UNIVERSIDADE FEDERAL FLUMINENSE

BRUNO DA COSTA MOREIRA

Usando Diferenças Finitas e Equação de Onda em uma Abordagem em GPU para Áudio em Games

NITERÓI

2012

UNIVERSIDADE FEDERAL FLUMINENSE

BRUNO DA COSTA MOREIRA

Usando Diferenças Finitas e Equação de Onda em uma Abordagem em GPU para Áudio em Games

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Computação Visual

Orientador:

Esteban Walter Gonzalez Clua

NITERÓI

2012

Bruno da Costa Moreira

Usando Diferenças Finitas e Equação de Onda em uma Abordagem em GPU para Áudio em Games

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Computação Visual

Aprovada em Janeiro de 2012.

BANCA EXAMINADORA

Prof. Esteban Walter Gonzalez Clua - Orientador, IC-UFF

Prof. Anselmo Antunes Montenegro, IC-UFF

Prof. Paulo Antonio Andrade Esquef, LNCC

Niterói

2012

Para minha família e amigos.

Agradecimentos

Presto meus agradecimentos aos meus maiores apoios durante todo o mestrado.

A minha mãe Vera Regina, ao meu pai Valdelar Moreira e ao meu irmão Vinicius Moreira por serem o meu suporte familiar durante toda minha trajetória de estudos independente se eram épocas de estresse ou tranqüilidade.

Aos meus colegas de estudos de mestrado,graduação,laboratório e alguns até colegas também de trabalhos: Giancarlo Taveira, Heraldo Borges, Marcelo Niero, Felipe Rolim, André Brandão, Tiago Bonini, Christian Ruff, Thales Sabino, Marcelo Zamith, Diego Brandão e alguns outros que não estão aqui citados, mas que também além de grandes amigos foram de grande importância para levar com dedicação e seriedade toda essa etapa de estudos e formação acadêmica da minha vida.

Ao meu orientador Esteban Clua, uma excelente pessoa que sempre esteve disposto a ajudar e me fazer crescer dentro da área de atuação e até mesmo fora dela.

A amiga Mariana Galindo, que sempre me incentivou e motivou durante o mestrado e principalmente durante a escrita desta dissertação. A outros grandes amigos que não de faculdade como: Bernardo Villaça, Igor Veras, Daniel Ribeiro, Flavio Braune, Tiago Braune, Gustavo Vargas, Felipe Kelm, Nicolas Dias, e outros tantos que sempre foram essenciais para os momentos de descontração e diversão, mesmo que muitas das vezes até atrapalhando os estudos. Mas que sempre se fazem necessários quando a cabeça parece que não vai agüentar mais de tanto compromisso.

A todos os demais que, de alguma forma, contribuíram para a realização desse trabalho.

Para todos os citados, obrigado!

Um agradecimento também ao CNPq, FAPERJ e Petrobras pelo suporte financeiro desse trabalho.

Resumo

Se por um lado a síntese de imagens na área de jogos e simulação já foi bastante explorada nos últimos anos, o mesmo não ocorre com a síntese realista de som. Isto ocorre devido a complexidade de sua propagação e sua interação com objetos de uma cena. O objetivo deste trabalho é mostrar avanços nos estudos de propagação de ondas acústicas para aplicações que exigem processamento em tempo real, tal como jogos e realidade virtual. Inicialmente apresentamos uma relação de trabalhos anteriores, e um deles é utilizado e adaptado para fazer a reprodução do áudio propriamente dita. Em seguida, propomos um método para acelerar a reprodução do áudio e apresentamos uma implementação paralela do mesmo, para uma arquitetura de GPU, com o objetivo de torna-la mais próxima do tempo real. Em nossa proposta apresentamos uma adaptação na equação de propagação de onda simples para uma equação de propagação com atenuação por distância. As implementações e propostas apresentadas obtém ganhos consideráveis de performance, conforme será mostrado ao final desta dissertação. Este trabalho, de maneira geral, une diversas etapas da reprodução do áudio com contribuições em diferentes pontos da implementação, de forma a criar uma base para o avanço dos estudos com o método de diferenças finitas utilizado. Embora os resultados obtidos ainda não estejam com taxas interativas, obtemos ganhos consideráveis e apontamos trabalhos futuros que poderão garantir este requisito.

Palavras-chave: Propagação de Onda Sonora, Método de Diferenças Finitas, Computação em GPU , Síntese de som.

Abstract

If on the one hand the image synthesis in games and simulation have been widely explored in recent years, on the other hand the same does not occur with the synthesis of realistic sound. This is due to the complexity of its propagation and its interaction with objects in a scene. The main goal of the work is to show the improvements in the study of acoustic wave propagation for applications that require real-time processing, such as games and virtual reality. First, a list of previous works is shown, and one of them is used and adapted to realize the complete audio reproduction. Then, an approach is proposed to accelerate the audio reproduction and it is shown how a parallel implementation of it was done, to a GPU architecture, in order to make it more close to real time. Also, the wave equation that governs the method is replaced by a more realistic wave equation that considers damping. Gains are obtained when talking about performance and precision with the methods presented. The dissertation puts together different stages in the audio auralization with contributions at different points of the implementation, creating a base for the progress in the studies that use the finite-difference methods. Although the results are not yet at interactive rates, we obtain considerable gains and pointed out future work that should ensure this requirement.

Keywords: Sound Wave Propagation, Finite Difference Method, GPU Computing, Sound Synthesis.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

2-D	:	Duas Dimensões
3-D	:	Três Dimensões
CPU	:	Unidade Central de Processamento
FD	:	Diferenças Finitas
FDM	:	Método de Diferenças Finitas
FLOAT	:	Tipo de Variável de Número em Ponto Flutuante
GPU	:	Unidade de Processamento Gráfica
GA	:	Acústica Geométrica
INT	:	Tipo de Variável de Número Inteiro
NA	:	Acústica Numérica
PDE	:	Equações Diferenciais Parciais
WAV	:	Formato de Arquivo de Áudio

Lista de Publicações

Os estudos dessa dissertação possibilitaram a publicação do seguinte artigo:

[P1] Moreira, B.C. ; Brandão, D. ; Kuryla, C. ; E. C. Gonzales ; Kischinhevsky, M. . An Architecture Using Finite Difference Method to Calculate Realistic Sounds Equalization in Games (to appear). In: X Simpósio Brasileiro de Games e Entretenimento DIgital, 2011, Salvador. Proceedings of Sbgames 2011.

Conteúdo

Capítulo 1

Introdução

A bilionária indústria de jogos está sempre avançando junto com a tecnologia, sendo em alguns casos até a principal protagonista no seu progresso. Na medida em que os hardwares se tornam mais potentes e com capacidade de processar uma maior quantidade de cálculos por segundo, novas técnicas são estudadas e propostas para deixar os jogos com um realismo cada vez maior. Porém, o que se pode observar, é que os maiores esforços, tanto da indústria quanto da comunidade acadêmica, estão concentrados em atingir e aperfeiçoar o realismo nos gráficos, procurando e propondo novos efeitos de renderização gráfica e efeitos físicos que afetem diretamente em um retorno visual, tais como cálculos de colisão mais precisos para que os objetos se movam na cena de forma mais correta e intuitiva. A resposta visual é geralmente mais atrativa e portanto desperta mais atenção dos estudiosos.

Entretanto, o que se vê em relação ao áudio geralmente são bibliotecas e processos que tratam apenas o seu posicionamento espacial, de forma a aumentar a experiência imersiva. Porém esse tipo de recurso não leva em consideração diversos efeitos do som, tal como o tempo que se leva para cruzar o ambiente (um raio, por exemplo, em que o som chega depois do efeito visual da claridade), e efeitos como refração do som, absorção, atenuação por distância, difusão e eco que surgem a partir da propagação e interação da onda sonora no ambiente.

Esta dissertação se propõe a introduzir modelos de propagação de ondas acústicas de maneira próxima aos modelos físicos conhecidos, mostrando métodos simplificados que podem ser utilizados para cenários de jogos e realidade virtual. Isto significa que dada uma configuração de domínio com geometria, posicionamento do ouvinte, fonte sonora e um sinal de som, propomos um método para reproduzir como este som deveria ser escutado na posição do ouvinte dado o sinal emitido na fonte e propagado pelo meio. Ressaltamos

que esta proposta tem como principal objetivo inserir este recurso em cenários que não requerem rigor físico nem matemático no seu cálculo, porém apenas aumentar o poder de imersão e interação. Na realidade, como os recursos computacionais e o tempo são limitados, é impossível alcançar o objetivo de uma reprodução de áudio perfeitamente correta levando em conta todos os fatores influenciadores de um ambiente. Portanto, técnicas e métodos são criados e utilizados para simplificar o modelo de forma a obter um tempo de processamento rápido o suficiente para utilização em tempo real. Porém, neste trabalho temos o cuidado para que o emprego dessas técnicas seja balanceado de forma a não prejudicar o resultado final do som.

Alguns dos efeitos supracitados considerados nos métodos utilizados nessa dissertação são:

- difração
- eco
- atenuação do som por distância
- refração
- tempo que o som leva para cruzar o ambiente

Apesar de nossos métodos propostos não permitirem ainda o processamento em tempo real, esta dissertação mostra avanços e adaptações em relação às técnicas utilizadas, obtendo resultados bastante próximos a uma taxa interativa, tornando viável num futuro próximo a propagação de som dentro de um ambiente de jogo, sem sacrificar a performance computacional e ainda obter resultados muito mais realistas dos que se tem hoje nestas aplicações.

A proposta apresentada, além de permitir que criadores de jogos possam ter um novo elemento a ser considerado na concepção dos cenários imersivos (materiais acústicos da cena), são criadas inúmeras outras possibilidades de design onde se pode usar o som como elemento de jogabilidade e interatividade. Acreditamos que esta área possa trazer um novo paradigma nos jogos das próximas gerações e criar experiências que farão o jogador ter um novo patamar de imersão.

Outro ponto positivo pode ser ressaltado quando analisamos como um áudio é gerado e utilizado dentro de um ambiente virtual. Cada áudio específico deve ser equalizado ou gravado manualmente de acordo com cada situação e ambientação que o jogo ou ambiente

virtual poderá submetê-lo. Esse processo é trabalhoso e pode necessitar de profissionais e designers de som, demandando uma carga de trabalho grande e custosa. Além disso, este tipo de profissional muitas vezes não se encontra presente em uma equipe de desenvolvedores de jogos. Com a proposta apresentada e seu objetivo final, esse processo pode vir a se tornar um processo muito mais simples e consequentemente menos custoso, onde a preocupação estará voltada para o momento de desenvolver a cena de jogo. Da mesma forma que o artista trabalha nas propriedades do material de um objeto como cor e textura, trabalharia nas suas propriedades acústicas. Assim, apenas áudios bases seriam necessários e então modificados automaticamente para gerar o som como deveria ser ouvido.

1.1 Escopo da Dissertação

Como ponto de partida, esta dissertação extende um trabalho anterior onde é apresentada a propagação de ondas acústicas [26]. A propagação de ondas acústicas é usada para diversos problemas, como calcular camadas geológicas de forma a possibilitar uma avaliação de quanto o solo está propenso a possuir reservas de petróleo. Em [26], a propagação de ondas é usada na renderização de efeitos visuais e detecção da direção real que o som chega no ouvinte. Como o método desenvolvido não é usado para reprodução do áudio propagado, mas apenas para análise e efeitos visuais, as partes cabíveis e contribuições desta dissertação são:

- Integração do algoritmo com biblioteca de áudio
- Adaptação do algoritmo para reconstrução de áudio
- Aperfeiçoamento da equação de onda utilizada
- Criação de um método que reconstrói o áudio utilizando uma análise de comportamento parcial do método de propagação de ondas
- Otimização deste método através de uma abordagem paralela para GPU

A biblioteca integrada irrKlang [10] permite a utilização de arquivos de áudio como dado de entrada do algoritmo. A partir deste momento pode-se então fazer a adaptação para gerar os áudios de saída. Com esta parte feita, é possível focar então nos objetivos de melhora de performance(vide método criado e otimização do mesmo) e qualidade(vide equação).

A modelagem de fontes sonoras está fora do escopo da dissertação, onde por razões de otimização foram consideradas fontes emissoras de som pontuais posicionadas no domínio utilizado.

A direção do som que chega no ouvinte também está fora do escopo desta dissertação. Apesar de estender um algoritmo que realiza esse cálculo, o som final no presente trabalho considera apenas um ponto como sendo o local onde o ouvinte se encontra (diferente do trabalho em [26] que considera múltiplos pontos) e portanto limita-se a computar apenas como o som é escutado e não de onde procede. Modificações são necessárias para utilizar o método previamente implantado em [26] para detectar a direção de onde vem a frente de onda acústica que resultará no áudio final. A partir daí, seria necessário utilizar alguma biblioteca de áudio posicional que faça com que o ouvinte tenha a sensação do som vir de uma direção específica.

1.2 Equação de Propagação da Onda

Pela física, o som é a transferência de energia feita através de uma onda mecânica. É fácil visualizar este fenômeno ao observar uma onda se propagando na superfície da água, uma vez que esta consiste também na transferência de energia. Para tentar modelar este fenômeno computacionalmente é necessário um modelo matemático, sendo este um dos grandes motivos para os estudos em propagação de ondas ter sido deixado de lado quando se fala em games: sua complexidade de cálculo matemático e computacional.

Para resolver esse modelo, dois métodos são comumente utilizados: elementos finitos e diferenças finitas, sendo que a abordagem por elementos finitos é mais custosa que a de diferenças finitas com abordagem explícita, e portanto, o segundo é escolhido para este trabalho. Mas mesmo sendo o método menos custoso, os cálculos que o método de diferenças finitas exige ainda são bastante complexos. Em se tratando de uma aplicação interativa, isto é agravado, já que se requer uma quantidade enorme de iterações por segundo, tornando o tema um grande desafio para jogos e aplicações interativas.

Porém, com o avanço de arquiteturas paralelas, tais como a Unidade de Processamento Gráfica(GPU) [7] e a linguagem de programação para GPUs, CUDA [16], tem-se a possibilidade de paralelizar estes cálculos e, mediante algumas simplificações dos modelos, poder incluir este recurso nestas aplicações. Devido ao baixo custo e acessibilidade das GPUs, pode-se pensar que estas soluções que requerem alto poder computacional serão possíveis de serem computadas em computadores pessoais e consoles de vídeo-games.

O trabalho estendido nessa dissertação [26], e que motivou a mesma, usa uma equação de onda bem simples que leva em consideração apenas a sua propagação(transferência de energia) pelo meio, mas já utiliza uma implementação baseada em GPU. Entretanto, como o fenômeno físico leva em consideração muitos outros fatores., uma modelagem precisa exigiria um modelo muito extenso e talvez inviável para tempo real e taxas interativas.

Esta dissertação foi feita levando em consideração que aos poucos esse modelo matemático deve ser melhorado. Modelos mais simples são considerados para testes iniciais, porém foi julgado de importância pelo autor que uma melhora nesta parte do algoritmo seria essencial para demonstrar a flexibilidade do resultado apresentado. Apesar das simplificações apresentadas, é possível aumentar cada vez mais a complexidade do método em busca de mais realismo e ir ponderando com a performance que se consegue alcançar.

Portanto, após os primeiros testes, é alterada também a equação de onda utilizada no trabalho, adicionando então a contribuição de um modelo que agora calcula não só a propagação da onda pelo ambiente, mas também a atenuação que a onda sonora sofre após uma determinada distância percorrida devido a perda de energia ao longo do deslocamento.

1.3 Reprodução do Áudio

Além de realizar a implementação dos modelos de propagação de onda, é necessário o entendimento do que é um arquivo de áudio e como é a estrutura da dados que o armazena e é interpretada pelo computador, de forma que a sua utilização possa ser incorporada de maneira agregada ao método de propagação.

Algumas bibliotecas auxiliam o trabalho de interpretação dessas estruturas de dados que formam os arquivos de áudio, facilitando a utilização de funções como de leitura de arquivo, carregamento em memória, modificação dos arquivos em memória entre outros. A biblioteca Irrklang [10] foi usada no presente trabalho. Dentro do escopo do trabalho a biblioteca foi satisfatória para todas as funções necessárias.

Após o carregamento dos arquivos de som, é feita a integração com o método para que os dados de entrada da simulação sejam processados a partir das amostras das amplitudes do arquivo. Após implementar o código que permite ler o arquivo de áudio como dado de entrada na posição considerada como a da fonte sonora no método, é indispensável processar a parte que permite capturar os dados do áudio no destino e a posição do ouvinte, de forma a gerar o som de saída.

Como já citado, o método de diferenças finitas é o usado para implementar a propagação do som no ambiente. Como o trabalho original [26] não utilizava a renderização do som propriamente, mas apenas os dados da propagação para efeito visuais e alguns cálculos, foram necessárias algumas modificações no método e nos valores da equação de onda utilizada para que fosse possível reproduzir de forma completa um arquivo de áudio chegando em uma posição do domínio diferente da posição da fonte sonora sem ocorrer perda de amostras.

Neste trabalho será apresentado como essa alteração foi feita, e como fica um arquivo de áudio totalmente processado pelo método de diferenças finitas e pronto para ser reproduzido na sua forma realística.

1.4 Método Proposto

Mesmo com a reprodução total do arquivo de áudio pelo método de diferenças finitas sendo feita em GPU, foi observado que o mesmo não tinha taxas interativas quando usado para renderização do som. As taxas de amostragem necessárias para a reprodução de um arquivo de áudio tornam o método muito mais custoso para renderização de som do que para efeitos visuais.

Esta dissertação propõe como outra contribuição, um método simples criado para fazer a reprodução simulada que se baseia no comportamento de uma onda realmente propagada pelo ambiente, e através da análise desse comportamento o mesmo é reproduzido para todo o arquivo de áudio que se deseja modificar. Os resultados não são fisicamente reais, pois o processamento por este método não é mais feito por diferenças finitas, porém esta aproximação interpolada pode servir para aplicações que não exijam precisão, tais como jogos e realidade virtual.

Tomaremos como base de referência desta dissertação o método de processamento total por diferenças finitas como sendo o método real e completo que, dentro do escopo deste trabalho, será considerado como o realístico. Chamaremos de método proposto ou simulado, o método simplificado e criado para fazer uma simulação de forma mais rápida que ainda obtenha resultados próximos do método ao qual ele foi baseado.

Algumas propriedades físicas do som e da transferência de energia mecânica foram levadas em consideração durante o desenvolvimento do método, tal como a propriedade de superposição da onda mecânica e de perda de energia como calor.

1.5 Otimização do Método Proposto em GPU

O método proposto para fazer uma simulação de forma mais rápida e ainda baseada na propagação real da onda teve sua primeira implementação ainda na CPU, sendo os primeiros resultados apresentados nesta dissertação e também em [P1].

Consideraremos que um dos principais requisitos no processo de implementação de jogos é o tempo real e o esforço para redução do tempo de processamento. Isto se deve não somente porque se quer taxas interativas, mas também porque os jogos necessitam processar muitos outros elementos de forma concorrente (computação gráfica, física, inteligência artificial, etc).

O método criado já foi pensado de forma a ser paralelizável. Assim, ganhos poderiam ser obtidos com a sua portabilidade para GPU. Uma importante contribuição dessa dissertação é como essa portabilidade pode ser realizada para esta plataforma. Resultados são apresentados e comentados mostrando a viabilidade desse processo.

1.6 Organização da Dissertação

No Capítulo 2 são apresentadas as pesquisas relacionadas com o tema estudado. A equação de onda utilizada no trabalho e como foi feita a mudança para sua melhoria estão expostas no Capítulo 3. Como são estruturados e tratados os arquivos e áudio, bem como as mudanças feitas no trabalho em que a dissertação se baseia para realizar a reprodução por diferenças finitas de um arquivo de áudio estão detalhadas no Capítulo 4. O Capítulo 5 apresenta a implementação em CPU e GPU do método simples criado para renderizar o som de forma mais rápida sem grande perda de qualidade. No Capítulo 6 são expostos os resultados de performance e qualidade do trabalho realizado e as conclusões tiradas dos mesmos. Por fim, no Capítulo 7 serão apresentadas diversas formas de melhorar ainda mais os métodos propostos e questões que ficaram em aberto.

Capítulo 2

Pesquisas Relacionadas

As pesquisas relacionadas ao tema desta dissertação são discutidas nessa seção. Primeiramente apresenta-se o comportamento das ondas acústicas, os fenômenos físicos que as envolvem e as estruturas de representação do som no ambiente computacional. Em seguida são apresentados as técnicas utilizadas para renderizar som de forma realística num ambiente computacional, bem como alguns trabalhos que buscam sintetizar o som em tempo real com o objetivo de aplicações interativas, tal como jogos e realidade virtual. Por fim, com os métodos a serem adotados bem definidos, faz-se uma explanação sobre a equação de onda para identificar algumas técnicas implantadas no trabalho.

2.1 Reprodução Acústica do Som

Para o cálculo computacional do som é necessário antes de mais nada um estudo sobre o seu comportamento físico e de como a onda se propaga, bem como os fenômenos relacionados. Em [4] os autores apresentam técnicas relevantes para este trabalho, referente a Design de Som, onde discute-se a criação de elementos acústicos sintetizados e não gravados. Para tanto, apresenta-se um programa, denominado *Pure Data*, e apresentam-se conceitos gerais sobre cálculo de diversos efeitos, fenômenos de propagação, percepção sonora, respostas fisiológicas e natureza do som. Deste livro tirou-se a base de todo o conhecimento para entender as ondas sonoras.

Em [4] também são apresentadas sugestões de como tratar o áudio computacionalmente, bem como sugestão de estruturas de dados simples usadas para armazená-lo, como por exemplo arquivos texto ou vetores. Além de mostrar também como esses arquivos são interpretados, teorias de amostragem de som e como sua discretização é usada para suprir as estruturas em questão.

2.2 Técnicas Computacionais Para Reprodução Acústica

As técnicas para simular propagação de som podem ser classificadas em duas categorias: Acústica Geométrica (GA) e Acústica Numérica (NA). Trabalhos baseados em métodos de acústica numérica são em geral computacionalmente mais custosos e portanto menos apropriados para utilização em tempo real, enquanto que abordagens baseadas em técnicas geométricas não são tão custosas mas não representam a precisamente a física do som pois deixam de considerar certos efeitos como difração e dispersão.

Acústicas geométricas assumem a propagação retilínea das ondas sonoras, portanto, muitas abordagens utilizam técnicas chamadas de traçados de raios (*ray-casting*)[21] e traçados de feixes(*beam tracing*)[5, 1, 24].

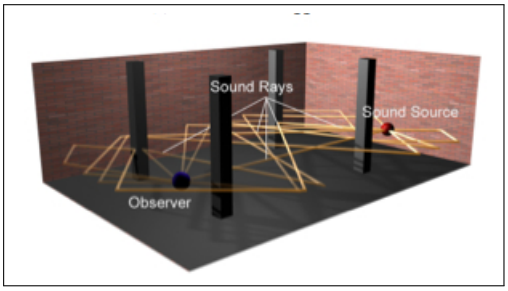

Figura 2.1: Técnica de traçado de raios em [21].

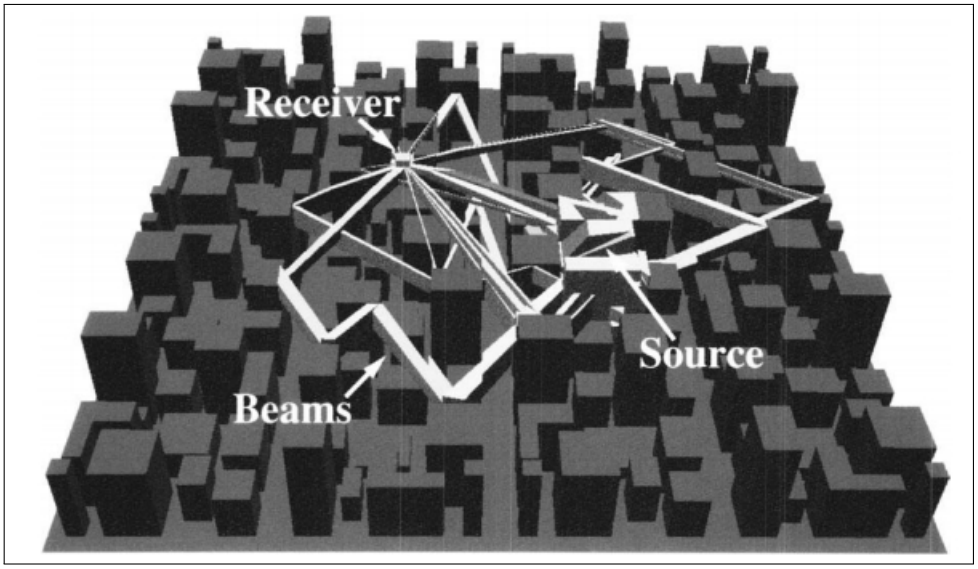

Figura 2.2: Técnica de traçado de feixes em [5].

Na Acústica Numérica resolve-se diretamente a equação da onda levando em conta a propagação física do som pela cena. Havendo recursos computacionais suficientes, todos

os fenômenos podem ser considerados, incluindo difração e dispersão em cenas complexas, por exemplo. Abordagens numéricas não são afetadas pela complexidade de polígonos da cena modelada e analisada, ao contrário, essas abordagens se tornam mais custosas e sua escalabilidade se dá através das dimensões físicas do ambiente.

Dois métodos comumente utilizados para abordagens numéricas são diferenças finitas e elementos finitos. Vários trabalhos demonstram o uso desses métodos, conforme discutido em [2, 14, 18, 23, 12], mas nenhum deles é utilizado em tempo real.

Pode-se perceber por alguns desses trabalhos que geralmente o tratamento de som é usado para avaliação acústica de ambientes no momento do projeto de sua arquitetura, uma vez que esse tipo de avaliação não requer do tempo real, já que pretende-se apenas analisar o comportamento do som dentro de ambientes que serão usados para dar aulas, palestras, realizar concertos musicais ou que demonstrem algum outro tipo de importância entre o que vai ser apresentado e como as pessoas que estão no ambiente estarão ouvindo.

Como som é considerado de baixa prioridade na produção de jogos realísticos, poucos trabalhos foram encontrados focados no objetivo de renderizar sons fisicamente corretos em tempo real [19, 24]. Mais ainda: os recursos computacionais disponíveis de GPU e CPU são tipicamente utilizados para fazer os cálculos físicos (como mecânicas de colisão) e gráficos. Como a arquitetura básica da linguagem CUDA, usada para programar aplicações para a GPU permite apenas um kernel rodando de uma vez, a maioria dos motores de jogos e aplicações comerciais preferem usar esse poder de processamento para essas tarefas tradicionais, não dispondo muitos recursos para a computação do som. Entretanto, novas arquiteturas de GPUs estão permitindo a execução de kernels concorrentes, o que torna possível o processamento do áudio em concorrência com a computação gráfica e outros cálculos de física.

Dentre os trabalhos que procuram renderizar sons em jogos, [24] utiliza uma abordagem com um método geométrico, onde a quantidade de polígonos na cena influencia a sua complexidade. O algoritmo faz um pré-processamento com a redução de polígonos da cena para em seguida aplicar o método geométrico de propagação de onda correspondente. Em [24], já é usada uma abordagem em GPU. Já em [19] é usado um algoritmo, denominado como decomposição retangular, que apresenta uma melhora de processamento de até 100 vezes sobre o método simples de diferenças finitas. Este algoritmo usado em [19] foi apresentado em [17]. Mesmo já tendo uma ótima performance, [19] não implementa o seu método em GPU, permitindo que diversas melhorias possam ser feitas.

Em [3, 26, 22], eficientes implementações de diferenças finitas são apresentadas em

GPU, dando uma grande expectativa na utilização deste método. Algumas técnicas foram desenvolvidas em [3] e demonstram, por exemplo, que ao usar a memória compartilhada da GPU tem-se o maior ganho de processamento no método de diferenças finitas, pois diminui o tempo de transferência de dados que é o gargalo quando usando memória global.

A partir da observação desses trabalhos [24, 19, 3, 26], são escolhidos os métodos e técnicas que vão formar a base desta dissertação. Uma nova abordagem para a reprodução das ondas acústicas através de diferenças finitas é apresentada e otimizada. O trabalho aqui apresentado se baseia em [26] para dar continuidade aos estudos com essa combinação de métodos.

2.3 Equação de Ondas

A equação de onda clássica que representa apenas a propagação da onda no meio ambiente sem considerar fenômenos de atenuação pode ser encontrada em [26]. Neste trabalho o problema de propagação de ondas é simulado utilizando o método de diferenças finitas. Este modelo no entanto, não trata da perda de energia, fenômeno que ocorre naturalmente a medida que a onda se dispersa. Isto significa que ao utilizarmos tal modelo para simular a propagação de onda em uma sala fechada, a perturbação da onda sonora geraria uma propagação de uma onda que nunca desapareceria completamente do ambiente.

Sabe-se que os recursos no ambiente computacional são limitados. Métodos que discretizam o domínio para fazer cálculos sobre os pontos do mesmo não podem ter um tamanho de domínio infinito, portanto, os domínios utilizados devem ser limitados pelos recursos computacionais disponíveis. Com o objetivo de fazer a propagação em domínios limitados, onde a continuação do mesmo para o infinito é simulada, o método matemático utilizado necessita de uma condição de borda. Condição de borda é uma técnica que evita que as ondas propagadas sejam refletidas no contorno para dentro do domínio, gerando um fenômeno não físico.

Tal como em [26], este trabalho considera a condição de borda proposta em [20]. Esta condição é determinada pela decomposição da equação de onda escalar de uma dimensão, gerando o produto de dois termos, cada um representando o espalhamento da frente de onda em uma direção. A Equação 2.1 representa essa condição quando a propagação de onda ocorre horizontalmente para a direita. A condição de borda para ondas movendo-se horizontalmente para a esquerda pode ser obtida analogamente [20]. As condições no topo e base(valores máximos e mínimos verticais) do modelo são dados pelas condições

de Dirichlet [6].

$$\frac{\partial P}{\partial \overrightarrow{n}} = \frac{1}{c}\frac{\partial P}{\partial t} \tag{2.1}$$

Como uma contribuição para melhora do método utilizado, uma equação diferente é escolhida para substituir a equação de propagação simples proposta em [26]. Esta equação, apresentada em [25], leva em consideração o amortecimento da onda com a distância percorrida, isto é, a perda de energia para o calor a medida que a onda vai se propagando. A implementação da mesma torna o resultado obtido pela propagação mais realista e faz desaparecer efeitos indesejáveis supracitados, tal como o efeito de propagação em ambientes fechados e que nunca se dispersam. Este efeito geraria um resultado totalmente indesejado e incorreto se a simulação corresponder a um áudio com grande sequencia temporal, tal como uma música ambiente. Nestes casos, após algum tempo, a energia gerada pela fonte emissora estaria tão acumulada que os valores já não indicariam mais uma variação normal de som.

Capítulo 3

Equação de Ondas

Neste capítulo será apresentado o modelo matemático que representa fisicamente a dispersão de ondas acústicas e a abordagem numérica adotada para sua computação. É importante ressaltar que para todos cálculos deste trabalho, serão consideradas as variações ao longo dos eixos x e y, sendo usados portanto, modelos matemáticos bidimensionais (2-D).

3.1 Modelo Simples

A equação de onda é uma equação diferencial parcial de segunda ordem que descreve o comportamento do som ao longo do tempo. O campo acústico é descrito por $P(x, y, t)$ e $u(x, y, t)$, onde P representa o campo de pressão e u o deslocamento das partículas. A relação entre P e u é dada por $P(x, y, t) = -k \bigtriangledown^2 u(x, y, t)$. Além disso, a equação de onda 2-D pode ser representada pela Equação 3.1.

$$\frac{\partial^2 P(x,y,t)}{\partial t^2} = c^2(x,y)[\frac{\partial^2 P(x,y,t)}{\partial x^2} + \frac{\partial^2 P(x,y,t)}{\partial y^2}] + f(x,y,t) \tag{3.1}$$

Onde x e y são coordenadas cartesianas, t é o tempo, $c(x, y)$ é a velocidade de uma onda acústica no meio, e $f(x, y, t)$ é o termo que vai adicionar ao método valores da fonte sonora.

Para resolver numericamente equações diferenciais parciais(PDE), diversas técnicas podem ser implantadas. O método escolhido é baseado no Método de Diferenças Finitas(FDM), que substitui o cálculo para solução no domínio continuo por um numero finito e discreto de pontos, chamados de pontos do grid, que cobrem todo o domínio. Após a

discretização das PDEs, obtém-se um conjunto de equações, equações de diferenças finitas(FD), que aproximam a equação diferencial em todos os pontos do grid. Cada equação de FD em volta de um ponto do grid é escrita como uma expressão algébrica que envolve os pontos vizinhos para substituir as derivadas espaciais.

A aproximação das PDEs nos pontos do grid é feita por diferenças centrais, que oferecem uma segunda ordem de precisão espacial. A técnica usada para a discretização do tempo também é de segunda ordem neste trabalho. A expressão algébrica própria pode ser obtida a partir da expansão da série de Taylor. Usando a segunda ordem para aproximação no espaço e tempo, como explicado, e assumindo $h = \Delta x = \Delta y$ e $t = n\Delta t$, a equação da onda pode ser reescrita na sua forma discreta como na Equação 3.2.

$$P_{i,j}^{n+1} = 2P_{i,j}^{n} - P_{i,j}^{n-1} + \Delta t F_{i,j}^{n} + \frac{A}{12}[P_{i-1,j}^{n} + P_{i+1,j}^{n} - 4P_{i,j}^{n} + P_{i,j-1}^{n} + P_{i,j+1}^{n}] \tag{3.2}$$

Onde $A = (\frac{c(x,y)\Delta t}{h})^2$, $n = 1, 2, 3, \ldots$ e representa o instante de tempo. Visto que o campo de velocidade não varia no tempo, ele não é uma função do tempo[6].

Um detalhe importante de se destacar é que como o presente trabalho lida com áudio, os valores de Δt, Δx e Δy devem ser escolhidos cuidadosamente para que o som possa ser processado. Por exemplo, para o valor de Δt deve-se respeitar a taxa de amostragem do arquivo de áudio que se pretende gerar, conforme pode ser visto na Equação 3.3. Desta forma a leitura do campo de pressão pode ser feita em todos os instantes de tempo necessários para preenchimento do arquivo de áudio.

$$\Delta t = \frac{1s}{n} \tag{3.3}$$

Sendo n é o número de amostras por segundo.

A expressão discreta da Equação 3.2 provê a computação explícita de valores aproximados no instante de tempo $(n + 1)$, uma vez que os instantes de tempo $(n - 1)$ e (n) estão disponíveis para todos os pontos do grid. Os pontos de forma explícita são uma propriedade conveniente que permite o uso de um paralelismo eficiente, já que todos os valores de pontos do grid no instante de tempo $(n + 1)$-ésimo são computados de forma independente. Outro aspecto das aproximações discretas por equações diferenciais é que algumas estratégias de discretização implícita acarretam em uma estabilidade incondicional, enquanto a utilizada provê uma estabilidade condicional. Isto é, os instantes de tempo

devem ser menores que um limite superior, e caso o valor fique acima do mesmo a solução numérica pode extrapolar (o limite superior pode ser encontrado a partir do critério de Von Neumann), ou seja, a amplitude pode aumentar de forma não física com ganhos de energia para o sistema. Por outro lado, a estabilidade incondicional permitiria uma variação maior de instantes de tempo que no limite máximo do método explícito. Entretanto, o método implícito não é eficientemente paralelizável em GPUs. Consequentemente a técnica explícita da Equação 3.2 foi a primeira escolhida, e uma escolha cuidadosa dos instantes de tempo foi feita não só para garantir a reprodução do áudio, como supracitado, mas para também seguir a condição de estabilidade do método[6].

3.2 Modelo com Atenuação

O modelo apresentado pela Equação 3.1 e usado em [26, 15] é o modelo que conta com a forma mais simples de propagação de onda. Neste trabalho, incrementamos este modelo, inserindo o fator de atenuação por distância, obtendo bons resultados. Este fenômeno pode ser modelado utilizando a equação de amortecimento, que nada mais é do que a Equação 3.1 com um termo de amortecimento em k [25]. Assim, esta equação pode ser re-escrita pela Equação 3.4.

$$\frac{\partial^2 P(x,y,t)}{\partial t^2} - k\frac{\partial P(x,y,t)}{\partial t} = c^2(x,y)[\frac{\partial^2 P(x,y,t)}{\partial x^2} + \frac{\partial^2 P(x,y,t)}{\partial y^2}] + f(x,y,t) \tag{3.4}$$

O termo k varia de acordo com o meio em que a propagação de onda está inserida. Como é considerado para este trabalho apenas um único meio, o valor k escolhido e utilizado ao longo de todos os testes realizados é o mesmo.

Pelo mesmo processo de diferenças finitas utilizado para o modelo simples, produz-se o que seria a equação discreta análoga à Equação 3.4, novamente assumindo $\Delta x = \Delta y$:

$$\frac{P_{i,j}^{n+1} - 2P_{i,j}^{n} + P_{i,j}^{n-1}}{\Delta t^2} + k\frac{P_{i,j}^{n} - P_{i,j}^{n-1}}{\Delta t} = c^2\frac{-4P_{i,j}^{n} + P_{i+1,j}^{n} + P_{i-1,j}^{n} + P_{i,j+1}^{n} + P_{i,j-1}^{n}}{\Delta x^2} + F_{i,j}^{n} \tag{3.5}$$

Após a reorganização dos termos da Equação 3.5 podemos chegar finalmente a equação que faz o cálculo do campo de pressão com atenuação por distância a cada instante de tempo para todos os pontos do grid, chegando assim a Equação 3.6, que será usada neste trabalho:

$$P_{i,j}^{n+1} = P_{i,j}^{n}+(1-k\Delta t)(P_{i,j}^{n}-P_{i,j}^{n-1})+\frac{\Delta t^2 c^2}{\Delta x^2}(-4P_{i,j}^{n}+P_{i+1,j}^{n}+P_{i-1,j}^{n}+P_{i,j+1}^{n}+P_{i,j-1}^{n})+\Delta t^2 * F_{i,j}^{n} \tag{3.6}$$

Chegamos a esta equação após a primeira etapa de testes, onde parte do método proposto ainda se encontrava em CPU. Para tal, novamente são respeitadas todas as condições de estabilidade necessárias para que o método não forneça resultados errados devido a estouros no cálculo[6].

3.3 Condição de Borda

A condição de borda adotada utilizada para ambas as equações, tanto para o caso com atenuação(Equação 3.6) como para o sem atenuação(3.2),já estava implementada em [26]. "Esta condição é determinada pela decomposição da equação de onda escalar de uma dimensão, gerando o produto de dois termos, cada um representando o espalhamento da frente de onda em uma direção" [26]. A Equação 2.1, também descrita em [26], com o mesmo processo de discretização utilizado nas Equações 3.2 e 3.5 pode ser transformada na Equação 3.7 que segue[20], sendo $h = \Delta x = \Delta y$:

$$P_{1,j}^{n+1} = P_{1,j}^{n} + P_{2,j}^{n} - P_{2,j}^{n-1} + \frac{c\Delta t}{h}(P_{2,j}^{n} - P_{1,j}^{n} - (P_{3,j}^{n-1} - P_{2,j}^{n-1})) \tag{3.7}$$

A Equação 3.7 mostra o comportamento que deve ser empregado na borda da esquerda do domínio. De forma análoga podemos chegar nas Equações 3.8,3.9 e 3.10 que indicam respectivamente os comportamentos a serem empregados nas bordas do domínio: lateral direita, superior e inferior.

$$P_{M+1,j}^{n+1} = P_{M+1,j}^{n} + P_{M,j}^{n} - P_{M,j}^{n-1} - \frac{c\Delta t}{h}(P_{M+1,j}^{n} - P_{M,j}^{n} - (P_{M,j}^{n-1} - P_{M-1,j}^{n-1})) \tag{3.8}$$

Sendo M o tamanho do domínio na horizontal.

$$P_{i,1}^{n+1} = P_{i,1}^{n} + P_{i,2}^{n} - P_{i,2}^{n-1} + \frac{c\Delta t}{h}(P_{i,2}^{n} - P_{i,1}^{n} - (P_{i,3}^{n-1} - P_{i,2}^{n-1})) \tag{3.9}$$

$$P_{i,M+1}^{n+1} = P_{i,M+1}^{n} + P_{i,M}^{n} - P_{i,M}^{n-1} - \frac{c\Delta t}{h}(P_{i,M+1}^{n} - P_{i,M}^{n} - (P_{i,M}^{n-1} - P_{i,M-1}^{n-1})) \qquad (3.10)$$

Sendo M o tamanho do domínio na vertical.

Com o emprego destas equações para as bordas, torna-se possível a simulação da propagação de ondas acústicas em ambientes abertos, pois a onda reproduz um comportamento como se tivesse saído do sistema e ido para o infinito.

Capítulo 4

Arquivo de Áudio

Nesta seção será apresentada a estrutura de um arquivo de áudio, bem como seu funcionamento e como o método proposto processa este áudio para gerar o efeito realístico do som, mediante seu processamento por diferenças finitas.

4.1 Arquivo de Áudio

Para fazer a reprodução do áudio e sua reconstrução é necessário descrever a estrutura de dados que armazena as informações sobre um áudio no ambiente computacional.

Um arquivo de áudio é composto por uma seqüência de valores de amostras que representam a discretização da forma de onda – amplitude – passando pelo ponto de captura em determinado instante de tempo [4]. Além de um cabeçalho que o define, este tipo de arquivo possui dois parâmetros: o número de amostras por segundo, e a precisão da amostra. O primeiro indica quantos valores de amplitude são analisados por segundo. Quanto maior a quantidade de amostras por segundo, maior o número de freqüências que aquele arquivo de áudio pode representar. De acordo com a teoria de amostragem uniforme(*sampling theory*), o número de pontos de amostra por segundo deve ser pelo menos duas vezes maior que a maior freqüência que se deseja representar em um sinal, chamando-se a este valor de limite Nyquist. Com o objetivo de cobrir toda a faixa de freqüências da audição humana, que está situada entre 20Hz e 20kHz, é necessário ter pelo menos 40mil amostras por segundo [4].

O segundo parâmetro descreve a ordem da aproximação, que constitui precisão de uma amostra (resolução ou tamanho de amostra), isto é, o número de bits que são reservados para representar um único valor de amostra(16, 32, 64-bits, etc.). Quanto maior o valor

por amostra, mais preciso o áudio será. Entretanto, a precisão também depende do hardware de entrada e saída de áudio, e muitas vezes um valor de precisão muito alto por cada amostra não é necessário, pois não poderá ser reproduzido com esta mesma taxa.

4.2 Processamento do Sinal de Áudio

Este trabalho pretende utilizar as estruturas de dados típicas para representar o áudio para, mediante o uso do método de propagação de ondas por diferenças finitas, realizar uma reconstrução do mesmo. O primeiro passo foi fazer a reconstrução total do áudio utilizando a equação simples do algoritmo já portado em GPU e apresentado em [26]. Entretanto, o método apresentado nesse artigo não possui os parâmetros corretos para que fazer a leitura de acordo da quantidade necessária de amostras de um arquivo de áudio. Além disso, em [26] também não há uma estratégia para integrar um arquivo de áudio com as equações de propagação.

A biblioteca irrklang [10] foi então adotada e integrada ao método e utilizada para a leitura dos arquivos de áudio. A biblioteca permite, por exemplo, que com a utilização do Código 1, possa ser feito o carregamento de um arquivo *WAV* e o armazenamento dos seus valores de amostras em um vetor. Este procedimento possibilita a utilização desse vetor como fonte de entrada de dados para o método de diferenças finitas.

```
1 engine = createIrrKlangDevice();
2 sndSource = engine->addSoundSourceFromFile( "SoundFileName.wav",
  ESM_NO_STREAMING, false );
3 sndSource->setForcedStreamingThreshold( -1 );
4 sampleData = (int16_t*) sndSource->getSampleData();
```

Algoritmo 1: Utilização da biblioteca Irrklang juntamente com o modelo proposto.

Sobre os parâmetros para a leitura de acordo da quantidade necessária de amostras, foi percebido que a implementação anterior de [26] não necessitava de uma grande preocupação com os valores de Δt e Δx do método, pois como o objetivo era o efeito visual e a computação da direção da frente de onda os valores de intervalo de tempo não eram precisos e não eram necessárias muitas amostras por segundo. Como dito no Capítulo 3, os valores devem seguir a Equação 3.3, dependendo da quantidade de amostras do arquivo de áudio utilizado. Como um processamento de 40mil amostras por segundo, por exemplo, a equação de diferenças finitas deve ser calibrada com um Δt muito pequeno para que não sejam perdidos os instantes de tempo que devem ser utilizados na leitura. Isso

aumenta o custo computacional, porque mais iterações são necessárias para a criação de cada segundo do arquivo de áudio. Este cuidado com os parâmetros do método é tomado em todos os testes deste trabalho.

Com a entrada de áudio e os parâmetros do método configurados, tem-se então toda a estrutura necessária para partir para a próxima etapa: realizar o processamento de áudio por diferenças finitas. Para isso, primeiro deve ser feito o mapeamento do grid de domínio das diferenças finitas na cena da geometria 3D da aplicação, como mostrado na Figura 4.1. Em seguida, devem ser criados pulsos que correspondem as amostras do arquivo de áudio em um ponto arbitrário do ambiente. Esse ponto é designado como fonte sonora, de tal forma que a cada iteração o método vai calcular a propagação da energia que é emitida nesse local (como mostrado na Figura 4.2). Os pulsos continuam sendo inseridos até que o arquivo de áudio termine, momento este que significa que a a fonte parou de emitir os sons.

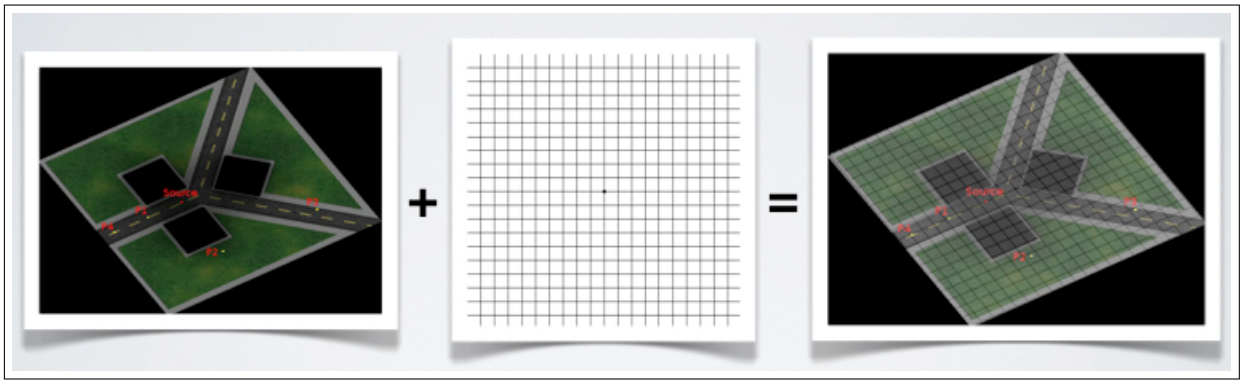

Figura 4.1: Mapeamento do grid na cena de jogo para permitir a análise da propagação de ondas acústicas de forma clara de acordo com as posições de interesse da fonte e ouvinte.

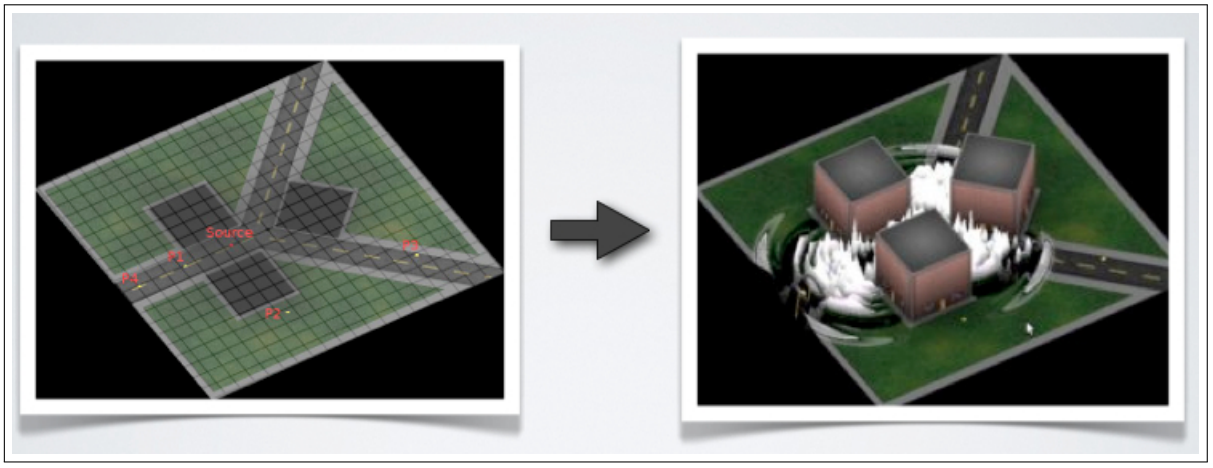

Figura 4.2: Representação visual da propagação de onda em um ambiente externo saindo de um ponto considerado como fonte de emissão sonora.

Para cada iteração do método de diferenças finitas, as amplitudes de onda propagadas

por todo o domínio para um determinado intervalo de tempo são calculadas. Então, as amostras de som que chegam num ponto específico podem ser lidas e interpretadas, usando o método considerado, através da leitura do valor de amplitude naquele ponto em um determinado instante de tempo. Esta leitura dos cálculos do método de diferenças finitas que se encontram em GPU, é feita através da chamada de função a seguir:

```
CHECK_ERROR( cudaMemcpy( &amplitude, &gpu_U1[X_SRC +
Y_SRC * XPOINTS], sizeof( float ), cudaMemcpyDeviceToHost ) )
```

Para calcular o comportamento real de um som em um ponto de interesse, as amostras nesse ponto são lidas periodicamente usando um intervalo de tempo correspondente para a precisão dada/desejada de freqüência de áudio, tal como por exemplo 40 mil amostras por segundo. Os valores de amostra obtidos da leitura são então armazenados e usados para gerar um novo arquivo de som modificado. Este áudio contem o som percebido no local, considerando a real propagação de ondas pela cena.

Na Figura 4.3 um áudio original e um áudio processado são mostrados sobrepostos. O experimento desta figura usou uma fonte sonora na posição $(x, y) = (64, 20)$ em um ambiente aberto com o ouvinte posicionado na coordenada $x = 64$ e $y = 64$, simulando uma distância de 44 metros.

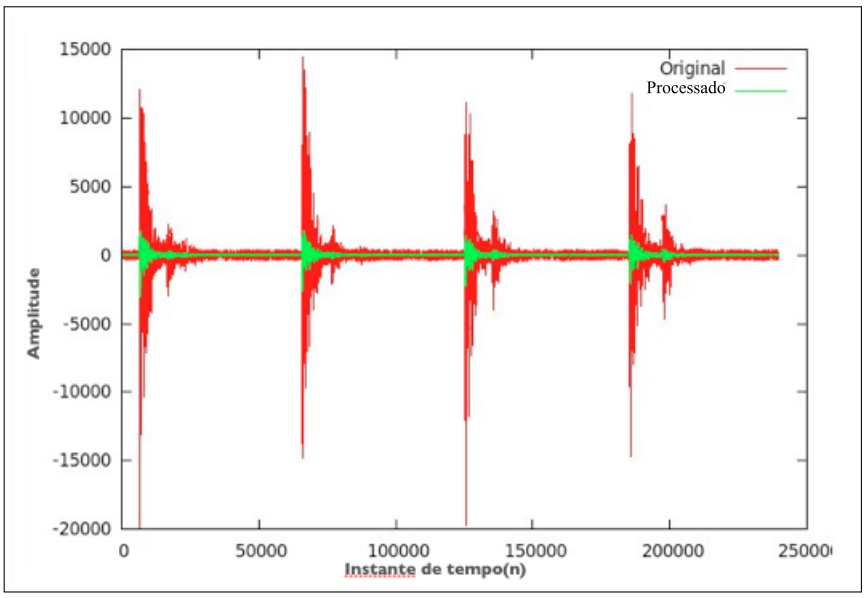

Figura 4.3: Comparação entre a variação de sinal do arquivo original *footsteps.wav* e a variação de sinal do arquivo de som processado por diferenças finitas.

A Figura 4.3 confirma o comportamento intuitivo de que dado ouvinte e fonte sonora separados por uma distância grande, a amplitude sonora é atenuada pela propagação da onda pelo ambiente. A Figura 4.3 demonstra também que as variações de amplitude do

som processado e do original são similares, o que indica que quando o áudio é tocado, ele é muito próximo do áudio original, porém com com uma intensidade mais baixa.

Para fins de análise e efeitos visuais, objetivos do trabalho [26], o algoritmo pode ser considerado de processamento em tempo real, pois não exige uma taxa de amostragem tão alta. Porém, com as funções implementadas e os testes feitos, foi descoberto que para uma taxa de amostragem de 44kHz (que requer um Δt muito pequeno e portanto muitas iterações por segundo) o método em GPU apresentado em [26] não consegue um processamento em tempo real para a renderização do som. Considerando isto, no próximo capítulo deste trabalho propomos uma abordagem para fazer essa captura de um áudio de saída de uma forma mais rápida com um processamento parcial da propagação de onda e ao mesmo tempo gerando resultados aceitáveis dentro da margem de percepção e próximos a como o ser humano escutaria o som se ele tivesse sido totalmente processado.

Capítulo 5

Processamento Acústico Baseado em Comportamento de Pulso

Esta seção descreve em detalhes o método desenvolvido neste trabalho [15]. É apresentada uma implementação em CPU e um método paralelo em GPU, de forma a aumentar a sua performance. No Capítulo 6 serão apresentados os resultados e ganhos de tempo vindos dessa melhora.

As técnicas utilizadas nesta seção são técnicas relacionadas com base em teoria de sinais e sistemas lineares. [8, 9, 13]

5.1 Método Proposto e sua Implementação em CPU

Diferente do método de processamento de áudio total por diferenças finitas que possui uma fase única de processamento, o método proposto baseado em comportamento de pulso consiste em duas fases bem definidas: análise de apenas um pulso de onda e a generalização do comportamento do pulso para um arquivo de áudio completo. Podemos ver a comparação geral entre os métodos através da Figura 5.1.

Ao iniciar o método, na etapa 1, o pulso é emitido na posição da fonte e analisado por um intervalo de t segundos e servirá como uma espécie de amostragem da propagação, que depois será generalizada para todo o áudio, porém sem ter que realizar o cálculo da propagação constantemente. Este pulso é criado a partir da inserção de um valor grande de amplitude em um único instante de tempo na posição da fonte, isto é, a fonte emite este valor por apenas um intervalo de Δt, e no instante seguinte a fonte já é zerada e para de emitir qualquer valor. A propagação desse único valor emitido é feita então pelo método no ambiente.

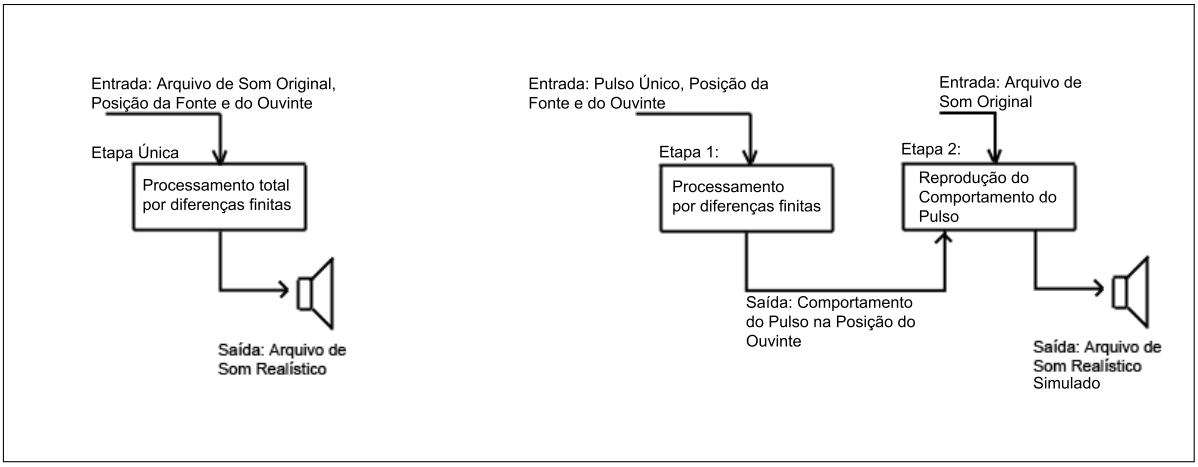

Figura 5.1: Na esquerda o diagrama resumido do Método de Processamento Total por Diferenças Finitas. Na direita o diagrama resumido do Método de Processamento Baseado em Comportamento de Pulso proposto neste trabalho.

A análise de um pulso simples é feita lendo os valores de amplitude que chegam em determinado ponto da cena (x, y) onde o ouvinte está posicionado. Isto é feito da mesma maneira que no caso de processamento real do som, como descrito no Capítulo 4, porém para um intervalo de tempo pequeno. Após o cálculo da propagação do pulso único, os valores observados são comparados com a amplitude original inicial. Esta comparação irá gerar os valores proporcionais para cada instante de tempo subseqüente ao instante inicial, para o intervalo de tempo previamente determinada (por exemplo 200ms). Um exemplo de valores proporcionais do comportamento de um pulso único propagado é exposto na Figura 5.2.

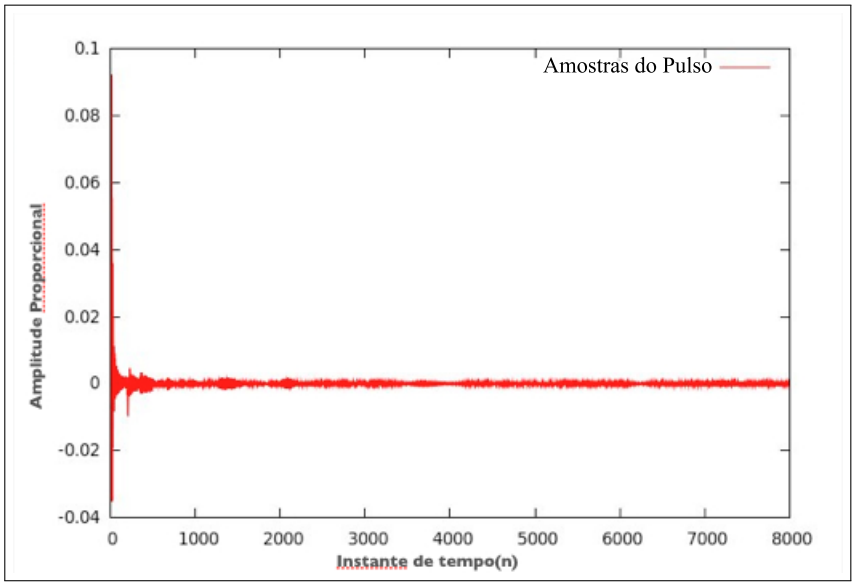

Figura 5.2: Comportamento analisado na propagação de um único pulso.

Neste trabalho foram feitos testes com diferentes taxas de amostragem dos arquivos de

áudio utilizados, portanto mesmo que para ambos os testes seja considerado um mesmo intervalo de tempo na análise do pulso, o número de leituras feitas para o pulso pode variar. Nos testes feitos para uma taxa de amostragem de 40kHz foi feita uma leitura de 8000 amostras para o pulso, enquanto que nos testes que consideram uma taxa de amostragem de 11025Hz foram realizadas 2205 amostras(totalizando um tempo de 200ms de análise em ambos os casos).

Este intervalo de tempo de análise foi escolhido arbitrariamente apenas como base para obtenção dos resultados. O critério levado em consideração foi apenas o de selecionar um tempo que não fosse muito curto, de forma a não ser curto o suficiente para perder alguns efeitos como as reverberações finais e ecos, mas que ao mesmo tempo não fosse muito longo, de forma a não perder muito tempo com cálculos do comportamento do som em um momento em que o mesmo já tenha sido propagado e atenuado de forma a não influenciar mais significativamente no resultado.

Depois da etapa de análise, o método entra na fase de reconstrução. Na etapa 2, de reconstrução, o algoritmo varre todo o arquivo de áudio original, e para cada amostra, aplica os valores proporcionais que representariam o mesmo comportamento do pulso naquela amostra se aquela amplitude da amostra fosse a utilizada na etapa de análise. Esses valores(que representam o comportamento de cada amostra cruzando o ambiente) são armazenados na memória e usados para gerar o novo som. Portanto, cada amostra do arquivo original gera uma janela de tempo de valores representando seu comportamento individual. Esta janela afetará os próximos 200ms (considerando o tempo adotado para observação do pulso deste trabalho) no som gerado. Como se pode ver na Figura 5.3, esses comportamentos de cada amostra vão se sobrepor, e quando isto ocorre os valores são somados. Essa soma pode ser feita de acordo com a propriedade de superposição das ondas acústicas[4]. A Figura 5.3 mostra como a heurística da etapa de reconstrução aqui proposta funciona.

Apresentado o método e suas etapas, pode-se ver no Código 2, a implementação simples que mostra como a fase de reprodução do áudio da heurística é calculada em CPU:

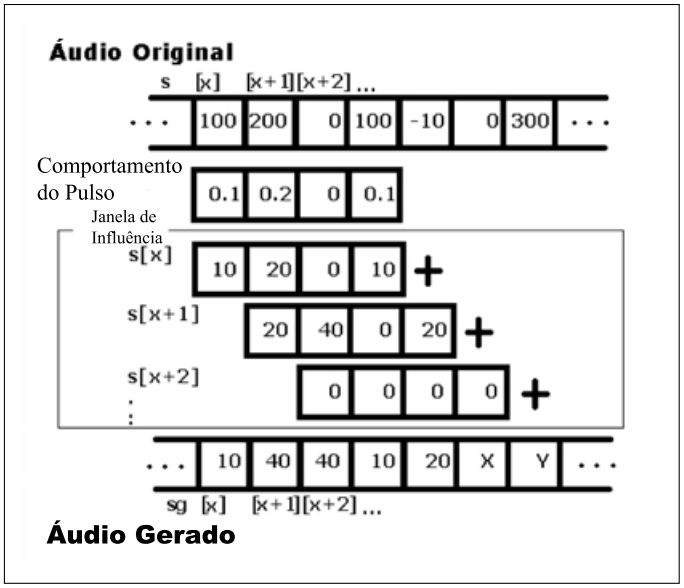

Figura 5.3: Representação visual da heurística, usando um pulso com apenas 4 amostras para exemplificação. A imagem mostra a influencia da amostra S[x], S[x+1] e S[X+2] no áudio gerado, dado o comportamento de pulso.

```
1 for todas amostras do áudio original do
2     for numero de amostras do pulso analisado do
3         if i + j dentro do tamanho do vetor then
4             listenerSampleData[i + j] ←
              listenerSampleData[i + j] + audioOriginal[i] ∗ vAmostras[j];
5         end
6     end
7 end
```

Algoritmo 2: Segunda etapa da heurística em CPU. Fase de reprodução.

Pode-se observar pelas repetições aninhadas que o método apresenta características de um algoritmo paralelizável, sendo as GPUs ótimas candidatas para este processamento. Esta versão do algoritmo apresentada acima é a mesma usada em [15] e usada nos primeiros testes que serão apresentados no Capítulo 6.

5.2 Modelo Paralelo em GPU

Por possuir características paralelizáveis o presente trabalho apresenta o mesmo método anterior implementado em GPU. Esta abordagem permite um ganho muito grande na segunda etapa da heurística, a reconstrução do áudio.

A primeira etapa do método, a análise do pulso, como já dito anteriormente, utiliza a propagação de onda por diferenças finitas e portanto já se encontra em GPU[26]. Algumas considerações devem ser lembradas ao portar este algoritmo para GPU. A primeira preocupação surge na divisão de blocos e threads com as devidas proteções para o algoritmo não ultrapassar os limites das estruturas de dados. Neste trabalho a divisão de blocos e threads para o cálculo em GPU é baseado no número de amostras do pulso analisado. Como o mesmo é fixo, não é de preocupação a reorganização dos blocos e threads de forma dinâmica.

Outra, é a proteção para que nunca threads diferentes escrevam em um mesmo campo no momento de gerar o arquivo de áudio de saída. Se isto ocorresse, valores concorrentes poderiam ser sobrescritos de forma a alterar o resultado final. Sabemos que esta é uma preocupação relevante, pois como já visto, as janelas de comportamento geradas para cada amostra se sobrepõe muitas vezes, e portanto diversas amostras do arquivo original precisarão acessar um mesmo campo do arquivo gerado.

Uma terceira preocupação é a alocação de memória. No algoritmo criado neste trabalho, o vetor com os valores proporcionais do pulso analisado são colocados na memória global da GPU para que todas threads possam fazer o acesso simultâneo. Também é colocado na memória global o vetor que vai salvar o arquivo de áudio resultante. Os demais valores necessários para os cálculos, como índices e o valor da amostra do arquivo de áudio original são passados durante a chamada do kernel por parâmetros e portanto ficam situados localmente na função.

O Código 3, similar ao de CPU, é o utilizado para fazer o processamento do método em GPU. Internamente a esse código teremos apenas o kernel da GPU que realiza o cálculo que gera o valor proporcional a ser inserido no arquivo de áudio resultante dado i e j correspondentes (sendo i a posição da amostra no arquivo original e j a posição dentro do comportamento do pulso analisado).

```
1 for todas amostras do áudio original do
2 |   callCUDAMethod( gpu_Am, gpu_Rf, int(audioOriginal[i]) , i ,
  |   nAmostrasAudio)
3 end
```

Algoritmo 3: Segunda etapa da heurística em CPU. Fase de reprodução

Cada índice da repetição, que percorre todo o arquivo de áudio, irá chamar o número de threads referentes ao numero de amostras do comportamento do pulso. Portanto, no exemplo do trabalho em que se usa 2205 amostras no pulso, para cada índice percorrido

2205 threads são chamadas para ser executadas em paralelo, fazendo assim o cálculo do comportamento de uma única amostra do arquivo original em 2205 posições do arquivo resultante de uma única vez. Ao terminar o cálculo das 2205 posições, as threads serão sincronizadas. Olhando para a Figura 5.3 pode-se ver que os valores em cada s[índice] são os calculados ao mesmo tempo. Vale lembrar que os mesmos não são armazenados em nenhum vetor s, mas sim somados diretamente no vetor de áudio resultante. Os vetores s da Figura 5.3 são meramente ilustrativos. Com esta solução, garantimos que a escrita no arquivo gerado será realizada ao mesmo tempo apenas por threads que vão escrever em lugares diferentes do vetor de saída, evitando resultados inesperados.

Com os algoritmos acima demonstrados, tornam-se possíveis as avaliações e testes apresentados no próximo capítulo.

Capítulo 6

Resultados e Conclusões

Nesta capítulo iremos apresentar os resultados e discussões que levarão a algumas conclusões desta dissertação. Os testes foram divididos em duas etapas: A primeira etapa engloba testes realizados a partir da criação do método proposto e sua implementação em CPU com a utilização da equação de onda simples[15]. A segunda etapa conta com a avaliação de testes onde a fase de reconstrução do áudio do método é portada para GPU, e a equação de onda simples é substituída pela modelagem com atenuação por distância. Em ambas as etapas serão apresentados e discutidos dois tipos de testes: performance e qualidade (erro).

A implementação realizada para todos os tipos de testes e simulações do trabalho foi feita em C++ utilizando OpenGL para a visualização e NVIDIA CUDA API para o processamento em GPU. Como já citado em outros capítulos, IrrKlang [10] foi usada como biblioteca de apoio para manipulação de arquivos de audio, tanto para leitura quanto para reprodução. O ambiente computacional empregado contou com um TESLA C1060 composto por 30 multiprocessadores, 16 núcleos em cada multiprocessador, e uma CPU de quatro núcleos com 2.4GHz, e 4GB DDR3 de memória. O sistema operacional usado foi o Linux.

6.1 Equação de Onda Simples com Método Proposto em CPU

Esta seção descreve os resultados dos testes feitos após a primeira versão do algoritmo ter sido desenvolvida [15]. Neste momento a integração com a biblioteca de áudio já estava feita, assim como a primeira versão do método proposto que conta com a parte de análise de pulso em GPU e a parte de reprodução do áudio ainda em CPU. Ressaltamos também

que nesta etapa a equação de onda utilizada é equação de propagação de onda simples, sem atenuação, extraída de [26], que serviu de base para esta pesquisa.

6.1.1 Análise de Performance

Considerando que o presente trabalho tem o objetivo de vir a ser um método implantado em games e, portanto, o tempo real é o fator principal a ser considerado, a primeira análise a ser feita é a de performance.

Para os testes comparativos (Tabela 6.1) foi empregada uma cena com as seguintes características: fonte sonora na posição $(x, y) = (64, 20)$ em um ambiente aberto com o ouvinte posicionado em $(x, y) = (64, 64)$), totalizando uma distância de 44 metros entre um e outro. O tamanho do ambiente respeita um grid de tamanho 128x128, onde cada unidade representa um metro. Quatro arquivos de som diferentes foram usados para comparar o tempo de processamento para diferentes quantidades de amostras por arquivo. Como pode ser visto na Tabela 6.1, o tempo de processamento do som usando o método proposto no trabalho é muito menor se comparado ao gasto para fazer o processamento total do áudio apenas pelo método de diferenças finitas. Observa-se aqui, um ganho de até treze vezes no melhor dos casos expostos. Porém, nota-se com estes resultados que apesar do grande ganho, ainda não podem ser considerados em tempo real, onde o tempo de processamento deveria ser menor que o tamanho total de tempo do áudio.

Tabela 6.1: Resultados comparativos de diferentes arquivos de áudio com números variados de amostras. Os resultados estão em segundos e representam o tempo gasto para sintetizar o áudio completo ou reconstruí-lo usando a heurística proposta no trabalho. Erros entre ambos os métodos estão expostos na última coluna.

Arquivo / Tempo Total Aproximado	Número de Amostras	Tempo para Renderizar o Comportamento Real do Som	Tempo para Renderizar pelo Método de Análise de Pulso Proposto	Erro	Erro(dB)
bell.wav / 00:01	32449	84,685	23,295	116	1,99
baby.wav / 00:01	169984	444,234	39,382	45	1,07
footsteeps.wav / 00:02	239616	609,182	47,258	11	1,61
crowdtalk.wav / 01:00	5765766	14702,703 (estimado)	710,932	N/D	N/D

É importante notar também, como descrito no Capítulo 5, que o método proposto possui dois passos: análise e reconstrução. Foi observado, que para todos os arquivos, o tempo do passo de análise levou 19 segundos para todas as simulações. Isto acontece pois o tempo de análise de pulso é o mesmo e a taxa de amostragem, neste caso de 40kHz, também é constante, deixando o resto do tempo para a segunda etapa. Ressaltamos que em trabalhos futuros é possível uma abordagem que faça um pré processamento desta primeira etapa, de forma a suprimir este tempo fixo para qualquer arquivo de áudio. Mais sobre essa sugestão será explanado no Capítulo 7.

6.1.2 Análise de Qualidade

Quanto a qualidade do som, os resultados obtidos são encorajadores para os testes feitos. Na Figura 6.1 é apresentado um comparativo de áudio original *footsteps* contra as duas formas de renderização do som: o comportamento real por diferenças finitas e o método proposto. Pode-se observar pelo gráfico que a diferença entre ambos é mínima. Para uma melhor comparação, a Figura 6.2 mostra apenas os áudios gerados pelos dois métodos, sem o arquivo original.

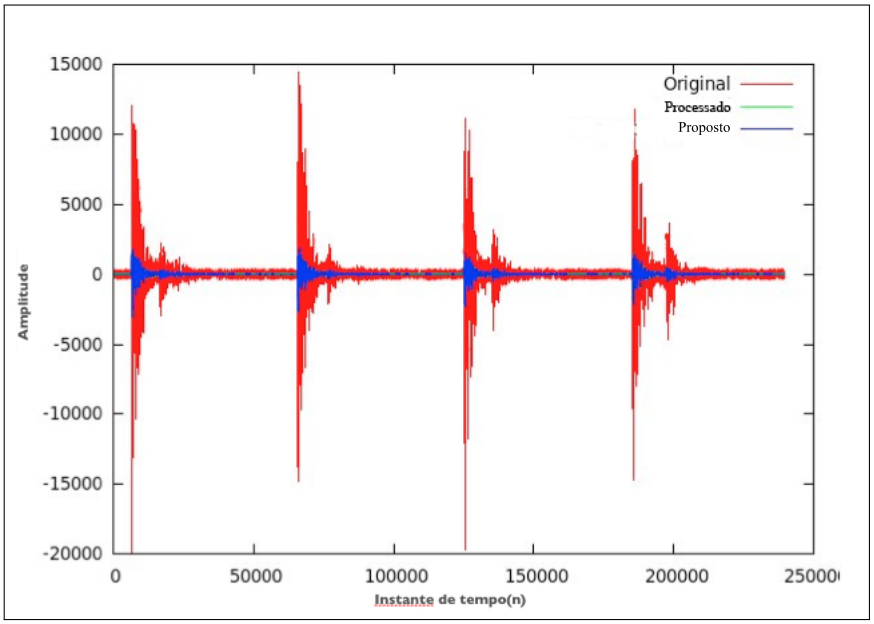

Figura 6.1: Gráfico comparativo do arquivo *footsteps.wav*. Em vermelho o som original. O verde representa o comportamento real para um áudio atingindo um ouvinte em $(x, y) = (64, 64)$. Em azul como o som chega pelo método proposto.

Para uma análise mais acurada, usando os valores dos áudios gerados, uma análise de erro é feita para mostrar o quão próximo o método proposto neste trabalho está em relação ao resultado real por diferenças finitas. Esse erro é calculado pela Equação 6.1:

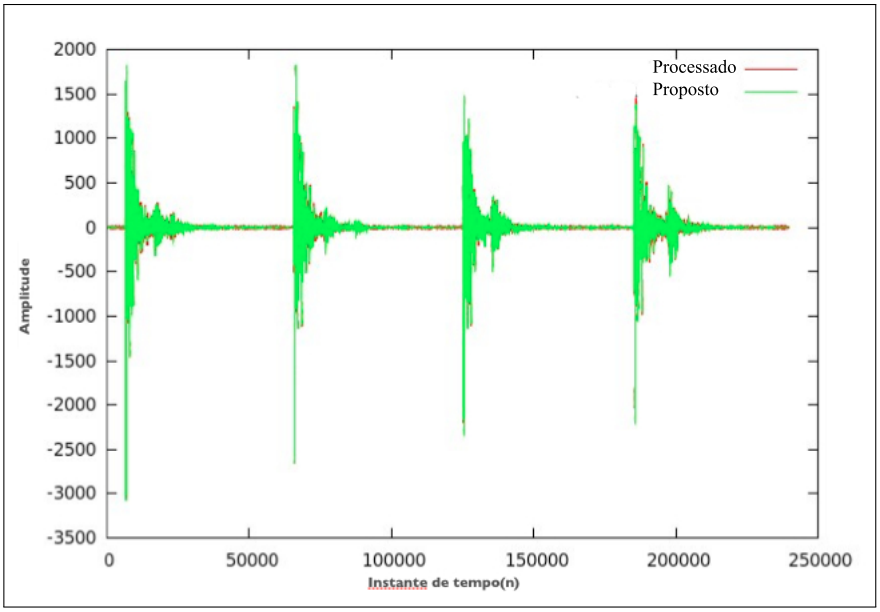

Figura 6.2: Gráfico comparativo do arquivo *footsteps.wav*. O vermelho representa o comportamento real do áudio chegando no ouvinte em $(x, y) = (64, 64)$ após o processamento por diferenças finitas. Em verde, como o som chega pelo método aqui proposto.

$$Erro = \frac{1}{m} * \sum_{k=1}^{m} |(u[k]^p - u[k]^n)| \tag{6.1}$$

Nesta Equação 6.1,m é o número de amostras de audio, u^p é o valor da amostra no método totalmente processado, e u^n é o valor da amostra calculado no método por comportamento de pulso proposto. A Equação 6.1 calcula o erro em amplitude entre os arquivos.

Os resultados do erro podem ser vistos nas últimas colunas da Tabela 6.1. Para analisar se estes erros são considerados relevantes ou não, podemos levar em conta, conforme [11], que o limite para o ouvido humano perceber uma alteração de som é de aproximadamente 1dB sobre as frequências audíveis. Esta interpretação então é feita baseando-se na proximidade do erro com essa margem de 1dB. Como visto, a Equação 6.1 fornece os valores de erro em amplitude. Para saber se o erro está imperceptível ao ouvido humano ou não, é necessário fazer uma conversão de amplitude para decibel utilizando a Equação 6.2.

$$dB = 20 * log_{10}(\frac{P}{P_{max}}) \tag{6.2}$$

Onde $Pmax$ é a amplitude referência e P a amplitude sendo comparada [4]. Os erros em decibéis encontrados foram pequenos e os resultados quando comparados com o método do comportamento real se mostraram praticamente imperceptíveis ao que se deveria escutar.

Esta é uma indicação de que o método proposto provê bons resultados práticos e se essa variação de erros, entre 0 e 2dBs, persistir para uma grande base de dados, pode ser concluído que o som renderizado dessa forma pode ser apropriado a jogos e aplicações interativas.

6.2 Equação de Onda Com Atenuação e Implementação Paralela do Método Proposto em GPU

Esta seção descreve os resultados dos testes realizados com a versão paralela do algoritmo desenvolvido. Aqui, o método proposto já conta com a parte de análise de pulso e a reprodução do áudio em GPU. Mostramos aqui também uma implementação mais precisa da equação da onda, devido a inclusão da atenuação por distância, sendo esta uma contribuição também deste trabalho. Os efeitos que se consegue incluir são especialmente a simulação da perda de energia por calor das partículas durante a transferência da mesma.

6.2.1 Análise de Performance

A primeira análise feita foi na diferença de tempo de processamento entre o algoritmo com a equação de onda simples e o com equação de onda com amortecimento por distância. Neste primeiro teste, independente dos ambientes testados, não foi obtida nenhuma mudança no tempo de processamento entre as implementações, e por esse motivo, tabelas e valores comparativos não serão exibidos para esse teste. Conclui-se que a melhora para a equação mais precisa não é um fator preocupante para a performance do método.

Com essa constatação foi possível seguir para os próximos testes que visam avaliar os resultados das análises de tempo de processamento e de erro encontrado entre o áudio totalmente processado pelo método de diferenças finitas e o áudio gerado em duas etapas pelo método proposto em GPU.

Para os testes que geraram os resultados da Tabela 6.2 foram utilizados 4 arquivos de audio agora com uma taxa de amostragem de 11kHz e 16bits por amostra. Os 4 arquivos são os mesmos utilizados na Seção 6.2, porém convertidos para a taxa citada. Nas colunas que tratam do método proposto os áudios foram processados com uma análise de pulso de 200ms. Dessa vez, como a taxa de amostragem é menor, para reproduzir o comportamento de 200ms no pulso, agora são necessárias apenas 2205 amostras. Foi utilizada uma cena com um grid de tamanho 128x128.

Tabela 6.2: Teste comparativo de tempo para os diversos métodos. Os tempos estão em segundos. Em colunas: (2) Método de reprodução total por diferenças finitas; (3 - 4) Método implementado em [15] com segunda etapa de reconstrução do áudio em CPU; (5 - 6) Método melhorado nesta etapa do trabalho com a segunda etapa de reconstrução do áudio em GPU

Arquivo(wav) / Tempo / Amostras	Tempo de renderização: reprodução total por diferenças finitas	Tempo de renderização: Método de analise de pulso EM CPU - Etapa de Analise / Reprodução	Tempo de renderização: Método de analise de pulso EM CPU- Tempo total	Tempo de renderização: Método de analise de pulso EM GPU - Etapa de Analise / Reprodução	Tempo de renderização: Método de analise de pulso EM GPU - Tempo total
bell / 00:01 / 16224	37,056	5,042 / 0,461	5,503	4,95 / 0,291	5,241
baby / 00:01 / 39042	88,273	5,366 / 1,146	6,512	5,124 / 0,704	5,828
footsteeps / 00:02 / 55036	126,004	5,101 / 1,628	6,729	4,954 / 0,991	5,945
crowdtalk / 01:00 / 1324324	2896,59	5,172 / 39,776	44,948	5,149 / 23,845	28,994

Pode-se notar pela Tabela 6.2 que a etapa de reprodução do comportamento do pulso teve uma melhora significante quando o algoritmo foi passado para GPU, obtendo ganhos de aproximadamente 1,7 vezes nessa etapa. O algoritmo em GPU ainda é uma primeira abordagem paralela, e portanto o código ainda deve permitir otimizações a serem feitas que tragam algum ganho. Também é importante perceber que observando-se para a segunda etapa do método proposto percebe-se que os tempos obtidos são sempre menores que o tempo total de duração do áudio, enquanto isso não necessariamente acontece na primeira etapa (aprox. 5 segundos, independente do tamanho do som, a dependência aqui é com o tamanho do pulso analisado). Este fato leva à conclusão de que o gargalo para o tempo real está na primeira etapa, e que a segunda etapa poderia ser considerada como suficiente para o tempo real. Já que leva menos tempo que o próprio arquivo de áudio, a mesma poderia estar sendo realizada em um buffer enquanto o arquivo resultante já poderia estar tocando.

Alguns testes foram feitos para diferentes tamanhos de grid como 256x256, 1024x1024,

e 2048x2048 com a finalidade de comparar o tempo de processamento com o aumento do domínio. A Tabela 6.3 mostra os resultados obtidos para um grid 256x256. Pode-se observar que apenas a reprodução total por diferenças finitas e a primeira etapa do método proposto(análise de pulso) apresentavam maior tempo de processamento com a mudança para grids maiores. Isto é facilmente explicável, visto que a segunda etapa do algoritmo proposto depende apenas do tamanho do arquivo de áudio e do tamanho do pulso. Pode-se perceber também que a mudança de tempo ainda está escalando de forma pouco significativa visto que o método tratado neste trabalho ainda está no espaço 2D. Portanto, testes exaustivos para observar a mudança de performance com o tamanho do domínio não foram feitos, sendo que estes se tornam de maior interesse quando o método tiver sido transcrito para uma versão 3D.

Tabela 6.3: Testes de tempo realizados para grid de tamanho 256x256. Os tempos estão em segundos. Em colunas: (2) Método de reprodução total por diferenças finitas; (3 - 4) Método deste trabalho com reconstrução do áudio em GPU

Nome Arquivo(wav) / Tempo Total(aprox) / Numero de amostras	Tempo de renderização: reprodução total por diferenças finitas em grid 256x256	Tempo de renderização: Método de analise de pulso EM GPU em grid 256x256 - Etapa de Analise / Reprodução	Tempo de renderização: Método de analise de pulso EM GPU em grid 256x256 - Tempo total
bell / 00:01 / 16224	37,046	5,312 / 0,292	5,604
baby / 00:01 / 39042	88,786	5,309 / 0,702	6,011
footsteeps / 00:02 / 55036	124,858	5,016 / 1,006	6,022
crowdtalk / 01:00 / 1324324	2975,723	5 / 23,923	28,923

6.2.2 Análise de Qualidade

Em relação a qualidade do som gerado, os resultados foram interessantes e relevantes. Para chegar aos resultados, mais uma vez, é feita uma comparação entre o som totalmente processado no método preciso de diferenças finitas e o som gerado pelo método proposto. Essa diferença é chamada de erro, visto que é considerado que o processamento total por diferença finitas gera o que seria, para o caso estudado, o som fisicamente realístico. Na Figura 6.3 podemos ver um outro exemplo da representação gráfica de um som totalmente processado comparado a um som processado pelo método, agora para o arquivo de áudio *Bell.wav*.

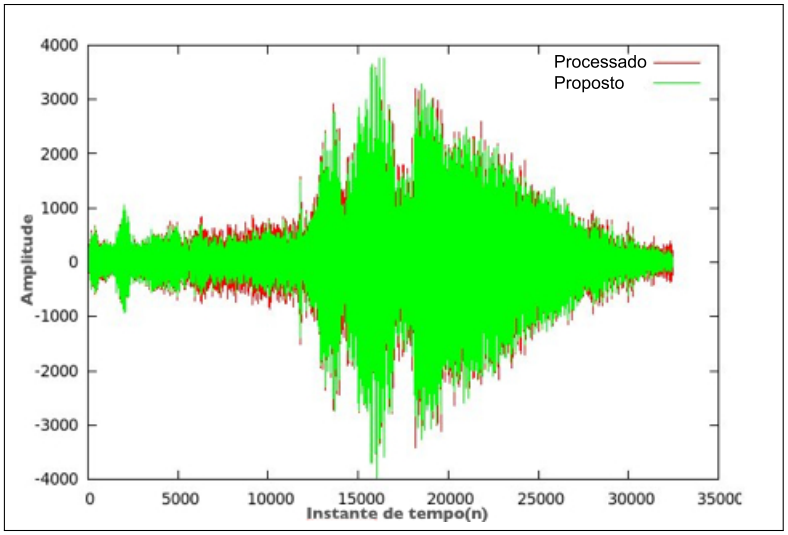

Figura 6.3: Gráfico comparativo do arquivo *bell.wav*. Vermelho representa o comportamento real do áudio chegando no ouvinte em $(x, y) = (64, 64)$ após o processamento por diferenças finitas. O verde representa como o som chega ao ouvinte pelo método proposto.

Como um incremento na base de testes, nesta etapa foram criados 3 ambientes para a realização dos mesmos. Na primeira etapa com o método ainda em CPU e a equação de onda simples[15], os resultados foram gerados em apenas um ambiente externo simples. Nesta etapa do trabalho tomou-se o cuidado de fazer a utilização de ambientes mais complexos, ou seja, que possuem uma geometria mais detalhada, como prédios e muros que possam provocar reflexões e desvios nas ondas sonoras. A Figura 6.4 mostra os 3 ambientes utilizados.

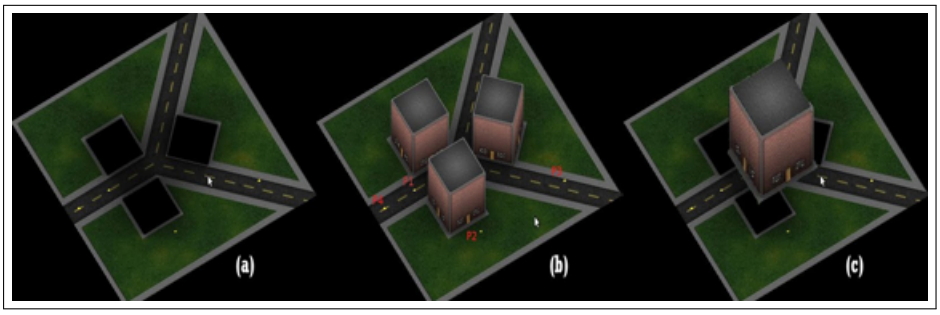

Figura 6.4: (a) ambiente externo sem prédios; (b) ambiente externo com prédios; (c) ambiente com uma representação gráfica de um prédio para simular um galpão no centro. Mesmo que graficamente não representado, é como se o prédio no centro fosse um galpão vazio com uma porta aberta na parte da frente.

Os mesmos quatro arquivos de áudio foram testados em todos os ambientes, sendo que para cada um foram feitas observações em duas posições diferentes de fonte e ouvinte, uma primeira com a fonte e ouvinte próximos e uma segunda com uma grande distância entre ambos. Sabendo que a representação gráfica dos testes não é o suficiente para uma

avaliação precisa, o erro entre cada arquivo gerado pelo método proposto e pelo processamento total é calculado seguindo novamente as Equações 6.1 e 6.2. Para não deixar de exibir os resultados visuais dos testes, os gráficos que os representam são relevantes e estão expostos no Apêndice A.

Para apresentar todos os testes de qualidade feitos, foram montadas diferentes tabelas para cada arquivo de áudio. São elas as Tabelas 6.4, 6.5, 6.6, 6.7.

Tabela 6.4: Demonstrativo dos resultados de erro para os diferentes testes para o arquivo *Bell.wav*

Tipo de Teste (Bell.wav)	Erro Médio	Amplitude média do som processado	Erro médio em dB(feito com média)
Sem Prédios Perto	1,64	90,17	0,156
Sem Prédios Longe	1,71	20,25	0,704
Com Prédios Perto	0,87	21,47	0,345
Com Prédios Longe	0,64	1,65	2,847
Galpão Ouvinte Fora	0,79	3,12	1,960
Galpão Ouvinte Dentro	9,72	109,61	0,737

Tabela 6.5: Demonstrativo dos resultados de erro para os diferentes testes para o arquivo *Baby.wav*

Tipo de Teste (Baby.wav)	Erro Médio	Amplitude média do som processado	Erro médio em dB(feito com média)
Sem Prédios Perto	1,29	147,74	0,075
Sem Prédios Longe	1,24	41,47	0,255
Com Prédios Perto	1,13	58,05	0,167
Com Prédios Longe	0,69	13,27	0,440
Galpao Ouvinte Fora	1,95	23,19	0,701
Galpao Ouvinte Dentro	35,48	515,27	0,578

Observa-se que os erros foram pequenos e até menores quando comparados com os da primeira etapa [15], devido a uma pequena correção do cálculo na GPU que estava sendo feito com números inteiros(int) e deveriam estar sendo feitos em pontos flutuantes(float). O objetivo da implementação com números inteiros era de aumentar a performance, mas esse ganho não estava acontecendo enquanto os resultados estavam piorando consideravelmente, visto que as contas quando feitas com números inteiros geravam um acumulo de erros de truncamento e estes ocasionavam em uma grande diferença no arquivo final. O

Tabela 6.6: Demonstrativo dos resultados de erro para os diferentes testes para o arquivo *Footsteps.wav*

Tipo de Teste (Footsteps.wav)	Erro Médio	Amplitude média do som processado	Erro médio em dB(feito com média)
Sem Predios Perto	0,52	99,09	0,045
Sem Predios Longe	0,53	31,06	0,146
Com Predios Perto	0,48	28,32	0,145
Com Predios Longe	0,27	2,72	0,822
Galpao Ouvinte Fora	1,04	7,04	1,196
Galpao Ouvinte Dentro	34,68	279,83	1,014

Tabela 6.7: Demonstrativo dos resultados de erro para os diferentes testes para o arquivo *Crowdtalk.wav*

Tipo de Teste (Crowdtalk.wav)	Erro Médio	Amplitude média do som processado	Erro médio em dB(feito com média)
Sem Predios Perto	0,63	261,84	0,020
Sem Predios Longe	0,62	82,46	0,065
Com Predios Perto	0,56	145,25	0,033
Com Predios Longe	0,54	16,62	0,277
Galpao Ouvinte Fora	2,62	33,48	0,654
Galpao Ouvinte Dentro	66,49	786,05	0,705

resultados obtidos, mais uma vez levando em consideração a colocação sobre a percepção humana em relação a 1dB[11], mostram uma diferença imperceptível na maioria dos testes entre o algoritmo que processa todo o áudio e o método proposto.

Os maiores erros foram encontrados basicamente em duas situações: Em testes onde o ouvinte se encontra muito distante da fonte sonora, pois a amplitude chega muito baixa e o cálculo de conversão pra decibel, que usa uma escala proporcional, acaba acentuando as diferenças em valores menores e quando eram tratados ambientes com eco, como nos testes em que o ouvinte e a fonte estavam dentro do galpão. Este caso sugere um aumento de tempo de análise do pulso na primeira etapa do processo, pois esse erro ocorre devido as reverberações atrasadas que influenciam bastante o som. O tempo de pulso analisado pode estar pequeno de tal forma que elas estão sendo desconsideradas.

6.3 Resumo dos Resultados e Conclusões

Em ambas as etapas de teste e com as diferentes equações empregadas, os resultados encontrados foram bastante promissores. Até onde os testes mostraram, pelos erros serem praticamente imperceptíveis, o método se torna facilmente utilizáveis em jogos, sendo que a performance melhorou consideravelmente em todos os testes realizados. O autor acredita que com mais alguns avanços nos estudos e implantação de algumas técnicas e suas respectivas implementações em GPU se possa chegar ao tempo real com a abordagem que esta sendo seguida. No entanto, parece que por enquanto para alcançar o tempo real com o método proposto e otimizado neste trabalho, algum pré-processamento que armazene os valores de comportamento de pulso para todos os pontos do domínio(ou pelo menos para os de maior interesse) na primeira etapa do algoritmo deverá ser feito. Mesmo com o avanço das placas gráficas e os recursos de cálculos paralelos disponíveis que permitem que o algoritmo de propagação de onda com atenuação por diferenças finitas seja executado em GPU, o mesmo ainda se mostra muito pesado para o tempo real no estágio em que se encontra.

Capítulo 7

Trabalhos Futuros

O tema abordado neste trabalho é ainda pouco explorado, principalmente quando aplicado a jogos e realidade virtual. Dado seu caráter de novidade, acreditamos que com este trabalho muitas fronteiras foram traçadas e muitos trabalhos futuros podem vir a surgir. Esta seção apresenta as sugestões do autor para a continuação do trabalho.

7.1 Equação da Onda

A primeira sugestão é aumentar a precisão física do cálculo da equação de onda utilizada. Até [26, 15] a equação de onda utilizada é a de propagação de onda simples, que por exemplo, não leva em consideração absorção e atenuação. Neste trabalho já está implementada a equação de propagação de onda que leva em consideração a atenuação por distância, que modela a perda de energia da onda para calor na medida que se propaga pelo ambiente. Modelos mais precisos ainda existem e a sua implementação pode tornar o resultado apresentado cada vez mais realístico. Um exemplo seria de adotar uma equação que leve em consideração também a absorção de energia durante a troca de meios. Ou seja, um modelo que faça a computação para meios heterogêneos.

7.2 Método Baseado no Comportamento do Pulso

Para as etapas do algoritmo proposto algumas sugestões são feitas. Na primeira etapa de análise do pulso foi possível observar pela análise de erro que o tamanho do pulso pode causar erros maiores ou menores dependendo do ambiente a qual o emissor de som e o ouvinte estão inseridos. No caso de um ambiente fechado, por exemplo, é mais interessante que tenhamos uma análise de pulso maior para não perder as reverberações finais da onda

sonora que causam o efeito de eco. Já num ambiente aberto a onda sonora geralmente não volta várias vezes onde o ouvinte está e portanto é como se ela dispersasse antes, exigindo uma análise menor do pulso. O autor sugere que se crie algum método que, através de alguma heurística, analise e defina de forma adaptativa valores diferentes de tempo de análise do pulso. Quanto menor o tempo de análise, mais rápida é a segunda fase de reconstrução, e quanto maior, mais preciso são os resultados.

Para a segunda etapa, é possível que seja feita ainda uma otimização refatorando o algoritmo de reprodução do comportamento do pulso. Como o algoritmo apresentado é uma primeira abordagem, estudos em relação ao tipo de memória onde se devem armazenar as varáveis é um exemplo de como o algoritmo poderia ser melhorado.

Outro tópico relevante a ser investigado, principalmente quando se trata de jogos, é a inclusão de emissores de som e ouvintes em movimento. Em jogos teremos praticamente sempre o jogador se movimentando pelo cenário, mesmo que as fontes sonoras se encontrem estáticas. É aconselhado aqui um estudo de como poderia ser adaptado o método para levar em consideração esta situação. Primeiro pode-se pensar apenas no ouvinte em movimento, e em seguida extender para a fonte em movimento. Um último passo seria a inserção de múltiplas fontes sonoras.

7.3 Pré Computação

Ainda na primeira etapa do método proposto(baseado no comportamento do pulso) onde ocorre a propagação do pulso por diferenças finitas, foi observado que a mesma é o gargalo da simulação quando se fala em tempo real. Sugere-se num primeiro momento considerar a cena de jogo estática e a partir daí fazer um pré-processamento dos valores de análise de pulso entre todos os pontos do domínio. Com os valores da primeira etapa armazenados, acredita-se que ao fazer a segunda etapa com a utilização de buffers, torna-se possível uma inserção de valores em um streaming de saída. Visto que na segunda etapa, para os padrões de testes realizados nesse trabalho, o tempo de reprodução é sempre menor que o tempo do áudio a ser tocado, seria possível, com a utilização destes buffers para streaming, uma reprodução do áudio em tempo real. Esta abordagem prejudicaria a utilização do método em cenas dinâmicas, porém estando em tempo real já seria de grande utilidade para muitos casos. Deve-se ter atenção nesta parte com a utilização de memória para armazenar os valores pré computados. É possível que compactações sejam necessárias para armazenar todos os valores do domínio.

7.4 Expansão do Método para 3D

Como o trabalho começou de um modelo mais simples da propagação de onda, o mesmo foi feito com base no método implementado em [26] que utiliza uma abordagem de propagação de onda num grid 2D. Para expandir o método para o 3D deve ser levado em conta um modelo matemático em que a equação de onda esteja variando nas 3 dimensões. Alguns trabalhos já apresentam esse tipo de modelo, como em [22]. Essa contribuição é um grande passo para a realização de uma simulação realística da propagação das ondas acústicas com o método proposto. A mesma pode também levantar vários questionamentos quanto ao método, visto que o custo do processamento da parte de análise de pulso aumentará consideravelmente.

Referências

[1] ANTONACCI, F.; FOCO, M.; SARTI, A.; TUBARO, S. Real time modeling of acoustic propagation in complex environments. *In Proceedings of 7th International Conference on Digital Audio Effects* (2004), 274–279.

[2] BALEVIC, A.; ROCKSTROH, L.; TAUSENDFREUND, A.; PATZELT, S.; GOCH, G.; SIMON, S. Accelerating simulations of light scattering based on finite-difference timedomain method with general purpose gpus. *n IEEE International Conference on Computational Science and Engineering 0* (2008), 327–334.

[3] BRANDAO, D.; ZAMITH, M.; CLUA, E.; MONTENEGRO, A.; BULCAO, A.; MADEIRA, D.; KISCHINHEVSKY, M.; LEAL-TOLEDO, R. Perfomance evaluation of optimized implementations of finite-difference method fow wave propagation problems on gpu architecture. *In Proceedings of the Computer Architecture and High Performance Computing Workshops, Sociedade Brasileira de Computação* (2010).

[4] FARNELL, A. *Designing Sound*. Applied Scientific Press, London, 2008.

[5] FUNKHOUSER, T.; TSINGOS, N.; ARLBOM, I.; ELKO, G.; SONDHI, M.; WEST, J.; PINGALI, G.; MIN, P.; NGAN, A. A beam tracing method for interactive architectural acoustics. *The Journal of the acoustical society of America 115*, 2 (2004), 739–756.

[6] GOLUB, G.; ORTEGA, J. Scientific computing and differential equations: an introduction to numerical methods. *1st ed. Academic Press* (1991).

[7] GPGPU. General-purpose computation using graphics hardware, February 2010.

[8] HAYES, M. *Schaum's Outline of Digital Signal Processing*, 1 ed. McGraw-Hill, USA, 1998.

[9] HAYKIN, S.; VEEN, B. V. *Sinais e Sistemas*. Bookman, Porto Alegre, RS, 1999.

[10] IrrKlang. A cross platform sound library for c++, c and all .net languages., 2010.

[11] JESTEADT, W.; WIER, C.; GREEN, D. Intensity discrimination as a function of frequency and sensation level. *The Journal of the acoustical society of America 61*, 1 (1977), 169–177.

[12] KOWALCZYK, K.; WALSTIJN, M. V. Virtual room acoustics using finite difference methods. *In Proceedings IEEE Int. Symp. Communication, Control Signal Processing (ISCCSP)*, 1504 (2008).

[13] LYONS, R. G. *Understanding Digital Signal Processing*, 3rd ed. Prentice Hall, Michigan, USA, 2011.

[14] MICHEA, D.; KOMATITSCH, D. Accelerating a threedimensional finite-difference wave propagation code using gpu graphics cards. *Geophysical Journal International*, 182 (2010), 389–402.

[15] Moreira, B.; Brandão, D.; Kuryla, C.; E., C.; Kischinhevsky, M. An architecture using finite difference method to calculate realistic sounds equalization in games. *X Simpósio Brasileiro de Games e Entretenimento DIgital. Proceedings of Sbgames 2011* (2011).

[16] NVIDIA. Cuda programming guide, 2010.

[17] RAGHUVANSHI, N.; NARAIN, R.; LIN, M. C. Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Transactions on Visualization and Computer Graphics 15*, 5 (2009), 789–801.

[18] RAGHUVANSHI, N.; NICO, G. Accelerated wavebased acoustics simulation. *ACM Symposium on Solid an Physical Modeling* (2008), 91–102.

[19] RAGHUVANSHI, N.; SNYDER, J.; MEHRA, R.; LIN, M.; GOVINDARAJU, N. Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. *ACM Transactions on Graphics 29*, 4 (July 2010).

[20] REYNOLDS, A. Boundary condition for the numerical solution of wave propagation problems. *Geophysics 1*, 43 (1978), 1099–1110.

[21] RÖBER, N.; KAMINSKI, U.; MASUCH, M. Ray acoustics using computer graphics technology. *In Proceedings of the 10th International Conference on Digital Audio Effects* (2007), 01–08.

[22] SABINO, T.; ZAMITH, M.; BRANDAO, D.; MONTENEGRO, A.; KISCHINHEVSKY, M.; LEAL-TOLEDO, R.; SILVEIRA, O.; BULCAO, A.; CLUA, E. Scalable simulation of 3d wave propagation in semi-infinite domains using the finite difference method on a gpu based cluster. *In Proceedings of the V e-Science Workshop 1* (2011), 110–118.

[23] SAVIOJA, L.; RINNE, T.; TAKALA, T. Simulation of room acoustics with a 3-d finite difference mesh. *n Proceedings of Internation Computer Music Conference (ICMC94)* (1994), 463–466.

[24] SILTANEN, S. *Efficient physics-based room-acoustics modeling and auralization.* Tese de Doutorado, Aalto University School of Science and Technology, Espoo, Finland, 2010.

[25] Y., N.; G.P., N. Fast water animation using the wave equation with damping. *Procs. 5th Int. Conf. on Computational Science ICCS 2005 3515* (May 2005), 232–239.

[26] ZAMITH, M.; PASSO, E.; BRANDAO, D.; CLUA, E.; MONTENEGRO, A.; KISCHINHEVSKY, M.; LEAL-TOLEDO, R. Sound wave propagation applied in games. *In Proceeding of IX Brazilian Symposium of Games and Digital Entertainment., Sociedade Brasileira de Computação* (2010).

APÊNDICE A – Gráficos dos Sons Processados

A seguir os gráficos que comparam os arquivos de áudio processados totalmente pelo método de diferenças finitas e os arquivos de áudio gerados pelo método proposto para uma mais rápida renderização. Esses gráficos servem de suporte visual para analisar e interpretar o comportamento do áudio processado, mas não são suficientes para provar o resultado do método. Análises númericas dos valores de amplitude dos mesmos são feitas e expostas na seção de resultados da dissertação.

Os gráficos expostos levam em consideração uma taxa de amostragem de 11kHz(11025 amostras por segundo) e 16bits de precisão por amostra do arquivo de áudio processado.

As Figuras A.1,A.2,A.3 e A.4 mostram os resultados para os testes no ambiente aberto sem prédios.

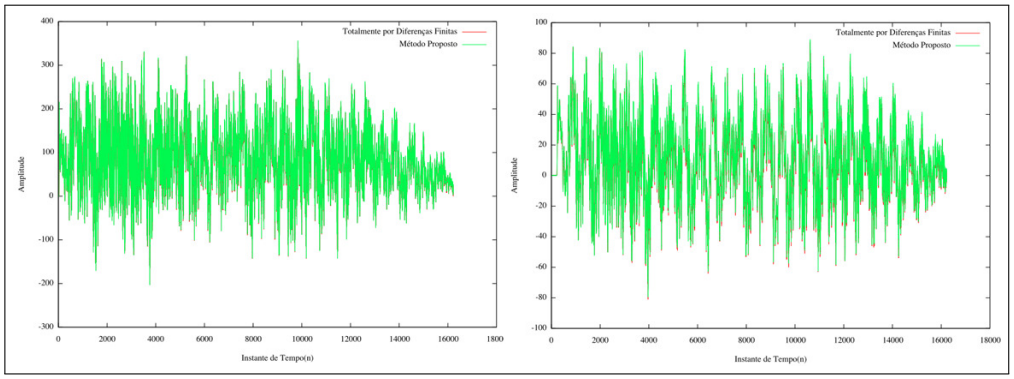

Figura A.1: Gráfico comparativo do arquivo *Bell.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(64,35) e fonte em (x,y)=(64,25). Na direita, ouvinte em (x,y)=(64,100) e fonte em (x,y)=(64,25).

As Figuras A.5,A.6,A.7 e A.8 mostram os resultados para os testes no ambiente aberto com prédios.

As Figuras A.9,A.10,A.11 e A.12 mostram os resultados para os testes no ambiente com o galpão ao centro.

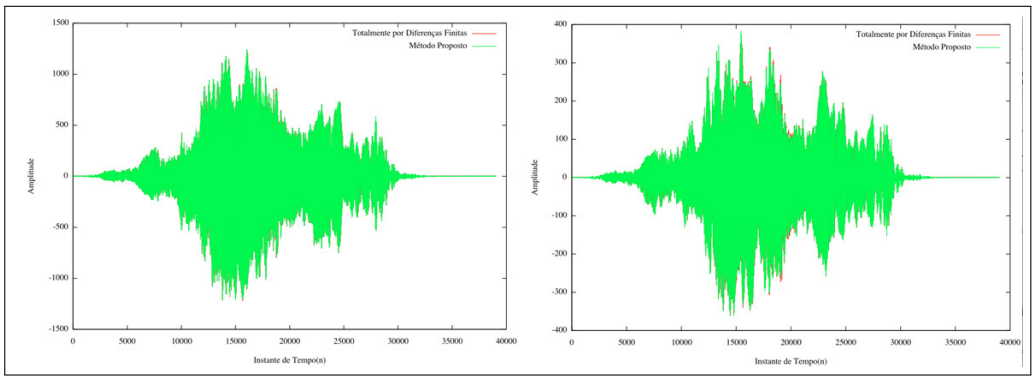

Figura A.2: Gráfico comparativo do arquivo *Baby.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(64,35) e fonte em (x,y)=(64,25). Na direita, ouvinte em (x,y)=(64,100) e fonte em (x,y)=(64,25).

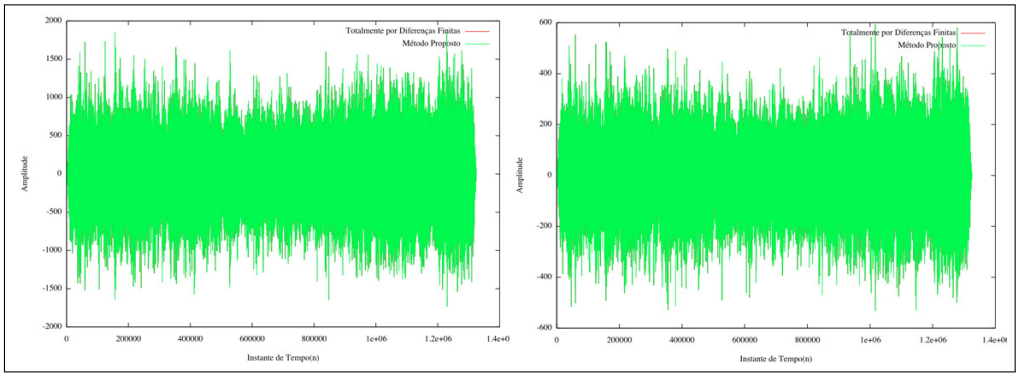

Figura A.3: Gráfico comparativo do arquivo *Crowd-Talking.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(64,35) e fonte em (x,y)=(64,25). Na direita, ouvinte em (x,y)=(64,100) e fonte em (x,y)=(64,25).

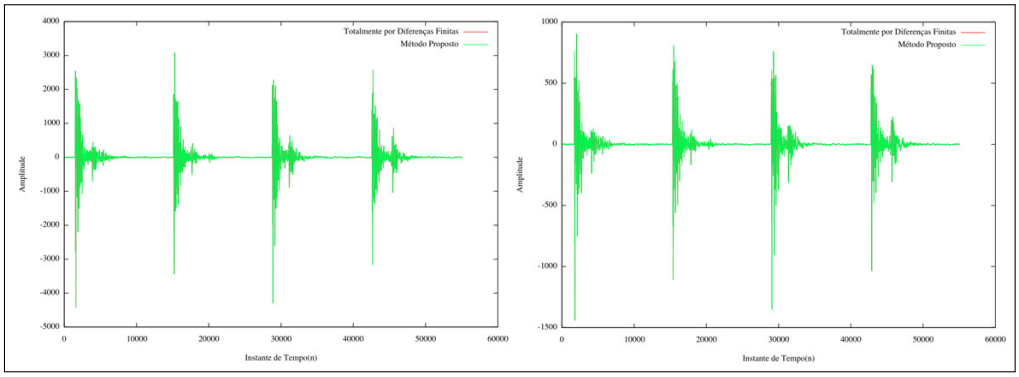

Figura A.4: Gráfico comparativo do arquivo *Footsteps.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método.Na esquerda, ouvinte em (x,y)=(64,35) e fonte em (x,y)=(64,25). Na direita, ouvinte em (x,y)=(64,100) e fonte em (x,y)=(64,25).

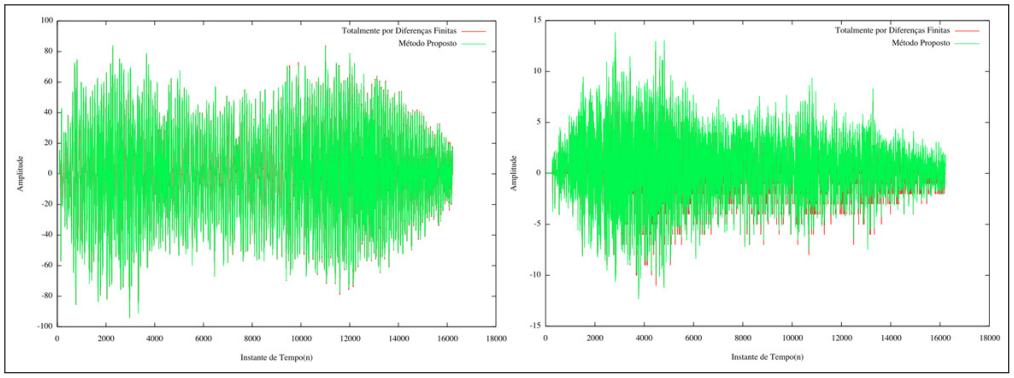

Figura A.5: Gráfico comparativo do arquivo *Bell.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(58,78) e fonte em (x,y)=(58,48). Na direita, ouvinte em (x,y)=(80,110) e fonte em (x,y)=(60,30).

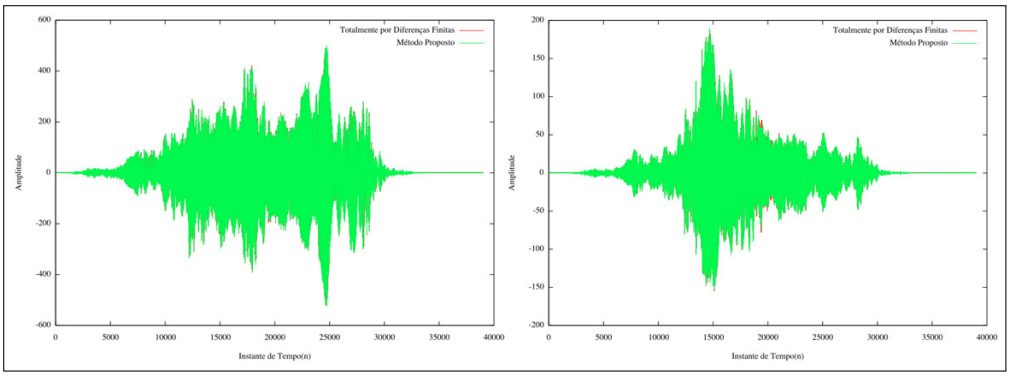

Figura A.6: Gráfico comparativo do arquivo *Baby.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(58,78) e fonte em (x,y)=(58,48). Na direita, ouvinte em (x,y)=(80,110) e fonte em (x,y)=(60,30).

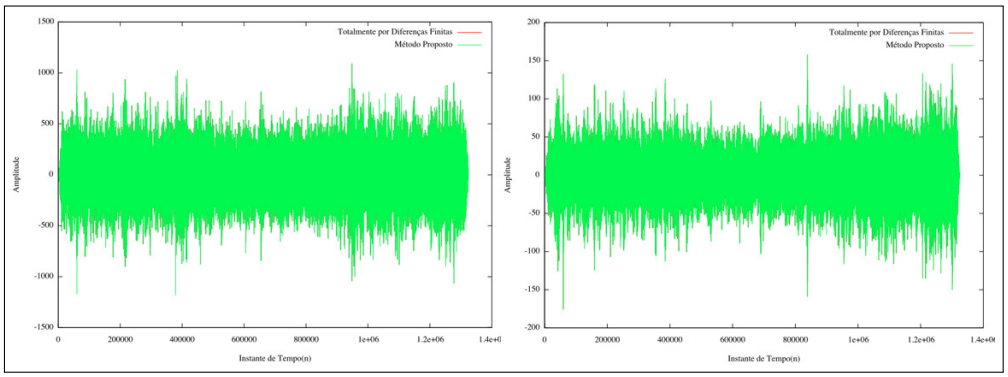

Figura A.7: Gráfico comparativo do arquivo *Crowd-Talking.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(58,78) e fonte em (x,y)=(58,48). Na direita, ouvinte em (x,y)=(80,110) e fonte em (x,y)=(60,30).

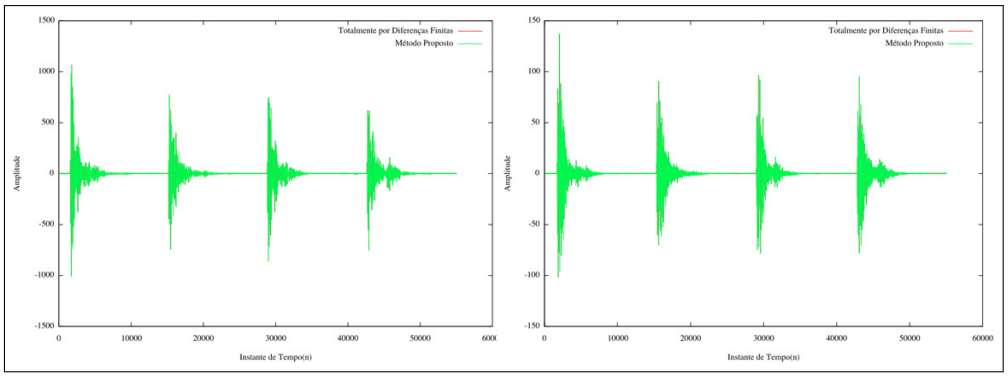

Figura A.8: Gráfico comparativo do arquivo *Footsteps.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(58,78) e fonte em (x,y)=(58,48). Na direita, ouvinte em (x,y)=(80,110) e fonte em (x,y)=(60,30).

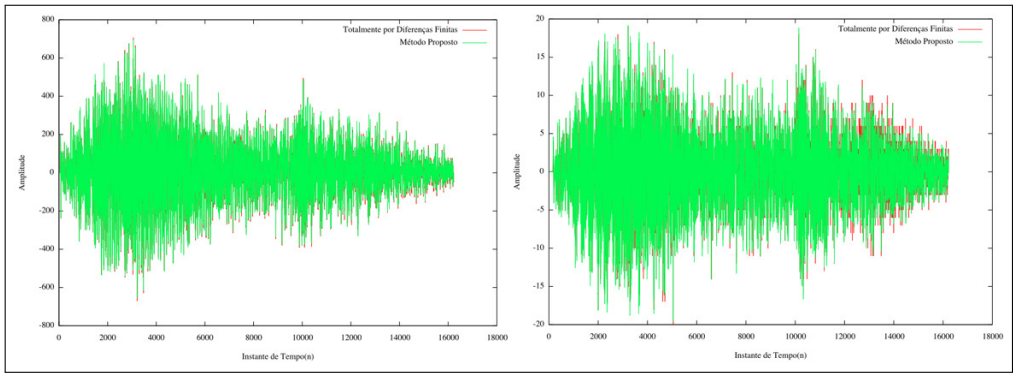

Figura A.9: Gráfico comparativo do arquivo *Bell.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(64,64) e fonte em (x,y)=(75,75). Ambos dentro do galpão. Na direita, ouvinte em (x,y)=(30,30) e fonte em (x,y)=(64,64). Fonte dentro e ouvinte fora do galpão.

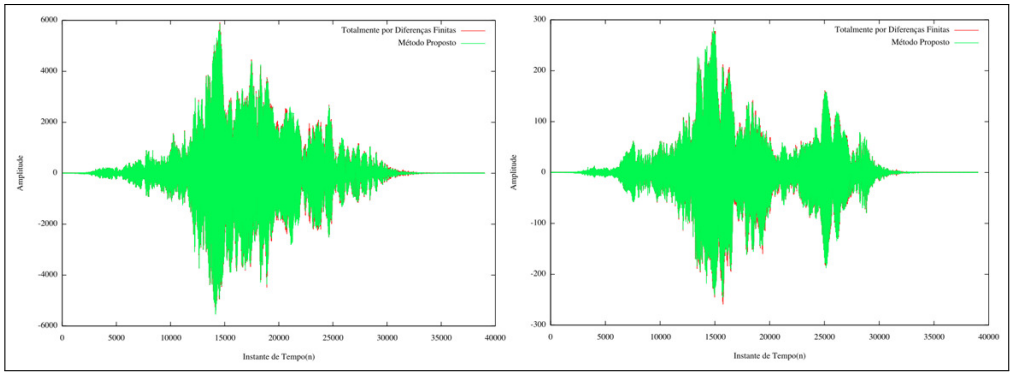

Figura A.10: Gráfico comparativo do arquivo *Baby.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(64,64) e fonte em (x,y)=(75,75). Ambos dentro do galpão. Na direita, ouvinte em (x,y)=(30,30) e fonte em (x,y)=(64,64). Fonte dentro e ouvinte fora do galpão.

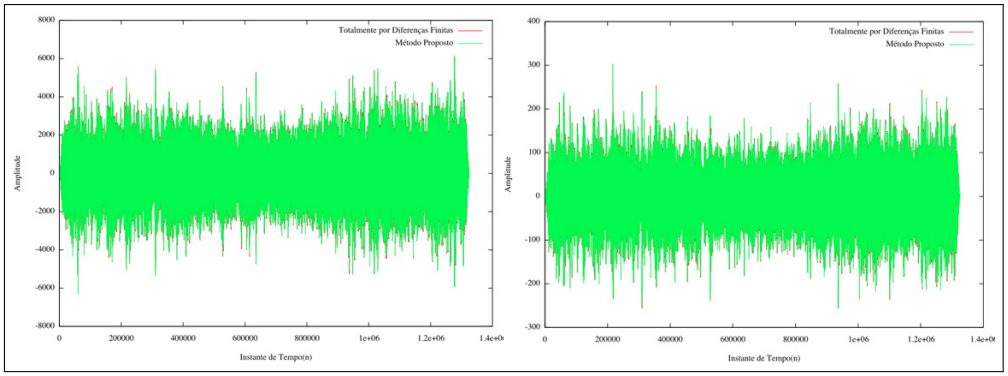

Figura A.11: Gráfico comparativo do arquivo *Crowd-Talking.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(64,64) e fonte em (x,y)=(75,75). Ambos dentro do galpão. Na direita, ouvinte em (x,y)=(30,30) e fonte em (x,y)=(64,64). Fonte dentro e ouvinte fora do galpão.

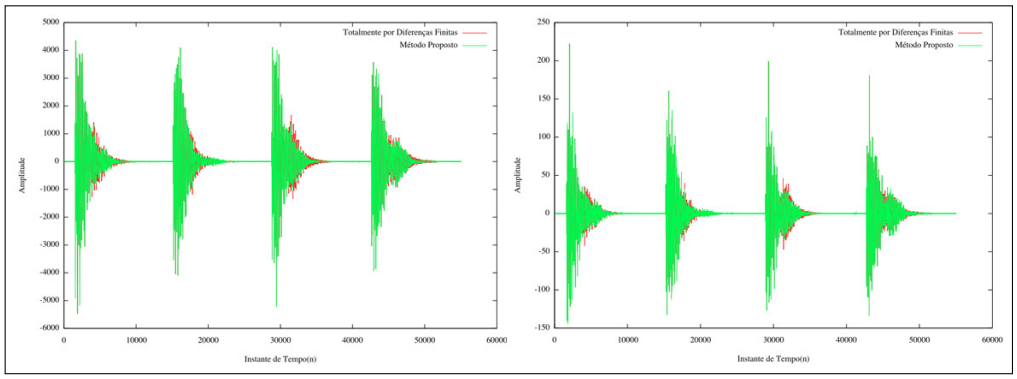

Figura A.12: Gráfico comparativo do arquivo *Footsteps.wav*. Em vermelho o som totalmente processado por diferenças finitas. Verde representa o processamento pelo método. Na esquerda, ouvinte em (x,y)=(64,64) e fonte em (x,y)=(75,75). Ambos dentro do galpão. Na direita, ouvinte em (x,y)=(30,30) e fonte em (x,y)=(64,64). Fonte dentro e ouvinte fora do galpão.

APÊNDICE B – Publicações Geradas pelo Trabalho

A publicação aqui incluída não faz parte de outras dissertações. A publicação é feita pelo autor da dissertação durante os estudos do tema. E ela é parte do trabalho aqui apresentado.

Publicação [P1]

A publicação [15] demonstra os primeiros resultados dos estudos. Apresentando o sucesso na integração da biblioteca de som. Na reprodução total e renderização do áudio através de diferenças finitas. Mostrando porque a mesma não é válida para processamento em tempo real. É introduzido também o método proposto(Capítulo 5), porém ainda com parte dele implementado em CPU. Resultados em cenas simples são apresentados e mostram o potencial do método. O método, suas implementações e testes foram feitos por Bruno Moreira, mesmo autor desta dissertação e que escreveu 90% do artigo.

An Architecture Using a Finite Difference Method to Calculate Realistic Sound Equalization in Games

B. Moreira
E.W. C. Gonzales
M. Kischinhevsky
MediaLab
Computer Departament
Universidade Federal Fluminense

C.L.Kuryla*
Florida International University

D. Brandão**
Centro Federal de Ensino Tecnológico
Celso Suckow da Fonseca - CEFET/RJ
Computer Departament
Universidade Federal Fluminense

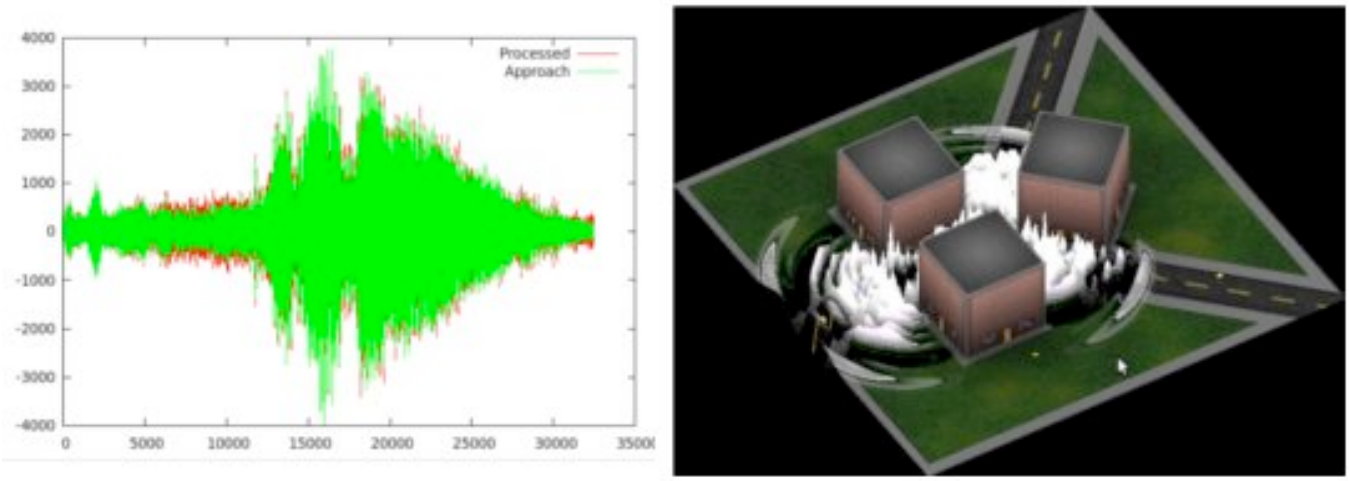

Figure 1: *Illustration of the proposed Sound propagation approach.*

Abstract

Most games and other interactive virtual environments focus on rendering natural phenomena in the most believable manner by using accurate visuals and physics. However, not much effort has been put into accounting for the physics of sound. The simulation of the real behavior of sound through an environment, when considering the speed of sound, reflection, and absorption, is computationally expensive and is usually left aside. In this work, an algorithm that calculates sound wave propagation using a finite difference method is used and extended to present a novel approach to sound rendering. This approach reaches the objective more quickly, and the sound generated has no perceptible loss of accuracy. The approach is designed to be implemented in GPU architectures and eventually enable real-time results.

Keywords:: Sound Wave Propagation, Finite Difference Method, GPGPU, Audio Effects

Author's Contact:

{bmoreira,esteban,kisch}@ic.uff.br
*ckury001@fiu.edu
**dbrandao@ic.uff.br

1 Introduction

Games are constantly being improved to describe reality in a more believable way. Many electronic games, especially those which are classified as simulation-games, rely on the modeling of natural phenomena in order to immerse the player in the game and promote the suspension of disbelief. However, simulating sound wave propagation has not been considered a priority, due in part to its computational complexity. Positional audio libraries have signifcantly contributed to this field, but the industry, as well as most academic research, has always been focused on providing more realistic graphics and, to some extent, rigid-body physics simulations. Positional audio libraries are now a common commodity and current implementations focus only on calculating intensity and pan based on the relative position and orientation of the sound source and the virtual listener. These libraries do not consider important issues such as the time the sound takes to travel through the environment (i.e. air), reflection, and the absorbency of the objects present in the scene. Taking scene geometry into account would greatly improve the immersion experience. Scene geometry affects the way sound travels through the virtual environment, which often changes the perceived equalization and the direction from which the listener hears the generated sound.

Despite the obvious benefits, simulation methods for such phenomena are computationally intensive, and running them in real-time, concurrently with all of the other expensive tasks, has not been a viable option. However, current Graphics Processing Unit (GPU) [GPGPU 2010] technology has the computing power to run and implement a simulation like this in real-time without compromising the overall performance, i.e. the GPU works as mathematics coprocessor.

When determining the scene geometries effect on sound traveling through an environment, it is necessary to use a mathematical model that represents this propagation physically. In order to solve this model computationally, two methods commonly used are finite difference and finite element methods. Finite element methods are more computationally expensive than finite difference methods with an explicit approach, so the choice of the second method is more appropriate for a game environment. [Zamith et al. 2010] implemented an algorithm that uses the finite difference method in a GPU. This work shows an accurate method for calculating the scattering of acoustic waves in a non-homogeneous medium and can be done in real-time.

In [Zamith et al. 2010], the sound wave propagation calculation was used to consider scene geometry while rendering visual effects and check the direction of the sound when it reached the listener. In this paper, the previous work is extended and a first approach is proposed using a finite difference method implemented in GPUs to do the actual rendering of a sound that travels through a medium and is heard by a listener at some distance.

After implementing code that took an audio file as input data for the source position, as in [Zamith et al. 2010], code was created to capture the audio data at the destination (listener position) and generate the output sound. When these functions were implemented, it was found that with a sample rate of 44 kHz, the method could reach a real-time processing speed for visual effects (as shown in Figure 1-right image), but not for sound rendering. This work proposes an approach to capture the output audio in a faster way by using a partial processing of the wave propagation, while still obtaining results which are perceptually acceptable and close to how a human would hear the sound if it were totally processed by the method.

2 Related Work

Techniques for simulating sound propagation may be classified into two categories: geometric acoustics (GA) and numerical acoustics (NA). Previous works based on numeric methods are in general too computationally expensive for real-time use, while approaches based on geometric techniques do not accurately represent the physics of sound because they overlook effects such as diffraction and scattering.

Geometric acoustics assumes rectilinear propagation of sound waves, so most approaches make use of ray-casting [Röber et al. 2007] and beam tracing [Funkhouser et al. 2004; Antonacci et al. 2004] techniques.

Numerical acoustics directly solves the wave equation governing physical propagation of sound in a scene. With enough computation, all wave phenomena can be captured, including diffraction and scattering in complex scenes. Numerical approaches are insensitive to the complexity and polygon count of a scene and instead scale with its physical dimensions. Two methods commonly used are finite difference and finite element methods. Several works demonstrate the use of these methods, such as [Balevic et al. 2008; Michea and Komatitsch 2010; Raghuvanshi and Nico 2008; Savioja et al. 1994; Kowalczyk and Walstijn 2008], but they are not suitable for real-time use.

Sound is considered to have low priority when making games realistic, so few works attempting to realistically render sound in games were found. [Raghuvanshi et al. 2010; Siltanen 2010] uses a technique with a pre-computed scenario, but a problem arises when considering memory usage and the changing state of the game at runtime. Moreover, all the GPU and CPU resources are typically used for physics and computer graphics calculation. Since CUDA architecture basically allows only one concurrent thread, most engines and commercial applications prefer to use the computer horsepower for these tasks, so not many resources are available for the sound computation. However, new architectures of GPUs are allowing the execution of concurrent kernels, which allows real-time sound rendering to be done in parallel with the computer graphics and typical physics calculations. No previous work was found that uses numeric methods without pre-computed scenarios to take the scene geometry into account and still reach a real-time sound rendering that is close to reality.

3 Acoustic Wave Equation

In this section, the mathematical model that physically represents the scattering of acoustic waves and the numerical approach adopted are presented. The wave equation is a second order differential equation that describes the behavior of sound waves over time. The acoustic wave field is described by $P(x, y, t)$ and $u(x, y, t)$, where P is the pressure field and u is the particles displacement.

The relation between P and u is given by $P(x, y, t) = -k\nabla^2 u(x, y, t)$. Thus, the 2D wave equation can be represented by Equation 1;

$$\frac{\partial^2 P(x,y,t)}{\partial t^2} = c^2 + \left[\frac{\partial^2 P(x,y,t)}{\partial x^2} + \frac{\partial^2 P(x,y,t)}{\partial y^2}\right] + f(x,y,t) \tag{1}$$

where x and y are Cartesian coordinates, t is time, $c = c(x, y)$ is the velocity of an acoustic wave, and $f(x, y, t)$is the source term.

To numerically solve the partial differential equation (PDE), several techniques may be employed. The method that was chosen is based on the Finite Difference Method (FDM), which replaces the quest for a solution on the continuous domain by one on a finite number of points, the grid points, which cover the whole domain. Specifically, after the discretization of the PDE, one obtains a set of equations, the finite-difference (FD) equations, which approximate the differential equation at all grid points. Each FD equation around a grid point is written as an algebraic expression which involves its neighboring points to replace the spatial derivatives.

The approximation of the PDE at the grid points is done through spatially centered differences, which offer second order spatial accuracy. The technique used for the time discretization in this work is second order as well. The proper algebraic expressions can be obtained with Taylor series expansions. Using a second order approximation for space and time, as explained, and assuming that $h = \Delta x = \Delta y$ and $t = n\Delta t$, the wave equation is rewritten in its discretized form as Equation 2;

$$\begin{gathered} P_{i,j}^{n+1} = 2P_{i,j}^{n} - P_{i,j}^{n-1} + \Delta t f(x,y,t) + \\ \frac{A}{12}\left[P_{i-1,j}^{n} + P_{i+1,j}^{n} - 4P_{i,j}^{n} + P_{i,j-1}^{n} + P_{i,j+1}^{n}\right] \end{gathered} \tag{2}$$

where $A = \left(\frac{c(x,y)\Delta t}{h}\right)^2$ and $n = 1, 2, 3, ...$ represents the time instant. Since the velocity field does not vary with time, it is not a function of time [Golub and Ortega 1991].

The discrete expression in Equation 2 provides explicit computation of approximate values at time instant $(n + 1)$, once those at time instants $(n - 1)$ and (n) are available at all grid points. The explicitness is a convenient feature that allows efficient use of parallelism since each grid points value at the $(n + 1)$th time instant is computed independently. Another aspect of discrete approximations for differential equations is that some implicit discretization strategies provide unconditional stability, while the one employed here provides conditional stability. That is, time steps have to be less than an upper limit, above which the numerical solution blows up (the upper limit can be found with the von Neumann criterion), i.e. its amplitude increases non-physically. On the other hand, unconditional stability would allow larger time steps than the explicit methods upper limit. However, the implicit method is not efficiently parallelized on GPUs. Consequently the explicit technique of Equation 2 was chosen, and a careful choice of time steps was made.

4 Finite Difference Method on GPU and Implementation Aspects

The Graphics Processing Unit (GPU) is a hardware specially designed to handle tasks related to visual effects. A GPU can be considered similar to a CPU, but with different processing units that are designed specifically for graphical data processing. Modern high-end GPUs have computational power far greater than CPUs.

Before CUDA, the concept of GPU Computing was to map the problem as a set of vertex and fragments to generate a texture representing the final desired solution. Since 2006, with the NVIDIA CUDA release[Nvidia 2010], the world of high-performance computing became more accessible. A GPU with hundreds of cores is now available for a tenth of the price of a cluster with the same computational power.

CUDA extends existing programming languages by adding a set of instructions that allow code execution on NVIDIA GPU's. As a consequence of its original design, which was for visual effects, GPU memory structure is divided into global, texture, and shared memories. Shared memory is a small but extremely fast memory. This speed comes from its physical proximity to the processing core. Texture memory is a read only memory which is slower than

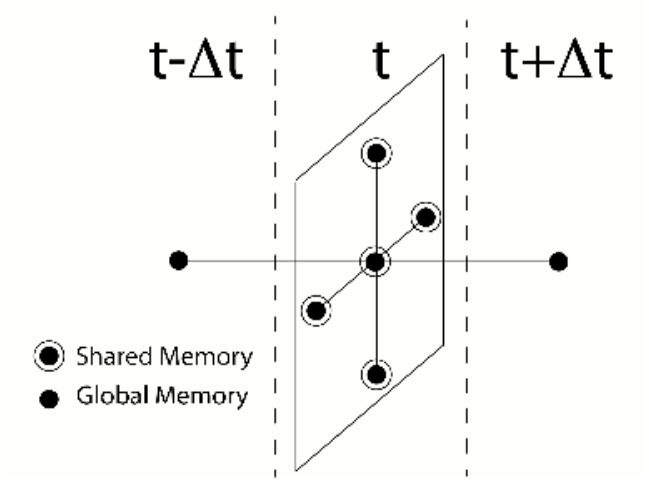

Figure 2: *Second order 2D finite difference operator. To compute the next step $(t + \Delta t)$ it is necessary to fetch the actual (t) and the past one $(t - \Delta t)$.*

the shared memory, but is almost twice as fast as the global memory.

As described in [Brandao et al. 2010; Sabino et al. 2011], efficient finite-difference implementations take advantage of shared memory. Using a single GPU, the amount of memory that needs to be allocated for the iterative solution of Equation 2 is equal to three times the domain size. The past and present time instants and the velocity field need to be available at any time. Figure 2 shows a 2D FD explicit scheme with a second order spatial FD operator.

A typical CUDA application workflow consists of four basic steps: (1) initialize the necessary data on the host (CPU side), (2) copy the data from the host to the device (GPU side), (3) invoke the kernel that will process the data in the device, and (4) read the data back to the host.

For the FD problem, the velocity field is initially allocated and sent to the device memory. Sufficient memory is initialized and allocated to hold values for past and present instants of time. The amplitude values for $P^0(x, y)$ and $P^1(x, y)$ are determined using a forward explicit approximation FD scheme. Equation 2 is then used for subsequent time steps.

A GPU can run millions of lightweight threads efficiently. To help with the management of these threads, CUDA uses the concept of a grid of thread blocks. Threads are organized into blocks, which in turn are organized in a grid. The proposed approach computes each new value $P^{(n+1)}(x, y)$ with only one thread. To compute the next step $(t + \Delta t)$ it is necessary to fetch the actual (t) and the previous one $(t - \Delta t)$.

For efficient memory access, shared memory is used to hold values of some value in a simulation step. This avoids unnecessary reads from global memory. Figure 2 shows that a thread must access up to five values of the same instant $P^{(n)}(x, y)$. Bringing these values into shared memory makes the accesses much faster. The values of $P^{(n-1)}(x, y)$ are fetched directly from global memory. Since all the threads of the same block will only access one position, a coalesced memory read taking maximum advantage of the performance of GPU architecture can be guaranteed.

5 Real Audio Reproduction

This section explains how an audio file works and how the implemented method processes it to generate what this work considers to be the real behavior of sound or the sound totally processed. This will provide a basis for the next section, which will explain the heuristic used for a faster calculation.

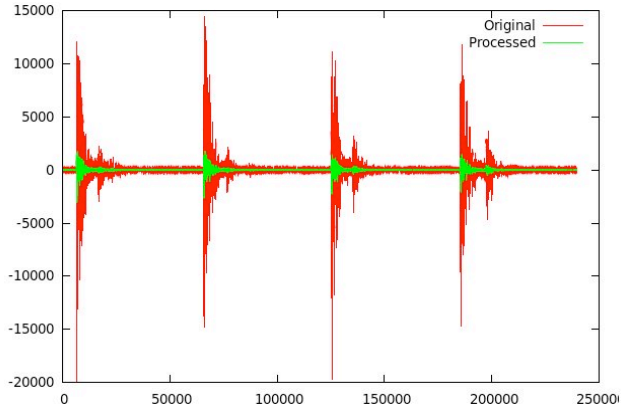

Figure 3: *The comparison between the original sound file footsteps.wav and the processed file. The x-axis represents the sample number and the y-axis is the amplitude.*

5.1 Audio file

An audio file is composed of sample values that represent the discretization of the wavelength passing through the point of capture at a given time interval [Farnell 2008]. This file has two parameters: the number of samples per second, and the accuracy of the sample. The former indicates how many wavelength values are analyzed per second. The higher the value of samples per second, the greater the number of frequencies the audio file can represent.

According to sampling theory, the number of sample points per second must be at least two times greater than the highest frequency in a signal. In order to cover the human hearing range, from 0Hz to 20kHz, it is necessary to have at least 40000 sample points every second [Farnell 2008]. The second parameter describes the order of approximation. It is the precision of a sample (resolution or sample size), i.e. the number of bits that are reserved to represent a single sample value (16,32,64-bits, etc.). The higher the value per sample, the more accurate the audio will be. However, the accuracy also depends on the audio input and output hardware, so a very high precision value for each sample time is usually not required.

5.2 Audio processing

To perform the audio processing using finite differences, pulses that correspond to the samples in a sound file are inserted at a point in the environment. This point is designated as a source (representing a radio or speaker, for example), so that at each iteration the method will calculate the propagation of the pulses that come out from the point. The pulses are given until the audio file ends, at which point, the source stops emitting sounds.

For each iteration of the finite difference method, the amplitude of the waves propagated through the whole domain for a chosen time interval can be calculated. Then, a sample of sound arriving at a specific place can be read, using the considered method, by reading the amplitude value at the specific place for a time instant. To calculate the real behavior of a sound at a point of interest, the samples at the point are read using the time interval required for the given/desired precision of audio frequency, for example, 40000 samples per second. Reading the samples during the time of interest will then give the real sound as it would be heard at the position where the read was done. The sample values are stored and then used to generate a new sound file. This new file contains the sound observed when considering the real propagation of the waves through the scene.

In Figure 3 an original and processed audio file are shown together. This figure experiment used a sound source at position $(x, y) = (64, 20)$ in an open scene with the listener positioned at $(x, y) = (64, 64)$, simulating a distance around 44 meters.

The above figure confirms the intuition that when the receiver and source are separated by a sufficient distance, the sound amplitudes

are attenuated because of the dispersion of waves through the environment. The figure also demonstrates that the amplitude variations of the processed and original sounds are very similar, which means that when the new sound is played, it closely resembles the original sound, but with a lower volume.

The problem with this real reproduction of sound is the amount of computing time required. It is not fast enough for a real-time algorithm, and as the aim here is for use in games, it must happen in real-time to be considered plausible for use. The results in the following sections show the average processing time for different audio files under the conditions described for Figure 3. The times are clearly not viable for use in real-time applications.

The objective is now to create a method that not only considerably reduces the processing time for the rendering of audio files, but also maintains sufficient audio quality, so that the quality loss when using the method is imperceptible to humans. This way, the resultant audio file will sound realistic (or very close) when heard. The proposed method in this work is still not suitable for real-time use. However, it opens new possibilities for studies and improvements in the algorithm, possibly enabling future work to attain a processing time suitable for the purposes and requirements of games.

6 The Proposed Approach

The implementation of the method that considers the wave propagation through the scene, and the subsequent analysis of the resulting sounds after they were completely processed, revealed the necessity for the development of a new approach which could still make use of the behavior of the propagation of a real wave, but possibly reproduce the behavior for the whole audio file. This could result in a significant reduction in the audio rendering time.

To address this improvement, a new approach was developed. A method was created to analyze only one pulse of a wave, and then reproduce the pulse behavior for an entire audio file. The pulse is emitted from the source position and analyzed over an interval of t seconds. In this work, the time interval used is 200ms, represented by 8000 samples considering a system sample rate of 40kHz. This interval was chosen only as an interval for first results; the criteria taken into consideration to select this time were only that it was not too short, so it was not short enough to lose some effects like late reverberation and echo, and that it was not too long, so it was not long enough to interfere with the calculation of the behavior of the wave by sampling at a time after the wave had already spread out and attenuated.

This single pulse analysis is done by reading the amplitude values that reach the point (x,y) where the listener is positioned. This is done in the same way as it is done for the entire audio processing as described in Section 5.2. An example of the behavior of a single propagated pulse is shown in Figure 4. After the single pulse propagation is calculated, the values are compared to the original pulse value, and the comparison yields the proportional values based on the propagation time and the original amplitude.

After the analysis step comes the reconstruction phase. In the reconstruction phase, the system scans the whole original audio file, and for each sample, applies the values that would represent the same behavior as calculated through the pulse analysis. These values (that represent the behavior of each sample) are stored in memory and used to generate the new sound. Each sample generates a window of values representing its individual behavior; this window will affect the next 200ms of sound (this was the chosen time of pulse observation). As shown below, this behavior will overlap for each sample, and when this happens the values are added. This summation can be done because of the superposition property of acoustic waves[Farnell 2008]. Figure 5 shows how the heuristic works.

7 Results and Conclusion

Our implementation was made in C++ using OpenGL for visualization and NVIDIA CUDA API for GPU processing. We used irrKlang to manipulate audio files for reading and playing. The

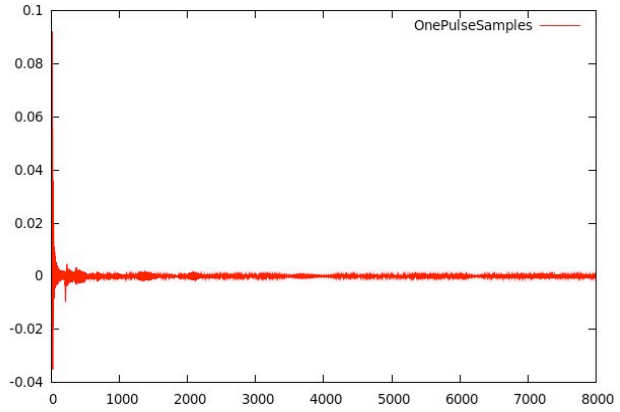

Figure 4: *The behavior of a single pulse propagation analyzed for 200ms.*

computational environment employed was a TESLA C1060 composed of 30 multiprocessors, and a quad core CPU with 2.4GHz, 4GB DDR3 memory, running on a Linux OS.

For the comparative tests (see Table 1) the scene used was constructed as follows: the sound source was at position (x,y) = (64,20) in an open environment, with the listener established at (x,y) = (64,64), simulating a distance of 44 meters from one to the other. Four different audio files were used to compare the processing time for different quantities of samples per file. As shown in Table 1, significantly less time is needed to reproduce the sound when using the approach presented in this work than the time needed to totally process the audio for the real behavior.

Table 1: *Comparative results of different audio files with different numbers of samples in a* 128×128 *grid. The results are in seconds and represent the time taken to render the complete audio or to render the audio using this work's heuristics. Errors between the methods are shown in last two columns.*

File Name Approx Total Time	Number of Samples	Time to render: real behavior of sound file(s)	Time to render: pulse analysis heuristics (s)	Error	Error (db)
bell.wav / 00:01	32449	84,685	23,295	116	1.99
baby.wav / 00:01	169984	444,234	39,382	45	1.07
footsteeps .wav / 00:02	239616	609,182	47,258	11	1.61
crowdtalk .wav / 01:00	5765766	14702,703 (estimated - not rendered)	710,932	N/A	N/A

As described in Section 6, the proposed method has two steps: analysis and reconstruction. For all of the files, the analysis step took 19 seconds (because the listener and source were in the same position in every trial), and the rest of the time was the reconstruction phase. To obtain a faster result, one suggestion is to have a preprocessing step to minimize (or even almost eliminate) the time required for the analysis phase. Another suggestion is to use a GPU calculation kernel that could solve the reconstruction step operations in parallel.

In reference to the sound quality, the test results were very encouraging. Figure 7 (at the end of the paper) shows a comparison between the original footstep audio file and the two methods used to render the processed sound: the real behavior obtained by the total processing and the behavior rendered by the proposed method. Observation shows the difficulty to even discern the difference between the two methods used to generate the processed sound since the results are so close. Figure 8 (also at the end) only shows the two methods, without the original file, for an easier comparison. Two additional audio file tests are shown in Figure 9 and Figure 10 (also at the end of the paper).

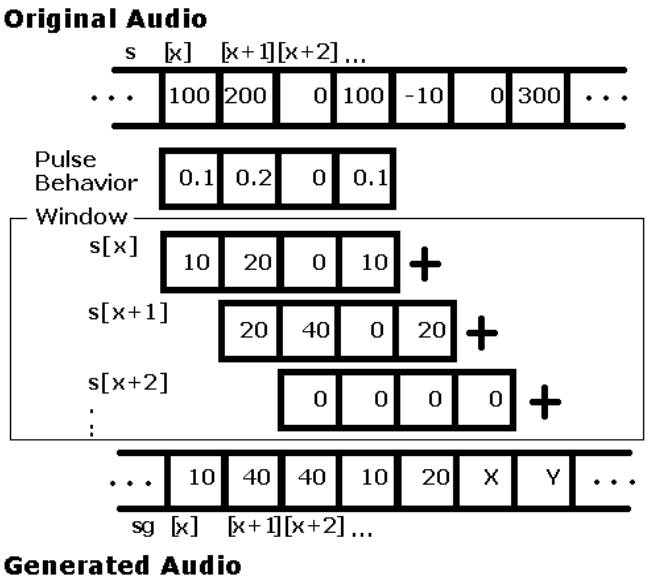

Figure 5: *Visual representation of the heuristic, using an example of a pulse with 4 samples shown.*

Observation of the superimposed graphs clearly shows that the results were very close, but the graphs are only part of the complete analysis. An error analysis was performed on the values to determine how close the proposed method came to the real, totally-processed result, as well as the perceptibility of the difference. The calculated errors are shown in the last two columns of Table 1. The errors in the amplitude values were first calculated using Equation 3:

$$Error = \frac{1}{m} * \sum_{1}^{m} (u^p - u^n) \tag{3}$$

where m is the number of audio samples, u^p is the sample value in the totally processed method, and u^n is the sample value the new proposed method. Equation 3 gives the error in the form of amplitude values, so further analysis regarding the perceptibility of the difference was possible. The relationship between amplitude and decibel is given by Equation 4:

$$dB = 20 * log_{10}\left(\frac{P}{P_{\max}}\right), \tag{4}$$

where $P_{\max}$ is the reference amplitude and P is the amplitude being compared [Farnell 2008]. Using Equation 4, the values in the last column of Table 1 were generated. The threshold for human loudness discrimination is roughly 1 dB over all audible frequencies [Jesteadt et al. 1977], so the differences between the results obtained from the new method and the real behavior method were almost imperceptible. Considering these tests, the proposed method provides good results in practice, and if this range of error persists for a large database, then sound rendered this way could be suitable for a game.

Additional tests were performed in an open scene, but with the addition of buildings, to examine the results obtained when other acoustic effects are present due to a different geometry. Effects such as reflections and diffractions can be observed in this type of simulation. Figure 6 shows one of the tests as an example. This test is not expressed in Table 1, but the figure shows that the rendering still produces a close result when using the proposed method.

As suggestions for future works, the authors would be interested in an improvement in the wave propagation method used, which in this work was the [Zamith et al. 2010] method, because [Zamith et al. 2010] uses a simple finite difference method of wave propagation that, for example, does not consider attenuation and absorption. Another suggestion for future work is to include an adaptive algorithm to calculate the number of samples needed for the pulse

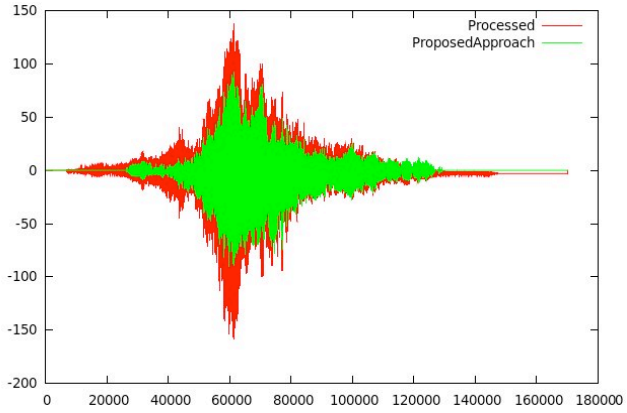

Figure 6: *Comparative graphic of the baby.wav file. The red is the audio as it reaches the listener in the position (x,y) = (84,20) after total processing, considering the source to be at (x,y) = (64,64). The green is the process done by the proposed method. In this test the environment had buildings and the listener was positioned behind a building relative to the source position used.*

in the analysis step. Using as few samples as possible without affecting the result would greatly improve the processing time for the reconstruction phase.

Acknowledgements

The authors gratefully acknowledge CNPq, CAPES and FAPERJ for the financial support of this work.

References

ANTONACCI, F., FOCO, M., SARTI, A., AND TUBARO, S. 2004. Real time modeling of acoustic propagation in complex environments. In *Proceedings of 7th International Conference on Digital Audio Effects*, 274–279.

BALEVIC, A., ROCKSTROH, L., TAUSENDFREUND, A., PATZELT, S., GOCH, G., AND SIMON, S. 2008. Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose gpus. In *IEEE International Conference on Computational Science and Engineering*, vol. 0, IEE, 327–334.

BRANDAO, D., ZAMITH, M., CLUA, E., MONTENEGRO, A., BULCAO, A., MADEIRA, D., KISCHINHEVSKY, M., AND LEAL-TOLEDO, R. 2010. Perfomance evaluation of optimized implementations of finite-difference method fow wave propagation problems on gpu architecture. In *Proceedings of the Computer Architecture and High Performance Computing Workshops*, Sociedade Brasileira de Computação.

FARNELL, A. 2008. *Designing Sound*. Applied Scientific Press, London.

FUNKHOUSER, T., TSINGOS, N., ARLBOM, I., ELKO, G., SONDHI, M., WEST, J., PINGALI, G., MIN, P., AND NGAN, A. 2004. A beam tracing method for interactive architectural acoustics. *The Journal of the acoustical society of America 115*, 2, 739–756.

GOLUB, G., AND ORTEGA, J. 1991. *Scientific Computing and Differential Equations: an Introduction to Numerical Methods*, 1st ed. Academic Press.

GPGPU, 2010. General-purpose computation using graphics hardware. `www.gpgpu.org`, February.

JESTEADT, W., WIER, C., AND GREEN, D. 1977. Intensity discrimination as a function of frequency and sensation level. *The Journal of the acoustical society of America 61*, 1, 169–177.

KOWALCZYK, K., AND WALSTIJN, M. V. 2008. Virtual room acoustics using finite difference methods. In *Proceedings IEEE Int. Symp. Communication, Control Signal Processing (ISCCSP)*, 1504.

MICHEA, D., AND KOMATITSCH, D. 2010. Accelerating a three-dimensional finite-difference wave propagation code using gpu graphics cards. *Geophysical Journal International 182*, 389–402.

NVIDIA. 2010. *Cuda Programming Guide*. Nvidia.

RAGHUVANSHI, N., AND NICO, G. 2008. Accelerated wave-based acoustics simulation. In *Proceedings of the 2008 ACM Symposium on Solid an Physical Modeling*, 91–102.

RAGHUVANSHI, N., SNYDER, J., MEHRA, R., LIN, M., AND GOVINDARAJU, N. 2010. Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. *ACM Transactions on Graphics 29*, 4 (July), 11.

RÖBER, N., KAMINSKI, U., AND MASUCH, M. 2007. Ray acoustics using computer graphics technology. In *Proceedings of the 10th International Conference on Digital Audio Effects*, 01–08.

SABINO, T., ZAMITH, M., BRANDAO, D., MONTENEGRO, A., KISCHINHEVSKY, M., LEAL-TOLEDO, R., SILVEIRA, O., BULCAO, A., AND CLUA, E. 2011. Scalable simulation of 3d wave propagation in semi-infinite domains using the finite difference method on a gpu based cluster. In *Proceedings of the V e-Science Workshop*, vol. 1, 110–118.

SAVIOJA, L., RINNE, AND TAKALA, T. 1994. Simulation of room acoustics with a 3-d finite difference mesh. In *Proceedings of Internation Computer Music Conference (ICMC94)*, 463–466.

SILTANEN, S. 2010. *Efficient physics-based room-acoustics modeling and auralization.* Master's thesis, Aalto University School of Science and Technology, Espoo, Finland.

ZAMITH, M., PASSO, E., BRANDAO, D., CLUA, E., MONTENEGRO, A., KISCHINHEVSKY, M., AND LEAL-TOLEDO, R. 2010. Sound wave propagation applied in games. In *Proceeding of IX Brazilian Symposium of Games and Digital Entertainment.*, Sociedade Brasileira de Computação.

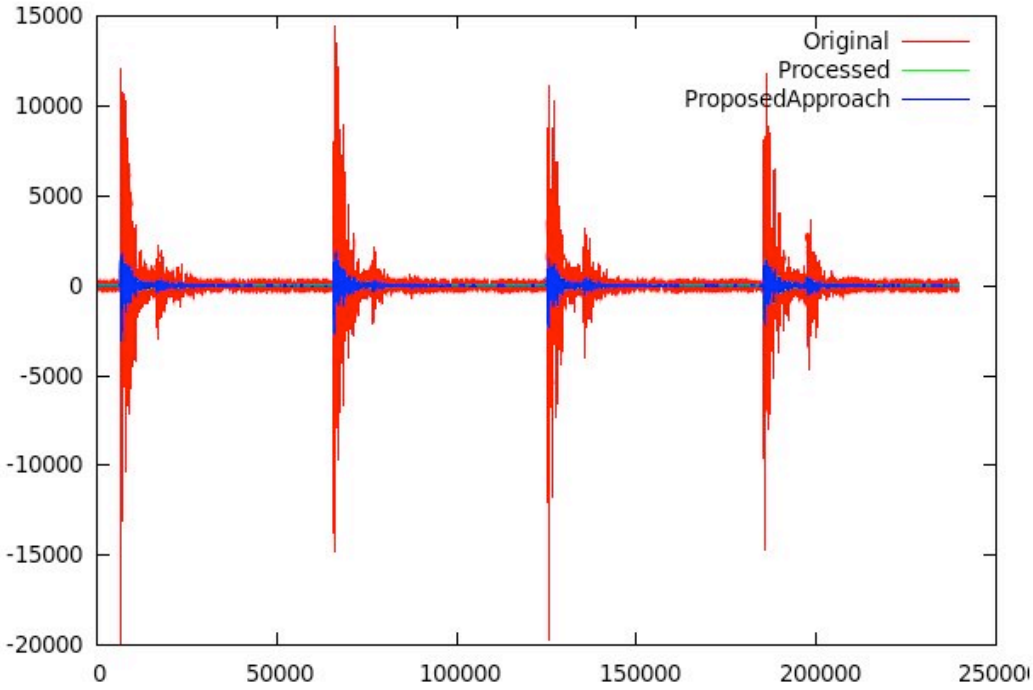

Figure 7: *Comparative graphic of the footsteps.wav file. Red represents the original sound. Green represents the real behavior audio arriving for a listener at (x,y) = (64,64). Blue represents how the sound arrives with the proposed method.*

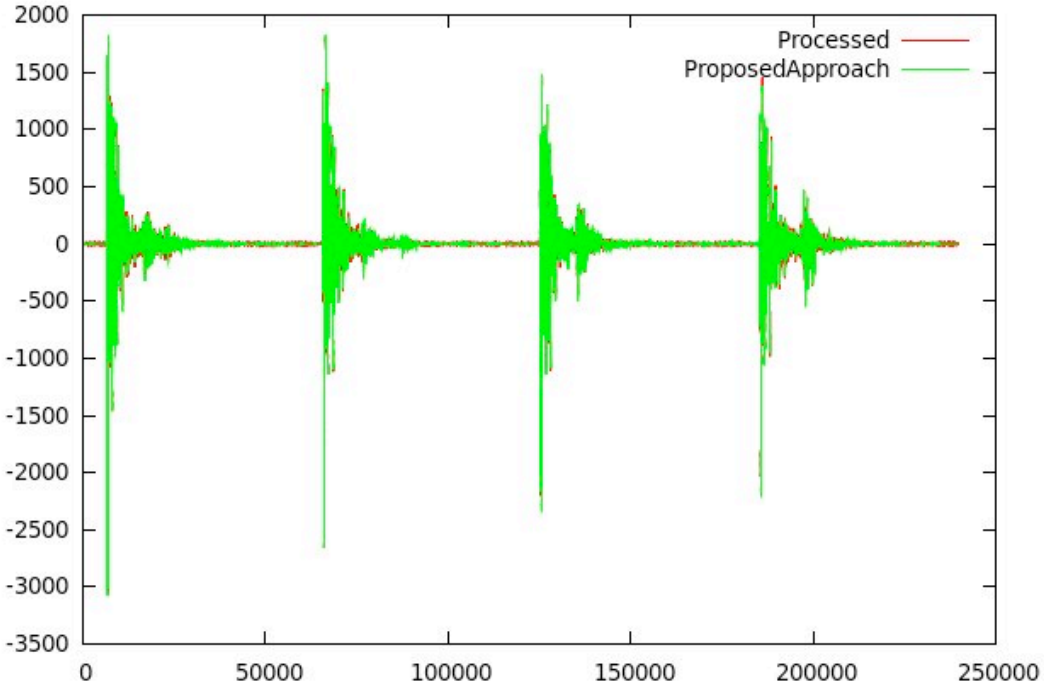

Figure 8: *Comparative graphic of the footsteps.wav file. Red represents the real behavior audio arriving for a listener at (x,y) = (64,64) after processing. Green represents how the sound arrives with the proposed method.*

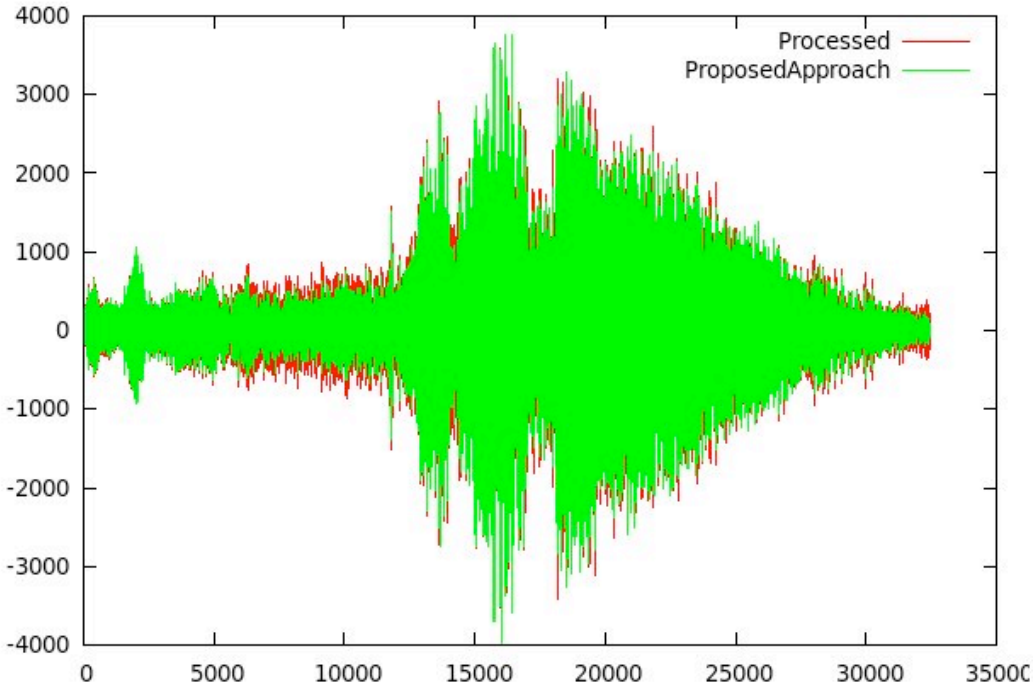

Figure 9: *Comparative graphic of the bell.wav file. Red represents the real behavior audio arriving for a listener at (x,y) = (64,64) after processing. Green represents how the sound arrives with the proposed method.*

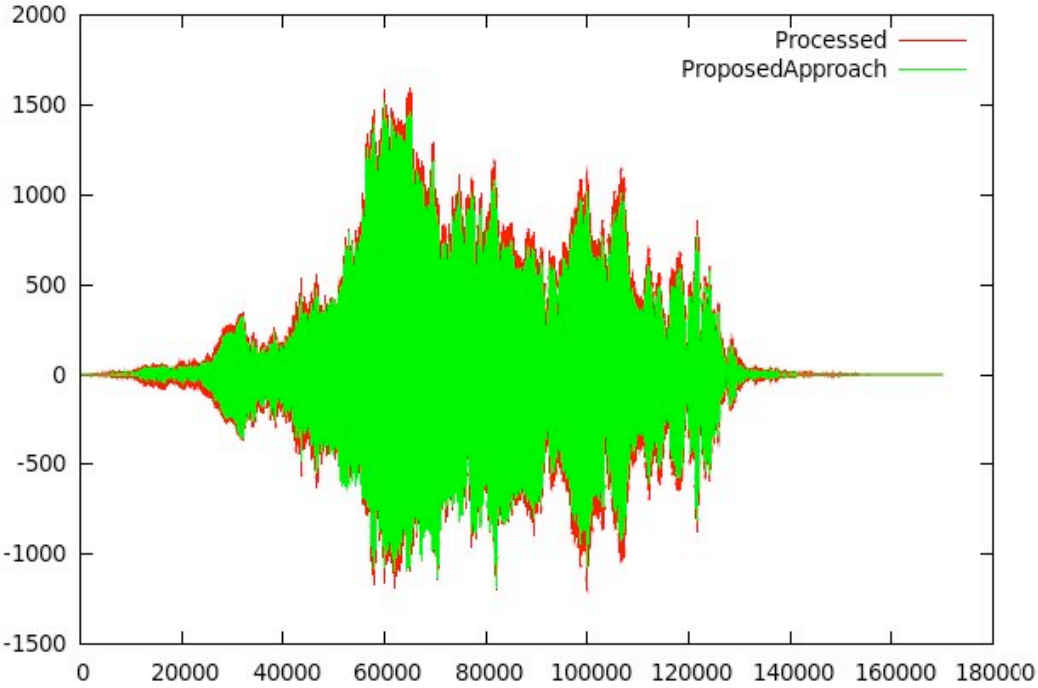

Figure 10: *Comparative graphic of the baby.wav file. Red represents the real behavior audio arriving for a listener at (x,y) = (64,64) after processing. Green represents how the sound arrives with the proposed method.*