressando as funções de transferência dos filtros de análise como:

$$H_i(z) = H_{i,0}(z^2) + z^{-1}H_{i,1}(z^2) \quad , i = 0, 1 \tag{4.24}$$

e definindo $\mathbf{H}_p(z) = [H_{i,j}(z)]$ como a matriz polifásica de tipo 1 do banco de análise, a função de transferência implementada pela estrutura da Figura 4.9 com apenas dois canais é dada por:

$$R(z) = \begin{bmatrix} G_0(z^2) & G_1(z^2) \end{bmatrix} \mathbf{H}_p(z^2) \begin{bmatrix} 1 \\ z^{-1} \end{bmatrix} \tag{4.25}$$

No intuito de identificar um sistema FIR desconhecido por meio dos coeficientes dos sub-filtros $G_0(z^2)$ e $G_1(z^2)$, escreve-se a função de transferência desse sistema desconhecido da seguinte maneira:

$$P(z) = \begin{bmatrix} P_0(z^2) & P_1(z^2) \end{bmatrix} \begin{bmatrix} 1 \\ z^{-1} \end{bmatrix} \tag{4.26}$$

onde $P_0(z^2)$ e $P_1(z^2)$ são os componentes polifásicos do tipo 1 de $P(z)$. No caso desta função de transferência possuir M coeficientes polifásicos, ela pode ser representada pela Figura 4.16 [53]. A partir das equações anteriores, é possível verificar que a estrutura de dois canais $R(z)$ modela o sistema FIR desconhecido $P(z)$ quando

$$\begin{bmatrix} G_0(z^2) & G_1(z^2) \end{bmatrix} \mathbf{H}_p(z^2) = \begin{bmatrix} P_0(z^2) & P_1(z^2) \end{bmatrix} \tag{4.27}$$

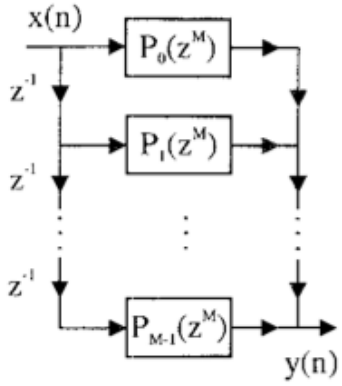

Figura 4.16: Representação da função de transferência com M componentes polifásicos. Fonte: *New Results on Adaptative Filtering Using Filter Banks* [53].

Por meio dos filtros FIR causais $G_0(z^2)$ e $G_1(z^2)$, não é possível encontrar esta igualdade. Entretanto, para $\mathbf{F}_p(z^2)$ tal que,

$$\mathbf{F}_p(z^2)\mathbf{H}_p(z^2) = z^{-\Delta}\mathbf{I} \tag{4.28}$$

onde $\mathbf{H}_p(z^2)$ e $\mathbf{F}_p(z^2)$ são as matrizes polifásicas (com atraso [51]) dos bancos de filtro de análise e síntese de um sistema de reconstrução perfeita respectivamente, então,

$$\left[\begin{array}{cc} G_0(z^2) & G_1(z^2) \end{array}\right] = \left[\begin{array}{cc} P_0(z^2) & P_1(z^2) \end{array}\right] \mathbf{F}_p(z^2) \tag{4.29}$$

Assim, a função de transferência da estutura de dois canais será $P(z)z^{-\Delta}$ e esta estrutura de dois canais implementa corretamente qualquer sistema FIR $P(z)$ mas com um acréscimo de atraso constante de Δ amostras no sinal [11].

No caso de haver M componentes polifásicos, ao incluir uma matriz polifásica $\mathbf{H}_p(z^M)$, seguida de outra matriz polifásica $\mathbf{F}_p(z^M)$, como mostrado na Figura 4.17, a função de transferência desse sistema não é alterada, exceto pela inclusão do atraso Δ [53]. Estes atrasos dependem do comprimento dos filtros protótipos e do número de estágios da árvore de filtros.

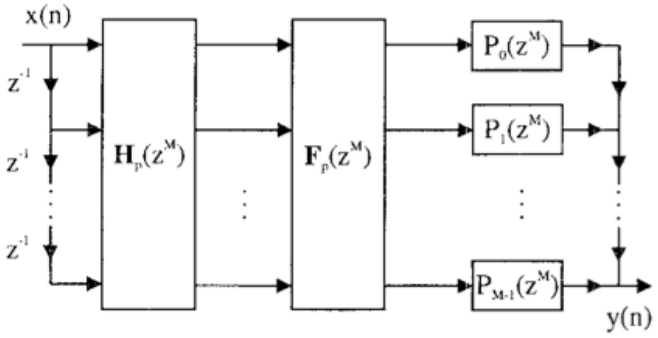

Figura 4.17: Representação da função de transferência com atraso. Fonte: *New Results on Adaptative Filtering Using Filter Banks* [53].

A resposta impulsiva resultante desse sistema pode ser obtida pela soma das convoluções dos filtros de análise com os coeficientes esparsos [12].

$$r(n) = \sum_{m=0}^{M-1} h_m(n) * g_m(n) \tag{4.30}$$

A representação polifásica

Para explicar a ideia principal da representação polifásica, [50] considera-se um filtro $H(z) = \sum_{n=-\infty}^{\infty} h(n)z^{-n}$ e separa-se os coeficientes pares de $h(n)$ dos coeficientes ímpares conforme a equação abaixo. Isto gera duas fases: a par e a ímpar.

$$H(z) = \sum_{n=-\infty}^{\infty} h(2n)z^{-2n} + z^{-1} \sum_{n=-\infty}^{\infty} h(2n+1)z^{-2n} \tag{4.31}$$

É possível escrever $H(z)$ como:

$$H(z) = E_0(z^2) + z^{-1}E_1(z^2) \tag{4.32}$$

tal que,

$$E_0(z) = \sum_{n=-\infty}^{\infty} h(2n)z^{-n}, \quad E_1(z) = \sum_{n=-\infty}^{\infty} h(2n+1)z^{-n} \tag{4.33}$$

De maneira geral, tem-se o seguinte:

$$\begin{aligned} H(z) \;=\; & \sum_{n=-\infty}^{\infty} h(nM)z^{-nM} \\ & + z^{-1} \sum_{n=-\infty}^{\infty} h(nM+1)z^{-nM} \\ & \vdots \\ & + z^{-(M-1)} \sum_{n=-\infty}^{\infty} h(nM+M-1)z^{-nM} \end{aligned} \tag{4.34}$$

ou seja,

$$H(z) = \sum_{l=0}^{M-1} z^{-l}E_l(z^M) \quad (polifase \quad de \quad tipo1) \tag{4.35}$$

onde

$$E_l(z) = \sum_{n=-\infty}^{\infty} e_l(n)z^{-n} \tag{4.36}$$

e $e_l(n) \triangleq h(Mn + l)$, para $0 \leq l \leq M-1$. Assim, $H(z)$ é a representação polifásica de tipo 1 e $E_l(z)$ são os componentes polifásicos de $H(z)$. As matrizes polifásicas de H_0 e H_1,

quando combinadas em uma única matriz, são representadas da seguinte maneira [51]:

$$\mathbf{H}_p(z) = \begin{bmatrix} \mathbf{H_0}_p(z) \\ \mathbf{H_1}_p(z) \end{bmatrix} = \begin{bmatrix} H_{0_0}(z) & H_{0_1}(z) \\ H_{1_0}(z) & H_{1_1}(z) \end{bmatrix} = \begin{bmatrix} H_{0_{par}}(z) & H_{0_{impar}}(z) \\ H_{1_{par}}(z) & H_{1_{impar}}(z) \end{bmatrix} \tag{4.37}$$

O exemplo a seguir ilustra, na prática, como os componentes polifásicos são representados. Considere um filtro $H(z) = 1 + 2z^{-1} + 3z^{-2} + 4z^{-3}$. Seus componentes polifásicos desse filtro são: $E_0(z) = 1 + 3z^{-1}$ (pares) e $E_1(z) = 2 + 4z^{-1}$ (ímpares).

4.3.3 Mapeamento dos Pesos

Conforme apresentado na Seção 4.3, existe um valor de peso w que é multiplicado com seus respectivos filtros esparsos para cada sub-banda e posição no triângulo de interpolação. O método utilizado para a obtenção dos pesos de cada sub-banda é o de otimização linear através de mínimos quadrados em pontos em que os coeficientes são conhecidos. Entretanto, esta abordagem não contempla os pesos de posições desconhecidas. Assim, torna-se necessária a criação de uma função de interpolação destes pesos tendo como referência os pesos obtidos nas direções conhecidas. Com isso, é possível reconstruir as HRIRs das posições não conhecidas [13].

Escolheu-se o método de interpolação polinomial *cubic spline* pois, dentre os testes realizados em [13], ele se mostrou mais eficiente. Este método consiste em gerar os pesos de todos os azimutes para cada sub-banda em um ângulo de elevação fixo. Em [14], o autor nomeia este procedimento de *azimuth interpolation*. Para o cálculo dos pesos em elevações intermediárias, que não são múltiplas de 10°, procedimento semelhante pode ser realizado. Para cada sub-banda e uma posição de azimute constante, podem ser calculados os pesos para diferentes ângulos de elevação. Isto é chamado de *elevation interpolation*.

Neste mecanismo de interpolação de HRTFs no domínio da transformada wavelet, a interpolação na elevação acontece da seguinte maneira. Dado um azimute θ, calcula-se a média ponderada dos resultados da interpolação na elevação para as duas elevações mais próximas.

$$\tilde{w}(\theta, \phi) = \alpha \hat{w}(\theta, \phi_L) + (1 - \alpha)\hat{w}(\theta, \phi_H) \tag{4.38}$$

onde $\hat{w}$ é o resultado da interpolação na elevação para o azimute θ e as elevações mais próxima da elevação de ϕ. ϕ_L e ϕ_H são, respectivamente, as elevações mais próximas abaixo e acima da elevação de P. α é calculado conforme equação abaixo:

$$\alpha = \frac{\phi_H - \phi}{\phi_H - \phi_L} \tag{4.39}$$

Capítulo 5

Computação Paralela com o uso de Unidades de Processamento Gráfico

Nos últimos anos, o interesse no desenvolvimento e utilização de unidades de processamento gráfico, GPUs, em projetos que demandam alto poder de processamento cresceu bastante. O rápido aumento de performance juntamente com as melhorias em sua programabilidade associados ao baixo custo, quando comparadas a outras tecnologias, abriu caminhos para que esta se tornasse uma plataforma de pesquisa e desenvolvimento de aplicações intensas computacionalmente, nos mais diversos domínios de aplicação.

O poder de processamento de placas gráficas cresce mais rapidamente que o poder de processamento das unidades centrais de processamento, CPUs, devido a diferenças arquiteturais entre elas. A Figura 5.1 mostra este crescimento. Enquanto placas gráficas superam a marca de 1500 multiplicado por 10^6 operações em ponto flutuante por segundo, *Giga Floating point Operations Per Second* (GFLOPS)[1], as CPUs ainda estão abaixo de 250 GFLOPS.

As CPUs são projetadas para alcançarem melhor performance na execução de códigos sequenciais, tendo que lidar também com *caching* de dados e controle de fluxo (*branch predictions* - previsão de desvios). Já as GPUs são especializadas em computação paralela e projetadas de tal forma que mais transistores são dedicados ao processamento de dados, ao invés de utilizá-los no controle de fluxo e *caching* de informações [54, 55]. A Figura 5.2 apresenta as principais diferenças arquiteturais entre CPU e GPU. Enquanto na primeira o espaço destinado ao controle e a cache de dados é maior que o espaço destinado às unidades lógicas e aritméticas, do inglês *Arithmetic Logic Unit*s (ALUs), na segunda, muito mais espaço é destinado para unidades de processamento de dados do que ao controle de execução e a cache de informações.

A GPU é ideal para problemas em que o mesmo programa pode ser executado paralelamente em vários elementos de dados e que tenha alta densidade aritmética - o número de operações aritméticas deve ser maior que o número de operações em memória. O processamento paralelo de dados permite que aplicações que necessitam processar um grande conjunto de informações sejam executadas mais rapidamente. Desta maneira, o mesmo programa é executado sobre elementos de dados diferentes e não precisa de um mecanismo altamente eficiente de controle de fluxo. Muitos algoritmos fora do contexto de renderização e processamento de imagens são acelerados pelo processamento paralelo

[1] 1500×10^6 operações em ponto flutuante a cada segundo.

Figura 5.1: Operações em ponto flutuante por segundo. Fonte: *NVIDIA CUDA C Programming Guide. Version 4.0* [54].

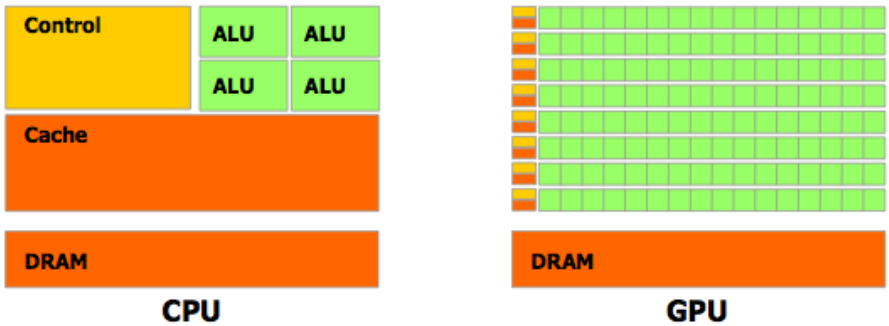

Figura 5.2: Diferenças arquiteturais entre CPU e GPU. Fonte: *NVIDIA CUDA C Programming Guide. Version 4.0* [54].

de dados, dentre eles, os softwares voltados para a resolução de problemas da biologia (biologia computacional), softwares de processamento de sinais, simulações físicas, dentre outros [54].

5.1 Breve Histórico da Computação com GPU

É interessante entender o processo evolutivo desde os primeiros pipelines gráficos até os modelos de programação conhecidos atualmente tais como CUDA™ e OpenCL™. Isso permite o entendimento do contexto que possibilitou a evolução dos dispositivos gráficos e também facilita a criação de melhores projetos para as futuras versões das GPUs.

Durante duas décadas, de 80 e 90, houve a predominância de hardwares gráficos configuráveis mas não programáveis. Neste período, funções fixas (*fixed-functions*) eram utilizadas e as interfaces de programação de aplicações, *Application Programming Interfaces* (APIs), tornaram-se populares. Dentre elas, destacam-se o DirectX™ da Microsoft e o OpenGL™, um padrão aberto bastante popular e suportado por uma grande quantidade de fabricantes. Durante esta fase, a interface para o host recebia comandos, que chamavam funções da API gráfica, e dados gráficos provenientes da CPU. A interface para o host também era responsável por retornar o estado e resultados de volta à CPU após a execução dos comandos. Durante estas duas décadas, cada nova geração de hardwares oferecia novas melhorias aos diferentes estágios do pipeline gráfico. Entretanto, essa evolução não acompanhava a demanda dos programadores por funcionalidades mais sofisticadas [56].

Assim que mecanismos de programação para GPU surgiram, pesquisadores foram atraidos pela possibilidade de utilizar esse hardware gráfico para o processamento e solução de problemas de natureza diferente da gráfica. Como inicialmente as únicas formas de construir aplicações que utilizassem o hardware gráfico era por meio das APIs gráficas padrões como, por exemplo, OpenGL e DirectX, pesquisadores exploraram a computação de propósito geral, ou seja, de propósito que não seja gráfico, utilizando estas APIs. Assim, embora as aplicações que eles desenvolviam em GPU não fossem aplicações gráficas, estas eram progamadas de maneira que se parecesse com uma renderização gráfica tradicional para a GPU [57].

Os pesquisadores perceberam, no começo do século XXI, que as GPUs foram projetadas para produzir uma cor para cada pixel presente na tela como resultado de operações realizadas sobre valores de entrada. Assim, a partir de algumas entradas, a cor de saída era calculada pela GPU. Como o cálculo executado sobre esses parâmetros de entrada eram controlados pelos programadores, estes pesquisadores observaram que os parâmetros de entrada (cores, textura e outros) poderiam ser qualquer tipo de dado. Assim, a cor retornada pela GPU para cada um dos pixels da tela poderia ser interpretada como o resultado de qualquer operação que foi executada em GPU solicitada pelo programador.

Entretanto, apesar de apresentar resultados motivadores, este modelo de programação era muito restrito. Além da incapacidade de cálculo de valores em ponto flutuante de algumas GPUs, quando o programa trazia resultados incorretos ou falhava na sua finalização, não existia um método eficiente de verificar como o código estava sendo executado naquele dispositivo. Além disso, o pesquisador que desejasse criar aplicações para serem executadas em GPU deveria aprender OpenGL ou DirectX, o que era um obstáculo para uma maior aceitação dessa arquitetura [57].

5.2 CUDA

No final do ano de 2006 a NVIDIA lançou a primeira GPU construída com a arquitetura CUDA, incluindo muitos componentes para a computação em GPU de forma a suprir as necessidades e limitações até então presentes no desenvolvimento de aplicações que utilizam este tipo de unidades de processamento. Esta arquitetura traz um novo modelo de programação paralela e um novo conjunto de intruções, o que a faz resolver problemas computacionais complexos de uma maneira mais eficiente que em uma CPU [54, 57].

CUDA permite que desenvolvedores utilizem C como linguagem de programação de alto-nível, o que populariza ainda mais o uso deste hardware gráfico no desenvolvimento de aplicações computacionalmente complexas. Além de C, a arquitetura de computação paralela CUDA também provê interfaces para outras linguagens como FORTRAN [54].

O modelo de programação paralela CUDA foi projetado para permitir que aplicações sejam escaláveis em seu paralelismo independentemente do número de núcleos disponíveis para processamento. As abstrações criadas por meio de um conjunto de extensões da linguagem C para o desenvolvimento em CUDA guiam o programador para a divisão de problemas em sub-problemas de forma que cada um daqueles possam ser executados em paralelo por blocos de threads. Além disso, cada sub-problema pode ser resolvido em paralelo por todas as threads que compõem o bloco. Neste contexto, cada bloco de threads pode ser atribuído a qualquer núcleo de processador que esteja disponível, permitindo que GPUs com mais núcleos executem um programa em menos tempo que uma GPU com menos núcleos executaria o mesmo programa [54].

Além da criação de novas *keywords* adicionadas à linguagem C padrão, da criação de um compilador específico para esta extensão e da construção de um driver de hardware que explora todo o poder computacional massivo da arquitetura CUDA, os usuários não precisam se preocupar mais em conhecer interfaces de programação tais como OpenGL ou DirectX para criar aplicações de propósito geral. Nem é mais preciso que o programa seja forçado a se parecer com uma tarefa gráfica [57].

5.2.1 Modelo de Programação CUDA

O modelo de programação em CUDA exige o conhecimento de alguns conceitos principais e o entendimento de como estes são representados na linguagem de programação C. Segundo [54], os principais conceitos por trás do modelo de programação CUDA são:

Kernels: São funções em C, definidas por meio da utilização da *keyword* `__global__`, que são executadas N vezes em paralelo por N threads CUDA diferentes. Estas funções são executadas em um número de blocos e threads definidos pelo programador utilizando a sintaxe `<<<num_of_blocks, num_of_threads>>>`. Cada thread que executa o kernel possui um identificador único acessível pela variável `threadIdx`. Essa estrutura está representada pela Figura 5.3.

Hierarquia de Threads: O identificador de threads `threadIdx` mencionado anteriormente pode ser interpretado como um vetor de três dimensões, permitindo, assim, que estas threads sejam indexadas em uma, duas e até três dimensões, formando blocos de threads de uma a três dimensões também. O número de threads em cada

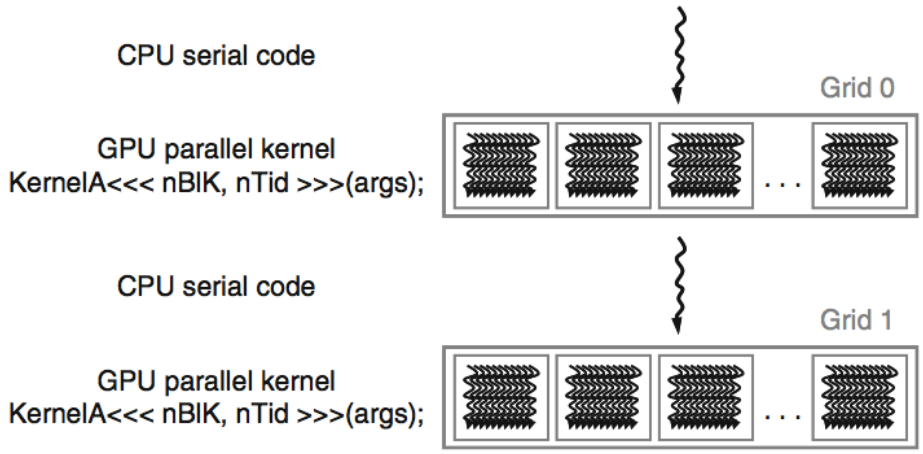

Figura 5.3: Execução de um programa CUDA onde o código serial executado em CPU invoca uma função que será processada em GPU em uma malha de $nBlK$ blocos com $nTid$ threads cada um. Fonte: *Programming Massively Parallel Processors - A Hands-on Approach* [56].

bloco é limitado de acordo com a capacidade da placa de vídeo, assumindo que um bloco encontra-se no mesmo núcleo (*core*) do processador. Em resumo, é possível ter um conjunto de threads organizados em uma, duas ou três dimensões dentro de um bloco de threads que também pode ser disposto de uma a três dimensões. Estes blocos, por conseguinte, são organizados em um *grid* de uma ou duas dimensões conforme apresentado pela Figura 5.4. Da mesma forma que as threads dentro de um bloco possuem uma variável que representa seu índice, um bloco dentro de um grid é identificado pela variável `blockIdx`.

Hierarquia de Memória: Programas desenvolvidos em CUDA podem acessar diferentes espaços de memória em sua execução. Cada thread tem acesso à memória global, possui seu espaço de memória local e blocos possuem espaços de memória compartilhada acessíveis pelas threads dentro deste bloco. Existe mais um espaço de memória utilizado apenas para leitura e voltado para a melhoria de performance em tipos específicos de aplicação: memórias *constant*. A memória *constant* (constante) é geralmente usada para dados que não serão alterados na execução do kernel. Por possuir cache, pode apresentar menor tráfego de informação entre a memória em leituras de dados que já foram realizadas anteriormente [54]. Em outras palavras, a GPU pode ler e escrever dados em registradores e memória local alocados a cada thread, memória compartilhada alocada ao bloco em que se encontram essas threads e memória global alocada para o grid de blocos. A Figura 5.5 apresenta uma visão do modelo de memória de um dispositivo CUDA.

Programação Heterogênea: O modelo de programação em CUDA faz com que um programa seja executado em diferentes ambientes simultaneamente. Isso acontece pois kernels são executados na GPU (*device*) e o resto do programa é executado na CPU (*host*). Esta heterogeneidade não acontece apenas no âmbito do processador, mas também no contexto da memória que é utilizada. Ou seja, na parte do programa

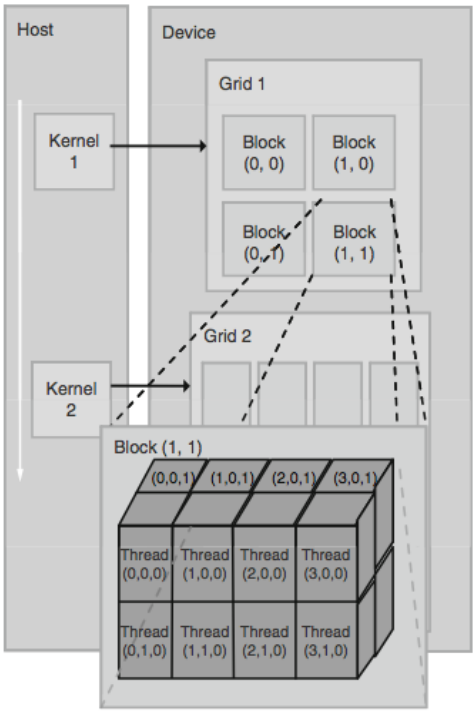

Figura 5.4: Organização de threads em CUDA. Fonte: *Programming Massively Parallel Processors - A Hands-on Approach* [56].

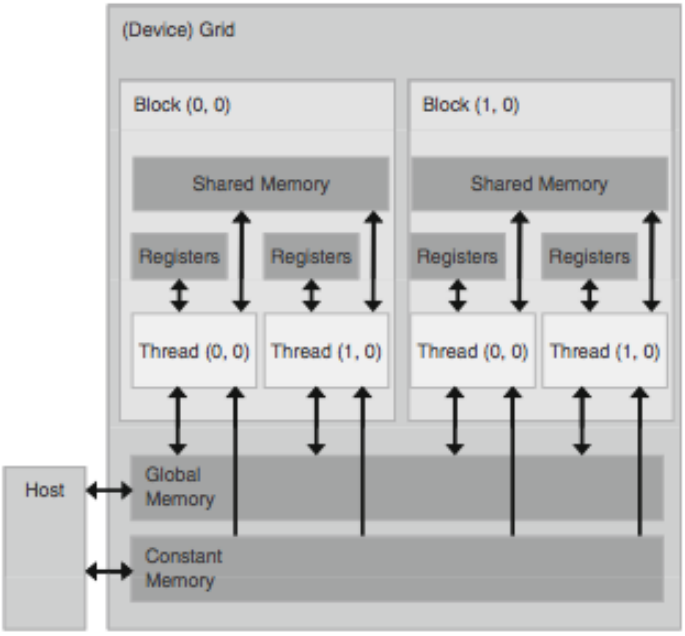

Figura 5.5: Modelo de memória de um dispositivo CUDA. Fonte: *Programming Massively Parallel Processors - A Hands-on Approach* [56].

que é executada na CPU, a memória do *host* é utilizada. Já na execução dos kernels, a memória do *device* é utilizada. A Figura 5.3 apresenta o fluxo de execução de um programa desenvolvido em CUDA e mostra esse conceito de heterogeneidade, em que partes do programa são executadas em CPU e outras partes são executadas em GPU.

Tendo em vista estas particularidades inerentes do modelo de programação da arquitetura CUDA, é preciso estar ciente do quê pode ser otimizado quando se utiliza GPUs para a solução de problemas que demandam poder massivo de processamento. Caso contrário a implementação pode até ser menos eficiente do que uma implementação feita exclusivamente em CPU.

5.3 Processamento de Áudio com GPU

Com o advento das interfaces de programação de propósito geral para GPUs, cada vez mais percebe-se o seu uso na solução de problemas de natureza que não seja gráfica. É comum encontrar aplicações da computação de propósito geral com unidades de processamento gráfico, *General-Purpose Computing on Graphics Processing Unit* (GPGPU), no campo da bio-informática e de simulações físicas. Entretanto, é menos comum encontrar aplicações em GPU voltadas para o processamento de sinais de áudio.

Um artigo apresentado em 2004 [58] descreve os resultados de experimentos realizados com GPU para o processamento de áudio 3D. Aplicações que demandam essa capacidade requerem um número significativo de operações e cálculos numéricos. Segundo esse trabalho, as GPUs podem ser utilizadas para o processamento de áudio com performance comparável ou melhor que a performance do mesmo sistema executado em CPUs de última geração. Além disso, os autores também dizem que as GPUs mais recentes executam outras tarefas de maneira mais eficiente que as CPUs, incluindo o algoritmo da transformada rápida de Fourier - FFT. Entretanto, esse mesmo trabalho alerta sobre possíveis limitações aos modos de acesso à memória e à quantidade de informações que podem ser transmitidas eficientemente do dispositivo para o hardware de áudio localizado no host.

Mais recentemente, em 2009, um trabalho de mestrado desenvolvido na Dinamarca [59] apresentou algoritmos de processamento de áudio implementados em GPU. O objetivo desse trabalho era o de mostrar que alguns algoritmos eram executados mais rapidamente em GPU enquanto outros eram executados de maneira mais rápida em CPU, sempre considerando a implementação de algoritmos otimizados tanto para GPU quanto para CPU. Esse trabalho, segundo o autor, é um estudo da viabilidade e uma análise de eficiência na utilização desses algoritmos em GPU para usos práticos. Para esse trabalho os algoritmos utilizados foram:

- Equalizador
- *Delay* (atraso)
- Reverb
- Compressão
- Visualização de áudio no domínio da frequência

Esse trabalho apresentou os seguintes resultados para cada um dos algoritmos testados:

Equalizador: o algoritmo do equalizador é inerentemente sequencial e não pode ser computado em paralelo. Não há nenhum caso em que o algoritmo executado em GPU terá melhor performance que o mesmo algoritmo executado em CPU. O tempo de execução em GPU é linearmente proporcional ao comprimento do sinal.

Delay: foram testados diferentes métodos que implementam o efeito de atraso de um sinal. Dentre eles, todos os métodos implementados em GPU se mostraram mais lentos que os métodos implementados em CPU. Segundo o autor, isso não se deve a lentidão no processamento, mas sim, ao overhead causado pelas operações em memória devido à transferência de dados do host para a GPU e o resultado da GPU para o host. Em GPUs mais novas e capazes de transmitir dados de forma assíncrona para a memória, pode ser possível que a operação de delay seja mais rápida que a implementação em CPU, dependendo do tamanho do atraso que deseja-se simular. Além disso, a execução desse algoritmo em GPU pode ser interessante quando a CPU está ocupada em outro processamento. Em resumo, o algoritmo de delay não é computacionalmente intenso, portanto, no geral, a GPU não é uma boa opção para executá-lo.

Reverb: o algoritmo para criação do efeito chamado "reverb", de reverberização, pode ser paralelizado completamente. Em todos os casos é mais rápido executar o algoritmo de reverb na GPU.

Compressão: nesse caso, as implementações em CPU são sempre mais rápidas que as implementações em GPU. Mais uma vez, se os recursos da CPU estão escassos, é interessante delegar esse processamento para a GPU. Novamente, há overhead no procedimento de cópia de dados em memória do host para a GPU e do resultado da GPU para o host, que seria beneficiado por cópias assíncronas em memória.

Visualização no domínio da frequência: a performance de visualização de dados no domínio da frequência segue a performance do algoritmo da transformada de Fourier. Assim, essa visualização será significativamente melhor se executada em GPU. Se for preciso apresentar os dados no domínio da frequência, não será preciso copiá-los da GPU para o host pois eles já estarão disponíveis na placa de vídeo.

Em resumo, o trabalho [59] mostra que o Reverb é o único algoritmo que tem clara vantagem sobre os demais quando comparadas suas execuções em GPU e CPU. Para os algoritmos de delay e compressão a execução em GPU é mais lenta que em CPU mas é uma alternativa caso a CPU esteja sobrecarregada com outras tarefas. O equalizador é puramente sequencial e portanto não se beneficia da capacidade computacional e de paralelismo que a GPU oferece.

Ainda mais recente é o trabalho em andamento de André Bianchi da Universidade de São Paulo [60]. Neste trabalho, o autor estuda o processamento de áudio digital em três arquiteturas distintas, dentre elas, a GPU. Além disso o autor apenas apresenta propostas de análise que serão feitas em seu trabalho, mas ainda não possui nenhum resultado relevante.

Capítulo 6

Proposta e Implementação

O desenvolvimento desse trabalho é resultado da busca por uma biblioteca em C capaz de realizar a interpolação de HRTFs e síntese de áudio tridimensional em tempo real. Também é o resultado da busca por uma maneira mais eficiente de se calcular a interpolação de HRTFs no domínio das transformadas wavelet e contribuir para a construção do conhecimento na utilização de GPUs no processamento de sinais de áudio.

Além de todo o processo de fundamentação teórica, em que as possibilidades para a sua execução foram analisadas e avaliadas, também manteve-se um grande interesse na implementação de um artefato de software que pudesse ser utilizado pela comunidade interessada no assunto. Tal artefato servirá para avaliações e testes de posicionamento de fontes sonoras e também pode servir como base para futuros trabalhos na área de auralização e simulação acústica de salas. Com o ferramental necessário, é possível a criação deste artefato: um mecanismo eficiente e intuitivo de mixagem de áudio que possibilite o posicionamento de fontes sonoras em pontos virtuais do espaço tridimensional ao redor do ouvinte de maneira que este tenha a impressão real de que tais fontes se encontram em posições determinadas. O resultado deve ser gerado em dois canais de saída o que, se comparado a sistemas de *home-theater*, por exemplo, que visam reproduzir essa espacialidade do som por meio de mais de dois canais, representa um ganho considerável em espaço de armazenamento do resultado gerado.

Nesse contexto, os maiores desafios deste projeto é implementar o algoritmo de interpolação de HRTFs no domínio da transformada wavelet, e a síntese sonora tridimensional em tempo real em resposta às mudanças de posições da fonte sonora no espaço tridimensional, utilizando a arquitetura de computação paralela da NVIDIA: CUDA. O paralelismo massivo que a GPU proporciona possibilita que um grande volume de dados, como os sinais de áudio, sejam processados em tempo hábil. Para tanto, pretende-se criar uma interface interativa em que o usuário possa posicionar facilmente a fonte sonora em uma posição relativa a uma posição de origem utilizando um dispositivo de entrada como um mouse, por exemplo.

O processo de criação do artefato de software mencionado anteriormente divide-se em três etapas principais que evidenciam o andamento, as tomadas de decisão e os artefatos intermediários que foram produzidos. O ambiente de desenvolvimento utilizado nesse projeto é apresentado na Seção 6.1. Na Seção 6.2, descreve-se o desenvolvimento de um software para a validação do algoritmo e testes iniciais de posicionamento de uma fonte sonora utilizando a linguagem de computação técnica MATLAB. Em seguida, na Seção

6.3, o processo de reescrita do código de interpolação de HRTFs utilizando a linguagem C é apresentado. O desafio nesta etapa, além de ter que adaptar todo o código para outra linguagem, foi o de implementar funções já disponibilizadas pelo conjunto de bibliotecas disponíveis no MATLAB. Finalmente na Seção 6.4 alguns pontos do programa escrito em C foram paralelizados utilizando a tecnologia CUDA, permitindo, assim, maior eficiência na execução de alguns procedimentos. É nessa seção que percebe-se uma das maiores contribuições desse trabalho: o processamento em paralelo por meio do uso de GPU. Os resultados são discutidos na Seção 6.5 onde também são apresentadas algumas informações quanto à qualidade do sinal gerado.

6.1 Ambiente de Desenvolvimento

Foram utilizados para o desenvolvimento desse trabalho, a seguinte configuração de hardware e software:

Hardware e Sistema Operacional	
Sistema Operacional:	Mac OS X Version 10.6.8
Processador:	2.66 GHz Intel Core i7
Memória:	8GB 1067 MHz DDR3

Tabela 6.1: Hardware e Sistema Operacional

NVIDIA - CUDA	
Modelo:	NVIDIA GeForce GT 330M
Memória:	512MB
CUDA Driver Version:	4.0.19
GPU Driver Version:	1.6.40.0 (256.00.35f12)
CUDA Cores:	48

Tabela 6.2: NVIDIA - CUDA

Softwares e bibliotecas	
MATLAB:	7.4
Xcode:	3.2.6
libSDL:	1.2.14
FFTW:	3.3.1

Tabela 6.3: Softwares e bibliotecas utilizados

6.2 Implementação em MATLAB

Após a análise e avaliação dos algoritmos de interpolação de HRTFs que poderiam ser utilizados e a opção pelo trabalho desenvolvido e descrito no artigo publicado no *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics* de 2009 [14], foi feito contato com o autor do artigo para esclarecimento de dúvidas. Além disso, verificou-se com o mesmo, a possibilidade de acesso ao código até então desenvolvido por ele como parte dos testes e validação de sua tese. O autor mostrou-se disposto a ajudar e disponibilizou para estudo uma parte do código escrito na linguagem de computação técnica MATLAB.

O desenvolvimento desse trabalho foi iniciado, então, tomando como base a implementação em MATLAB fornecida pelo professor Dr. Julio Cesar Boscher Torres do Departamento de Expressão Gráfica da Escola Politécnica da Universidade Federal do Rio de Janeiro [1, 11, 12, 13, 14, 61]. A partir do seu trabalho iniciou-se uma série de estudos para entendimento do funcionamento do programa e, também, a escrita de novas funções para reprodução fiel do procedimento de interpolação de HRTFs no domínio da transformada wavelet conforme descrito no artigo [14].

Esta seção mostra como se dá a organização do código enviado pelo autor dos trabalhos [1, 11, 12, 13, 14]. Também apresenta algumas implementações feitas sobre o código original para garantir que o trabalho desenvolvido em [14] fosse corretamente reproduzido. Além disso, são apresentadas algumas modificações propostas pelo autor desse trabalho no intuito de se obter resultados cuja avaliação subjetiva fosse, de certa forma, mais consistente e aceitável.

6.2.1 Organização

O código enviado pelo professor Julio, está organizado conforme Tabela 6.4. Trata-se de um conjunto de funções MATLAB implementadas em arquivos com extensão *.m* que atuam em uníssono para o cálculo das HRIRs. A função principal, chamada de *whrtf*, da maneira que está implementada, é capaz apenas de recuperar os coeficientes esparsos a partir da HRIR de uma posição conhecida no espaço tridimensional e também de reconstruir a HRIR original a partir dos coeficientes esparsos obtidos anteriormente. A HRIR desta posição conhecida é obtida do trabalho desenvolvido e documentado pelo relatório técnico [39].

As funções fornecidas pelo professor Julio, autor do artigo [14] no qual baseia-se o desenvolvimento deste trabalho, oferecem apenas a capacidade de encontrar os coeficientes esparsos de uma HRIR conhecida e a conversão desses coeficientes novamente em HRIR. Para recriar o ambiente descrito nesse trabalho foi necessário implementar outras funções que permitiriam, assim, interpolar HRIRs. Tudo o que foi implementado para complementar o código fornecido pelo professor Julio está descrito na Seção 6.2.2.

6.2.2 Complementação do Código em MATLAB

A seção anterior apresentou a organização do código que serviu como base para a implementação do mecanismo de interpolação de HRIRs no domínio da transformada wavelet. Conforme apresentado, o código disponibilizado não contém toda a implementação

whrtf.m	arquivo principal para o cálculo da HRIR no domínio da transformada wavelet para um determinado azimute e elevação conhecidos.
find_delay.m	função utilizada para o cálculo do atraso ITD do sinal em cada ouvido, tendo como base a HRIR de uma determinada posição.
readhrtf.m	função utilizada para a leitura dos arquivos de HRIR gerados pelo estudo realizado em [39].
shift.m	função que desloca os elementos de um vetor para a direita ou esquerda. É utilizada para simular atrasos no sinal.
coef_spars.m	função utilizada para calcular os coeficientes de uma HRTF conhecida.
dec_poly.m	função utilizada para calcular os coeficientes correspondentes a cada sub-banda da transformada wavelet.
lefilt.m	função que lê os filtros utilizados no processo de decimação polinomial (vide Seção 4.3.1).
calc_delta.m	função que calcula o atraso inserido em cada sub-banda devido à aplicação da transformada wavelet, formada pelo cascateamento de diversos filtros (vide Seção 4.3.1).
cascata.m	função que retorna a estrutura em árvore por meio da decomposição do filtro passa-baixas.
spars.m	função que realiza o espargimento de dados de um vetor de acordo com o coeficiente de esparsidade.
resp_imp.m	função que retorna a resposta impulsiva (HRIR) a partir dos coeficientes esparsos.
polyphase.m	função que retorna as componentes polifásicas do tipo 1 ou 2 de um filtro.
calc_w.m	função que retorna, de acordo com quatro pontos conhecidos (A, B, C e P), os valores de peso dessa configuração.
calculaITD.m	função que calcula o atraso do sinal sonoro em cada orelha, baseando-se no posicionamento da fonte em relação à cabeça.

Tabela 6.4: Estrutura organizacional do código escrito em MATLAB disponibilizado pelo professor Dr. Julio Cesar Boscher Torres do Departamento de Expressão Gráfica da Escola Politécnica da Universidade Federal do Rio de Janeiro [61]

mencionada pelo artigo [14]. Nesta seção, serão apresentados alguns algoritmos e funções adicionados ao projeto no intuito de recriar o mecanismo de interpolação proposto.

Até então tem-se em funcionamento o mecanismo de conversão de HRIRs em coeficientes e o mecanismo de conversão desses coeficientes em HRIRs. Segundo o artigo [14], diferentemente de outros métodos no domínio do tempo, o método de interpolação proposto além de utilizar as HRIRs de três vértices A, B e C de um triângulo, utiliza outra HRIR de um ponto P conhecido interno a este triângulo. Assim, para solucionar o sistema sub-determinado criado pelas equações $G_{P,k}(z) = w_{A,k}G_{A,k}(z)+w_{B,k}G_{B,k}(z)+w_{C,k}G_{C,k}(z)$, onde $G_{P,k}(z)$ representa os coeficientes da sub-banda k no ponto P e $w_{A,k}$, $w_{B,k}$ e $w_{C,k}$ representam os pesos dos filtros esparsos $G_{A,k}(z)$, $G_{B,k}(z)$ e $G_{C,k}(z)$ respectivamente, foi preciso implementar uma função que utilizasse o algoritmo de minimização linear por quadrados mínimos. Assim, encontrou-se os valores dos pesos $w_{A,k}$, $w_{B,k}$ e $w_{C,k}$ para cada sub-banda k [14].

Até esse momento, era possível realizar o cálculo de HRIRs no domínio da transformada wavelet apenas para posições conhecidas, ou seja, posições em que HRIRs já haviam sido medidas (vide trabalho [39]), o que era útil apenas para verificar a validade deste procedimento. Para encontrar os dados de HRIRs de posições não conhecidas, ainda era preciso implementar o mecanismo de interpolação no azimute e na elevação.

O método de interpolação no azimute foi implementado conforme descrito no artigo [14] e apresentado também na Seção 4.3.3. Para uma elevação conhecida ϕ_M, foram calculados todos os pesos referentes aos azimutes conhecidos para esta elevação. Então, supondo a elevação $\phi_M = 0°$, para os azimutes $\theta = \{0°, 5°, \ldots, 355°\}$, cada um dos pesos foram calculados utilizando o algoritmo de minimização linear por quadrados mínimos mencionado anteriormente na Seção 4.3.3 e também armazenados em uma estrutura de dados para acesso posterior. Sobre os valores obtidos, foi aplicado o método de interpolação por spline cúbica e os valores de pesos para azimutes não conhecidos foram, então, obtidos. Desta forma, para três pontos conhecidos A, B e C, e com as informações sobre seus coeficientes esparsos ($G_{A,k}(z)$, $G_{B,k}(z)$ e $G_{C,k}(z)$) e sobre os valores de peso para cada um dos pontos ($w_{A,k}$, $w_{B,k}$ e $w_{C,k}$) em cada uma das sub-bandas k, foi possível encontrar a HRIR correspondente ao ponto P desconhecido.

Com este procedimento não foi possível reproduzir com exatidão os valores dos pesos encontrados pelo autor do artigo [14] em seu trabalho. Analisando o gráfico dos pesos em função do azimute, para a elevação 0° foi obtido um comportamento semelhante ao encontrado pelo autor, entretanto, com valores diferentes. Para sanar essa questão, o autor disponibilizou o código para o cálculo dos pesos conforme especificado no parágrafo anterior. A função *calc_w*, apresentada na Tabela 6.4, calcula os coeficientes esparsos de quatro pontos conhecidos A, B, C e P - onde P é um ponto interno ao triângulo formado pelos vértices A, B e C - e calcula por meio do algoritmo de minimização por quadrados mínimos os valores dos pesos para os três vértices do triângulo e para cada sub-banda.

Para a interpolação na elevação, implementou-se uma função que realiza a média ponderada entre os valores dos pesos em um dado azimute nas elevações mais próximas à elevação procurada, conforme equação 4.38.

Seguindo o caminho natural do desenvolvimento desse algoritmo, foi preciso testar as HRIRs interpoladas sobre sinais reais de áudio para que verificações subjetivas fossem realizadas. Com isso, seria possível ouvir o sinal de áudio posicionado virtualmente em um determinado ponto do espaço tridimensional ao redor do ouvinte. Para isso, construiu-se

uma função principal *main* responsável por validar os parâmetros de entrada tais como elevação e azimute de forma que estes estivessem dentro dos limites de angulação disponíveis. Além disso, esta função recupera os valores de HRIR correspondentes às duas orelhas e aplica cada função aos sinais de áudio, realizando a síntese de áudio biaural e reproduzindo-o pela saída de áudio disponível pelo computador. Esse programa principal está representado de forma simplificada pelo Algoritmo 1.

Algoritmo 1 main

```
1:                                  ▷ elev e azim são os parâmetros de entrada da função
2: → Validação dos valores de entrada elev e azim
3: flip_azim ← 360 − azim
4: IN ← wavread(filename)                          ▷ lê arquivo de áudio no formato .wav
5: Gl ← getSparseCoefficients(elev, azim)          ▷ coeficientes para a orelha esquerda
6: Gr ← getSparseCoefficients(elev, flip_azim)     ▷ coeficientes para a orelha direita
7: L ← conv(IN[1], resp_imp(Gl))                   ▷ convolução do áudio com a HRIR esquerda
8: R ← conv(IN[2], resp_imp(Gr))                   ▷ convolução do áudio com a HRIR direita
9: play([L,R], FS) ▷ função que reproduz o áudio resultante a uma determinada taxa de
   amostragem FS
```

Após alguns testes subjetivos realizados com sinais de áudio reais localizados em uma posição fixa no espaço (utilizando o mecanismo de interpolação implementado), decidiu-se avaliar a qualidade do deslocamento do som pelo espaço. Para isso, foram gerados sinais de áudio que variavam em 1° de uma direção a outra no azimute. Para simular esse deslocamento sem que estalos causados pela mudança de direção pudessem ser percebidos, foi utilizado o método *overlap-save* [13, 34] apresentado na Seção 3.3.1. Esse método foi implementado seguindo as equações 3.11, 3.12 e 3.13 e é representado pelo Algoritmo 2 a seguir.

No Algoritmo 2, divide-se o sinal de entrada X em seções de tamanho L tal que cada seção se sobrepõe à seção anterior por $(P-1)$ pontos. A convolução de cada seção de L pontos com uma sequência de P pontos onde P é menor que L (ou seja, P < L) produz um resultado em que os primeiros $(P-1)$ pontos são incorretos e, por isso, são descartados (vide Seção 3.3.1). Nesse algoritmo, divide-se o sinal de entrada em blocos de acordo com o número de filtros utilizados. Assim, há um bloco de tamanho L para para cada filtro. O bloco de código iniciado pelo comando **for** na linha dez do algoritmo é reponsável por gerar as seções Xr de L pontos cada uma a partir do sinal de entrada X, conforme definido pela equação 3.12. Em seguida, é feito o cálculo da convolução circular de L pontos entre a seção Xr obtida anteriormente e o filtro h de tamanho P. Ao resultado dessa convolução, é adicionado o atraso referente à diferença de tempo interaural do sinal sonoro recebido pelas orelhas. Em seguida, esse resultado é concatenado a uma variável que será utilizada para reproduzir o sinal gerado após a aplicação de todos esses filtros sobre o sinal de entrada X, como resultado da aplicação do método *overlap-save*.

Até então era possível realizar a interpolação de HRIRs utilizando o método proposto pelo artigo [14]. Inclusive com a aplicação do algoritmo de minimização linear por mínimos quadrados para o cálculo dos pesos ótimos aplicado aos coeficientes de cada ponto do triângulo e em cada sub-banda, conforme proposto. Com as HRIRs interpoladas para todos os azimutes em uma determinada elevação, foi feito um teste com um sinal de áudio

Algoritmo 2 overlap_save

```
1:                                  ▷ filters e X são os parâmetros de entrada da função
2: filters contém os filtros, as HRIRs que serão utilizadas nessa função. Cada um com
   tamanho P.
3: X é o sinal de áudio que se quer filtrar.
4: totalLengthX ← length(X)                                   ▷ tamanho do sinal X
5: numOfBlocks ← size(filters)                                ▷ número de blocos
6: L ← ceil(totalLengthX / numOfBlocks)                       ▷ número de amostras por bloco
7: for r ← 1 to numOfBlocks do
8:     h ← filters[r]                  ▷ recupera o filtro que será utilizado nessa iteração
9:     P ← length(h)                                          ▷ P é o tamanho do filtro
10:    for n ← 1 to L do
11:        index ← (n − 1 + (r − 1) ∗ (L − P + 1) − P + 1)
12:        if index ≤ 0 then
13:            Xr(r, n) ← 0                                   ▷ Xr tem tamanho L
14:        else
15:            Xr(r, n) ← X(index)
16:        end if
17:    end for
18:    delta_L ← calculaITD(azim)
19:    cconvL ← cconv(Xr[r], h, L)      ▷ convolução circular de L pontos entre a seção
   Xr[r] e o filtro h
20:    aux ← shift(cconvL, delta_L)
21:    Y ← aux[P:end]                                         ▷ Y é o resultado da execução
22: end for
```

real onde simulou-se o deslocamento da fonte sonora ao redor do ouvinte. Esse deslocamento ocorreu de grau em grau e utilizou o método *overlap-save* descrito anteriormente. As HRIRs e o sinal resultante foram calculados para posteriormente serem processados e enviados aos devidos canais de saída. Vale mencionar que os resultados encontrados não foram convincentes pois a mudança de posição da fonte sonora apresentava variações bruscas na qualidade do sinal e na sensação do posicionamento real dessa fonte. Avaliações subjetivas levaram à conclusão de que: ou o procedimento ou a implementação não estavam corretos.

Visando obter resultados mais satisfatórios e uniformes, optou-se por simplificar a implementação e seguir, em alguns aspectos, o modelo proposto pelo método Bilinear apresentado na Seção 4.2.1. Essa implementação está descrita na Seção 6.2.3.

6.2.3 Mudança no cálculo dos pesos

A alteração no método utilizado para o cálculo dos pesos aplicados aos coeficientes de cada sub-banda tem o objetivo de simplificar a implementação, haja vista que o intuito desse trabalho é mostrar que o algoritmo de interpolação de HRTFs no domínio da transformada wavelet pode ser otimizado por meio do uso de unidades de processamento gráfico. Além disso, o procedimento de cálculo dos valores dos pesos, apesar de ser importante, não é o foco do presente trabalho. Dito isso, esta seção apresenta uma adaptação do método bilinear para o cálculo dos valores dos pesos.

No método bilinear, são escolhidos os quatro pontos mais próximos da posição que deseja-se calcular. As equações 4.1 e 4.2, em conjunto, descrevem como os valores dos pesos são calculados e por conseguinte o valor da HRIR da posição desejada. Nesse mesmo método, tais pesos são aplicados diretamente à HRIR de cada posição.

Nesse trabalho, conforme já mencionado anteriormente, propõe-se para o cálculo dos pesos, o mesmo esquema proposto pelo método bilinear. Entretanto, por influenciar, como um todo, o método de interpolação, é possível considerar que implementa-se aqui o chamado método bilinear no domínio da transformada wavelet: uma mescla dos conceitos apresentados nos trabalhos [45, 46] e [1, 12, 13, 14]. Nesse método, seja um ponto P posicionado em azimute e elevação desconhecidos o qual deseja-se encontrar sua HRIR. Para esse cálculo, seleciona-se os quatro pontos (A, B, C e D) mais próximos do ponto P e aplica-se a transformada wavelet implementada por um banco de filtros de análise conforme apresentado na Seção 4.3.1. Nesse modelo, tem-se três situações possíveis para o posicionamento do ponto P desconhecido:

- Ponto P localizado em uma elevação ϕ constante em que os valores de HRIRs para determinados azimutes são conhecidos. Esta situação está representada graficamente pela Figura 6.1. Para este caso, tem-se, pela equação 4.2, que $c_\theta \neq 0$ e $c_\phi = 0$ e, assim, $\hat{h}(k) = (1 - c_\theta)h_a(k) + c_\theta h_b(k)$ pela equação 4.1.
- Ponto P localizado em um azimute θ constante em que os valores de HRIRs para determinadas elevações são conhecidos. Esta situação é apresentada pela Figura 6.2. Neste caso, tem-se, pelo uso da equação 4.2, que $c_\theta = 0$ e $c_\phi \neq 0$ e, assim, $\hat{h}(k) = (1 - c_\phi)h_a(k) + c_\phi h_d(k)$ pela equação 4.1.

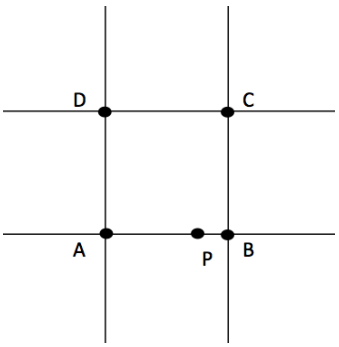

Figura 6.1: Situação em que não há variação na elevação. Portanto basta calcular a ponderação com base na distância angular relativa aos dois pontos mais próximos ao ponto P na mesma elevação.

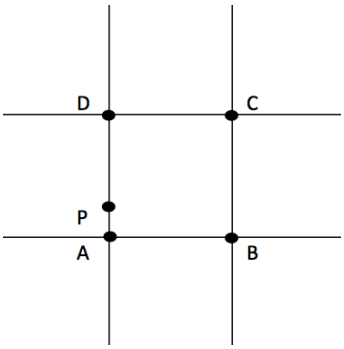

Figura 6.2: Situação em que não há variação no azimute. Portanto basta calcular a ponderação com base na distância angular relativa aos dois pontos mais próximos ao ponto P no mesmo azimute.

- Ponto P localizado em uma posição em que azimute e elevação não são conhecidos. Este caso está representado pela Figura 6.3. Por meio da equação 4.2, $c_\theta \neq 0$ e $c_\phi \neq 0$. Assim, vale a equação 4.1 completa.

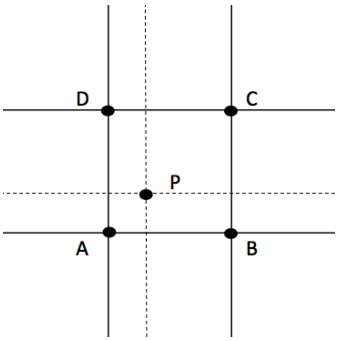

Figura 6.3: Azimute e elevação não são conhecidos.

No método proposto, para cada ponto, os coeficientes das cinco sub-bandas serão calculados conforme equação 6.1.

$$G_{P,k}(z) = w_{A,k}G_{A,k}(z) + w_{B,k}G_{B,k}(z) + w_{C,k}G_{C,k}(z) + w_{D,k}G_{D,k}(z) \tag{6.1}$$

em que, k representa a sub-banda wavelet, $G_{P,k}(z)$ são os coeficientes para o k-ésimo sub-filtro no ponto P e $w_{A,k}$, $w_{B,k}$, $w_{C,k}$ e $w_{D,k}$ são os pesos, aplicados conforme equação 4.1, dos seus respectivos filtros esparsos $G_{A,k}(z)$, $G_{B,k}(z)$, $G_{C,k}(z)$ e $G_{D,k}(z)$.

A implementação desse método híbrido simplificou o cálculo dos valores de pesos e propiciou a interpolação de funções de transferência relacionadas à cabeça. Essas funções, quando aplicadas a um sinal de áudio no processo de síntese biaural, permitiram a transição do objeto sonoro de uma posição à outra com suavidade e sem que interrupções ou outros efeitos indesejados ocorressem. Nessa implementação, a restrição referente ao incremento no azimute no método bilinear foi desconsiderada em favor de uma abordagem mais simples, semelhante à abordagem de interpolação na elevação apresentada na Seção 4.3.3, mas no contexto do azimute.

6.2.4 Testes

Os testes realizados com o código em MATLAB foram conduzidos para validar a implementação do mecanismo de interpolação de HRTFs. Nesse procedimento, todas as HRTFs foram calculadas e posteriormente o sinal de áudio foi divido em 360 blocos, filtrados por cada uma das HRTFs interpoladas. O áudio resultante após a aplicação de um par de HRTFs para cada uma das 360 posições (com elevação constante) ao redor do ouvinte, permite que se perceba a fonte sonora movendo-se suavemente por essa trajetória.

Com a realização desses testes, foi obtido um resultado satisfatório quanto ao posicionamento do objeto sonoro. O algoritmo de interpolação de HRTFs no domínio da transformada wavelet foi validado em relação à sensação de direcionalidade que esse método provoca. Também, validou-se a implementação do método *overlap-save* para a mudança de HRTFs entre diferentes posições.

6.3 Biblioteca em linguagem C

A segunda etapa da implementação desse projeto é caracterizada pela implementação em linguagem C de todo o código produzido na etapa anterior, em MATLAB. Aqui foram utilizadas outras tecnologias tanto para a criação de uma interface gráfica que permitisse ao usuário testar as mudanças de posicionamento e percebê-las em tempo real quanto para a reprodução do áudio biaural. Nesta seção, serão apresentadas algumas dessas tecnologias.

A Seção 6.3.1 apresenta uma descrição mais detalhada de como o código foi escrito em C, abordando os algoritmos fundamentais e descrevendo como o método de interpolação de HRTFs no domínio da transformada wavelet é implementado. Já a Seção 6.3.2 mostra a tecnologia utilizada para a criação de uma interface gráfica capaz de receber ações de um usuário tais que proporcionem alterações em tempo real no sinal que está sendo reproduzido. A Seção 6.3.3 apresenta a biblioteca utilizada para implementar a operação de convolução via FFT. Na Seção 6.3.4, a biblioteca utilizada para reprodução de áudio é apresentada e são feitas referências a alguns exemplos de código. Finalmente, na Seção 6.3.5, apresenta-se a motivação para a realização de alguns testes com mais de uma thread sendo executadas concorrentemente em CPU.

6.3.1 Organização

Da mesma forma que fora feita na implementação em MATLAB, as funções apresentadas pela Tabela 6.4 foram implementadas em C. De forma abstrata e simplificada, é possível descrever a função *whrtf*, mencionada anteriormente, conforme especificado pelo Algoritmo 3. O objetivo principal dessa função é de calcular os coeficientes (neste trabalho, definido como G) de uma posição conhecida por meio da execução da função *coef_spars* e, a partir desses coeficientes, obter a resposta impulsiva (definida como Hr) como resultado da execução da função *resp_imp*. Estes são os dois passos mais importantes da *whrtf* e podem ser desacoplados um do outro, ou seja, ambos podem ser executados independentemente desde que os parâmetros necessários para a execução de cada procedimento sejam respeitados. Além disso, é dessa forma, desacoplando essas duas funções principais, que consegue-se encontrar HRIRs de posições desconhecidas. Em outras palavras, HRIRs de posições que não foram medidas pelo estudo desenvolvido no trabalho [39] ou em outros estudos similares.

Algoritmo 3 whrtf

1: ▷ **elev** e **azim** são parâmetros de entrada da função
2: $D \leftarrow$ find_delay(elev, azim) ▷ Calcula ITD (*Interaural time difference*)
3: $Ho \leftarrow$ readhrtf(elev, azim) ▷ Lê HRIR da posição especificada
4: $Ho \leftarrow$ shift(Ho, D) ▷ Adianta-se o sinal na orelha mais próxima à fonte sonora.
5: $G \leftarrow$ coef_spars(Wavelet, Ho) ▷ *Wavelet* representa os filtros wavelets que serão utilizados
6: $Hr \leftarrow$ resp_imp(Wavelet, G)

Conforme mencionado anteriormente, o primeiro passo mais importante na execução da função *whrtf* é o procedimento de obtenção dos coeficientes de uma HRIR. Isso é

representado pela chamada executada na quinta linha ($G \leftarrow$ coef_spars(Wavelet, Ho)) do Algoritmo 3. A função *coef_spars* é apresentada de maneira simplificada pelo Algoritmo 4.

Algoritmo 4 coef_spars

```
1:                                  ▷ filtros e hrtf são parâmetros de entrada da função
2: J ← length(filtros)              ▷ calcula-se o número de estágios de acordo com os filtros
   utilizados
3: sist ← hrtf
4: for j ← 1 to J do
5:     H ← lefilt(filtros[j]);
6:     G_aux ← dec_poly(H, sist)
7:     G[J-j+1] ← G_aux[2]
8:     sist ← G_aux[1]
9: end for
10: G[0] ← sist
```

No Algoritmo 4, a variável *filtros* é definida como sendo os filtros utilizados para o cálculo dos coeficientes da HRIR. Para estar em conformidade com o trabalho descrito pelo artigo [14], utilizou-se wavelets Daubechies de tamanho igual a oito, com quatro níveis de decomposição resultando em cinco sub-bandas para a decomposição das HRIRs. A escolha desses filtros protótipos se deu porque, originalmente, o processo de desenvolvimento de um mecanismo de interpolação eficiente de HRIRs com a utilização de transformadas wavelet apresentado nos trabalhos [1, 12, 13, 14], tinha o intuito de reduzir o modelo da HRIR gerada por meio da eliminação de coeficientes com baixo valor de energia. Isso reduz o custo computacional de processamento mas mantém cerca de 95% da energia original da HRTF [14]. Buscando encontrar uma transformada wavelet que permitisse reduzir ao máximo o número de coeficientes do modelo da HRTF sem comprometer as características espectrais da HRTF original, foram consideradas as escolhas das funções de base (filtros protótipos que variam de acordo com a família wavelet escolhida) e a estrutura de decomposição (divisão das frequências conforme Figura 4.10) [12]. Nos trabalhos [1, 12], foram investigadas duas famílias de transformadas wavelet: Biortogonal e Daubechies. De acordo com esses estudos, percebeu-se que os filtros protótipos biortogonais concentravam mais energia nas frequências mais altas enquanto os filtros protótipos da família Daubechies permitiam uma distribuição de energia mais uniforme. Por esse motivo, optou-se pelo uso dos filtros Daubechies de tamanho oito.

Na execução do Algoritmo 4, a cada iteração do laço apresentado na linha 4, aplica-se a função *dec_poly* sobre os parâmetros H, que representa o filtro Daubechies 8 e *sist* que representa a HRIR original de uma posição conhecida. Visualmente, acontece o mesmo fluxo apresentado na Figura 4.10. A estrutura de decomposição nesse caso é chamada de *direta* ou *estrutura em oitavas*. Nessa forma de decomposição, as frequências mais baixas são decompostas a cada novo estágio, o que permite uma resolução cada vez maior nessa faixa de frequência. Dessa maneira, os resultados das decomposições das frequências mais altas são armazenados como os coeficientes para cada sub-banda de acordo com a iteração. A Figura 6.4 apresenta visualmente o processo de execução do Algoritmo 4.

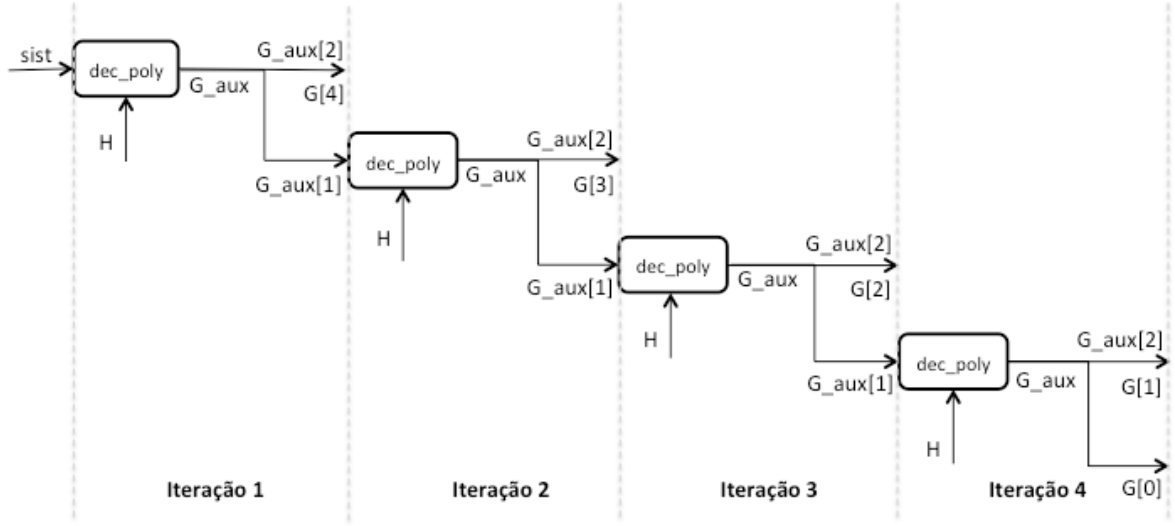

Figura 6.4: Representação visual da execução da função *coef_spars*.

Um esboço da função *dec_poly* é apresentado a seguir pelo Algoritmo 5. Essa função é responsável por decompor, por meio do filtro Daubechies de tamanho oito, o sistema representado pela HRIR. Além disso, nessa mesma função, implementa-se o cálculo de algumas operações definidas anteriormente nas Seções 4.3.1 e 4.3.2.

Algoritmo 5 dec_poly

1: **filtro** e **hrtf** são parâmetros de entrada da função
2: → obtém-se os filtros de síntese (F) e análise (H) a partir do **filtro** de entrada de acordo com as equações 4.16, 4.17 e 4.18
3: → decomposição polifásica dos bancos de síntese e análise (vide Seção 4.3.2)
4: → decomposição polifásica da **hrtf**
5: → cálculo dos coeficientes conforme equação 4.29

Até esse momento, foi apresentado o procedimento que recupera os coeficientes de cada sub-banda. Esse procedimento é importante porque é a partir de coeficientes de posições conhecidas é que se torna possível o cálculo dos coeficientes de uma posição não conhecida. A partir daí, recupera-se a HRTF correspondente a esses coeficientes.

O segundo passo mais importante na execução da função *whrtf* é o procedimento de obtenção da HRIR a partir de seus coeficientes esparsos. Isso é representado pela chamada executada na linha seis ($Hr \leftarrow$ resp_imp(Wavelet, G)) do Algoritmo 3. A função *resp_imp* é apresentada de maneira simplificada pelo Algoritmo 6.

O resultado retornado pela função *resp_imp* é a HRIR correspondente aos coeficientes G. Como se pode ver, a função *cascata* é aplicada ao parâmetro *filtros* (que é representada novamente por quatro estágios compostos pelos filtros protótipos da família Daubechies de tamanho oito), obtendo assim, a estrutura H formada também por cinco níveis, cinco sub-bandas. Em seguida, no algoritmo, os coeficientes de esparsidade são obtidos da mesma maneira como foram definidos pela equação 4.15. Nas próximas linhas, faz-se o espargimento dos coeficientes representados por G e posteriormente a convolução entre o resultado do cascateamento do filtro Daubechies e os coeficientes esparsos. Tudo isso para

Algoritmo 6 resp_imp

1: ▷ **filtros** e **G** são parâmetros de entrada da função
2: $H \leftarrow$ cascata(filtros)
3: $M \leftarrow$ size(H)
4: $L[1] \leftarrow 2^{(M-1)}$
5: **for** $n \leftarrow 2$ **to** M **do** ▷ obtém-se os coeficientes de esparsidade para cada sub-banda
6: L[n] $\leftarrow 2^{(M-n+1)}$
7: **end for**
8: **for** $n \leftarrow 1$ **to** M **do**
9: G_t[n] $\leftarrow$ spars(G[n], L[n])
10: R[n] $\leftarrow$ conv(H[n], G_t[n])
11: R_sizes[n] $\leftarrow$ length(R[n])
12: **end for**
13: Max_R $\leftarrow$ max(R_sizes)
14: // cria-se a matriz Mh[M][Max_R] onde a linha n contém R[n]
15: // somam-se as colunas da matriz Mh, obtendo um único vetor Xr que é retornado como resultado da execução da função

cada sub-banda. Com o resultado da convolução em cada sub-banda, cria-se uma matriz em que cada linha representa o resultado dessa convolução para cada sub-banda. Após criar essa estrutura, os elementos de cada coluna são somados, obtendo, assim, a HRIR resultante dos coeficientes esparsos.

A função *resp_imp* apresenta duas outras funções interessantes e importantes para o entendimento do funcionamento desse processo de recuperação de HRIRs a partir de seus coeficientes esparsos. São elas: *cascata* e *spars*, que são apresentadas pelos algoritmos 7 e 8 respectivamente, a seguir.

Algoritmo 7 cascata

1: ▷ **filtros** é o parâmetro de entrada da função
2: $J \leftarrow$ length(filtros) ▷ identifica o número de estágios
3: $H \leftarrow$ lefilt(filtros[1])
4: $G[0] \leftarrow$ H[2] ▷ passa-altas
5: $Gl \leftarrow$ H[1] ▷ passa-baixas
6: **for** $j \leftarrow 2$ **to** J **do**
7: $Gl\_old \leftarrow Gl$
8: $Gl \leftarrow$ conv(Gl_old, spars(H[1], $2^{(j-1)}$))
9: $G[j-1] \leftarrow$ conv(Gl_old, spars(H[2], $2^{(j-1)}$))
10: **end for**
11: $G[J] \leftarrow$ conv(Gl_old, spars(H[1], $2^{(J-1)}$))

O Algoritmo 7 funciona de maneira semelhante ao apresentado nas Figuras 4.10 e 6.4. O filtro passa-baixas é convoluído com sua representação esparsa, gerando um conjunto de dados que será utilizado na próxima iteração. Além disso, esse mesmo filtro passa-baixas é convoluído com a representação esparsa do filtro passa-altas e é armazenado em $G[j-1]$, onde j é o contador que identifica a iteração. Já o Algoritmo 8, que representa a função

Algoritmo 8 spars

```
1:                                  ▷ array e sp são os parâmetros de entrada da função
2: N ← size(array)                  ▷ identifica o número de elementos do array
3: y ← zeros[(N − 1) ∗ sp + 1]      ▷ cria um novo vetor com (N − 1) ∗ sp + 1 posições
4: for n ← 1 to N do
5:     y[(n − 1) ∗ sp + 1] ← array[n]
6: end for
```

spars, é responsável por adicionar zeros entre os valores originais de um vetor conforme o coeficiente de esparsidade, de acordo com a equação 4.15.

6.3.2 Cocoa

No projeto desenvolvido em MATLAB, não foi criada uma interface gráfica que permitisse a interação de algum usuário de forma a propiciar a mudança de posicionamento dinamicamente. O propósito da primeira etapa foi de apenas validar a implementação do algoritmo de interpolação de HRTFs e outras funções que seriam úteis ao processamento em tempo real. Já na segunda fase do desenvolvimento, foi criada uma interface gráfica para permitir a interatividade com um usuário. Essa interface foi implementada utilizando o framework Cocoa [62] disponível no sistema operacional Mac OS X [63].

Cocoa é um framework de desenvolvimento formado por bibliotecas e APIs que provêem acesso a vários componentes do sistema operacional tais como: áudio, video, rede, aplicações de usuários, gerenciamento de dados e interface gráfica. Está escrito em Objective-C, um subconjunto do *American National Standards Insitute* (ANSI) C, e foi extendido com algumas características sintáticas e semânticas derivadas do Smalltalk de forma a suportar o paradigma de programação orientado a objetos [64]. A implementação em Cocoa não exige que toda a aplicação seja desenvolvida utilizando esse framework. Por ter origem no ANSI C padrão, é possível mesclar código de bibliotecas escritas em C ou C++.

Esse framework utiliza o padrão de projeto chamado *Model-View-Controller*, Modelo-Visão-Controlador (MVC). Os modelos encapsulam os dados da aplicação e as visões apresentam e oferecem mecanismos de edição dos dados. As controladoras fazem a mediação da lógica de negócio entre modelos e visões [62]. Esse padrão de projeto permite separar efetivamente as responsabilidades de cada camada. Dessa forma, o mesmo programa pode ser implementado em outro ambiente que utiliza outra tecnologia para a criação de telas sem que o núcleo de funcionalidades precise ser reimplementado.

Nesse trabalho, o framework Cocoa foi utilizado para criar as telas que compõem a interface gráfica com o usuário. Além disso, também foi utilizado para a criação de threads e gerenciamento de arquivos. A Figura 6.5 a seguir mostra a tela criada para ser apresentada enquanto as HRTFs são interpoladas. Em seguida, após o processamento das HRTFs, a tela inicial do programa, representada pela Figura 6.6 é apresentada.

O usuário, após ser apresentado a esta tela, deve selecionar um arquivo de áudio em formato *WAVEform audio format* (WAVE) para que os controles de execução e de posicionamento sejam habilitados. Assim que um arquivo de áudio WAVE qualquer é selecionado, tem-se a situação representada pela Figura 6.7. O programa também utiliza o framework Cocoa para gerenciar threads de execução. A Figura 6.8 a seguir mostra o

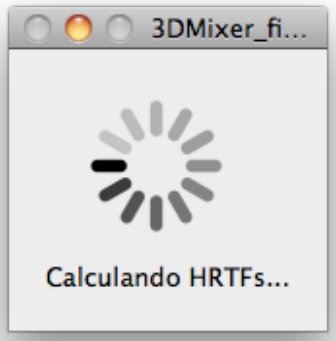

Figura 6.5: Tela apresentada ao usuário enquanto as HRTFs são interpoladas.

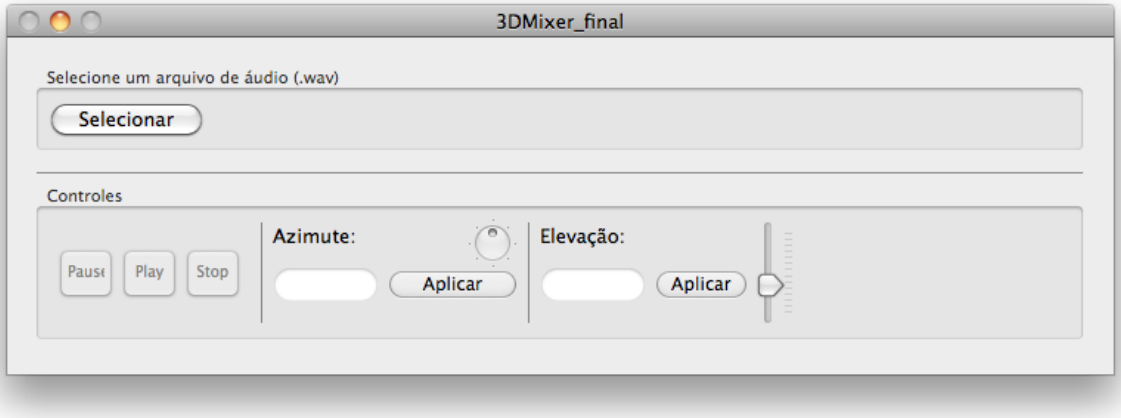

Figura 6.6: Tela apresentada ao usuário após o processamento das HRTFs.

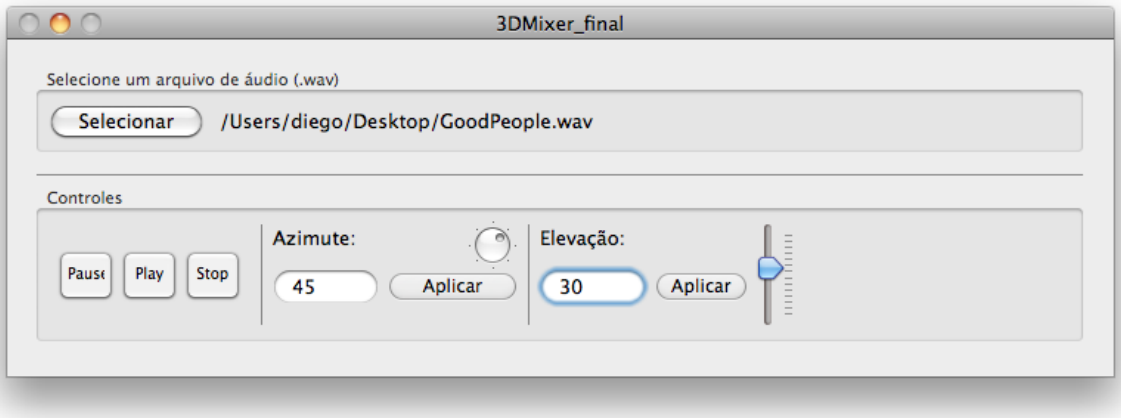

Figura 6.7: Usuário seleciona um arquivo de áudio e os controles são habilitados para alteração.

processo de interpolação das HRTFs sendo executado em paralelo com a thread principal, que gerencia a atualização da interface gráfica. Ao final desse processamento, a thread auxiliar utilizada nessa interpolação é finalizada.

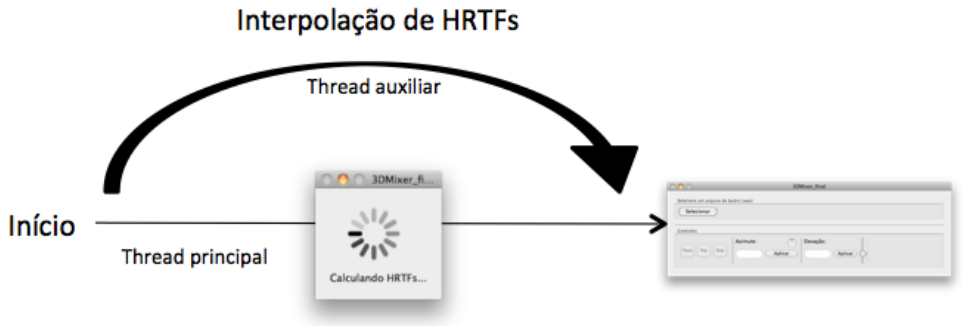

Figura 6.8: Thread auxiliar utilizada para o cálculo das HRTFs enquanto a thread principal trabalha na atualização da interface gráfica.

6.3.3 FFTW

No processo de execução da síntese biaural, é preciso aplicar o par de HRTFs da posição desejada ao sinal de áudio. Isso é feito por meio da operação de convolução via FFT conforme foi apresentada na Seção 3.2.6. A convolução via FFT, por ser mais rápida, foi implementada com o auxílio de uma biblioteca desenvolvida no MIT chamada *Fastest Fourier Transform in the West* (FFTW). Essa biblioteca, segundo alguns testes de performance, se mostra superior quando comparada a outras bibliotecas que possuem o mesmo propósito de cálculo da transformada de Fourier [65].

A FFTW é capaz de calcular a transformada discreta de Fourier em uma ou mais dimensões de tamanho arbitrário tanto para dados do conjunto dos números reais quanto para dados do conjunto dos números complexos. O bloco de código apresentado na Seção A.1 mostra como a convolução via FFT está implementada com o uso da biblioteca FFTW.

Tal função recebe como parâmetros de entrada:

- o sinal
- tamanho do sinal
- o filtro que será utilizado para realizar a convolução com o sinal
- tamanho do filtro

O sinal e o filtro são convertidos para uma representação em números complexos e a transformada de Fourier é aplicada sobre ambos. Multiplicam-se os resultados das duas operações anteriores e aplica-se a transformada inversa de Fourier sobre o resultado dessa multiplicação. Assim, tem-se a convolução via FFT.

6.3.4 LibSDL

Para a reprodução do áudio biaural, a biblioteca *Simple DirectMedia Layer* (libSDL) foi escolhida pois é uma biblioteca multimidia robusta que pode ser utilizada em dife-

rentes plataformas. Essa biblioteca está escrita em C e provê acesso de baixo nível aos dispositivos de áudio, teclado, mouse, dentre outros. É utilizada no desenvolvimento de softwares em geral e em alguns jogos [66].

As seções A.2 e A.3 apresentam duas funções utilizadas nesse projeto para a reprodução de áudio biaural. A função A.2 é responsável por preparar o dispositivo de áudio para ser utilizado. Já a função A.3 é a responsável por aplicar o par de HRTFs sobre um pequeno trecho do sinal de áudio e tornar o resultado dessa operação disponível no buffer de reprodução.

6.3.5 Testes com Múltiplas Threads

Na mesma etapa do desenvolvimento da biblioteca em C, descrita na Seção 6.3, foram realizados testes com a execução de não apenas uma única thread para o cálculo de WHRTFs, HRTFs interpoladas no domínio wavelet, mas sim, com múltiplas threads. O intuito dessa sub-etapa foi verificar se haveria ganho em performance ou *throughput* ao executar o procedimento para o cálculo de WHRTFs concorrentemente na CPU. Conforme será apresentado na Seção 6.5 adiante, há vantagens em disparar mais de uma thread concorrentemente para o cálculo de WHRTFs distintas. Ao se fazer isso, consegue-se que o tempo necessário para o cálculo de cada uma individualmente decresça.

6.4 Implementação em CUDA

A terceira etapa da implementação desse projeto consiste na utilização da GPU para o processamento em paralelo de alguns procedimentos. De acordo com o que foi apresentado na Seção 6.2, tem-se dois passos importantes no processo de interpolação de HRIRs. O primeiro passo é o cálculo dos coeficientes esparsos e o segundo é a obtenção da HRIR de uma posição a partir desses coeficientes. Tais coeficientes são apresentados como um conjunto de vetores que representam as sub-bandas criadas após o cálculo da transformada wavelet (implementada por um banco de filtros). Esse procedimento é descrito pelo Algoritmo 4.

Nessa etapa do projeto, o foco está sobre pontos no código passíveis de paralelismo, ou seja, trechos de código que podem ser otimizados por meio da execução em paralelo. A partir de uma breve análise do Algoritmo 4 é possível perceber que ele não é ideal para ser paralelizado pois cada iteração depende de dados produzidos pela iteração anterior. Seguindo essa mesma ideia, percebe-se que o Algoritmo 6 pode ser paralelizado facilmente pois cada iteração pode ser executada independentemente da outra.

A função descrita pela listagem na Seção A.4 mostra a implementação do Algoritmo 6, que calcula a resposta impulsiva referente a uma posição a partir de um conjunto de coeficientes. Nessa função, os dados necessários para esse cálculo são enviados para a memória do dispositivo de vídeo. Após esse procedimento, a função que de fato executa o processamento em paralelo é então invocada. Essa função está representada pela listagem na Seção A.5. O resultado desse processamento é copiado do dispositivo para a memória do host para ser utilizado pelas operações de posicionamento do objeto sonoro em tempo real.

Nesse trabalho utiliza-se filtros Daubechies de tamanho oito. Os quatro estágios ocasionados por esse filtro geram cinco sub-bandas e, logo, cinco vetores representando cada

sub-banda dos coeficientes relativos a uma HRIR. A função representada pela listagem na Seção A.5 é executada pelo código apresentado na listagem A.4 em cinco blocos de threads na GPU. Dessa forma, cada bloco é responsável por uma única sub-banda. Além disso, utiliza-se o número máximo de threads por bloco que a GPU disponibiliza. É importante mencionar que essa implementação não aproveita ao máximo o potencial que a GPU oferece. Até esse momento utiliza-se apenas cinco blocos de threads para o processamento de uma única HRIR. Tomando como exemplo a GPU utilizada nesse trabalho e detalhada na Tabela 6.2, é possível configurar uma estrutura de blocos de threads com as seguintes dimensões: 512 x 512 x 64. Ou seja, para aproveitar ao máximo a estrutura de blocos que o dispositivo de vídeo oferece, 16777216 blocos de threads podem ser utilizados para o processamento em um único grid. Entretanto essa configuração ainda não foi explorada pois é preciso que o código seja alterado para atender a essa estrutura de maior complexidade. Porém é uma das possibilidades futuras de evolução desse trabalho.

Após apresentar as funções chaves que compõem esse trabalho, segue uma breve exposição de como esse algoritmo de interpolação de HRTFs em GPU está posicionado dentro do programa criado para a simulação do posicionamento de uma fonte sonora em tempo real. A arquitetura de todo o programa, incluindo o processo de interpolação de HRTFs está representada pela Figura 6.9.

A arquitetura apresentada na Figura 6.9 mostra que, a partir das HRIRs existentes (que podem ser obtidas de um banco de HRIRs de algum trabalho já desenvolvido), obtém-se os seus respectivos coeficientes. A seguir, a partir desses coeficientes, obtém-se o valor das HRIRs interpoladas, processo este executado no dispositivo de vídeo, na GPU. O sinal de áudio é então dividido em pequenos blocos e cada um desses blocos sofre convolução com o par de HRIRs da posição selecionada pelo usuário naquele momento. O resultado dessa convolução é o sinal de áudio reproduzido nos canais esquerdo e direito de um fone de ouvido. Como se pode ver, a operação de convolução dos blocos do sinal de áudio com as HRIRs interpoladas está acontecendo no host e não na GPU. Foi escolhido dessa maneira pois de acordo com um estudo apresentado na referência [67] e que fez a comparação entre performance de execução entre a biblioteca FFTW e cuFFT (biblioteca CUDA que implementa as operações relacionadas à FFT), o cálculo da FFT cujo tamanho não é uma potência de dois é realizado mais rapidamente na CPU utilizando a biblioteca FFTW. Além disso, no artigo [68] listado nas referências, os autores fazem um estudo sobre aplicações que precisam realizar rapidamente uma grande quantidade de convoluções baseadas em FFT. Segundo eles, não há uma melhora significativa de performance com a utilização de GPU. Essa situação se encaixa no contexto desse projeto pois aqui é necessário responder rapidamente às mudanças de direção solicitadas pelo usuário e a cada novo instante de tempo um novo par de HRIRs deve ser aplicado sobre um pequeno bloco de áudio.

De acordo com o mesmo artigo [68], muitas aplicações necessitam executar rapidamente várias operações de convolução baseada em FFT com pequenos conjuntos de dados. Para que esse cenário seja executado eficientemente, é importante, dentre outros:

- maximizar o paralelismo independente no algoritmo
- minimizar o tempo de transferência de dados entre CPU e GPU
- ajustar as configurações para otimizar a execução

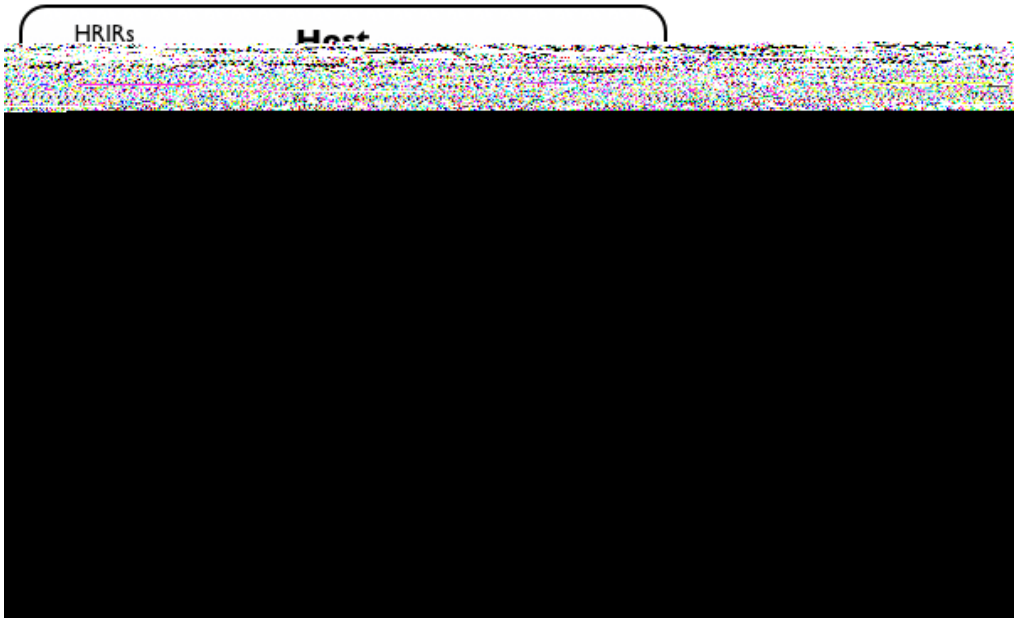

Figura 6.9: Descrição da arquitetura. Na etapa 1, a partir das HRIRs existentes calculam-se seus respectivos coeficientes. Na etapa 2 esses coeficientes são processados pela GPU e retornam as HRIRs interpoladas correspondentes na etapa 3. Pequenos blocos de sinal de áudio são separados na etapa 4 e sofrem a convolução com um par de HRIRs na etapa 5. O resultado dessa convolução, apresentado na etapa 6, é reproduzido por fones de ouvido conforme apresentado na etapa 7.

Entretanto, ainda assim a transferência de dados tomaria grande parte do tempo de execução. O artigo [68] chega a conclusão de que para superar essas limitações, a primeira opção é que uma nova biblioteca que execute a operação de convolução baseada em FFT sobre pequenos conjuntos de dados seja criada e otimizada. Outra medida que poderia beneficiar não só esse caso mas também vários outros no contexto da programação de propósito geral em GPU seria prover um canal de comunicação mais rápido entre CPU e GPU. Assim não se perderia tanto tempo trafegando os dados entre os dispositivos.

Neste trabalho, tentou-se implementar outras partes do algoritmo de interpolação de HRTFs em GPU. Dentre elas, alguns procedimentos executados no cálculo dos coeficientes esparsos. Entretanto, após testes e tentativas em melhorar o algoritmo, chegou-se a conclusão de que tais procedimentos seriam melhor processados em CPU. O tempo gasto na transmissão dos dados entre CPU e GPU não compensava a melhoria em performance e o esforço despendido para implementar um kernel que seria executado em paralelo pela GPU. Em outras palavras, o tempo que era ganho na execução em paralelo era perdido na transmissão dos dados entre CPU e GPU.

Também, nessa etapa, executou-se procedimento semelhante ao apresentado na Seção 6.3.5 anterior. Da mesma forma, obteve-se uma melhoria ao disparar threads concorrentes no host para o cálculo de WHRTFs de diferentes posições em GPU. Tais resultados serão apresentados na Seção 6.5 a seguir.

6.5 Resultados

Testes cujo foco eram o tempo de execução foram realizados nas três etapas de desenvolvimento desse trabalho e consistiram de execuções dos programas para o cálculo de WHRTFs de posições cujas HRIRs já estavam disponíveis. Os primeiros testes realizados com a implementação do algoritmo de interpolação de HRTFs foram feitos na primeira etapa de desenvolvimento com o código escrito em MATLAB. Naquele momento, após alguns refinamentos do código, era possível calcular uma WHRTF a partir de uma HRIR conhecida em aproximadamente 0,67 segundo, o que é obviamente, impraticável em um sistema de áudio em tempo real.

Na segunda etapa da implementação, onde todo o código escrito em MATLAB foi re-escrito na linguagem C, obteve-se uma notável melhoria de tempo de execução para o cálculo de uma WHRTF se comparado ao código em MATLAB. O tempo médio necessário para o cálculo de uma única WHRTF nessa implementação é de aproximadamente 0,007 segundo. Ainda nessa etapa de testes, verificou-se o comportamento do algoritmo quando mais de uma thread executava o mesmo procedimento em CPU, conforme apresentado na Seção 6.3.5. Como exemplo, ao iniciar 16 threads, onde cada uma é responsável pelo cálculo de uma WHRTF, constatou-se que o tempo médio gasto em cada thread foi menor que o tempo consumido na execução de apenas uma única thread. Em suma, disparar 16 threads concorrentes para o cálculo de 16 WHRTFs é mais interessante que disparar apenas 1 thread para o cálculo de uma única WHRTF. Neste caso, 16 WHRTFs são calculadas em 0,053 segundo, o que corresponde a aproximadamente 0,003 segundo por thread. O gráfico da Figura 6.10 apresenta o tempo gasto na execução para diferentes números de threads, onde cada thread é responsável pelo cálculo de uma WHRTF.

Já na terceira etapa da implementação desse projeto, onde utilizou-se CUDA para o processamento em paralelo de certas partes do código, o tempo médio empregado no

cálculo de uma WHRTF, utilizando apenas uma thread em CPU, foi de 0,003 segundo. Do mesmo modo descrito no parágrafo anterior, testou-se a execução e verificou-se o comportamento de mais de uma thread em CPU executando o cálculo de WHRTFs. Na situação em que 16 threads em CPU foram utilizadas para o cálculo de 16 valores de WHRTFs, obteve-se os resultados em aproximadamente 0,029 segundo, o que corresponde a aproximadamente 0,0018 segundo por thread.

O gráfico representado pela Figura 6.10 mostra um comparativo de performance entre a execução em C e CUDA. Percebe-se que a execução em GPU é mais eficiente que a execução em CPU. Os valores representados no gráfico da Figura 6.10 foram obtidos por meio de sucessivas execuções do programa com o número de threads correspondente. A seguir, calculou-se a média desses conjuntos de valores para a implementação em C e em CUDA, apresentados nas Seções B.1 e B.2, respectivamente. Convém notar que o uso da GPU para o cálculo da interpolação de HRTFs é interessante pois a GPU nem sempre é utilizada a todo instante. Em muitos momentos esse recurso está ocioso e, portanto, pode ser utilizado para o processamento massivo de dados.

Visando entender o comportamento do procedimento que calcula a WHRTF, executou-se o mesmo processo repetidas vezes até que se obtesse um conjunto de dados relevantes para uma conclusão. A Figura 6.11 mostra um gráfico que compara as execuções em C e em CUDA para uma única thread. Já a Figura 6.12 revela um gráfico para o procedimento que calcula 16 WHRTFs utilizando as duas implementações: em C e em CUDA. Ambos mostram vinte execuções do programa em sequência. Mais exemplos com diferentes números de threads são apresentados no Apêndice B desse mesmo trabalho.

Nas Figuras 6.11 e 6.12 verifica-se que algumas execuções em CUDA demandaram tempo superior ao da execução em C. Isso pode acontecer quando o dispositivo de vídeo está ocupado executando outras tarefas diferentes da interpolação.

No programa criado nesse trabalho e que oferece a possibilidade de interação com o usuário, permitindo que o mesmo possa informar o posicionamento da fonte sonora em tempo real, é preciso que se tenha um conjunto de HRIRs correspondentes à todo o espaço ao seu redor. Assim, para o cálculo de 360 valores de azimute para cada valor de elevação entre o intervalo [-40°, 80°], o programa desenvolvido utilizando a tecnologia CUDA, é executado em aproximadamente 67 segundos. Assim, considerando as 43560 diferentes posições e, sabendo que para cada posição é preciso calcular duas WHRTFs diferentes, uma para cada orelha, tem-se que 87120 WHRTFs são calculadas nesses 67 segundos. Com esse resultado, conclui-se que cada WHRTF é calculada em aproximadamente apenas 0,0008 segundo.

A Figura 6.13 a seguir apresenta o tempo de execução do programa apelidado de *3D Mixer* em 31 execuções independentes. A cada execução, todas as 87120 WHRTFs são calculadas. Os valores representados no gráfico da Figura 6.13 estão apresentados na tabela 6.5.

Também foram realizadas avaliações subjetivas quanto ao posicionamento da fonte sonora como resultado do processo de síntese biaural por meio da aplicação das HRTFs interpoladas sobre o sinal de áudio. Essas avaliações consistem de audições do sinal resultante da síntese biaural. Em todas as implementações (MATLAB, C e CUDA), os resultados apresentados foram semelhantes e consistentes com a sensação de direcionalidade do som. No intuito de se avaliar a qualidade do sinal gerado após a aplicação

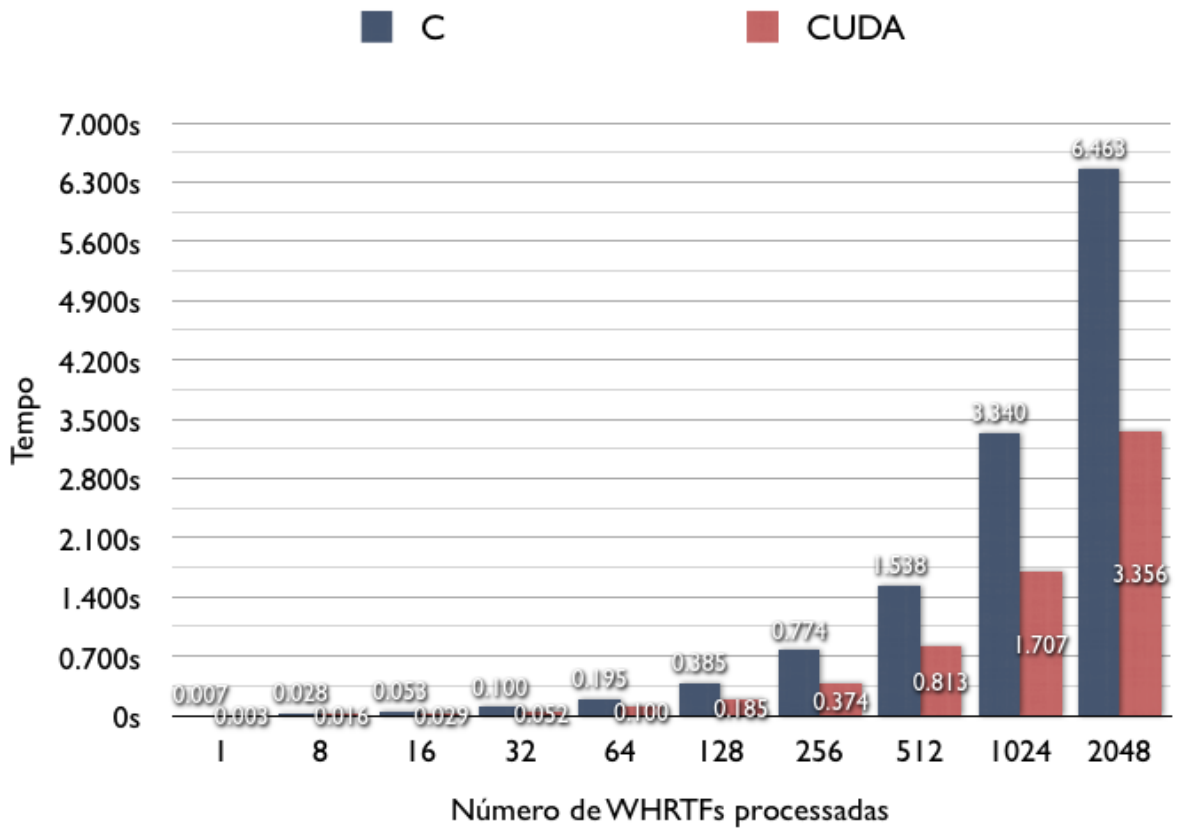

Figura 6.10: Tempo empregado no cálculo de uma a 2048 WHRTFs utilizando o código escrito apenas em C e o código escrito em CUDA. Cada WHRTF é executada por uma thread inicializada em CPU.

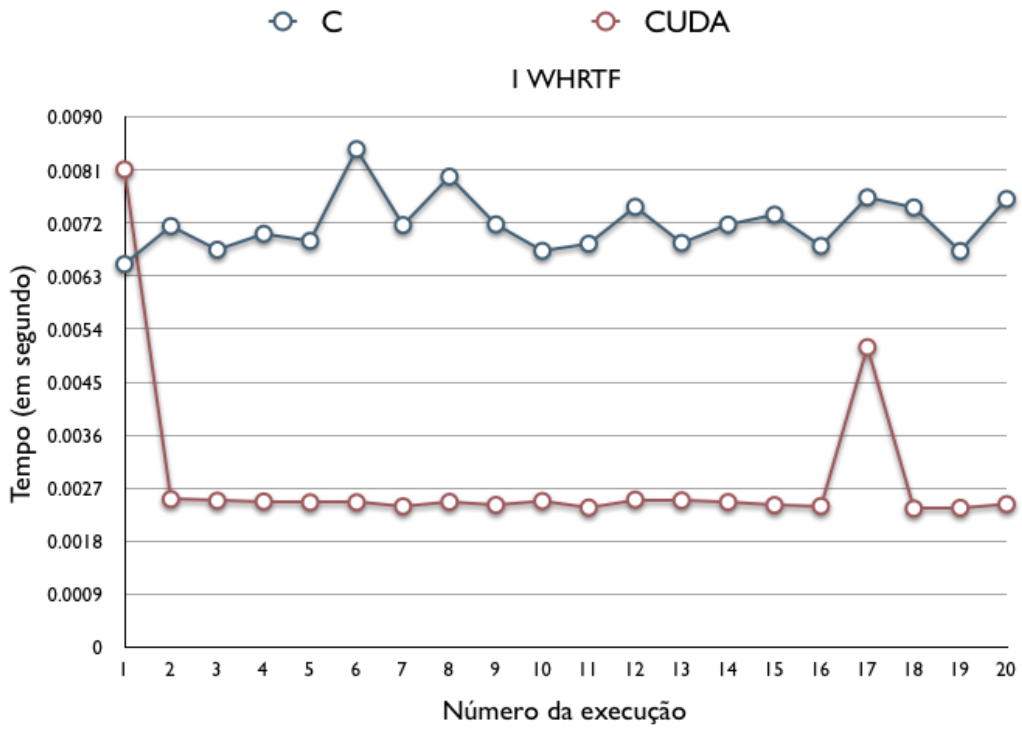

Figura 6.11: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de uma única WHRTF utilizando uma thread em CPU.

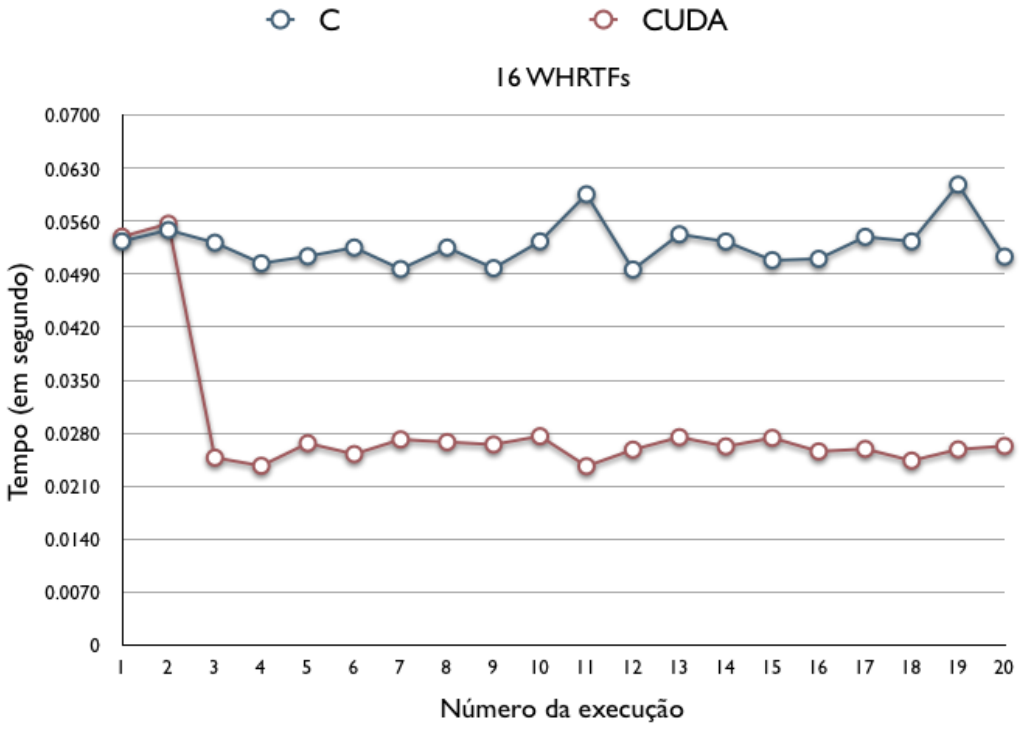

Figura 6.12: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 16 WHRTFs utilizando 16 threads em CPU.

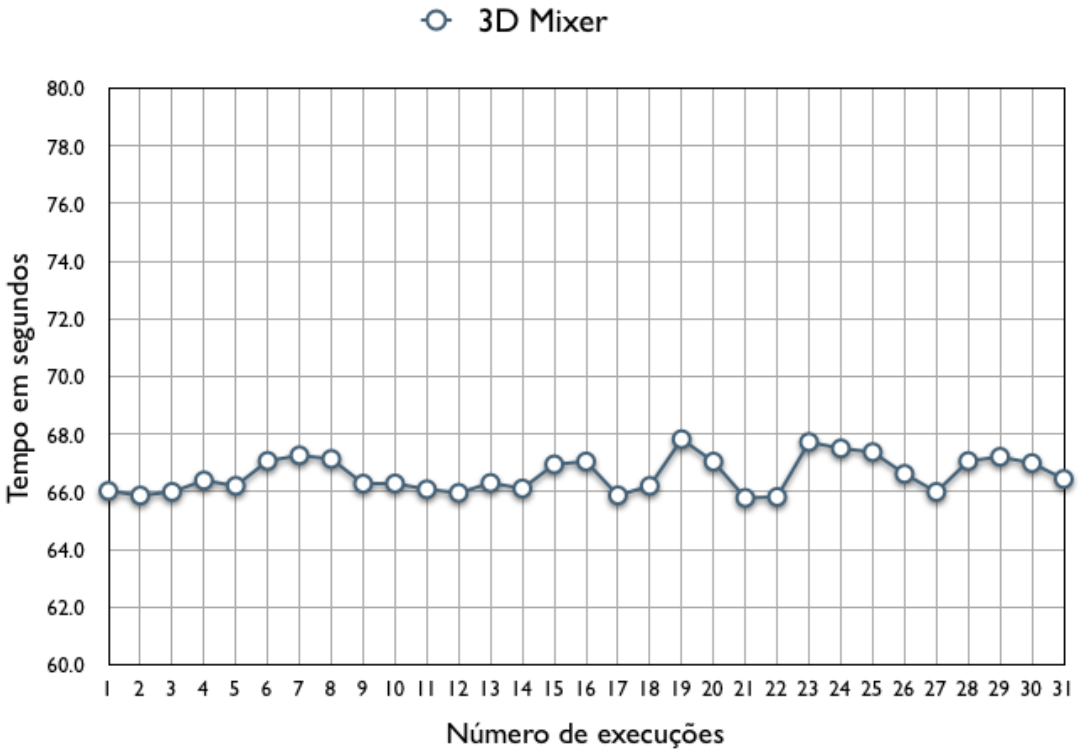

Figura 6.13: Tempo de cálculo de 87120 WHRTFs em 31 execuções independentes.

das HRIR interpoladas, utilizou-se o algoritmo *Perceptual Evaluation of Audio Quality* (PEAQ) conforme será apresentado na Seção 6.5.1 a seguir.

6.5.1 PEAQ - *Perceptual Evaluation of Audio Quality*

Com o objetivo de se obter a qualidade do áudio criado com o mecanismo de interpolação de HRTFs no domínio da transformada wavelet, utilizou-se um programa que implementa o algoritmo PEAQ. Esse algoritmo propõe uma métrica objetiva para avaliar subjetivamente a qualidade de um sinal de áudio ao invés de avaliá-lo apenas do ponto de vista do processamento digital de sinais.

Há algum tempo atrás, devido à falta de padrões internacionais e procedimentos de medidas para avaliação de áudio, o procedimento mais aceito era o teste de audição realizado por seres humanos. Por ser um método impreciso, pesquisas foram desenvolvidas e os primeiros métodos padronizados surgiram para avaliar a qualidade do sinal de áudio de telefones. Após mais alguns anos de pesquisa, a *International Telecommunication Union - Radiocommunication sector* (ITU-R) [69] recomendou a implementação do algoritmo PEAQ (ITU-R BS.1387) devido à necessidade urgente de se criar um padrão nessa área e tendo como inspiração seis métodos diferentes para a medição da qualidade de áudio [70].

Testes subjetivos demandam muito tempo para serem executados, são caros e não são práticos no dia-a-dia. A análise de resultados de testes subjetivos é baseada no grau de diferença subjetiva, *Subjective Difference Grade* (SDG), que é definido como a diferença entre um sinal de teste e um sinal de referência. Este é correspondente ao grau

1	66.030619
2	65.87051
3	65.99737
4	66.391513
5	66.206602
6	67.07077
7	67.263388
8	67.139825
9	66.282317
10	66.288137
11	66.087033
12	65.962272
13	66.306679
14	66.114221
15	66.950838
16	67.049881
17	65.876177
18	66.203637
19	67.823519
20	67.047367
21	65.786003
22	65.820763
23	67.726169
24	67.506224
25	67.376439
26	66.622056
27	65.99465
28	67.066328
29	67.213757
30	66.994556
31	66.445212

Tabela 6.5: Tempo para o cálculo de 87120 WHRTFs em 31 execuções independentes.

de diferença objetiva, ODG, que é a unidade adotada neste trabalho. A descrição desses graus é apresentada a seguir pela tabela 6.6.

Prejuízo	**ODG**
Imperceptível	0.0
Perceptível, mas não irritante	-1.0
Ligeiramente irritante	-2.0
Irritante	-3.0
Muito irritante	-4.0

Tabela 6.6: Grau de diferença objetiva. Fonte: *Estimating Perceptual Audio System Quality Using PEAQ Algorithm* [70]

O processo da percepção humana compara um sinal de referência com um sinal de teste e essa comparação nos permite estimar a diferença audível. Esse modelo de medição, que pode ser representado pela Figura 6.14, produz um número de variáveis, chamadas de *Model Output Variables* (MOV). No estágio final desse método, o algoritmo combina essas variáveis e produz um único valor de saída que corresponde à avaliação subjetiva da qualidade do áudio - ODG [70].

A Figura 6.14 apresenta o conceito básico para fazer medidas com o método PEAQ recomendado. Esse método pode ser aplicado em conjunto com diversos tipos de equipa-

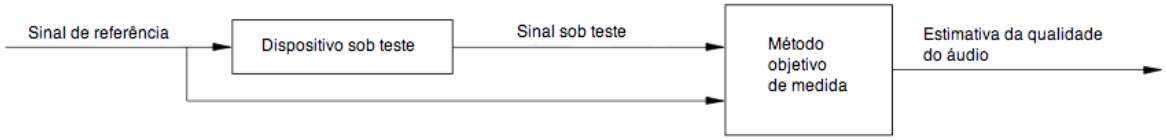

Figura 6.14: Conceito básico e geral para realizar medições objetivas utilizando um método objetivo de medida definido de acordo com o algoritmo escolhido. Fonte: *Recommendation ITU-R BS.1387-1* [71]

mentos de processamento de sinais de áudio tanto digitais quanto analógicos. Dentre suas aplicações, é possível destacar o auxílio à avaliação subjetiva e a avaliação de diferentes implementações [71], aplicações estas que se enquadram no escopo do presente trabalho.

Nesse trabalho, utilizou-se o método PEAQ para verificar a qualidade do sinal gerado após a aplicação das HRIRs criadas no processo de interpolação utilizando a implementação com as unidades de processamento gráfico (GPU). Assim, desse modo, os sinais de referência são os sinais de áudio sobre os quais HRIRs originais foram aplicadas. Já os sinais de teste, são os sinais de áudio gerados após a aplicação de HRIRs geradas a partir do mecanismo de interpolação de HRTFs implementado em GPU. A Figura 6.15 a seguir mostra os valores de ODG para cada azimute, em incrementos de 5°, e em cinco valores diferentes de elevação para a implementação feita em GPU. No teste realizado, foram gerados dois conjuntos de informações. O primeiro consiste do resultado da aplicação de HRIRs originais sobre um sinal de áudio de cerca de 3 minutos de duração com a implementação em CUDA. O segundo conjunto de dados consiste no sinal resultante da aplicação de HRIRs geradas pelo algoritmo de interpolação implementado em GPU sobre o mesmo sinal de áudio. Dessa maneira, tem-se todos os sinais de referência e sinais de teste que serão utilizados pelo algoritmo PEAQ. Assim, o gráfico apresentado pela Figura 6.15 mostra o resultado do algoritmo PEAQ nas 360 posições em questão (72 posições para cada elevação).

A tecnologia utilizada para obter os valores de ODG é protegida por patentes internacionais e, para explorá-la, é obritatória a obtenção de uma licença [71]. Para fins de pesquisa e estudo, existem programas e implementações livres que possuem as mesmas funcionalidades porém não foram validados junto à ITU-R, entretanto seu uso é incentivado. Dentre tais programas e implementações, tem-se o PEAQB [72] e o PQEVALAUDIO, disponibilizado pelo laboratório de telecomunicações e processamento de sinal da *McGill University* [73]. Este segundo software é utilizado nesse trabalho pois possui uma boa documentação e seu uso é bastante simples. Este programa requer que os sinais de referência e de teste estejam a uma taxa de amostragem de 48000 Hz. Para converter esses sinais de 44100 Hz para essa taxa de amostragem, utilizou-se primeiramente o programa RESAMPAUDIO, presente no pacote AFsp que é disponibilizado por essa mesma universidade [73]. Para obter os valores apresentados pelo gráfico da Figura 6.15, executou-se o programa PQEVALAUDIO para cada um dos sinais de referência e de teste, correspondendo aos mesmos valores de azimute e elevação.

É possível ver no gráfico da Figura 6.15 que os valores de ODG, ou seja, o prejuízo causado no sinal de áudio devido a aplicação de HRIRs interpoladas com o auxílio da GPU em comparação com a aplicação de HRIRs originais, encontram-se entre a faixa interpretada como “Perceptível, mas não irritante” e “Ligeiramente irritante”. Entretanto,

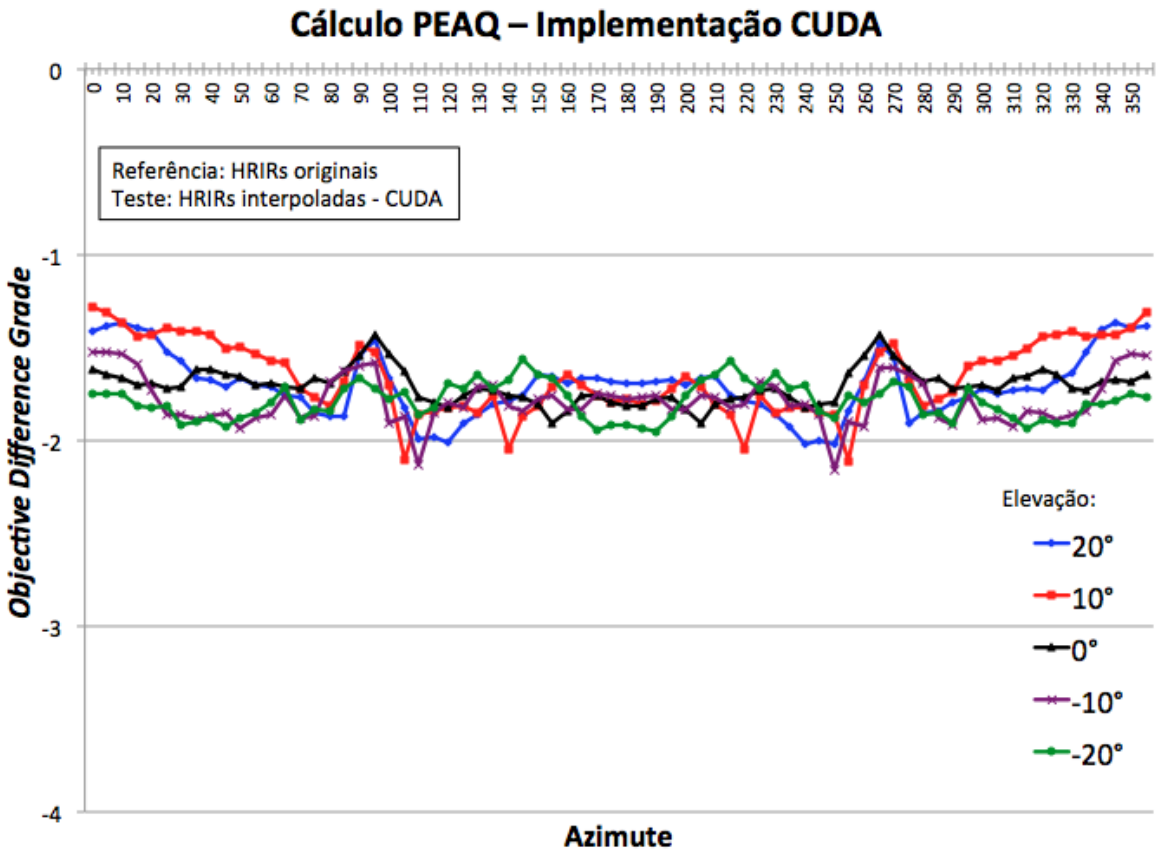

Figura 6.15: Comparação dos valores de ODG para elevações no intervalo [-20°, 20°] variando de 10° em 10°, e azimute variando de 5° em 5° na implementação em GPU.

percebe-se que os sinais de referência e teste perdem algumas faixas de frequências quando comparados ao sinal original. De acordo com os resultados apresentados pelo artigo [70], é possível dizer que o método de interpolação de HRIRs utilizando GPU, apresentado nesse trabalho, gera prejuízo em relação à aplicação de HRIRs originais sobre o mesmo sinal de áudio, próximo ao prejuízo causado pela aplicação de codecs MP3 e AAC a uma taxa de bits de 128kbps.

Também nota-se uma simetria dos resultados de 0° a 180° no azimute com os resultados de 180° a 355°. Isso se deve pelo fato das HRIRs serem simétricas, ou seja, a HRIR que modela a filtragem espectral na orelha esquerda de uma fonte sonora localizada a 60° à esquerda de um ouvinte, por exemplo, pode ser considerada idêntica à HRIR que modela a filtragem espectral na orelha direita de uma fonte sonora localizada também a 60° mas à direita do ouvinte.

Comparou-se também o sinal de áudio original, sem aplicação de nenhuma HRIR, com o sinal resultante da aplicação da HRIR original da posição 0° de azimute e 0° de elevação. O grau de diferença objetiva encontrado, o ODG, foi de -3.826. Esse valor situa-se entre as denominações "Irritante" e "Muito irritante". Em outras palavras, a aplicação da HRIR original sobre o sinal de áudio, por si só, já causa uma considerável degradação no sinal de áudio original.

6.5.2 Discussão acerca dos resultados

Considerando os testes realizados com apenas uma thread em execução para o cálculo de uma WHRTF, é possível constatar algumas particularidades em relação aos resultados encontrados ao comparar os tempos de execução em CPU e GPU. Considere então os tempos necessários para o cálculo de uma única WHRTF:

CPU:	0.007 s
GPU:	0.003 s

Tabela 6.7: Tempo (em segundos) necessário para o cálculo de uma WHRTF (1 thread) em CPU e GPU.

Quando se pensa em situações de processamento de áudio no ambiente de produção musical, logo vem à cabeça a necessidade de se ter um mecanismo de mixagem de áudio multicanal. Esse mecanismo faz com que canais dedicados a vários instrumentos, ou várias partes de um mesmo instrumento, sejam processados simultaneamente, possibilitando a audição dos mesmos em conjunto. Um exemplo desse resultado é o áudio gravado em um *Compact Disc* (CD). É sabido que os CDs possuem uma taxa de amostragem específica e que pode ser entendida no livro [74], onde o autor apresenta como o valor de 44.1 kHz foi escolhido. Dito isso, ao se pensar no uso de WHRTFs nesse ambiente de produção musical, pode-se pensar nos benefícios em utilizar o mecanismo implementado em GPU.

O código que faz o processamento do sinal de áudio em tempo real, processa blocos de 1024 amostras de áudio por vez. Considerando uma taxa de amostragem de 44100 Hz, 1024 amostras de áudio correspondem a aproximadamente 23 milisegundos (ms) de áudio. Assim, a cada 23 ms, um novo bloco de áudio é processado. Tomando como exemplo uma banda de rock que necessita de pelo menos cinco canais de áudio para suportar

os instrumentos que ela utiliza (nesse caso, uma bateria - considerando que ela utilize apenas um canal embora isso seja bastante improvável, um contra-baixo, duas guitarras e um microfone para voz), e supondo também que essa banda deseja se beneficiar do uso desse mecanismo de mixagem de áudio tridimensional em tempo real, é preciso que todos os instrumentos sejam processados paralelamente nesse intervalo de 23 ms. Para que cinco canais sejam processados sequencialmente em CPU, seria preciso, no mínimo, 35 ms. Entretanto, esse tempo é superior ao tempo de processamento de um bloco de áudio, que é de 23 ms. Nesse intervalo de tempo a CPU é capaz de processar apenas três instrumentos musicais. Com base nessas considerações, é possível afirmar que a abordagem em CPU não é viável para esta situação. Já se considerarmos o tempo necessário para o cálculo das WHRTFs em GPU, tem-se que o tempo para o processamento de cinco instrumentos musicais seria de aproximadamente 15 ms. Esse valor é inferior ao limite de 23 ms do processamento de um bloco de áudio. Dessa maneira, a GPU ainda seria capaz de processar mais dois instrumentos musicais além dos cinco já processados.

Supondo agora a utilização dessa ferramenta em mecanismos de alta definição tais como Blu-ray, HD DVD, dentre outros, também é possível tirar algumas conclusões. Sistemas de alta definição geralmente utilizam taxas de amostragem mais altas, por exemplo, 96 kHz. Baseando-se nessa taxa de amostragem de 96 kHz, considere a tabela 6.8 abaixo:

		# de WHRTFs processadas	
Amostras	**Tempo**	**CPU**	**GPU**
512	5.3 ms	0	1
1024	10.6 ms	1	3
2048	21.3 ms	3	7
4096	42.6 ms	6	14
8192	85.3 ms	12	28

Tabela 6.8: Número de WHRTFs processadas no intervalo de tempo correspondente ao número de amostras a uma taxa de amostragem de 96 kHz.

A Tabela 6.8 apresenta o número de WHRTFs possíveis que podem ser processadas em CPU e GPU tomando como base o tempo necessário para o cálculo de uma única WHRTF conforme apresentado pela tabela 6.7. Neste caso, são analisados os dados referentes a algum sistema com taxa de amostragem igual a 96 kHz. Para os diferentes números de amostra, a quantidade de WHRTFs que a GPU é capaz de processar é superior à CPU. No geral, a GPU processa 2.3 vezes mais WHRTFs que a CPU, ou seja, tem-se o dobro de *throughput* com a GPU.

Com essas informações, pode-se dizer que o uso da GPU no cálculo das WHRTFs é interessante pois possibilita o cálculo de mais WHRTFs que a CPU em um mesmo intervalo de tempo.

Capítulo 7

Conclusão e Trabalhos Futuros

Esse capítulo tem por objetivo concluir este estudo, apresentando e sintetizando o trabalho de pesquisa desenvolvido e listando as principais contribuições e resultados. Também nesse capítulo, as limitações encontradas são apresentadas bem como alguns possíveis trabalhos futuros e perspectivas de evolução deste.

7.1 Síntese do Projeto e Conclusões

Tendo em vista todo o assunto discutido nesse trabalho, pode-se dizer que o estudo aqui realizado alcançou o objetivo principal de implementar uma biblioteca que interpola HRTFs no domínio da transformada wavelet e realiza a síntese de áudio tridimensional em tempo real. Além disso, conclui-se que é possível otimizar o algoritmo de interpolação de HRTFs no domínio da transformada wavelet por meio do uso de unidades de processamento gráfico sem a perda de informações, já que nesse trabalho não foi preciso reduzir o número de coeficientes das HRIRs interpoladas no intuito de reduzir o volume de dados em processamento. A seguir recapitula-se cada uma das etapas no desenvolvimento desse trabalho.

Inicialmente, pesquisas sobre os sistemas de espacialização e auralização foram realizadas tendo como objetivo compreender os aspectos associados aos mecanismos de reprodução sonora desde os sistemas compostos por apenas um canal de saída até os sistemas multi-canais e biaurais. Assim, foram conhecidas as propriedades que nos permitem identificar o posicionamento de uma fonte sonora no campo ao nosso redor. Também, em virtude dessa pesquisa, os métodos de reprodução sonora para auralização foram conhecidos e manteve-se o foco no sistema de auralização biaural, objetivando a utilização de dois canais de saída para o desenvolvimento desse trabalho e visando a reprodução fiel da sensação de direcionalidade de uma fonte sonora.

Para entender como se dá o processo de síntese sonora tridimensional, que é a ferramenta utilizada para se obter um sistema de auralização eficiente, foram estudados os princípios da auralização bem como a maneira pela qual a síntese de áudio biaural é realizada. O propósito foi de encontrar a forma com que os sistemas de áudio biaurais funcionam pois assim se teria o conhecimento necessário para aplicar mudanças e melhorias ao método. Nessa etapa da pesquisa, procurou-se compreender também os rudimentos fundamentais para o entendimento do processamento de sinais digitais de áudio, haja

vista que a escolha de um método específico de convolução, por exemplo, pode implicar na degradação da performance do algoritmo em questão.

Após a pesquisa sobre os sistemas de auralização e como acontece a síntese de áudio biaural, foram estudadas as funções de transferência relativas à cabeça, que são as entidades fundamentais para este tipo de síntese mencionado anteriormente. Foi necessário pesquisar como se dá o processo de medição dessas funções e alguns dos métodos de interpolação que permitem obter os valores de funções de posições não conhecidas a partir de posições já conhecidas. A necessidade de entender alguns métodos de interpolação permitiu identificar aquele cujo algum trecho do algoritmo possibilitasse a execução em paralelo por meio do uso de unidades de processamento gráfico. Consequentemente, com a capacidade de se obter as funções referentes a qualquer posição no espaço tridimensional ao redor de um ouvinte, torna-se viável a criação de um sistema de áudio biaural.

Por fim, pesquisas sobre computação paralela por meio da utilização de unidades de processamento gráfico foram realizadas. O objetivo dessa pesquisa foi de entender o por quê dessas unidades também serem utilizadas para a solução de problemas não relacionados ao contexto gráfico, passando por um histórico da evolução desse tipo de unidade de processamento até a descrição do modelo de programação da arquitetura CUDA. Além disso, também pesquisou-se trabalhos anteriores relacionados ao processamento de áudio com o uso desse tipo de hardware. Como resultado dessa pesquisa, foram conhecidas as técnicas utilizadas no desenvolvimento de projetos de propósito geral com a utilização de GPU.

Ao selecionar o algoritmo de interpolação de HRTFs no domínio da transformada wavelet conforme o estudo apresentado na Seção 4.3, é possível dizer que esse projeto pode ser considerado, em parte, como uma evolução dos trabalhos anteriores [1, 11, 12, 13, 14].

A etapa de implementação desse trabalho foi sub-dividida em três sub-etapas de acordo com o ambiente de programação utilizado em cada uma. Na primeira, o código escrito em MATLAB serviu para esclarecer o funcionamento do algoritmo de interpolação selecionado para estudo e validar a sua funcionalidade. Também foi essencial no entendimento dos fundamentos básicos de processamento de sinais de áudio digital. Nessa mesma etapa, parte do método original (no que se refere ao cálculo dos pesos aplicados sobre as HRTFs utilizadas na interpolação) foi desconsiderada em favor da utilização de um modelo mais simples, embora aparentemente mais custoso computacionalmente. A seguir, todo o código utilizado na primeira etapa foi re-escrito em linguagem C na segunda etapa da implementação. Além disso, foram utilizadas tecnologias para a criação de uma interface gráfica (Cocoa), cálculo da transformada discreta de Fourier (FFTW) e reprodução de áudio (libSDL). Essa segunda etapa contribuiu para a solidificação do conhecimento do algoritmo de interpolação de HRTFs em questão e também permitiu a criação do protótipo do programa capaz de simular as mudanças de posicionamento de uma fonte sonora em tempo real. Estas duas etapas serviram como base para a implementação daquilo que é o foco desse trabalho: a implementação do algoritmo de interpolação de HRTFs no domínio da transformada wavelet utilizando unidades de processamento gráfico onde possível, visando a melhoria em performance. Nessa terceira etapa, conforme já mencionado no capítulo anterior, alguns trechos do algoritmo foram paralelizados utilizando a arquitetura CUDA. Assim, partes do código que antes eram executadas sequencialmente, agora são processadas em paralelo.

Após as três etapas implementadas, foram realizados testes de performance em que comparou-se o tempo gasto na execução do método de interpolação com os códigos criados na segunda e terceira etapas. Conforme apresentado na Seção 6.5, obteve-se uma melhora em performance com o programa implementado utilizando o paralelismo oferecido pela arquitetura CUDA. Conforme apresentado, a certas taxas de amostragem o *throughput* foi duas vezes superior quando executado em GPU. Além desses testes que avaliaram a performance, verificou-se também em todas as etapas, por meio de avaliação subjetiva, a síntese biaural de áudio com as HRTFs interpoladas. Em todas essas etapas, foram apresentados resultados semelhantes, ou seja, sinais de áudio que davam a mesma percepção de posicionamento. Também foi utilizado um método automatizado para avaliação subjetiva do resultado da aplicação das HRIRs sobre um sinal de áudio. Esse método, chamado de PEAQ, compara um sinal de referência com um de teste e retorna um valor chamado ODG, grau de diferença objetiva. Nesse trabalho, o sinal de referência é o sinal resultante da aplicação de uma HRIR original sobre um sinal de áudio. Já o sinal de teste é o sinal resultante da aplicação da HRIR interpolada sobre o mesmo sinal de áudio. Com base nesses dois sinais, de referência e teste e ambos referentes à mesma posição de elevação e azimute, obtém-se um valor que mede o prejuízo causado pela aplicação da HRIR interpolada em comparação à HRIR original. Nesse trabalho, a aplicação das HRIRs interpoladas sobre um sinal de áudio causou um prejuízo caracterizado entre "Perceptível, mas não irritante" e "Ligeiramente irritante".

Considerando os comentários anteriores, pode-se dizer que o trabalho desenvolvido permitiu alcançar os seguintes resultados:

1. Implementar uma biblioteca capaz de interpolar HRTFs no domínio da transformada wavelet e realizar a síntese de áudio tridimensional com desempenho em tempo real.

2. Interpolar HRTFs eficientemente com o uso de GPU. Ao implementar o método descrito no artigo [14] em um contexto de execução apenas em CPU e outro com funções sendo executadas em GPU, ainda que com algumas diferenças em relação ao texto original no que diz respeito ao cálculos dos pesos sobre os pontos utilizados para interpolação, mas visando a demonstração de que se obtém melhorias com a utilização de unidades de processamento gráfico, percebe-se que há um ganho em performance quando explora-se o paralelismo na execução de alguns trechos do código. A GPU é um recurso que está disponível na grande maioria dos computadores modernos e que geralmente está ocioso. Assim, é vantajoso utilizá-lo para o processamento massivo de dados.

3. Criar uma aplicação que permite reproduzir em tempo real o posicionamento selecionado por um usuário como resultado de sua interação com uma interface gráfica. O pré-processamento de HRTFs em menor tempo abre caminhos para o desenvolvimento de algoritmos que explorem ainda mais as GPUs para o processamento de áudio tridimensional e a construção de sistemas de auralização mais eficientes.

Tendo como base os resultados apresentados, outra importante contribuição desse trabalho foi a implementação de um sistema de interpolação de HRTFs em um ambiente híbrido composto pela interação entre uma unidade central de processamento e uma unidade de processamento gráfico. Isso permitiu o cálculo de HRTFs não conhecidas de maneira mais eficiente e possibilitou a construção de um programa capaz de reproduzir

em tempo real a mudança de posicionamento de uma fonte sonora como resultado da interação de um usuário com uma interface gráfica preparada para tal funcionalidade.

O código desenvolvido está disponibilizado como um projeto de código aberto (*open source*) em um repositório online que pode ser acessado pelo endereço eletrônico disponível na referência [75].

7.2 Limitações da Abordagem

Esse projeto, em razão de seu escopo e objetivos, apresenta limitações em alguns aspectos. Como primeiro exemplo, a implementação atual está limitada ao uso de apenas um filtro wavelet: Daubechies de tamanho oito. Embora o código inicial apresente a possibilidade de se utilizar outros filtros, como os biortogonais e Daubechies de tamanho quatro, essa capacidade não foi adicionada a partir da segunda etapa da implementação. Conforme já mencionado anteriormente, segundo os trabalhos [1, 12], os filtros protótipos da família Daubechies ofereciam distribuição de energia mais uniforme e, por esse motivo, optou-se pelo uso apenas desse filtro nesse trabalho.

A implementação além de estar limitada quanto à utilização dos filtros protótipos da transformada wavelet, também restringe-se ao uso de apenas um pequeno número de blocos de threads disponibilizados pela arquitetura CUDA. Esta limitação se deve pelo fato de que, para tornar o algoritmo mais eficiente e permitir que ele explore todo o potencial da GPU, seja necessário o projeto e a criação de um algoritmo mais complexo que o implementado atualmente e que demandaria mais tempo e conhecimento das particularidades de cada placa gráfica. Em outras palavras, a implementação de um método capaz de explorar todo o potencial da GPU no contexto da interpolação de HRTFs no domínio wavelet demanda conhecimento mais aprofundado de técnicas de paralelismo e otimização de dados em GPU. Em virtude do tempo disponível, optou-se apenas em implementar um método em GPU que pudesse trazer benefícios, em termos de performance, ao processo de interpolação.

Outro fator limitante desse trabalho é o fato deste poder funcionar apenas com um único valor para a taxa de amostragem. Seria interessante pesquisar HRTFs medidas em condições cuja a taxa de amostragem fosse maior, possibilitando assim, trabalhar com maior facilidade em sistemas de áudio de alta definição.

7.3 Trabalhos Futuros

A implementação proposta no presente trabalho ainda é passível de melhorias e otimizações. Entretanto, mostra-se relevante por ser mais um passo em direção ao uso de GPUs para o processamento de áudio e para a criação de sistemas de auralização eficientes.

Como trabalho futuro, pode-se projetar o algoritmo de interpolação de HRTFs para placas cuja capacidade seja 2.x. Com esse tipo de hardware, é possível executar kernels concorrentes, o que permitiria o uso de um número maior de blocos de threads para o processamento. Outra abordagem seria a reimplementação desse método de forma que um único kernel possa invocar um número maior de blocos de threads. Essa segunda estratégia demanda bom conhecimento referente à arquitetura de processamento massivo

de dados CUDA e seria a solução para uma das limitações apresentadas na Seção 7.2 anterior.

Outra funcionalidade que pode ser adicionada a esse trabalho futuramente é a capacidade de utilizar filtros protótipos de outros tipos e famílias de transformadas wavelet. Até o presente momento o algoritmo está limitado à utilização da família Daubechies de tamanho igual a oito. A utilização de outros filtros poderia propiciar melhorias tanto em termos de performance quanto em termos da qualidade do sinal produzido com o uso de outros filtros wavelet. Além disso, a implementação dessa funcionalidade irá permitir a comparação entre as diferentes famílias wavelet e a listagem de vantagens e desvantagens de cada uma delas.

Mais uma possibilidade de implementação de projetos tendo como base a pesquisa realizada e documentada nesse estudo é a implementação de uma aplicação de mixagem tridimensional de áudio. Para músicos e produtores musicais, explorar a capacidade de auralização onde a aplicação de funções sobre uma fonte sonora transmita uma sensação fiel de direcionalidade do som ao ouvinte é de extrema relevância pois irá oferecer novas maneiras que possam estimular diferentes (ou até mesmo inéditas) emoções nas pessoas. No campo da música, bem como no cinema, essas duas formas de arte, como formas de entretenimento, podem se beneficiar desse mecanismo ao criar um novo conceito no mercado em que os sistemas multicanais seriam substituídos por sistemas biaurais. As vantagens dessa utilização são interessantes pois, ao exigir o uso de fones de ouvido para a reprodução de áudio biaural, torna-se dispensável o uso de caixas de som contendo alto-falantes que geralmente são mais caros que fones de ouvido. Além disso, em locais em que deve haver a preocupação com a acústica de tal forma que o som não deva incomodar as instalações ao redor, a utilização de fones de ouvido se torna mais vantajosa e ideal à situação.

Também valeria a pena investigar meios pelos quais o sinal não sofresse tanto prejuízo devido a aplicação das características espectrais de posicionamento da fonte sonora, proporcionado pelo uso das HRTFs. Conforme resultados apresentados, a simples aplicação da HRIR original da posição em 0° de azimute e 0° de elevação faz com que o grau de diferença objetiva seja próximo do nível mais alto de prejuízo.

Além das possibilidades acima descritas, também é interessante a adaptação do sistema de posicionamento tridimensional de áudio desenvolvido nesse trabalho, em aplicações utilizadas para mixagem de áudio. A ideia é criar plugins capazes de serem adicionados em tecnologias já consolidadas no mercado de produção musical ou no cinema. Também há a possibilidade de ampliação da capacidade do aplicativo desenvolvido, para que ele passe a suportar mais de um canal de áudio.

Finalmente, outra possibilidade de evolução desse trabalho é a aplicação deste em sistemas reais de simulação acústica de salas e sistemas de realidade virtual. O algoritmo de interpolação de HRTFs em estudo nesse trabalho é apresentado na dissertação [1] como parte de um mecanismo de simulação acústica de salas. Portanto, seria interessante aplicar os resultados atingidos no presente trabalho em situações já documentadas anteriormente.

Apêndice A

Trechos de Código-fonte

A.1 Código em FFTW

A função A.1, recebe como argumento o sinal representado no domínio dos números reais. Primeiramente aloca-se espaço em memória para armazenar esse mesmo sinal de entrada no domínio dos números complexos. Após alocar espaço em memória, inicializa-se a representação do sinal no domínio dos números complexos a partir do mesmo no domínio dos reais. Em seguida, o mesmo é feito para o filtro. Entretanto, o filtro representado no domínio dos números complexos é criado a partir dos valores do filtro representado no domínio dos números reais e o seu tamanho é igualado ao tamanho do sinal. Para isso, zeros são incluídos até que o filtro tenha o mesmo tamanho do sinal de entrada.

Após esse procedimento de inicialização, os planos de execução da DFT em uma dimensão para o sinal e o filtro são criados e executados. Assim, aplica-se a FFT sobre o sinal e o filtro respectivamente. Os coeficientes resultantes da operação anterior são multiplicados e o resultado é, então, normalizado. Com esse resultado em mãos, cria-se o plano de execução da transformada inversa de Fourier e, enfim, executa-se esse procedimento.

Assim, o resultado da convolução pode ser recuperado e a função pode ser finalizada após a liberação dos recursos alocados até então.

Função A.1: convFFT - Convolução via FFT

```
float* convFFT(float* signal, unsigned int signalLength, float* filter, unsigned int
  filterLength) {
  // Allocate host memory for the signal
  fftw_complex *h_signal = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) *
    signalLength);
  fftw_complex *h_signal_out = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) *
    signalLength);;
  fftw_complex *h_signal_out2 = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) *
    signalLength);;

  // Initalize the memory for the signal
  for (unsigned int i = 0; i < signalLength; ++i) {
    h_signal[i][0] = signal[i];
    h_signal[i][1] = 0.0;
  }

  // Allocate host memory for the filter
  fftw_complex* h_filter_kernel = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
    signalLength);
  fftw_complex* h_filter_kernel_out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
    signalLength);
```

```
  // Initalize the memory for the filter
  for (unsigned int i = 0; i < signalLength; ++i) {
    if (i < filterLength) {
      h_filter_kernel[i][0] = filter[i];
      h_filter_kernel[i][1] = 0.0;
    } else {
      h_filter_kernel[i][0] = 0.0;
      h_filter_kernel[i][1] = 0.0;
    }
  }

  int new_size = signalLength;
  int mem_size = sizeof(fftw_complex) * new_size;

  fftw_plan planSignal, planFilter, planInv;
  planSignal = fftw_plan_dft_1d(new_size, (fftw_complex *) h_signal, (fftw_complex *)
    h_signal_out, FFTW_FORWARD, FFTW_MEASURE);
  planFilter = fftw_plan_dft_1d(new_size, (fftw_complex *) h_filter_kernel, (fftw_complex
     *) h_filter_kernel_out, FFTW_FORWARD, FFTW_MEASURE);

  fftw_execute(planSignal);
  fftw_execute(planFilter);

  // Multiply the coefficients together and normalize the result
  complexPointwiseMulAndScale(h_signal_out, h_filter_kernel_out, new_size, 1.0f /
    new_size);

  planInv = fftw_plan_dft_1d(new_size, (fftw_complex *) h_signal_out, (fftw_complex *)
    h_signal_out2, FFTW_BACKWARD, FFTW_MEASURE);
  fftw_execute(planInv);

  float* convolvedSignal = (float*) calloc(new_size, sizeof(float));
  for (int i = 0; i < new_size; i++) {
    convolvedSignal[i] = h_signal_out2[i][0];
  }

  fftw_destroy_plan(planSignal);
  fftw_destroy_plan(planFilter);
  fftw_destroy_plan(planInv);

  fftw_free(h_filter_kernel);
  fftw_free(h_filter_kernel_out);
  fftw_free(h_signal);
  fftw_free(h_signal_out);
  fftw_free(h_signal_out2);

  return convolvedSignal;
}
```

A.2 Código em libSDL

A função A.2, escrita em Objective-C, apresenta o código executado quando o usuário clica sobre o botão que inicializa a reprodução de áudio. A biblioteca é inicializada e o formato de áudio é configurado. O dispositivo de áudio é aberto e inicia-se a execução dos trechos de áudio até que todos os blocos que compõem esse sinal sejam reproduzidos. É interessante mencionar que antes do sinal ser reproduzido, cada bloco é processado pela função A.3 que será detalhada adiante.

Função A.2: playSelectedAudio - Função para reproduzir áudio utilizando libSDL

```
- (void) playSelectedAudio {
  Uint8* audioStream = wavFile.streamUint8;
  int audioLengthValue = wavFile.size;
```

```
  if( SDL_Init(SDL_INIT_TIMER | SDL_INIT_AUDIO ) < 0 ) {
    return;
  }

  SDL_AudioSpec wav_spec = wavFile.wavSpec;
  SDL_AudioSpec wanted, obtained;

  /* Set the audio format */
  wanted.freq = wav_spec.freq;
  wanted.format = wav_spec.format;
  wanted.channels = wav_spec.channels;  /* 1 = mono, 2 = stereo */
  wanted.samples = wav_spec.samples;  /* Good low-latency value for callback */
  wanted.callback = fillAudio;
  wanted.userdata = NULL;

  /* Open the audio device, forcing the desired format */
  if ( SDL_OpenAudio(&wanted, &obtained) < 0 ) {
    fprintf(stderr, "Couldn't open audio: %s\n", SDL_GetError());
    SDL_PauseAudio(0);
    return;
  }

  audio_len = audioLengthValue;
  audio_chunk = audioStream;
  audio_pos = audio_chunk;

  /* Do some processing */
  WavFile *wavfile = [self wavFile];
  originalStream = [wavfile stream];

  /* Let the callback function play the audio chunk */
  SDL_PauseAudio(0);

  /* Wait for sound to complete */
  while ( audio_len > 0 ) {
    SDL_Delay(100);         /* Sleep 1/10 second */
      }

  if (!stopWasPressed) {
    [self resetAudioSettings];
  }

  return;
}
```

A função abaixo é executada para processar blocos de tamanho igual a *len*. Nesse processo verifica-se se há dados a serem processados e, caso tenha, recupera-se as HRIRs referentes às orelhas esquerda e direita da posição selecionada pelo usuário. Aplica-se a operação de convolução com FFT dos filtros representados pelas HRIRs esquerda e direita sobre o sinal de áudio. Após esse procedimento, o sinal de áudio é mixado e reproduzido pela interface de áudio.

Função A.3: fillAudio - Função auxiliar utilizada pela função A.2

```
void fillAudio(void *udata, Uint8 *stream, int len) {
  /* Only play if we have data left */
  int indexAzim = staticAzimuth;
  int indexElev = staticElevation;

  WhrtfForPositionBean *whrtfForPositionLocal = whrtfs[indexElev+40][indexAzim];
  if ( audio_len == 0 ) {
    return;
  }

  // Mix as much data as possible
  len = ( len > audio_len ? audio_len : len );
```

```
  Uint8 *originalData = (Uint8 *) malloc(len * sizeof(Uint8));
  memcpy(originalData, audio_pos, len * sizeof(Uint8));

  int floatArrayLength;
  float** data = convertToFloatArray(originalData, len, &floatArrayLength);

  int PL = whrtfForPositionLocal.whrtfLeftLength;
  int PR = whrtfForPositionLocal.whrtfRightLength;
  int xrLength = floatArrayLength + PL;

  float** Xr = (float **) calloc(2, sizeof(float*));
  Xr[0] = (float*) calloc(xrLength, sizeof(float));
  Xr[1] = (float*) calloc(xrLength, sizeof(float));

  for (int n = 0; n < xrLength; n++) {
    int indexL = (n + (r)*(xrLength-PL+1)-PL);
    int indexR = (n + (r)*(xrLength-PR+1)-PR);

    // Left
    if (indexL < 0) {
      Xr[0][n] = 0.0;
    } else {
      Xr[0][n] = originalStream[0][indexL];
    }

    // Right
    if (indexR < 0) {
      Xr[1][n] = 0.0;
    } else {
      Xr[1][n] = originalStream[1][indexR];
    }
  }
  r++;

  int deltaL, deltaR;
  calculaITD(indexAzim, 'L', &deltaL, &deltaR);

  float** aux = (float**) calloc(2, sizeof(float*));

  float *convAux = convFFT(Xr[0], xrLength, whrtfForPositionLocal.whrtfLeft,
    whrtfForPositionLocal.whrtfLeftLength);
  aux[0] = shiftToRight(convAux, xrLength, deltaL);
  convAux = convFFT(Xr[1], xrLength, whrtfForPositionLocal.whrtfRight,
    whrtfForPositionLocal.whrtfRightLength);
  aux[1] = shiftToRight(convAux, xrLength, deltaR);

  float** aux2 = (float**) calloc(2, sizeof(float*));
  aux2[0] = (float*) calloc(floatArrayLength, sizeof(float));
  aux2[1] = (float*) calloc(floatArrayLength, sizeof(float));

  int j = 0;
  for (int i = PL; i < xrLength; i++) {
    aux2[0][j] = aux[0][i];
    aux2[1][j] = aux[1][i];
    j++;
  }

  Uint8 *audioStream  = convertToUint8Array(aux2, len);

  SDL_MixAudio(stream, audioStream, len, SDL_MIX_MAXVOLUME);

  free(convAux);
  free(aux[0]); free(aux[1]); free(aux);
  free(Xr[0]); free(Xr[1]); free(Xr);
  free(data[0]); free(data[1]); free(data);

  audio_pos += len;
  audio_len -= len;
}
```

A.3 Código em CUDA

As funções a seguir se propõem a fazer o que é especificado pela equação 4.30 responsável por calcular a HRIR a partir dos coeficientes esparsos por meio da convolução dos filtros de análise com tais coeficientes.

Função A.4: resp_imp - Função que calcula a HRIR a partir dos coeficientes esparsos

```
float* resp_imp(int elev, int azim, char* filtros[], int numFiltros, float** G, int*
  G_size, int* resultLength) {
  float** filterBank = (float**) calloc((numFiltros + 1), sizeof(float));
  int* filterBankLength = (int*) calloc((numFiltros + 1), sizeof(int));

  cascataSimple(filtros, numFiltros, filterBank, filterBankLength);

  int widthG = max(G_size, numFiltros+1) + max(atrasos, numFiltros+1);
  int height = numFiltros + 1;

  float *d_resultado, *h_resultado;
  float *d_G, *d_filterBank, *h_G, *h_filterBank;
  int *d_GSize, *d_filterBankLength, *h_rSize, *d_rSize;

  h_G = (float*) malloc(MAX_NUM_OF_THREADS * height * sizeof(float));
  h_filterBank = (float*) malloc(MAX_NUM_OF_THREADS * height * sizeof(float));
  h_rSize = (int*) calloc(height, sizeof(int));
  h_resultado = (float*) calloc(2 * MAX_NUM_OF_THREADS, sizeof(float));

  for (int i = 0; i < height; i++) {
    for (int j = 0; j < MAX_NUM_OF_THREADS; j++) {
      if (j < widthG) {
        h_G[j + i*MAX_NUM_OF_THREADS] = G[i][j];
      } else {
        h_G[j + i*MAX_NUM_OF_THREADS] = 0.0;
      }

      if (j < filterBankLength[i]) {
        h_filterBank[j + i*MAX_NUM_OF_THREADS] = filterBank[i][j];
      } else {
        h_filterBank[j + i*MAX_NUM_OF_THREADS] = 0.0;
      }
    }
  }

  cudaMalloc((void**) &d_GSize, height * sizeof(int));
  cudaMalloc((void**) &d_filterBankLength, height * sizeof(int));
  cudaMalloc((void**) &d_rSize, height * sizeof(int));
  cudaMalloc((void**) &d_resultado, 2 * MAX_NUM_OF_THREADS * sizeof(float));

  cudaMalloc((void**) &d_G, MAX_NUM_OF_THREADS * height * sizeof(float));
  cudaMalloc((void**) &d_filterBank, MAX_NUM_OF_THREADS * height * sizeof(float));

  cudaMemcpy(d_G, h_G, MAX_NUM_OF_THREADS * height * sizeof(float),
    cudaMemcpyHostToDevice);
  cudaMemcpy(d_GSize, G_size, height * sizeof(int), cudaMemcpyHostToDevice);
  cudaMemcpy(d_filterBank, h_filterBank, MAX_NUM_OF_THREADS * height * sizeof(float),
    cudaMemcpyHostToDevice);
  cudaMemcpy(d_filterBankLength, filterBankLength, height * sizeof(int),
    cudaMemcpyHostToDevice);

  respImpCuda<<<height, MAX_NUM_OF_THREADS>>>(height, widthG, d_G, d_GSize, d_filterBank,
     d_filterBankLength, d_rSize, d_resultado);

  cudaMemcpy(h_resultado, d_resultado, 2 * MAX_NUM_OF_THREADS * sizeof(float),
    cudaMemcpyDeviceToHost);
  cudaMemcpy(h_rSize, d_rSize, height * sizeof(int), cudaMemcpyDeviceToHost);

  int maxRSize = max(h_rSize, height);
  int maxFilterBankLength = max(filterBankLength, height);
```

```
  *resultLength = (maxRSize - maxFilterBankLength);
  float* respostaImpulsiva = (float*) calloc(*resultLength, sizeof(float));
  int j = 0;
  for (int i = maxFilterBankLength; i < maxRSize; i++) {
    respostaImpulsiva[j++] = h_resultado[i];
  }

  cudaFree(d_resultado);
  cudaFree(d_G); cudaFree(d_GSize);
  cudaFree(d_filterBank); cudaFree(d_filterBankLength);
  cudaFree(d_rSize);

  for (int i = 0; i < (numFiltros + 1); i++) {
    free(filterBank[i]);
  }

  free(filterBank); free(filterBankLength);
  free(h_G); free(h_filterBank);
  free(h_rSize); free(h_resultado);

  return respostaImpulsiva;
}
```

Função A.5: respImpCuda - Função CUDA executada em paralelo

```
__global__ void respImpCuda(int height, int widthG, float* dev_G, int* G_size, float*
  filterBank, int* filterBankLength, int* r_size, float* resultado) {
  int tidX = threadIdx.x + blockIdx.x * blockDim.x;
  int atrasos[5] = {1, 1, 8, 22, 50};

  L = (int) (blockIdx.x == 0) ? powf(2.0, height-1) : powf(2.0, height-blockIdx.x);
  partialVecLength = (G_size[blockIdx.x] + atrasos[blockIdx.x]);
  resultLength = (((partialVecLength - 1) * L) + 1);
  convLength = (filterBankLength[blockIdx.x] + resultLength - 1);
  r_size[blockIdx.x] = convLength;

  // Esparsando os coeficientes de G
  if (threadIdx.x % L == 0) {
    gaux[threadIdx.x] = dev_G[(threadIdx.x / L) + (blockIdx.x * blockDim.x)];
  } else {
    gaux[threadIdx.x] = 0.0;
  }
  if ((threadIdx.x + blockDim.x) < resultLength) {
    int index = threadIdx.x + blockDim.x;

    if (index % L == 0) {
      gaux[index] = dev_G[(index / L) + (blockIdx.x * blockDim.x)];
    } else {
      gaux[index] = 0.0;
    }
  }

  if (threadIdx.x < filterBankLength[blockIdx.x]) {
    fbaux[threadIdx.x] = filterBank[tidX];
  }

  if (threadIdx.x == 0) { // This is executed just once in a block
    maxLength = (filterBankLength[blockIdx.x] >= resultLength ? filterBankLength[blockIdx
      .x] : resultLength);
    minLength = (filterBankLength[blockIdx.x] <= resultLength ? filterBankLength[blockIdx
      .x] : resultLength);
  }

  __syncthreads();

  int index = 0;
  if (threadIdx.x < convLength) {
    index = threadIdx.x;
```

```
    float convResult = convCalc(index, fbaux, filterBankLength[blockIdx.x], gaux,
      resultLength, maxLength, minLength);
    convolutionResult[index + blockIdx.x * blockDim.x * 2] = convResult;

    if (index + blockDim.x < convLength) {
      index = index + blockDim.x;
      convResult = convCalc(index, fbaux, filterBankLength[blockIdx.x], gaux,
        resultLength, maxLength, minLength);
      convolutionResult[index + blockIdx.x * blockDim.x * 2] = convResult;
    }
  }

  index = threadIdx.x + blockDim.x;

  resultado[threadIdx.x] = convolutionResult[threadIdx.x + (0 * blockDim.x * 2)] +
              convolutionResult[threadIdx.x + (1 * blockDim.x * 2)] +
              convolutionResult[threadIdx.x + (2 * blockDim.x * 2)] +
              convolutionResult[threadIdx.x + (3 * blockDim.x * 2)] +
              convolutionResult[threadIdx.x + (4 * blockDim.x * 2)];

  resultado[index] = convolutionResult[index+ (0 * blockDim.x * 2)] +
            convolutionResult[index + (1 * blockDim.x * 2)] +
            convolutionResult[index + (2 * blockDim.x * 2)] +
            convolutionResult[index + (3 * blockDim.x * 2)] +
            convolutionResult[index + (4 * blockDim.x * 2)];
}
```

Apêndice B

Gráficos: C vs CUDA

Nesse apêndice serão apresentados alguns gráficos comparando o tempo gasto no cálculo de WHRTFs em C e CUDA, com um número variado de threads criadas na CPU em 20 execuções sucessivas. Cada thread é responsável pelo cálculo de uma WHRTF. Os gráficos abaixo foram criados com base nos valores apresentados pelas Tabelas B.1 e B.2.

Esses gráficos são parte dos resultados apresentados na Seção 6.5 dessa dissertação.

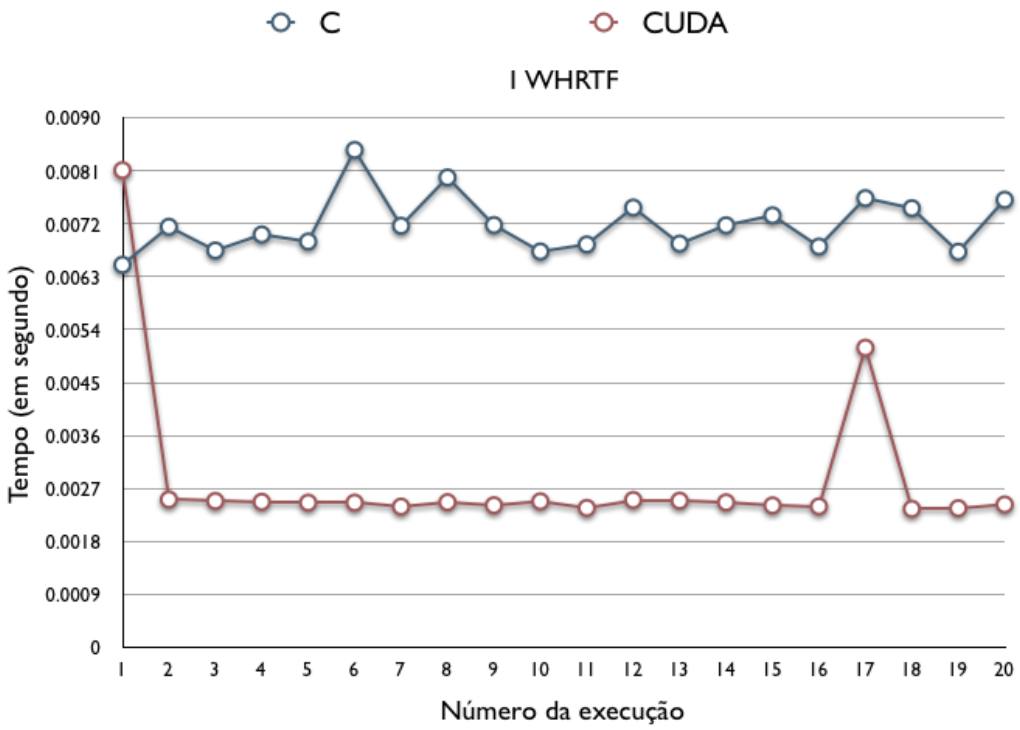

Figura B.1: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de uma única WHRTF.

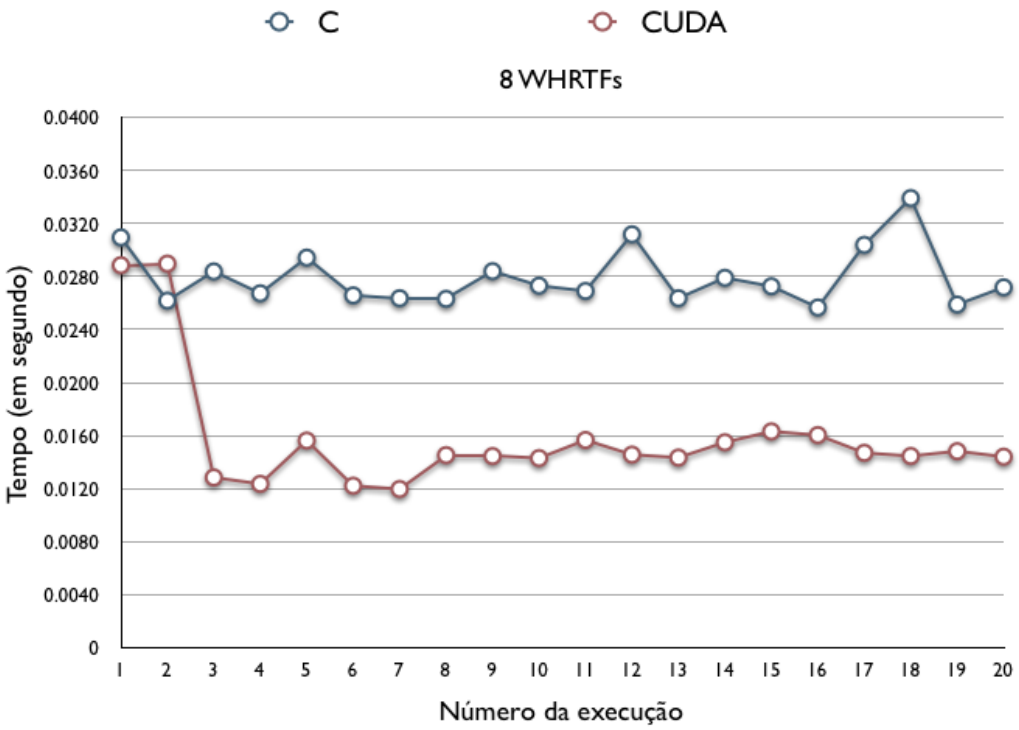

Figura B.2: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 8 WHRTFs.

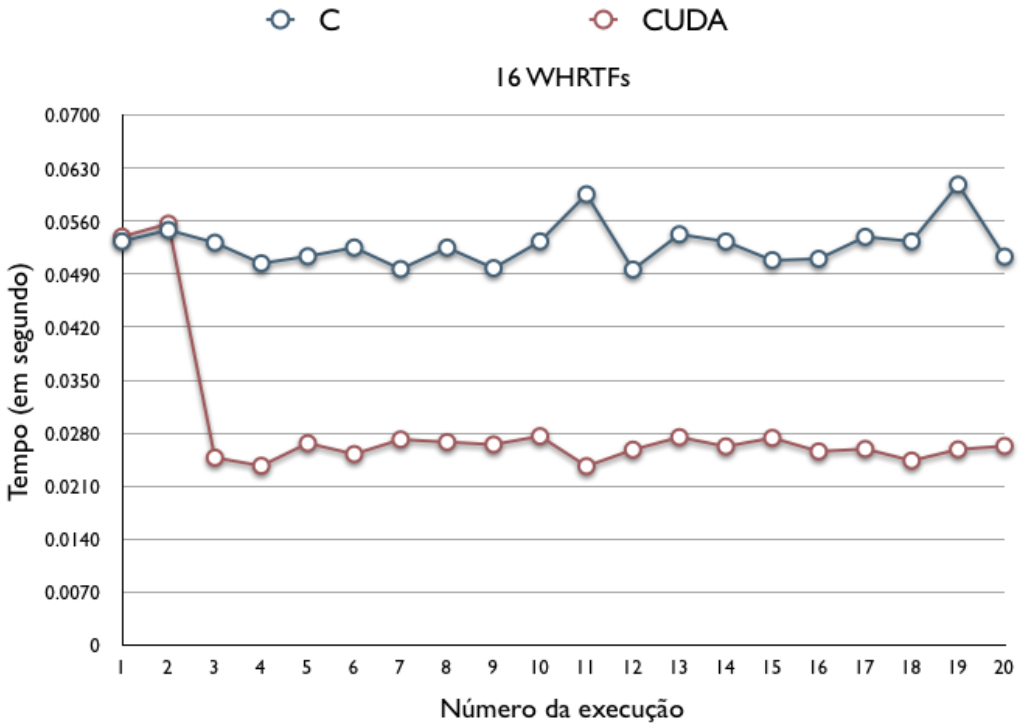

Figura B.3: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 16 WHRTFs.

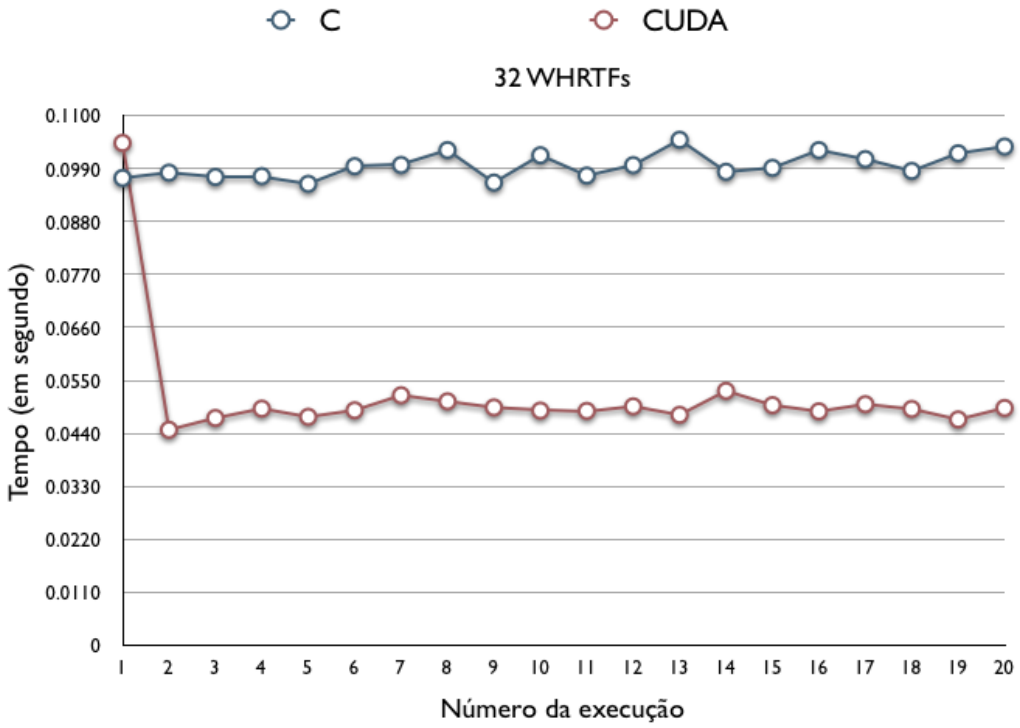

Figura B.4: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 32 WHRTFs.

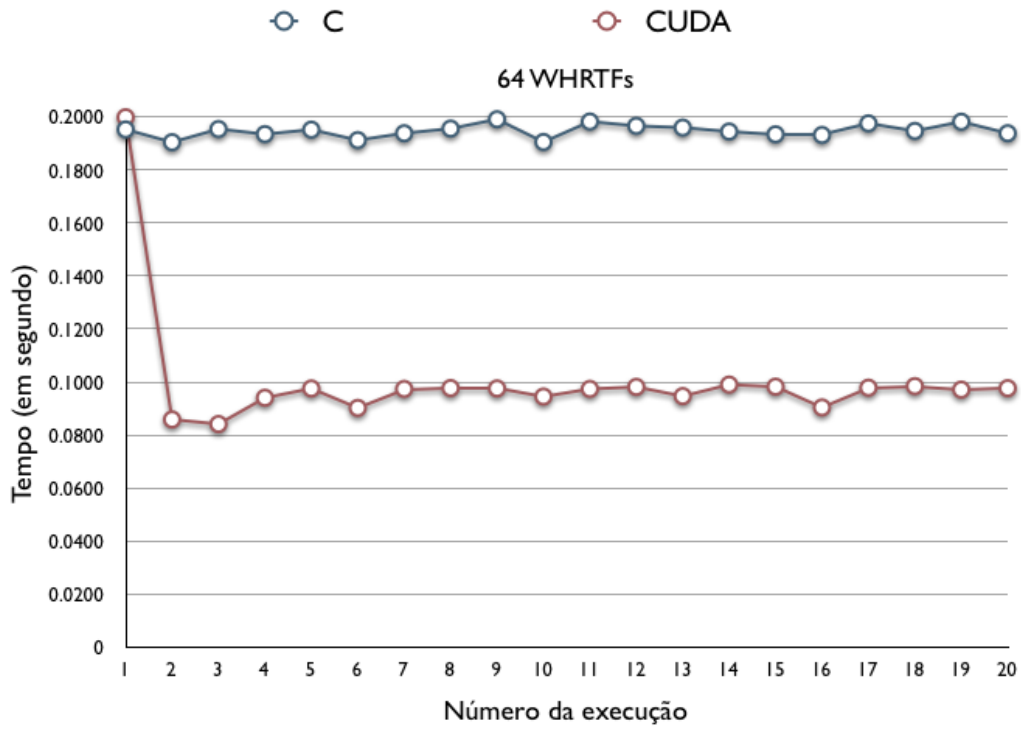

Figura B.5: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 64 WHRTFs.

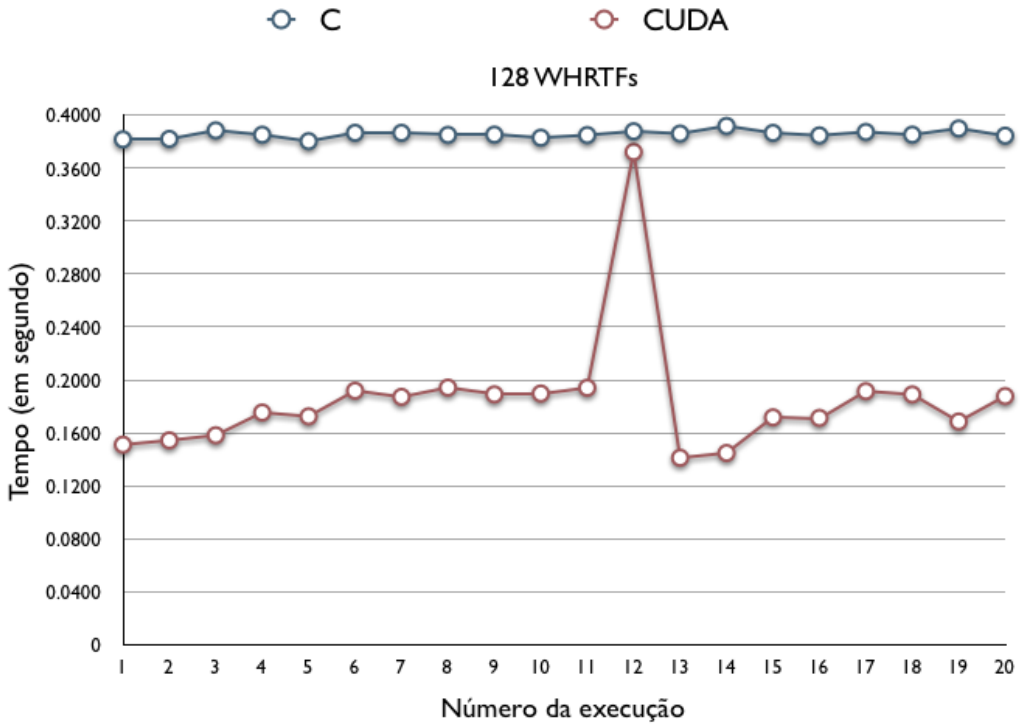

Figura B.6: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 128 WHRTFs.

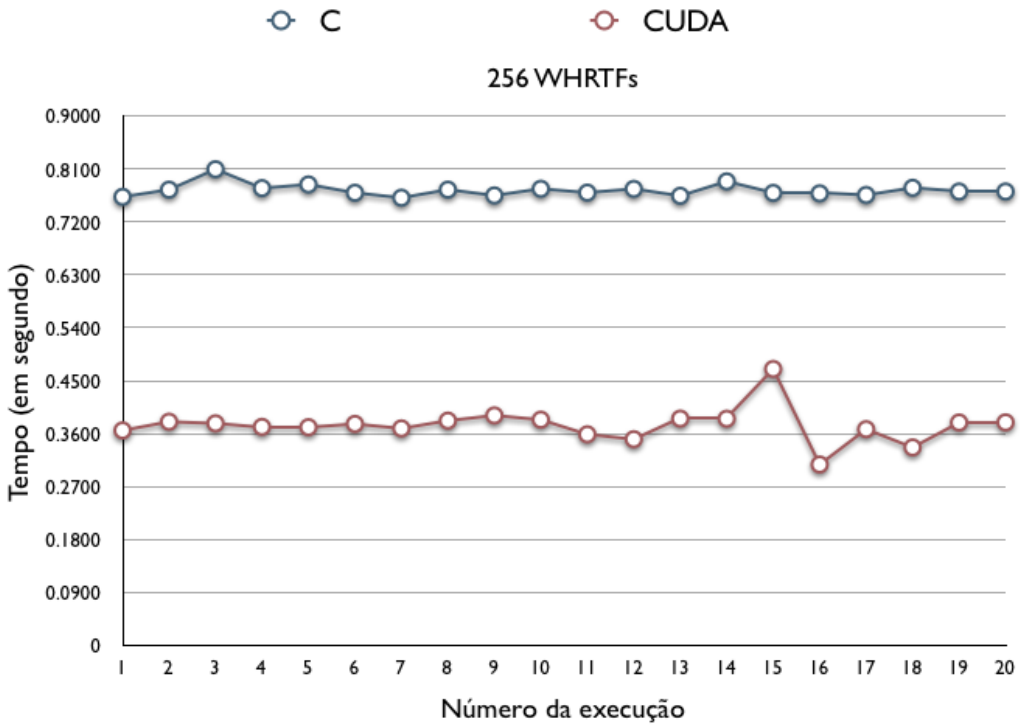

Figura B.7: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 256 WHRTFs.

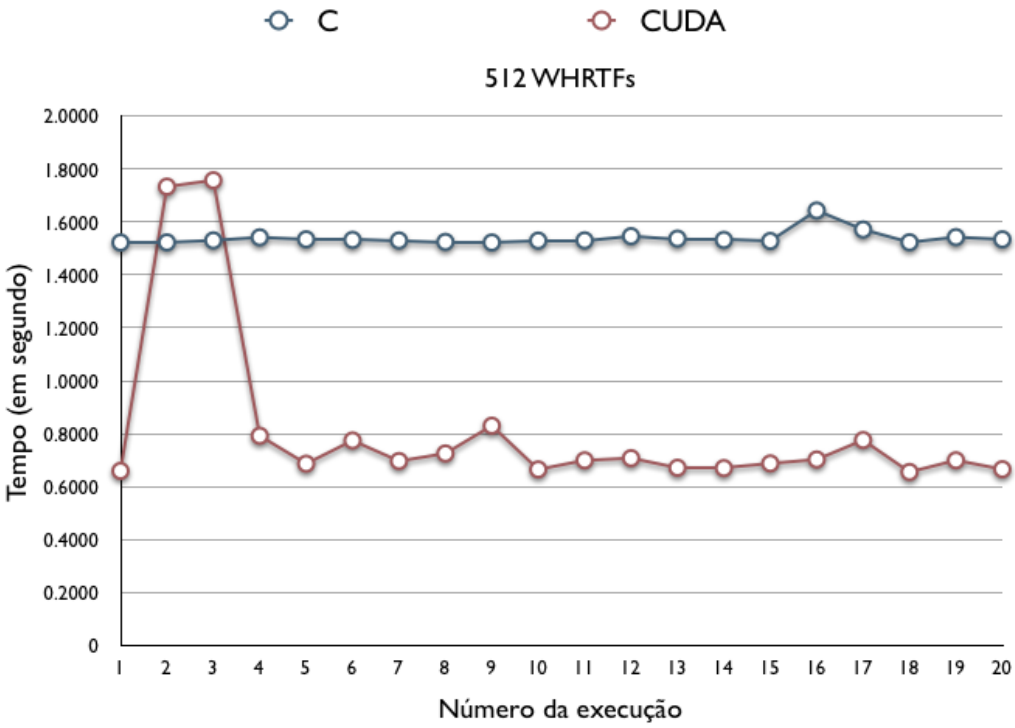

Figura B.8: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 512 WHRTFs.

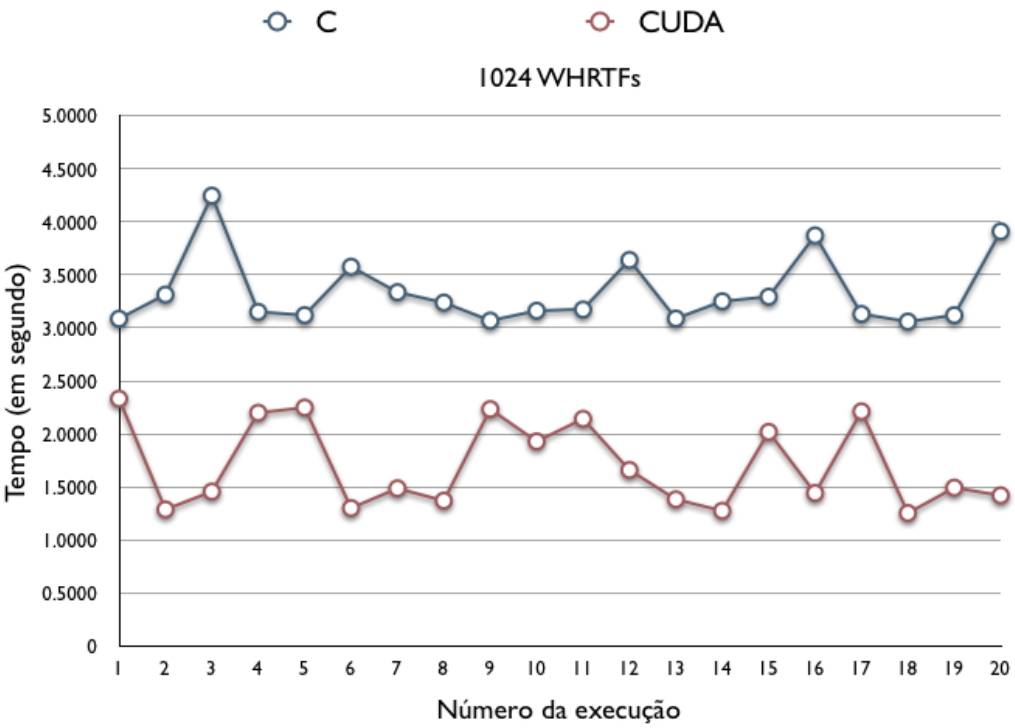

Figura B.9: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 1024 WHRTFs.

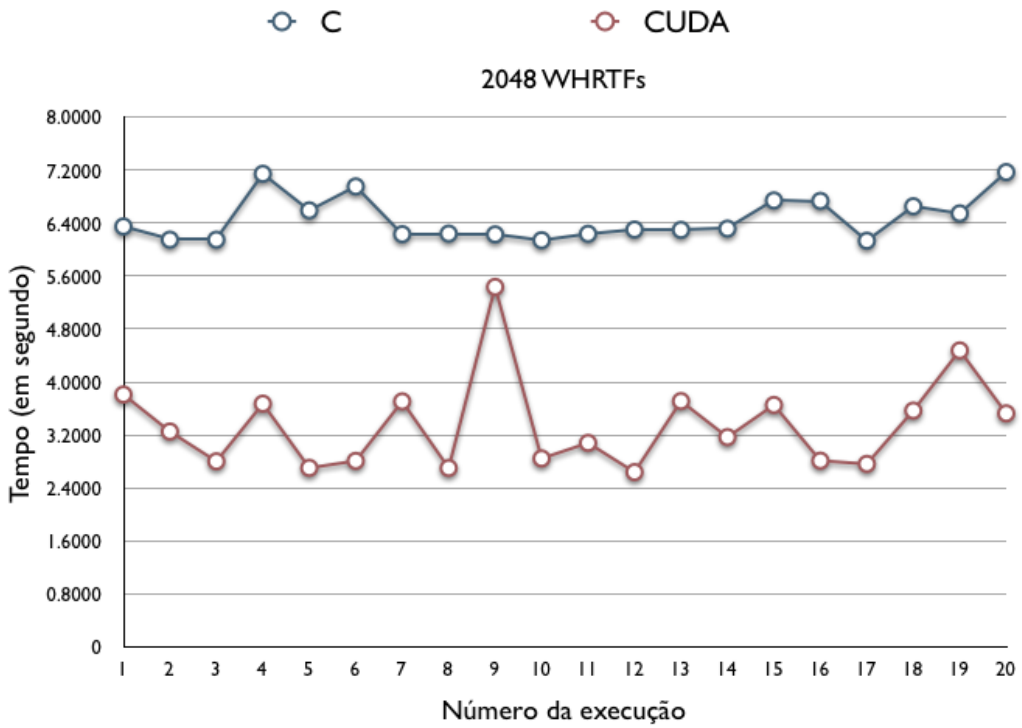

Figura B.10: Comparativo entre o tempo de execução em C e CUDA em 20 execuções sucessivas para o cálculo de 2048 WHRTFs.

C - # de WHRTFs									
1	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
0.0065	0.030938	0.053322	0.097012	0.195143	0.381504	0.762314	1.521762	3.084627	6.346336
0.007146	0.026172	0.054814	0.09812	0.190341	0.381842	0.774422	1.522211	3.312082	6.146906
0.006744	0.02837	0.053139	0.097223	0.195287	0.388285	0.809166	1.529638	4.244123	6.146451
0.007015	0.026722	0.050415	0.097387	0.193385	0.385008	0.777056	1.541157	3.149373	7.140499
0.006896	0.029412	0.051347	0.095801	0.195064	0.380151	0.783286	1.533763	3.118354	6.588726
0.008451	0.026567	0.052496	0.099463	0.191128	0.386437	0.769157	1.531565	3.574639	6.950282
0.007163	0.026353	0.049662	0.099704	0.193706	0.386438	0.760741	1.529369	3.334722	6.229543
0.007985	0.026303	0.052492	0.102801	0.195469	0.384878	0.774489	1.524056	3.236551	6.238053
0.007175	0.028389	0.049778	0.096048	0.198983	0.385072	0.764672	1.521835	3.066918	6.226173
0.006726	0.027298	0.053314	0.101726	0.190423	0.38276	0.775951	1.528893	3.16027	6.138789
0.006846	0.026898	0.059528	0.097515	0.198118	0.384734	0.769433	1.529685	3.171477	6.235222
0.007473	0.031168	0.049592	0.09967	0.19651	0.387598	0.775791	1.545626	3.638023	6.299431
0.00686	0.026353	0.054246	0.104938	0.195845	0.385866	0.764054	1.536381	3.086367	6.294404
0.007171	0.02789	0.05329	0.098295	0.194361	0.391631	0.788496	1.531368	3.249139	6.313447
0.00734	0.027253	0.050828	0.099108	0.193309	0.38638	0.76892	1.527409	3.292491	6.73882
0.00681	0.025653	0.051008	0.102785	0.193055	0.384487	0.768174	1.642681	3.869496	6.731912
0.007633	0.030374	0.053913	0.100926	0.197438	0.38711	0.765436	1.570621	3.129177	6.130782
0.007462	0.033899	0.053312	0.098471	0.194589	0.385058	0.777602	1.522624	3.05788	6.651086
0.006722	0.025876	0.060819	0.102136	0.198053	0.389731	0.77132	1.54196	3.118065	6.543319
0.007605	0.027172	0.051303	0.103536	0.193717	0.384292	0.77134	1.533652	3.90795	7.166474

Tabela B.1: Tempos de execução obtidos no cálculo, em C, dos respectivos números de WHRTFs.

CUDA - # de WHRTFs									
1	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
0.008107	0.028843	0.05393	0.104287	0.199747	0.151089	0.365392	0.65943	2.331459	3.809599
0.002512	0.02896	0.055647	0.044735	0.085852	0.154358	0.380169	1.732668	1.287799	3.251088
0.002486	0.012839	0.024753	0.047182	0.08412	0.158173	0.377375	1.756656	1.456396	2.801309
0.002472	0.012354	0.023681	0.049124	0.094028	0.175352	0.371372	0.791951	2.199	3.671479
0.002458	0.015622	0.026659	0.047428	0.097522	0.172447	0.370837	0.68657	2.247748	2.702551
0.002458	0.012214	0.025225	0.048784	0.090201	0.191819	0.376385	0.774199	1.300968	2.807906
0.002391	0.011966	0.02716	0.0519	0.097356	0.187152	0.368778	0.696614	1.4864	3.70581
0.002466	0.014522	0.026812	0.050638	0.097678	0.194155	0.381988	0.725163	1.370466	2.700433
0.002414	0.014449	0.026496	0.049373	0.097544	0.18903	0.390829	0.830018	2.23305	5.433791
0.00248	0.014298	0.027589	0.048722	0.094479	0.189669	0.38359	0.664981	1.928643	2.844555
0.002369	0.015659	0.023626	0.04866	0.097358	0.194012	0.358702	0.699354	2.141799	3.079378
0.002504	0.01456	0.025805	0.049598	0.098118	0.372049	0.350436	0.707448	1.659306	2.638454
0.002491	0.014337	0.027463	0.047835	0.094643	0.14118	0.385404	0.671275	1.383016	3.710282
0.002463	0.015489	0.026239	0.052814	0.098983	0.144789	0.385208	0.66985	1.275364	3.166575
0.002411	0.01631	0.027364	0.049846	0.098208	0.17169	0.469727	0.687649	2.018441	3.651921
0.002395	0.016034	0.025581	0.048569	0.090382	0.171269	0.307429	0.702437	1.44244	2.810933
0.005091	0.014695	0.025904	0.050042	0.097821	0.191433	0.367434	0.775691	2.211568	2.762532
0.002353	0.014456	0.024365	0.049055	0.098406	0.188994	0.336482	0.655287	1.253854	3.56731
0.002364	0.014819	0.025841	0.046882	0.097102	0.168557	0.378857	0.699475	1.492906	4.474867
0.002427	0.014404	0.026293	0.049261	0.097635	0.187713	0.379048	0.665681	1.420243	3.525398

Tabela B.2: Tempos de execução obtidos no cálculo, em CUDA, dos respectivos números de WHRTFs.

Bibliografia

[1] Julio Cesar Boscher Torres. "Sistema de Auralização Eficiente Utilizando Wavelets". Tese de doutorado. COPPE/UFRJ, 2004.

[2] Tansu Alpcan, Christian Bauckhage e Evangelos Kotsovinos. "Towards 3D Internet: Why, What, and How?" Em: *Cyberworlds, 2007. CW '07. International Conference on*. Out. de 2007, pp. 95 –99. DOI: 10.1109/CW.2007.62.

[3] R. Bolla et al. "Social networking and context management for the future 3D Internet". Em: *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International*. Jan. de 2009, pp. 1 –6. DOI: 10.1109/COMSNETS.2009.4808846.

[4] Inc. Linden Research. *Second Life website. http://secondlife.com/*. URL: http://secondlife.com/.

[5] Blizzard Entertainment. *World of Warcraft website.* URL: http://www.worldofwarcraft.com/index.xml.

[6] V.C. Stoianovici et al. "A Virtual Reality based human-network interaction system for 3D internet applications". Em: *Optimization of Electrical and Electronic Equipment (OPTIM), 2010 12th International Conference on*. Maio de 2010, pp. 1076 –1083. DOI: 10.1109/OPTIM.2010.5510551.

[7] *Virtual Barbershop - Cetera Algorithm*. Web Site. 2012. URL: http://www.sajithmr.me/cetera-algorithm.

[8] Brian Carty. "Movements in Binaural Space: Issues in HRTF Interpolation and Reverberation, with applications to Computer Music". Tese de doutorado. National University of Ireland, Maynooth, 2010.

[9] V. Ralph Algazi, Richard O. Duda e Dennis M. Thompson. "The use of Head-and-Torso Models for Improved Spatial Sound Synthesis". Em: *Audio Engineering Society, 113th Convention* (2002).

[10] Fakheredine Keyrouz e Klaus Diepold. "A Rational Hrtf Interpolation Approach for Fast Synthesis of Moving Sound". Em: *IEEE* (2006).

[11] M. R. Petraglia e J. C. B. Torres. "Performance analysis of adaptative filter structure employing wavelet and sparse subfilters". Em: *IEE Proceedings. Vision, Image and Signal Processing, Inglaterra* 149 (2002), pp. 115–119.

[12] J. C .B. Torres, Mariane Rembold Petraglia e R. A Tenenbaum. "An Efficient Wavelet-Based Model for Auralization". Em: *Acustica United with Acta Acustica, Dinamarca* 90.1 (2004), pp. 108–120.

[13] Julio Cesar Boscher Torres e Rafael Coelho Lavrado. "Interpolação de HRTFs através de Wavelets". Em: *XVII Congresso Brasileiro de Automática* (2008).

[14] J.C.B. Torres e M.R. Petraglia. "HRTF interpolation in the wavelet transform domain". Em: *Applications of Signal Processing to Audio and Acoustics, 2009. WASPAA '09. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 2009, pp. 293–296.

[15] Durand R. Begault. *3-D Sound for Virtual Reality and Multimedia*. Ames Research Center, Moffett Field, California: National Aeronautics e Space Administration, 2000.

[16] Regis Rossi Alves Faria. "Auralização em Ambientes Audiovisuais Imersivos". Tese de doutorado. Escola Politécnica da Universidade de São Paulo, 2005.

[17] Michael Gerzon. "Surround-sound psychoacoustics". Em: *Wireless World* (1974).

[18] Curtis Roads. *The Computer Music Tutorial*. The MIT Press, 1996.

[19] Friedrich A. Kittler. *Gramophone, Film, Typewritter*. Stanford University Press, 1999.

[20] William Howland Kenney. *Recorded Music in American Life: The Phonograph and Popular Memory, 1890-1945*. Oxford University Press, 1999.

[21] Gene Adair. *Thomas Alva Edison: Inventing the Electric Age*. Oxford University Press, 1996.

[22] John Watkinson. *The Art of Sound Reproduction*. Focal Press, 1998.

[23] Michael Vorlander. *Auralization: Fundamentals of Acoustic, Modelling, Simulation, Algorithms and Acoustical Virtual Reality*. Springer-Verlag Berlin Heidelberg, 2008.

[24] Ewan A. Macpherson e John C. Middlebrooks. "Listener weighting of cues for lateral angle: The duplex theory of sound localization revisited". Em: *The Journal of the Acoustical Society of America* 111.5 (2002), pp. 2219–2236.

[25] Jens Blaubert. *Spatial hearing: the psychophysics of human sound localization*. Revised. Cambridge: The MIT Press, 1997.

[26] Corey I. Cheng e Gregory H. Wakefield. *Introduction to Head-related Transfer Functions (HRTF'S): Representations of HRTF's in Time, Frequency, and Space*. Audio Engineering Society. 1999.

[27] Lord Rayleigh. *On our perception of sound direction*. Philosophical Magazine. 1907.

[28] Bohdan T. Kulakowski, John F. Gardner e J. Lowen Shearer. *Dynamic Modeling and Control of Engineering Systems*. Third. Cambridge University Press, 2007.

[29] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. www.DSPguide.com: www.DSPguide.com, 1997.

[30] Philippe de Larminat. *Analysis and Control of Linear Systems*. International Scientific e Technical Encyclopedia, 2007.

[31] Li Tan. *Digital Signal Processing: Fundamentals and Applications*. Academic Press, 2008.

[32] William H. Press et al. *Numerical Recipes in C - The art of Scientific Computing*. Second. CAMBRIDGE UNIVERSITY PRESS, 1992.

[33] Bill Gardner e Keith Martin. *Frequently asked questions about KEMAR HRTF data. http://sound.media.mit.edu/resources/KEMAR/KEMAR-FAQ.txt*. URL: `http://sound.media.mit.edu/resources/KEMAR/KEMAR-FAQ.txt` (acesso em 05/01/2011).

[34] Alan V. Oppenheim e Ronald W. Schafer with John R. Buck. *Discrete-time signal processing*. Prentice Hall, 1998.

[35] Ngai-Man Cheung, Steven Trautmann e Andrew Horner. "HEAD-RELATED TRANSFER FUNCTION MODELING IN 3-D SOUND SYSTEMS WITH GENETIC ALGORITHMS". Em: *IEEE* (1998).

[36] Kentaro Matsui e Akio Ando. "Estimation of individualized head-related transfer function based on principal component analysis". Em: *Acoustical Science and Technology* 30.5 (2009), pp. 338–347.

[37] Kosuke Tsujino et al. "Automatic filter design for 3-D sound movement in embedded applications". Em: *Acoustical Science and Technology* 28.4 (2007), pp. 219–229.

[38] Abhijit Kulkarni e H. Steven Colburn. "Infinite-impulse-response models of the head-related transfer function". Em: *The Journal of the Acoustical Society of America* 115.4 (2004), pp. 1714–1728.

[39] Bill Gardner e Keith Martin. *HRTF Measurements of a KEMAR Dummy-Head Microphone*. Rel. téc. 280. http://sound.media.mit.edu/resources/KEMAR.html: MIT Media Lab Perceptual Computing, 1994.

[40] IRCAM. *Listen HRTF Database*. 2003. URL: `http://recherche.ircam.fr/equipes/salles/listen/`.

[41] Liang Chen, Hongmei Hu e Zhenyang Wu. "Head-Related Impulse Response Interpolation in Virtual Sound System". Em: *IEEE - Fourth International Conference on Natural Computation* (2008).

[42] Elizabeth M. Wenzel e Scott H. Foster. "PERCEPTUAL CONSEQUENCES OF INTERPOLATING HEAD-RELATED TRANSFER FUNCTIONS DURING SPATIAL SYNTHESIS". Em: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics* (1993), pp. 102–105.

[43] M. Stular J. Sodnik R. Susnik e S. Tomazic. "Spatial sound resolution of an interpolated HRIR library". Em: *Applied Acoustics* 66 (2005), pp. 1219–1234.

[44] I. C. Corey e H. W. Gregory. "Moving sound source synthesis for binaural electroacoustic music using interpolated head-related transfer functions (HRTFs)". Em: *Computer Music Journal* 25 (2001), pp. 57–80.

[45] F. P. Freeland, L. W. P. Biscainho e P. S. R. Diniz. "Interpositional Transfer Function for 3D-Sound Generation". Em: *Journal of the Audio Engineering Society* 52.9 (2004), pp. 915–930.

[46] Tapio Lokki Lauri Savioja Jyri Huopaniemi e Riitta Väänänen. "Creating Interactive Virtual Acoustic Environments". Em: *J. Audio Eng. Soc.* 47.9 (1999).

[47] Crystal River Engineering. *Convolvotron*. URL: `http://www-cdr.stanford.edu/DesignSpace/sponsors/Convolvotron.html`.

[48] Marcelo Queiroz e Gustavo H. M. de Sousa. “STRUCTURED IIR MODELS FOR HRTF INTERPOLATION”. Em: *Proceedings of the International Computer Music Conference* (2010), pp. 262–269.

[49] S. Mehrotra, Wei ge Chen e Zhengyou Zhang. “Interpolation of combined head and room impulse response for audio spatialization”. Em: *Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on*. 2011, pp. 1 –6. DOI: 10.1109/MMSP.2011.6093794.

[50] P. P. Vaidyanathan. *Multirate systems and filter banks*. Prentice Hall, 1993.

[51] Gilbert Strang e Truong Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1997.

[52] C. Sidney Burrus, Ramesh A. Gopinath e Haitao Guo. *Introduction to Wavelets and Wavelets Transforms*. Prentice Hall, 1998.

[53] Mariane R. Petraglia e Rogerio G. Alves. “New Results on Adaptative Filtering Using Filter Banks”. Em: *IEEE International Symposium on Circuits and Systems, June 9-12, Hong Kong* (1997).

[54] *NVIDIA CUDA C Programming Guide*. Version 4.0. NVIDIA CUDA™. 2011.

[55] John D. Owens et al. “A Survey of General-Purpose Computation on Graphics Hardware”. Em: *Computer Graphics Forum* 26.1 (2007), pp. 80–113.

[56] David B. Kirk e Wen mei Hwu. *Programming Massively Parallel Processors - A Hands-on Approach*. Elsevier Inc., 2010.

[57] Jason Sanders e Edward Kandrot. *CUDA by Example - An Introduction to General-Purpose GPU Programming*. Addison-Wesley, 2010.

[58] Emmanuel Gallo e Nicolas Tsingos. “Efficient 3D Audio Processing with the GPU”. Em: *ACM Workshop on General Purpose Computing on Graphics Processors* (2004).

[59] Frederik Fabritius. “Audio Processing Algorithms on The GPU”. Diss. de mestrado. Technical University of Denmark, 2009.

[60] André Jucovsky Bianchi. *Processamento de áudio em tempo real em dispositivos não convencionais*. Texto de qualificação de Mestrado. 2011.

[61] Julio Cesar Boscher Torres. *Prof. Julio Cesar Boscher Torres*. Out. de 2011. URL: http://www.deg.poli.ufrj.br/~julio/.

[62] *Cocoa - Mac OS X*. 2012. URL: http://developer.apple.com/technologies/mac/cocoa.html.

[63] *Develop for Mac OS X*. 2012. URL: http://developer.apple.com/technologies/mac/.

[64] *Cocoa Fundamentals Guide*. Apple Inc. 2012.

[65] *FFTW*. 2012. URL: http://fftw.org/.

[66] *Simple Directmedia Layer*. 2012. URL: http://www.libsdl.org/.

[67] Hugh Merz. *CUFFT vs FFTW comparison*. Rel. téc. SHARCNET, Laurentian Universit, 2008.

[68] Shams A.H. Al Umairy et al. “On the use of small 2D convolutions on GPUs”. Em: *A4MMC 2010 - 1st Workshop on Applications for Multi and Many Core Processors* (2010).

[69] *International Telecommunication Union - Radiocommunication sector*. 2012. URL: http://www.itu.int/ITU-R/.

[70] Marija Šalovarda, Ivan Bolkovac e Hrvoje Domitrovic. “Estimating Perceptual Audio System Quality Using PEAQ Algorithm”. Em: *Applied Electromagnetics and Communications, 2005. ICECom 2005. 18th International Conference on* (2005).

[71] *RECOMMENDATION ITU-R BS.1387-1 Method for objective measurements of perceived audio quality*. ITU Radiocommunication sector. 2001.

[72] *Perceptual Evaluation of Audio Quality Beta*. URL: http://sourceforge.net/projects/peaqb/.

[73] *Audio File Programs and Routines*. Telecommunications and Signal Processing Laboratory Multimedia Signal Processing. 2012. URL: http://www-mmsp.ece.mcgill.ca/Documents/Software/Packages/AFsp/AFsp.html.

[74] John Watkinson. *The Art of Digital Audio*. 2nd. Focal Press, 1994.

[75] Diego Augusto Rodrigues Gomes. *Repositório do projeto 3DMixer no GitHub*. 2012. URL: https://github.com/diegoaugusto/3DMixer_final_GIT.