UNIVERSITY OF CALIFORNIA SAN DIEGO

Systems and Algorithms for Real-Time Audio Signal Processing

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Signal and Image Processing)

by

Louis A. Pisha

Committee in charge:

Professor Bhaskar D. Rao, Chair
Professor Shahrokh Yadegari, Co-Chair
Professor Truong Nguyen
Professor Piya Pal
Professor Miller Puckette

2022

The Dissertation of Louis A. Pisha is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2022

DEDICATION

To every PhD student who endeavors to carefully balance their passions with the constraints of funding and PI attention.

To my graduate student worker Union siblings at UC San Diego and across the country whose collective action has won, and will continue to win, improved wages and working conditions for all of us.

To everyone who works to elevate technology from merely useful to deeply meaningful.

And to J. and the others.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SUPPLEMENTAL FILES

File 3.1. VDaT demo video showing example scene

File 3.2. Python source code for offline diffraction simulation, including VDaT

File 4.1. SSNRD examples video, music sound source

File 4.2. SSNRD examples video, pink noise sound source

File 4.3. Python source code for SSNRD training and evaluation

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Shahrokh Yadegari for the academic, creative, and emotional support he offered me throughout my time at UC San Diego.

I would also like to acknowledge Research Scientist Harinath Garudadri, Professor Shahrokh Yadegari, Professor Jules Jaffe, and Professor Bhaskar D. Rao for their financial support of my studies at UC San Diego.

Chapter 1, in full, is a reprint of the material as it appears in IEEE Access. Pisha, Louis; Warchall, Julian; Zubatiy, Tamara; Hamilton, Sean; Lee, Ching-Hua; Chockalingam, Ganz; Mercier, Patrick P; Gupta, Rajesh; Rao, Bhaskar D; Garudadri, Harinath, IEEE, November 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in full, is a reprint of the material as it appears in the 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Pisha, Louis; Ligowski, Łukasz, IEEE, May 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in the Journal of the Acoustical Society of America (JASA) 148 (4). Pisha, Louis; Atre, Siddharth; Burnett, John; Yadegari, Shahrokh, The Acoustical Society of America, October 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, has been submitted for publication of the material as it may appear in IEEE Transactions on Visualization and Computer Graphics. Pisha, Louis; Yadegari, Shahrokh, IEEE, 2022. The dissertation author was the primary investigator and author of this paper.

VITA

2013	Bachelor of Arts in Liberal Arts St. John's College, Annapolis, MD
2013–2016	Additional undergraduate work in Electrical Engineering Stony Brook University, NY
2018	Master of Science in Electrical Engineering (Signal and Image Processing) Department of Electrical & Computer Engineering University of California San Diego
2022	Doctor of Philosophy in Electrical Engineering (Signal and Image Processing) Department of Electrical & Computer Engineering University of California San Diego

PUBLICATIONS

Louis Pisha, Krishna Chaithanya Vastare, Sergio Luna, Tamara Zubatiy, Ganz Chockalingam, and Harinath Garudadri. "Tools for assessing efficacy of hearing loss compensation" (presentation). 175th Meeting of the Acoustical Society of America, May 2018.

Louis Pisha, Sean Hamilton, Dhiman Sengupta, Ching-Hua Lee, Krishna Chaithanya Vastare, Sergio Luna, Tamara Zubatiy, Cagri Yalcin, Alex Grant, Mark Stambaugh, Arthur Boothroyd, Ganz Chockalingam, Rajesh Gupta, Bhaskar D. Rao, and Harinath Garudadri. "A Wearable Platform for Hearing Aids Research" (poster). 2018 International Hearing Aid Conference (IHCON).

Louis Pisha, Sean Hamilton, Dhiman Sengupta, Ching-Hua Lee, Krishna Chaithanya Vastare, Tamara Zubatiy, Sergio Luna, Cagri Yalcin, Alex Grant, Rajesh Gupta, Ganz Chockalingam, Bhaskar D. Rao, and Harinath Garudadri. "A Wearable Platform for Research in Augmented Hearing." 2018 IEEE Asilomar Conference on Signals, Systems, and Computers.

Louis Pisha. "Advancing the State of the Art in Real-Time Acoustic Modeling and Reconstruction for Source Separation" (presentation). ACTOR (Analysis, Creation, and Teaching of Orchestration) Project Workshop, July 13, 2019. IRCAM, Paris, France.

Louis Pisha, Julian Warchall, Tamara Zubatiy, Sean Hamilton, Ching-Hua Lee, Ganz Chockalingam, Patrick P. Mercier, Rajesh Gupta, Bhaskar D. Rao, and Harinath Garudadri. "A Wearable, Extensible, Open-Source Platform for Hearing Healthcare Research." IEEE Access, October 2019.

Louis Pisha, Siddharth Atre, John Burnett, and Shahrokh Yadegari. "Approximate diffraction

modeling for real-time sound propagation simulation." Journal of the Acoustical Society of America (JASA), October 8, 2020. pp. 1922-1933.

Louis Pisha and Łukasz Ligowski. "Accelerating non-power-of-2 size Fourier transforms with GPU Tensor Cores." 2021 IEEE International Parallel and Distributed Processing Symposium.

Shahrokh Yadegari, John Burnett, Grady Kestler, and Louis Pisha. "Spatial Audio and Sound Design in the Context of Games and Multimedia." Ed. Dr. Newton Lee. In *Encyclopedia of Computer Graphics and Games.* Springer Nature Switzerland AG, 2021.

Shahrokh Yadegari, John Burnett, Eito Murakami, Louis Pisha, Francesca Talenti, Juliette Regimbal, and Yongjae Yoo. "Becoming: An Interactive Musical Journey in VR." SIGGRAPH 2022 (August), Vancouver BC, Canada.

FIELDS OF STUDY

Major Field: Electrical Engineering (Signal and Image Processing)

Studies in Systems Design for Real-Time, Wearable Audio DSP (Hearing Aids)
Research Scientist Harinath Garudadri

Studies in Systems and Algorithms Design for Real-Time Acoustics & Spatialization
Professor Shahrokh Yadegari

Studies in Deep Neural Network Architectures for Real-Time Speech Processing
Research Scientist Harinath Garudadri and Professor Bhaskar D. Rao

Studies in Algorithms for Underwater Acoustics Simulation
Professor Jules Jaffe

ABSTRACT OF THE DISSERTATION

Systems and Algorithms for Real-Time Audio Signal Processing

by

Louis A. Pisha

Doctor of Philosophy in Electrical Engineering (Signal and Image Processing)

University of California San Diego, 2022

Professor Bhaskar D. Rao, Chair
Professor Shahrokh Yadegari, Co-Chair

Real-time systems are the canonical class of applications in signal processing. They drive the development of algorithms for approaching theoretical results within demanding practical constraints, and provide opportunities for devising clever ways to take advantage of hardware capabilities. State-of-the-art contributions are presented on three topics in this field.

The first contribution is the hardware, firmware, and software co-design of a wearable hearing aids research system. The system is open-source, easy to develop for, and much more powerful than traditional hearing aids. Its audio performance matches that of standard hearing aids and it can run custom DSP algorithms in usermode with only 2.4 ms of latency. The system

also includes local web-based user control and wearable electrophysiology.

The second contribution describes using GPU "Tensor Core" matrix multiply hardware to accelerate the computation of discrete Fourier transforms of sizes which are prime or have large prime factors. This includes mapping these sizes to the power-of-2-size Tensor Cores and emulating higher-precision arithmetic with lower-precision floating point numbers. For large batch sizes and for certain transform sizes which are odd or an odd number times 2 or 4, this approach produced state-of-the-art Fourier transform throughput.

Finally, two papers on algorithms design in real-time acoustic modeling for an audio spatialization system are presented. Two perceptually relevant types of diffraction are simulated with ray-based models of sound propagation. Existing methods have accuracy or performance limitations, especially in dynamic applications. A set of algorithms called Volumetric Diffraction and Transmission (VDaT) is introduced to approximate shadowed or near-shadowed diffraction by an occluding object. Similarly, Spatially Sampled Near-Reflective Diffraction (SSNRD) handles near-reflective diffraction involving the edges of reflecting objects. Both methods use ray tracing to spatially sample the scene, approximate ground truth results to within 1–3 dB, and have fast performance suitable for real-time applications. SSNRD also incorporates path generation algorithms, uses a small deep neural network (DNN) to compute the response of each acoustical path, and applies the GPU "RT core" real-time ray tracing hardware to spatial computing tasks beyond traditional ray tracing.

Introduction

Real-time applications present a unique set of challenges in systems and algorithms design. On the one hand, theoretically ideal methods may require far too many computational resources to be practically usable, requiring the development of clever approximate algorithms that produce adequate quality results within the performance constraints. On the other hand, algorithms can be designed to exploit the specific capabilities of the target computing hardware, accelerating their performance and potentially making an overall system design viable for the first time. This combination of constraints—designing algorithms to best fit both the application and the hardware—can make the "optimization problem" difficult, but also rewarding when the resulting system is able to solve a problem in a way that had never before been possible.

GPUs, especially NVIDIA GPUs with CUDA [1], are a fertile ground for this kind of work. The hardware is sufficiently different from a CPU to make programming it non-trivial, but also sufficiently more powerful for the effort to be worthwhile. New features, such as the Tensor Cores [2] for accelerated matrix multiply-adds and the RT cores [3] for real-time ray tracing, are added almost every generation, creating ample opportunities for new algorithmic approaches. Yet, these changes are nearly always made in a backwards-compatible way, and the same code will run on GPUs of any size or cost, so algorithms created using them are not at risk of being locked into one specific hardware model.

Audio can be a personally meaningful research field due to its connection to creative expression, in particular to music and interactive experiences (games, VR, etc.). However, it is also perhaps the canonical signal processing application, as many abstract concepts in the field can be made directly perceptible by applying them to audio and listening to the results.

Audio systems frequently must run in real time, often with higher update rates, lower latencies, and lower tolerance for failure to meet real-time constraints as compared to video or graphics. Furthermore, human hearing is essential to communication and quality of life, and hearing aids combine key aspects of audio signal processing and system design into a healthcare application that has real impacts on users' lives.

0.1 Contributions and Organization of Dissertation

This dissertation discusses systems and algorithms design contributions made across a range of topics in these areas of audio, accelerated computing, and real-time signal processing. Each of these contributions has advanced the state of the art in its respective area, and some of them have already borne fruit in influencing subsequent research.

Chapter 1 discusses the system design of a hearing aids research platform. Hearing aids are expensive [4], require time-consuming professional adjustment, and often perform poorly in noisy environments, among other complaints from hearing aid users [5]. The solutions to these issues—improved signal processing algorithms, self-fitting paradigms [6] [7], and low-cost off-the-shelf devices [8] [9]—are active research areas. However, this research work is impeded by hearing aid research systems often being proprietary, not sufficiently powerful, and not portable enough for easy use in the field. The presented system, called Open Speech Platform (OSP) [10], is designed to address all of these concerns. It consists of ear-level units in a traditional hearing aid form factor, wired to a small wearable box containing the battery and a powerful smartphone chipset. The audio hardware is high-quality but consumer-grade, and meets or exceeds the performance of off-the-shelf hearing aids on standard metrics. The system-on-chip runs Linux and can easily be programmed with new DSP algorithms in usermode, while maintaining extremely low latency of only a few milliseconds. The system also supports other sensors, including a one-wire wearable electrophysiology (EEG, ECG, etc.) system [11] which can run simultaneously with the hearing aid processing. The wearable box provides a

WiFi hotspot and serves researcher-created web apps which can interact with the hearing aid processing; the user can connect with their smartphone and do self-fitting, outcomes assessment, or other tasks while in the field. All of the hardware and software is open-source, and the platform has been in use by UC San Diego and its research partners to conduct audiological studies and investigate new hearing aid DSP algorithms. For more details on the OSP system design, see chapter 1.

Chapter 2 introduces algorithms for computing fast Fourier transforms (FFTs) on Tensor Core [2] matrix multiply-add hardware in NVIDIA GPUs. The discrete Fourier transform (DFT) is one of the most fundamental operations in digital signal processing, converting signals between the time and frequency domains. The Cooley-Tukey FFT algorithm [12] breaks up the computation of the DFT into steps based on the prime factors of the transform size, with powers of 2 being the most efficient. As a result, many applications use power-of-2 size signals in order to benefit from this algorithm, and power-of-2 FFT algorithms are often as fast as merely copying the data from and to memory. However, some applications [13] cannot arbitrarily choose the transform size, and may require transforms whose size is prime or has large prime factors. Tensor Cores accelerate matrix multiply operations, which the DFT is an example of, but their dimensions are all powers of 2 and they do not support 32-bit floating point computation. The presented algorithms [14] map DFTs of odd sizes to the Tensor Cores by taking advantage of the structure of the DFT matrix. Sizes which are an odd number times 2 or times 4 are also efficiently supported. In addition, algorithms are developed to approximate 32-bit floating point operations with the lower precision formats supported by the Tensor Core [15], with only a fraction of a bit of precision lost on average. For certain odd, odd times 2, or odd times 4 sizes, and for large batch sizes (large numbers of FFTs being computed at the same time), the presented approach surpasses the throughput of the previous state-of-the-art, becoming the fastest FFT ever published for these sizes. For more details, see chapter 2.

Chapters 3 and 4 describe algorithms for approximating wave effects when modeling sound propagation by rays in real-time applications. Many applications which involve a virtual

3D environment, from games and VR to architecture, require perceptually accurate simulation of acoustics in that environment. If the scene is fixed and only sound sources and receivers move, the acoustics can be accurately precomputed with wavefield methods [16–18], but if the scene may change dynamically, wavefield simulation requires far too much computation to do in real time [19]. As a result, methods are used which model the propagation of sound as rays and attempt to match wavefield results. Diffraction is inherently a wave phenomenon, and makes significant contributions to our experience of acoustics. It can be divided into two cases: when sound goes around or passes by an occluding object, and when sound reflects from a finite object whose size and shape affects the reflected sound. One popular approach to modeling these phenomena is edge diffraction methods, where ray paths are generated between the edges of meshes and the response of these paths is modeled. However, it is difficult to find the relevant set of edge diffraction paths, especially with detailed meshes which can move dynamically. Furthermore, when modeling the paths' response, a choice must be made between the Uniform Theory of Diffraction (UTD) model [20], which is fast but often inaccurate, or the Biot-Tolstoy-Medwin (BTM) model [21–23], which is accurate but requires a large amount of computation. The proposed methods, Volumetric Diffraction and Transmission (VDaT) [24] and Spatially Sampled Near-Reflective Diffraction (SSNRD), are a new type of ray-based approach to modeling the two cases of diffraction discussed above, with VDaT handling the occlusion case and SSNRD handling the reflections case. Both methods use spatial sampling—tracing rays into the scene in a pattern of concentric cylinders, around the path segment or reflection point respectively—to extract information about the local scene geometry. Then, both methods process these results through algorithms to produce estimates of the magnitude response of the diffracting or reflecting path. These algorithms are tuned to match the high-quality BTM edge-diffraction results within 1–3 dB, but they require much less computation than evaluating BTM directly. VDaT was developed first and supports sound transmission through obstacles as well as diffraction around them. It uses a combination of heuristics and a numerically optimized model to approximate BTM. SSNRD includes a set of algorithms for generating reflection paths

as needed for the modeling of diffraction in conjunction with reflections, and uses a small deep neural network (DNN) [25] to approximate the BTM results. It uses the RT core (real-time ray tracing hardware in NVIDIA RTX GPUs) [3] for multiple ray tracing steps as well as other spatial computing tasks beyond traditional ray tracing [26–30], and as a result it is able to handle scenes with millions of triangles. Together, SSNRD and VDaT can generate and simulate diffraction on thousands of acoustical paths at interactive rates in fully dynamic virtual scenes. For more details on VDaT and SSNRD, see chapter 3 and chapter 4 respectively.

Bibliography

[1] NVIDIA Corporation. "CUDA C Programming Guide". In: (2019). (Last viewed Jan. 20, 2019). URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[2] *NVIDIA A100 tensor core GPU architecture*. Tech. rep. NVIDIA Corporation, 2020. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf.

[3] NVIDIA Corporation. *NVIDIA Turing GPU architecture*. 2018. URL: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[4] Annie N Simpson, Lois J Matthews, Christy Cassarly, and Judy R Dubno. "Time from hearing aid candidacy to hearing aid adoption: a Longitudinal Cohort Study". In: *Ear and hearing* 40.3 (2019), pp. 468–476.

[5] Rebecca J Bennett, Ariane Laplante-Lévesque, Carly J Meyer, and Robert H Eikelboom. "Exploring hearing aid problems: perspectives of hearing aid owners and clinicians". In: *Ear and hearing* 39.1 (2018), pp. 172–187.

[6] Arthur Boothroyd and Carol Mackersie. "A "Goldilocks" approach to hearing-aid self-fitting: user interactions". In: *American journal of audiology* 26.3S (2017), pp. 430–435.

[7] Carol Mackersie, Arthur Boothroyd, and Alexandra Lithgow. "A "Goldilocks" approach to hearing aid self-fitting: ear-canal output and speech intelligibility index". In: *Ear and hearing* 40.1 (2019), pp. 107–115.

[8] SERGEI Kochkin. "MarkeTrak VIII: utilization of PSAPs and direct-mail hearing aids by people with hearing impairment". In: *Hearing Review* 17.6 (2010), pp. 12–16.

[9] Lisa Brody, Yu-Hsiang Wu, and Elizabeth Stangl. "A Comparison of Personal Sound Amplification Products and Hearing Aids in Ecologically Relevant Test Environments". In: *American journal of audiology* 27.4 (2018), pp. 581–593.

[10] Louis Pisha, Julian Warchall, Tamara Zubatiy, Sean Hamilton, Ching-Hua Lee, Ganz Chockalingam, Patrick P Mercier, Rajesh Gupta, Bhaskar D Rao, and Harinath Garudadri. "A wearable, extensible, open-source platform for hearing healthcare research". In: *IEEE Access* 7 (2019), pp. 162083–162101.

[11] Julian Warchall, Paul Theilmann, Yuxuan Ouyang, Harinath Garudadri, and Patrick P. Mercier. "A Rugged Wearable Modular ExG Platform Employing a Distributed Scalable Multi-Channel FM-ADC Achieving 101dB Input Dynamic Range and Motion-Artifact Resilience". In: *2019 IEEE International Solid- State Circuits Conference - (ISSCC)* (2019), pp. 362–363.

[12] James W Cooley and John W Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[13] Tai-Sung Lee, David S. Cerutti, Dan Mermelstein, Charles Lin, Scott LeGrand, Timothy J. Giese, Adrian Roitberg, David A. Case, Ross C. Walker, and Darrin M. York. "GPU-Accelerated Molecular Dynamics and Free Energy Methods in Amber18: Performance Enhancements and New Features". In: *Journal of Chemical Information and Modeling* 58.10 (2018). PMID: 30199633, pp. 2043–2050. DOI: 10.1021/acs.jcim.8b00462. eprint: https://doi.org/10.1021/acs.jcim.8b00462. URL: https://doi.org/10.1021/acs.jcim.8b00462.

[14] Louis Pisha and Łukasz Ligowski. "Accelerating non-power-of-2 size Fourier transforms with GPU tensor cores". In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2021, pp. 507–516.

[15] Anumeena Sorna, Xiaohe Cheng, Eduardo D'azevedo, Kwai Won, and Stanimire Tomov. "Optimizing the fast fourier transform using mixed precision on tensor core hardware". In: *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*. IEEE. 2018, pp. 3–7.

[16] Nikunj Raghuvanshi and John Snyder. "Parametric directional coding for precomputed sound propagation". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–14.

[17] Chakravarty R Alla Chaitanya, Nikunj Raghuvanshi, Keith W Godin, Zechen Zhang, Derek Nowrouzezahrai, and John M Snyder. "Directional sources and listeners in interactive sound propagation using reciprocal wave field coding". In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 44–1.

[18] Microsoft Game Dev. *Project Acoustics overview – What is Project Acoustics?* May 2022. URL: https://docs.microsoft.com/en-us/gaming/acoustics/what-is-acoustics.

[19] Jukka Saarelma, Jonathan Califa, and Ravish Mehra. "Challenges of Distributed Real-Time Finite-Difference Time-Domain Room Acoustic Simulation for Auralization". In: *Audio Engineering Society Conference: 2018 AES International Conference on Spatial Reproduction - Aesthetics and Science*. July 2018. URL: http://www.aes.org/e-lib/browse.cfm?elib=19609.

[20] Robert G Kouyoumjian and Prabhakar H Pathak. "A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface". In: *Proceedings of the IEEE* 62.11 (1974), pp. 1448–1461.

[21] Maurice Anthony Biot and Ivan Tolstoy. "Formulation of wave propagation in infinite media by normal coordinates with an application to diffraction". In: *J. Acoust. Soc. Am.* 29.3 (1957), pp. 381–391.

[22] U Peter Svensson, Roger I Fred, and John Vanderkooy. "An analytic secondary source model of edge diffraction impulse responses". In: *J. Acoust. Soc. Am.* 106.5 (1999), pp. 2331–2344.

[23] Paul T Calamia and U Peter Svensson. "Fast time-domain edge-diffraction calculations for interactive acoustic simulations". In: *EURASIP J. Applied Signal Proc.* 2007.1 (2007), pp. 186–186.

[24] Louis Pisha, Siddharth Atre, John Burnett, and Shahrokh Yadegari. "Approximate diffraction modeling for real-time sound propagation simulation". In: *J. Acoust. Soc. Am.* 148.4 (2020), pp. 1922–1933.

[25] Stefan Schubert, Peer Neubert, Johannes Pöschmann, and Peter Protzel. "Circular convolutional neural networks for panoramic images and laser data". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 653–660.

[26] Nathan Morrical, Ingo Wald, Will Usher, and Valerio Pascucci. "Accelerating unstructured mesh point location with RT cores". In: *IEEE Transactions on Visualization and Computer Graphics* (2020).

[27] Nate Morrical, Will Usher, Ingo Wald, and Valerio Pascucci. "Efficient space skipping and adaptive sampling of unstructured volumes using hardware accelerated ray tracing". In: *2019 IEEE Visualization Conference (VIS)*. IEEE. 2019, pp. 256–260.

[28] Nate Morrical and Stefan Zellmann. "Inverse Transform Sampling Using Ray Tracing Hardware". In: *Ray Tracing Gems II*. Springer, 2021, pp. 625–641.

[29] Stefan Zellmann, Martin Weier, and Ingo Wald. "Accelerating force-directed graph drawing with RT cores". In: *2020 IEEE Visualization Conference (VIS)*. IEEE. 2020, pp. 96–100.

[30] I Evangelou, G Papaioannou, K Vardis, and AA Vasilakis. "Fast Radius Search Exploiting Ray-Tracing Frameworks". In: *Journal of Computer Graphics Techniques Vol* 10.1 (2021).

Chapter 1

A Wearable, Extensible, Open-Source Platform for Hearing Healthcare Research

Reprinted from IEEE Access, November 2019, DOI 10.1109/ACCESS.2019.2951145

1.0 Abstract

Hearing loss is one of the most common conditions affecting older adults worldwide. Frequent complaints from the users of modern hearing aids include poor speech intelligibility in noisy environments and high cost, among other issues. However, the signal processing and audiological research needed to address these problems has long been hampered by proprietary development systems, underpowered embedded processors, and the difficulty of performing tests in real-world acoustical environments. To facilitate existing research in hearing healthcare and enable new investigations beyond what is currently possible, we have developed a modern, open-source hearing research platform, Open Speech Platform (OSP). This paper presents the system design of the complete OSP wearable platform, from hardware through firmware and software to user applications. The platform provides a complete suite of basic and advanced hearing aid features which can be adapted by researchers. It serves web apps directly from a hotspot on the wearable hardware, enabling users and researchers to control the system in real time. In addition, it can simultaneously acquire high-quality electroencephalography (EEG) or other electrophysiological signals closely synchronized to the audio. All of these features are

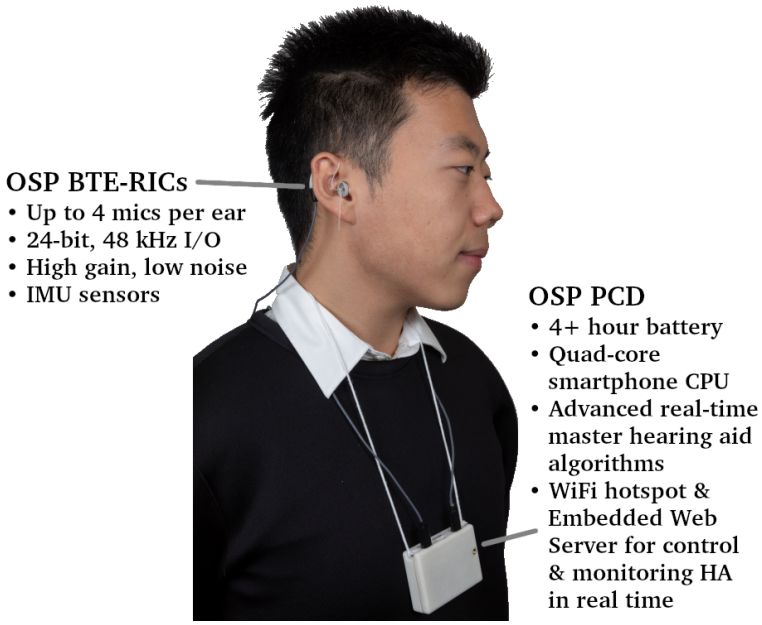

Figure 1.1. A user wearing the OSP wearable platform. The two hardware components shown are the behind-the-ear receiver-in-canal (BTE-RIC) transducers and the Processing and Communication Device (PCD).

provided in a wearable form factor with enough battery life for hours of operation in the field.

1.1 Introduction

Hearing is essential for communication, navigation, and quality of life. The healthy ear is able to operate in a wide variety of environments over a huge dynamic range due to its highly complex nonlinear, time-varying, and attention-controlled characteristics. As a result, when hearing impairments occur, they can rarely be corrected by simply amplifying the input sound. Hearing aids (HAs) have been under development from this starting point for the last forty years, and now incorporate multi-band processing, dynamic range compression, feedback and noise management, and other advanced features.

Unfortunately, there is substantial dissatisfaction with many aspects of HAs among the user community [1]. Key factors underlying this dissatisfaction include the following:

1. *Clinical challenges:* One example is that the current best practices in HL diagnosis

and intervention rely mostly on pure tone audiometry (PTA) [2], which characterize only the spectral aspects of HL, in clean conditions; the temporal dynamics in human perception of speech and music in clean and noisy environments are largely ignored. A different type of challenge is the typical need for users to see an audiologist to have fitting parameters adjusted; as an alternative, many researchers are investigating self-fitting procedures, environment-dependent profiles, and other ways to give the user control over their experience.

2. *Technical constraints:* HAs must provide sufficient battery power for processing and communication, in an acceptably small form factor, while introducing no more than 10 ms of latency [3] [4]. The overall latency requirement presents a significant challenge for noise mitigation algorithms and other advanced functions such as frequency lowering. Furthermore, binaural processing in HAs to take advantage of spatial information in noisy environments is a major challenge, because of the power requirements for wireless communication of full-band audio signals between the HAs and additional processing for adaptive beamforming.

3. *Research accessibility:* There are five major HA manufacturers: Phonak, Oticon, ReSound, Starkey, and WS Audiology. All of these manufacturers provide audiologists with tools for HA fitting, which can be used for certain kinds of clinical research. The manufacturers also sometimes provide their internal platforms for academic research in specific topics, such as directional microphones, noise management, programs for multiple listening environments, etc. However, each of these platforms is proprietary and unique, meaning that it is difficult to generalize research across the platforms, and infeasible to modify or experiment with the algorithms in ways not intended by the manufacturers.

4. *Cost:* There is an average 8.9-year delay between HA candidacy and HA adoption, with the biggest predictor of adoption delay being socioeconomic status [5]. This implies that the cost of HAs—which is often several thousand dollars—is a significant obstacle to

many users. This high cost is partly due to the technology, but also largely due to the closed ecosystem of medical-grade hearing instruments. In response, a new market in off-the-shelf hearing assisted devices has emerged [6] [7].

The National Institutes of Health (NIH) conducted a workshop in 2014 on open-source HA research platforms and published recommendations about their capabilities and features [8]. Our system, *Open Speech Platform (OSP)* [9], is designed to meet these recommendations, including the vision of "new types of basic psychophysical research studies beyond what is widely done today". OSP is a suite of comprehensive, open-source hardware and software tools for multidisciplinary research in hearing healthcare. The goals of OSP are to address the underlying causes behind the challenges described above, to facilitate existing research by audiologists and DSP engineers, and to enable new kinds of investigations between hearing and related disciplines.

The OSP hardware is comprised of:

1. a Processing and Communication Device ("PCD"), which is a small wearable box containing a smartphone chipset performing all the signal processing and wireless communication functions, plus the battery and supporting hardware

2. "hearing aid"-style audio transducer devices in behind-the-ear receiver-in-canal ("BTE-RIC") form factor, which connect to the PCD via a 4-wire cable. They support 4 microphones and one receiver (loudspeaker) per ear, plus an accelerometer/gyroscope (IMU) for measuring look direction and researching mobility disorders

3. an optional set of active biopotential electrodes for acquiring EEG or other electrophysiological signals, daisy-chained together and connected to acquisition hardware on the PCD via another 4-wire cable (together called "FM-ExG")

The OSP software components include:

1. Firmware for FPGAs in the PCD and BTE-RICs

2. An embedded Linux distribution running on the CPU within the PCD, including kernel modifications and custom drivers for the BTE-RICs

3. The OSP real-time master hearing aid (RT-MHA), which is a library of signal processing modules and a reference C++ program that performs basic and advanced HA signal processing in real time

4. The Embedded Web Server (EWS), which:

   (a) hosts a WiFi hotspot on the PCD

   (b) serves web apps to any browser-enabled device which connects to it, such as the user's smartphone

   (c) controls the RT-MHA parameters live based on user actions in the web apps

Taken together, OSP is a powerful research tool, in which all aspects of the assisted hearing experience—from the ear-level hardware to the signal processing algorithms to the way the user interacts with and controls their device—may be customized and used for research in the lab and in the field. The target audience of OSP is not just audiologists and speech DSP engineers, but also researchers in neuroscience, healthy aging, human-computer interaction, networking and edge/cloud processing, wearable electronics, and many other disciplines. Because OSP is open-source—all the software and hardware design files are released on our website [9]—researchers may modify and enhance whatever part of the system is relevant to their work, while leveraging past contributions made by other researchers.

Our development of OSP has resulted in novel developments in embedded systems design [10], portable electrophysiology [11] [12], adaptive filtering [13] [14], and other areas not yet published. Yet, the primary novelty of OSP—and its primary value to the community—is in its system design as a whole, and the capabilities it offers to researchers and users as a result of this design. As such, this paper describes the engineering design of all portions of the OSP platform, with an emphasis on how the design choices provide useful and advanced functionality.

In particular, we focus on aspects of the hardware that have not been reported on in previous publications, and we provide updates on continued development of other parts of OSP. Sec. 1.2 discusses the PCD, the software from FPGA through kernel level, the BTE-RICs, and other included sensors. Sec. 1.3 covers the FM-ExG. Sec. 1.4 reviews the RT-MHA and discusses new academic research on adaptive filters which has already been enabled by OSP. Sec. 1.5 describes the software architecture of the embedded web server (EWS) and the current set of provided web apps for audiologist and user engagement. Finally, Sec. 1.6 gives objective performance results for the hardware and software, showing its capacity for real-time, low-latency audio processing, the quality of the recorded electrophysiological signals, and the platform's usability for multidisciplinary clinical research.

1.1.1 Related Work

OSP intersects most aspects of the vast field of research on hearing healthcare. Thus, we will restrict our discussion in this section to systems for hearing research that perform real-time audio processing and have a portable or wearable component, as this is what OSP is at its core. The five major HA manufacturers each have their own proprietary systems of this kind, which they use for research on new clinical and technical challenges as they develop their advanced digital HAs. However, these systems are difficult to access for the research community at large, and difficult to modify and to obtain generalizable results from as discussed above.

As of 2014, no non-proprietary HA research system existed which met the needs of the HA research community, according to the aforementioned NIH workshop on this topic: "The NIDCD-supported research community has a critical need for an open, extensible, and portable device that supports acoustic signal processing in real time" [8]. As a result, in 2016 the NIH awarded six grants for development of open-source hearing aid research tools [15] [16]. Of these six, four—including OSP—are complete master hearing aid tools for research. The other three of these tools are:

Tympan

[17] includes a wearable processing unit based on Arduino Teensy [18] and a basic software library for HA processing. The strengths of this platform include flexibility with the transducers (the unit simply features standard 1/8" jacks) and battery (the user selects their own portable battery pack), low cost and use of readily available components, small size, easy development for beginners with the Arduino platform, and fast time-to-market. Its disadvantages include low audio quality, severely limited processing power, and support for only one input channel (microphone).

Open-MHA

[19] features an audio expansion board for BeagleBone Black, a Linux-based OS, and an extensive real-time and offline HA software suite. The advantages of this platform include good-quality audio, support for six-channel input, the well-documented nature of both BeagleBone Black and Linux, and the powerful master hearing aid DSP algorithms. Its downsides include somewhat limited processing power, the fact that its form factor is portable but not wearable, and the lack of ear-level transducers for users in the field. However, the open-source nature of these platforms allows the strengths of each to be combined: for instance, the Open-MHA DSP algorithms could in the future be ported to OSP hardware.

UT Dallas project

[20] is comprised of a cross-platform smartphone app for processing and commercial Bluetooth-enabled hearing aid transceivers. The advantages of this platform include its advanced speech enhancement algorithms, the complete absence of special-purpose hardware, the accessibility of smartphone development, and the use of industry-standard ear-level transducers (which are proven designs and ultimately the target hardware). Its weaknesses include its inability to process audio in real time (defined as a total microphone-to-loudspeaker delay of less than 10 ms while HA processing is occurring), the proprietary nature of the ear-level transducers, and the semi- or fully-closed smartphone operating systems and driver stack which make it difficult to

guarantee performance.

1.2 Wearable Hardware

1.2.1 Form Factor

As reported in [21], the software portions of OSP were first implemented on a laptop, with a studio audio interface and custom analog hardware for interfacing and the ear-level transducers. The OSP RT-MHA can still run on any Mac or Linux computer using any audio hardware supported by the respective OS. However, the potential of OSP is much more fully realized in its new wearable form factor which we initially discussed in [10].

As discussed in the Introduction, the battery size, available processing power, and communication abilities in commercial HAs are severely limited by the behind-the-ear or in-ear form factor they typically are available in. These factors in turn contribute to the cost and the difficulty of development (e.g. fixed-point embedded processors). For a research platform, we need much higher processing power, substantially improved wireless communication, relatively low cost, and easy development. These factors are much more important than the entire system fitting behind the ear, so we compromised on the form factor: we created a design which is still easily wearable but which is not limited to the space around the ear (Fig. 1.1). The processing, wireless communication, and battery for the OSP wearable system are housed in the Processing and Communication Device (PCD), which is a small box that may be worn around the neck or on a belt. The PCD is attached by wires to the BTE-RICs, which contain the audio transducers, codecs and interface hardware, and other sensors. Since the PCD processes the audio from both ears, it can use beamforming and other algorithms to take advantage of binaural information in the audio, something BTE or in-ear HAs would have to use wireless transmission to achieve. The aforementioned NIH workshop suggested that the form factor of BTE-RICs wired to a processing unit would be appropriate for a research system [8].

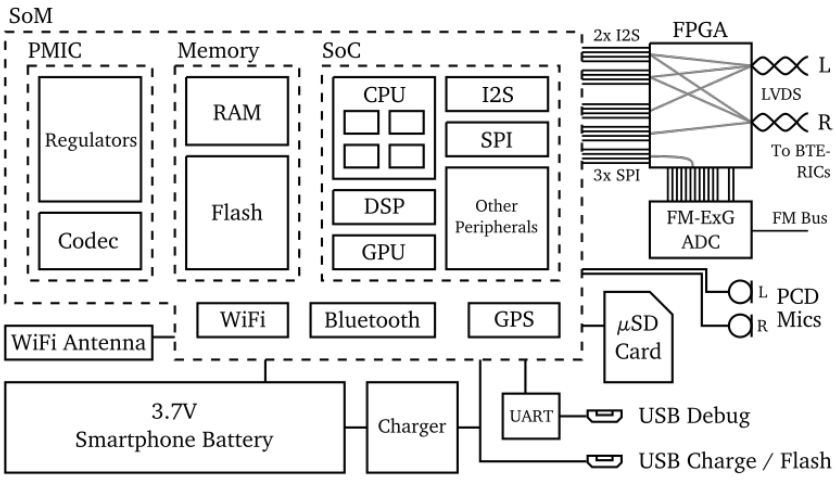

Figure 1.2. Block diagram of the OSP PCD (Processing and Communication Device).

1.2.2 Choice of Embedded Platform

Smartphone chipsets provide best-in-class computational performance per watt, diverse peripherals, and advanced wireless connectivity, so they are a natural choice for the embedded platform in the OSP wearable design. However, many smartphone chipsets are difficult to work with, due to the high degree of proprietary technology in modern smartphones. Furthermore, embedded systems development for hard-real-time, low-latency applications is typically done at a very low level. Low-level audio processing would be contrary to the goals of extensibility and controllability of OSP, but low latency and stability are still mandatory. Thus, the design task was primarily to (1) select a platform which is capable of high-level real-time processing and has all the necessary features, and then (2) adapt its hardware and software to the needs of OSP.

We selected the single-board computer system DragonBoard 410c from Arrow, based on the Snapdragon 410c chipset from Qualcomm. Because of the hobbyist-oriented nature of this product—it is intended to compete with platforms like Raspberry Pi and BeagleBone Black—a large support network for this chipset exists, including a well-maintained Debian branch. Moreover, several companies supply systems-on-module (SoMs) featuring the same chipset, which allow developers to move to an application-specific design without having to design a PCB hosting a complex modern system-on-chip (SoC), while maintaining software

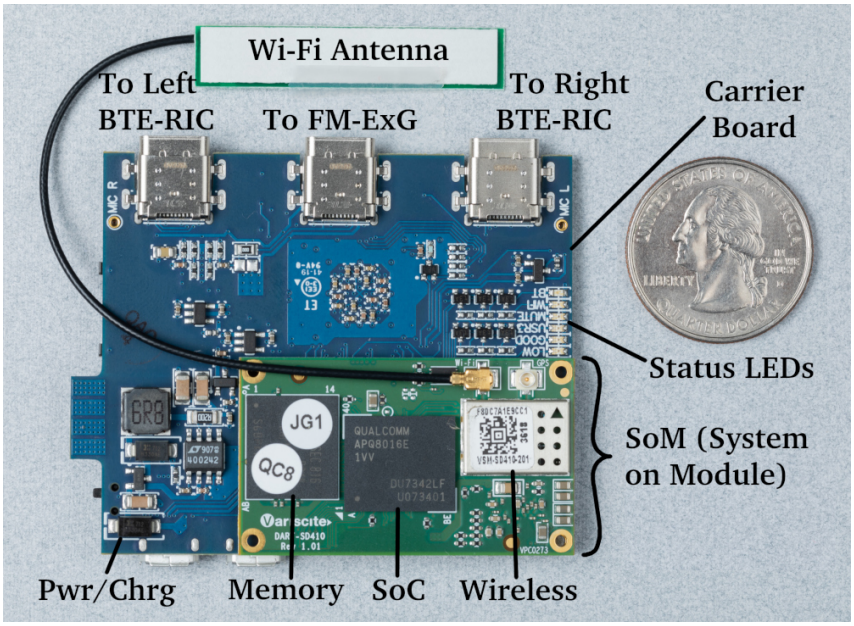

Figure 1.3. Components of the OSP PCD (Processing and Communication Device): the carrier board hosting the Snapdragon 410c SoM.

compatibility and most hardware compatibility with the DragonBoard. We chose the DART-SD410 from Variscite [22] as our SoM because it breaks out all the multichannel inter-IC sound (MI2S) peripheral lines from the SoC, unlike the DragonBoard and most other SoMs.

The Snapdragon 410c SoC (APQ8016) has four 64-bit ARM A53 cores at 1.2 GHz, plus DSP and GPU. Not only does a multicore CPU provide more processing power than a single-core CPU, it allows us to assign real-time portions of the HA processing to dedicated cores where they will not be interrupted, while the OS and EWS run on a different core (see Sec. 1.2.4). Key SoC peripherals include two multichannel inter-IC sound (MI2S) ports for audio I/O to the behind-the-ear receiver-in-canal (BTE-RIC) transducers; several SPI ports for peripheral control and communication; a microSD card for data logging; and a UART for the Linux terminal interface. Crucially, the MI2S ports are directly connected to the CPU, unlike in many smartphone chipsets where they are connected to the DSP. The latter would require at least some processing to be done on the DSP, which would substantially complicate the development process compared to running ordinary usermode code on the CPU, or add the

Figure 1.4. The OSP PCD disassembled, showing the battery, the back of the carrier board, and the plastic shell.

additional latency of transfers in each direction. The associated power management IC, PM8916, includes a separate lower-performance codec which is used to provide two microphones on the PCD. The SoC and associated wireless chips provide 2.4 GHz WiFi, Bluetooth, and GPS. Paired with the industry-standard networking software available for Linux, the WiFi can act as an access point and the system can serve web pages to clients which connect to it (Sec. 1.5).

We designed a carrier board to host the SoM (Fig. 1.3). This board also includes power supplies, the FPGA (Sec. 1.2.6), the other interface hardware and ports for the BTE-RICs and the FM-ExG, the microSD card slot and USB ports, and other basic system features. Adjacent to the carrier board is a 2000 mAh smartphone-type Li-Ion battery, which can be charged from a microUSB port or swapped out by the user. The carrier board, battery, and WiFi antenna are enclosed in a plastic case (Fig. 1.4) to form the PCD, which may be worn around the neck or on a belt. The PCD is roughly $73 \times 55 \times 20$ mm and has a mass of roughly 83 grams, representing a savings of 67% in weight and 72% in volume over the previous "portable" OSP hardware design [23].

1.2.3 Adapting Smartphone SoC Audio Hardware

As discussed above, the 410c platform was chosen for its power efficiency, high performance, wireless capabilities, and product ecosystem. However, the audio subsystem in the Snapdragon 410c was designed to support the needs of low cost smartphones; it was neither designed nor documented for general-purpose use. Our needs for audio I/O to the BTE-RICs in the HA application are substantially different from those of the smartphone applications the SoC's audio subsystem was designed for. Nevertheless, we were able to adapt this subsystem to the needs of OSP through a combination of reverse engineering and analysis of its partial documentation. Although some of the implementation details discussed here are specific to the 410c SoC, many of them would apply to a variety of single board computers and SoMs based on ARM processors running Linux. The OSP software comprising the RT-MHA and EWS is hardware-agnostic, and can run on Linux and OS X systems in addition to the embedded systems mentioned above.

Specifically, each BTE-RIC has one MI2S data line for microphone data and one for speaker data. The same speaker data line can be sent to both BTE-RICs, with the left and right receiver signals in the left and right time-division slots respectively. However, each of the two microphone data lines must be received by the SoC on separate MI2S data input pins, since they each already contain two mics' data. This means a total of two MI2S data input lines and one MI2S data output line are needed. Due to the design of the MI2S peripheral units in the SoC and the undocumented multiplexer block which connects them to the SoC's I/O pins, the only configuration which provides two MI2S data input lines is using two data lines of one MI2S unit in "receive" mode, and using a different unit in "transmit" mode for the data output line. Unfortunately, the Advanced Linux Sound Architecture (ALSA) kernel subsystem assumes that each codec has a unique data (I2S) port; in our case, two MI2S ports are being shared by two codecs. So, we had to build a custom ALSA driver for the BTE-RICs which registers two "virtual" audio devices—one for mics, and one for speakers—connected to the respective

MI2S peripherals. Each virtual device has its own "memory map" with registers controlling the appropriate functions; writes to and reads from these registers are rerouted in the driver to both codecs' SPI control ports as necessary. The result is that usermode software sees two devices, one with only audio inputs and one with only outputs, both of which function on their own or simultaneously.

1.2.4 Embedded Operating System

The embedded operating system used in OSP is based on the Debian 9 ("stretch") distribution of Linux for Snapdragon 410c (ARM64) provided by Linaro. Besides the custom audio driver mentioned above, we have tailor-built the kernel and configured the environment to meet the following goals:

1. Stable real-time performance
2. Low power consumption
3. Fast bootup
4. Small memory footprint
5. Security

These goals will be referenced by number in the following paragraphs.

Kernel

The kernel is configured with all core facilities and most drivers as built-in. Building as much code into the kernel binary rather than modules improves the bootup time (3). In the current configuration, there are a few drivers that remain modular due to the fact the driver cannot initialize until after the firmware is initialized and the device is powered up. Future work by our team and the community will be needed to modify these drivers to enable them to compile as built-in, which will allow module loading/unloading to be completely disabled. No dynamic

loading of kernel-mode code is a desire for security (5) since the kernel cannot be modified as easily at run-time which will help thwart specific threats, e.g., rootkits.

Any kernel facility that is not used by the system or driver that does not have the hardware present is not included. This optimization helps to achieve both (3) and (4) along with the additional benefit of decreasing build time of the kernel. Short build time is not a design goal but is a desirable metric that is crucial in reducing development and test time. Furthermore, we hope to show that by removing additional kernel features, the security posture of the system improves (5) by removing any attack vectors associated with those features.

To address (1), the `PREEMPT_RT` option has been enabled, which enables the kernel to become preemptable and shortens the critical sections within kernel code.

Environment

`systemd` has replaced the old `sysvinit` style `init` process that becomes `PID 1` when the kernel finishes its boot process, and handles the remaining portions of system boot. `systemd` is configured to only run on CPU core 0 through a configuration setting in `/etc/systemd/system.conf`. As a result of this configuration the `init` process and subsequent processes spawned by it will only run on core 0. This allows CPU cores 1-3 to be reserved for all real-time processing (1); user code handles their assignment to these cores when they are executed. Similarly, all interrupt handling is bound to core 0 to avoid interrupting the real-time processes running on cores 1-3.

Unnecessary and unused services are disabled to reduce power consumption (2) and enhance system security (5). The Bluetooth radio is also disabled by default for the same two reasons but can be enabled by a user if so desired. As seen in Table 1.3 below in Sec. 1.6, the idle power consumption is more than half of the total power consumption during full operation, so it is extremely important to eliminate unnecessary power sinks to improve battery life.

The system configures the WiFi interface as a hotspot after boot to allow for remote connectivity to the PCD, for the embedded web server (EWS) and for SSH for development. In conjunction with the hotspot, multicast DNS-Service Discovery (mDNS-SD, a.k.a Bonjour) is

enabled and configured to allow a user connected to the hotspot to easily access the EWS or SSH into the board using the hostname `ospboard.local`, without needing to know the IP address of the board. As a fallback for systems that do not support mDNS, e.g. Android, the IP address of the board is always the same when connected through the hotspot.

1.2.5 High-Performance BTE-RICs

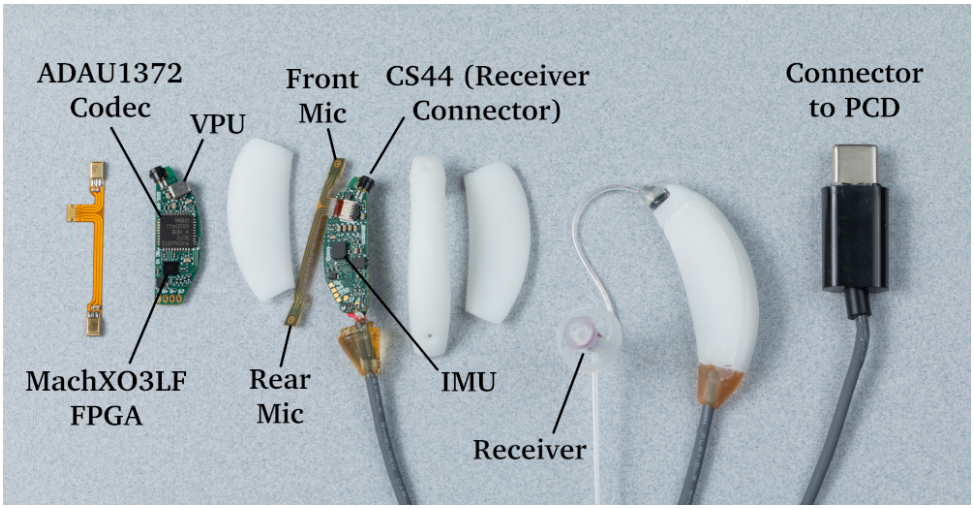

Figure 1.5. The OSP BTE-RICs, together and disassembled.

Along with the PCD, the other key hardware component of OSP is novel ear-level transducers in a behind-the-ear receiver-in-canal[1] (BTE-RIC) form factor (Fig. 1.5). These units are each connected to the PCD via a four-wire cable, and serve as the primary input and output for the system. They are composed of a rigid PCB for the electronics, a flex PCB for the microphones, a custom 3D-printed plastic shell, and a rugged 3D-printed strain relief [9].

Unlike in previous versions, the communication between the BTE-RICs and the PCD is digital—the codecs are within the BTE-RICs. The low-level digital interface is transparently facilitated by FPGAs in both the BTE-RICs and the PCD (Sec. 1.2.6). The decision to have digital communication with the BTE-RICs was made for several reasons. First, analog communication with the BTE-RICs would require at least six wires—a differential pair each for the microphone

[1]The output transducer, i.e. the loudspeaker, is called the "receiver" in the telephony and HA communities. This is typically a small speaker in a long, slender package that is in or just outside the ear canal.

and receiver, plus power and ground—plus even more wires for multiple microphones per ear. As discussed below, multiple audio inputs per ear is crucial for expansion of the hearing-related research OSP supports. Second, having the codec physically close to the transducers reduces the opportunity for noise and interference. Finally, the digital interface allows for additional sensors at the ear—starting with the IMU (Sec. 1.2.7)—without the need for any additional wires, thanks to the FPGAs.

The codec in each BTE-RIC is the high-performance but consumer-grade Analog Devices ADAU1372 [24], which provides a differential headphone driver for the receiver and four analog inputs per ear. By default, these are a front microphone, a rear microphone, an in-ear microphone, and a voice pick-up (VPU) transducer (Fig. 1.6); while the former two are common on hearing aids, the latter two are for specialized purposes, and are explained below. The I2S standard only supports two channels of audio per data line, so currently only two of these four inputs may be transmitted to the PCD at a time. However, the application may select via ALSA commands which two inputs these are, and future work will enable simultaneous capture of all four microphones (Sec. 1.2.6). All inputs and outputs are sampled at 48 kHz 24 bit; the codec also supports 96 kHz sampling, which will be supported by a future version of OSP for improved accuracy in beamforming.

Several types of audiological studies require measurement of the sound within the ear canal while a hearing assisted device is being worn. Purposes include calibration of the acoustics, Real Ear Insertion Gain (REIG) measurements during HA fitting [25], compensation for occlusion effects [26], and studying otoacoustic emissions [27] [28]. Typically, this measurement is performed with a probe placed into the ear canal as the HA is inserted; unfortunately, this method is time-consuming and precise positioning of the probe can be difficult [25]. To facilitate such studies, the BTE-RICs support a special receiver in development at Sonion [29] which has a microphone in the same package, facing into the ear canal. This allows the sound within the ear canal to be measured and monitored as a normal part of work with the platform—including in the field, which would normally be prohibitively difficult. The current design uses a CS44 connector

for the receivers, with a pinout that is compatible with a variety of regular receivers as well as with the embedded-mic receiver, thus not increasing costs for users who do not need this feature.

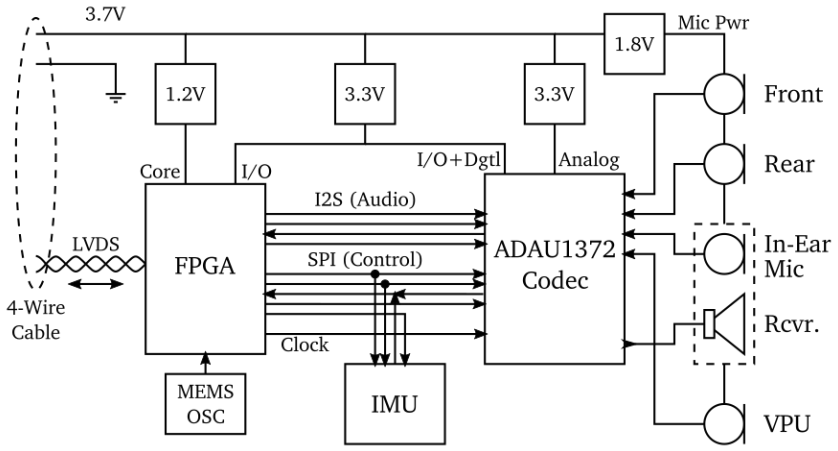

Figure 1.6. Block diagram of the OSP BTE-RICs.

A VPU (voice pick-up) is a bone conduction transducer that picks up the user's voice, while being highly immune to background noise (40-50 dB loss to ambient sound compared to conducted sound [30]). When mounted to a device which is in robust contact with the head, such as an in-ear hearing assisted device, it picks up the vibrations of the skull—that is, the user's voice—without any outside sound. While bone conduction microphones have made impressive advances, they still have reduced frequency range compared to air microphones, and their response to vibration is noticeably nonlinear. Thus, the VPU in this system effectively provides a measurement of the user's voice which is somewhat distorted but almost completely free of interference. This signal can be useful in several ways. First, adaptive systems such as beamforming and speech enhancement rely on accurate estimates of when the user is speaking (speech presence probability or SPP) in order to estimate the interfering noise. The VPU signal can provide an improved estimate of the SPP, so that the adaptation can be temporarily disabled while the user is speaking [31]. Second, other algorithms can be developed to improve the experience of listening to one's own voice, which is known to be adversely affected by HAs [26]. These may include reducing the gain while the user is speaking, DSP approaches to correct for

the presence of the HA in the canal, etc. Finally, algorithms—especially ones involving deep neural networks—can be developed to reconstruct an improved signal of the user's speech from the VPU signal [32], for purposes like telephony or virtual meeting settings.

In addition to the codec and audio hardware, each BTE-RIC also provides an inertial measurement unit (IMU), which is discussed in Sec. 1.2.7; separate analog and digital power supplies for additional noise suppression; and an FPGA, which is discussed next.

1.2.6 Custom Digital Interface

Both the PCD and each BTE-RIC contain an FPGA (Lattice MachXO3 series [33]). As discussed below, the form factor of the BTE-RICs containing the codecs with processing in the PCD would not be feasible without the FPGAs. Once they were present, they enabled additional features, including the FM-ExG (Sec. 1.3), so they have become a key component of the platform.

The original need for the FPGA came from the observation that the communication between the BTE-RICs and the PCD would require a large number of signal wires: bit clock, word clock, microphone data, and receiver data for I2S, and at least two lines for control signals to and from the codec and IMU (clock and data of I2C). Combined with power and ground wires, the cable to the BTE-RICs would have to have eight conductors. On top of this, neither I2S nor I2C are designed for transmission over wires of any significant length; while they would be likely to work in controlled conditions in the lab, they might not be robust in varying electromagnetic environments in the field. So, we decided to add an FPGA at each end, and transmit all the signals with a custom protocol over a single bidirectional twisted pair, reducing the number of conductors in the cable to four. The physical layer chosen is bus low-voltage differential signaling (BLVDS) [34] [35], a bidirectional version of the popular LVDS standard [36] used in many modern serial interfaces such as USB, SATA, and PCI Express. This interface uses standard CMOS drivers to transmit and analog differential amplifiers to receive; the FPGAs support this interface natively, only needing a few external resistors at each end to match the

impedance of the cable. Because the signal transmitted is differential, it is nearly immune to common-mode noise and interference; and since the cable is shielded and the conductors are twisted, there is very little opportunity for differential interference. As a result, this interface is perfect for high-speed communication over the roughly 1 m cable between the BTE-RIC and the PCD.

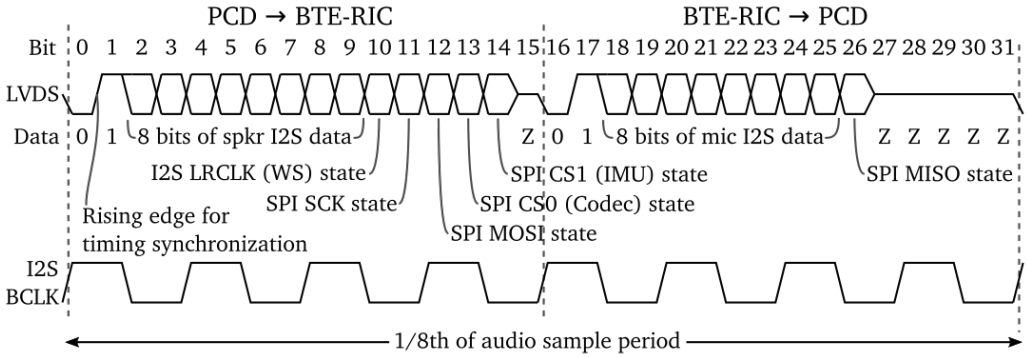

Figure 1.7. BLVDS protocol for communication between the PCD and BTE-RICs. 8 bits of I2S audio data are transmitted in each direction, plus a number of control signals, during the same time as 8 I2S bit clocks occur.

We created a custom communication protocol over BLVDS, designed to allow the SoC to transparently interact within the codec and IMU within each BTE-RIC (Fig. 1.7). There are three categories of signals which are multiplexed and packetized for transmission over LVDS: high-speed data, low-speed control, and clock. The microphone and receiver I2S data is the high-speed data; this is transmitted 8 bits at a time in each direction within each communication packet. The SPI control lines for the codec and IMU are the low-speed signals; the states of these signals are transmitted once per packet. Finally, the protocol allows the I2S bit clock in the BTE-RIC to be synchronized with that in the PCD, to correct for drift between the oscillators in the two devices. The FPGA in the BTE-RIC adjusts the sub-cycle timing of its I2S bit clock to match a known rising edge in the data stream from the PCD. Since the BTE-RIC sends back its own rising edge in its half of the packet, each FPGA can determine if the other is connected and properly responding, which allows for deterministic behavior at startup or any time communication is interrupted.

In future work, the FPGAs will also enable simultaneous capture from all four microphones on each BTE-RIC. The codec supports an extension to I2S called TDM which allows for four channels per data line. The SoC's I2S subsystem does not support this, but it does support two channels at twice the sample rate, which is the same data bitrate. For this mode, the FPGA in the PCD will send a "fake" word clock signal to the SoC which matches its expectations and "trick" it into accepting the data. The FPGA will also annotate the channel numbers in the lower, unused bits of the audio data—each sample is 32 bits but the ADC is only 24 bit—so that the application can distinguish them.

1.2.7 Simultaneous Ancillary Sensors

As described below, the OSP hardware platform currently supports three additional types of sensing capabilities, not traditionally associated with hearing aids research. Since OSP is designed to be a tool for new kinds of research beyond what is currently possible, these sensors may be used in conjunction with the audio transducers for new work in fields related to hearing, or on their own with OSP acting as a wearable acquisition and processing system. Furthermore, OSP can serve as a baseline open-source wearable hardware design, which can be modified by researchers who would like to add their own sensors for investigations into lifestyle, healthy aging, and many other health-related fields.

IMUs in BTE-RICs and PCD

Both the BTE-RICs and the PCD contain a Bosch BMI160 inertial measurement unit (IMU), which is a three-axis accelerometer plus three-axis gyroscope. The gyroscope data from the BTE-RICs provides reasonably accurate information about changes in head orientation. Assuming that target sound sources and interferers move much more slowly or rarely than the user's head, this allows changes in the user's look direction to be corrected for in algorithms which model the spatial positions of audio sources such as beamforming-based source separation or noise suppression. This has the potential to dramatically improve their convergence speed and

reduce their error rate, providing a better user experience.

In addition, there is another related healthcare application for the IMU data. Ability to maintain mobility—broadly defined as movement within one's environment—is an essential component of healthy aging, because it underlies many of the functions necessary for independence [37] [38]. In this context, gait disturbances are usually due to a combination of decreased physiological reserves and increased multisystem dysfunction [39]. The IMUs allow researchers to assess gait speed and monitor for unexplained gait disturbances during activities of daily living. Physical activity monitoring software could be developed to run in parallel with the hearing aid software and provide appropriate feedback to the user or researchers.

GPS

The SoM includes the radio hardware to support GPS-based location acquisition. Future work will focus on enabling GPS in software and acquiring useful data from it without disrupting real-time audio processing or consuming too much power.

FM-ExG hardware in PCD

The PCD's carrier board also includes a hardware subsystem for simultaneous biopotential acquisition. This consists of a fast-sampling ADC controlled by the on-board FPGA, which relays the data to the Snapdragon SoC via SPI. This system is discussed in the section below.

1.3 Simultaneous Multichannel Biopotential Signal Acquisition

1.3.1 Background

Acquisition and processing of biopotential or elecrophysiological signals—which we call "ExG", for EEG (electroencephalography), ECG/EKG (electrocardiography), EMG (electromyography), etc.—is a major field of study in emerging healthcare research. Simultaneous EEG and HA audio processing is of particular interest in pre-lingual pediatric hearing loss management,

as it could assist clinicians in fitting hearing aids to infants who are unable to self-report the efficacy of their hearing aid prescription, leading to a dramatic improvement in their quality of life [40]. Furthermore, in the future the process of HA tuning could be done adaptively via machine learning systems, which would monitor the experience of the user as measured by their EEG patterns. Unfortunately, EEG typically requires many electrodes with an independent wire for each, making acquisition systems large, expensive, and difficult to use especially in pediatric applications. While devices capable of concurrent hearing aid tuning and EEG do exist [41], to our knowledge no wearable or easily-portable devices of this kind are available to the research community. Other applications of wearable biopotential acquisition systems include monitoring conditions of concern such as heart ailments (ECG), muscle degeneration (EMG), or the progression of neurological disorders (EEG) such as Alzheimer's disease and Parkinson's [42]. In addition, there is emerging evidence that neurofeedback from EEG can be helpful as an intervention in many disease conditions [43] including epilepsy [44] and ADHD [45].

1.3.2 System Design

OSP incorporates a wearable biopotential acquisition system, which can run alongside the HA processing, and which only requires one small four-wire cable from the electrodes to the PCD. The design of this system is based upon the distributed FM-ADC architecture in [12]. The active electrodes feature high input dynamic range of around 100dB and no input gain stage. This allows them to support wet or dry electrodes, and they can be used for ECG, EMG, and EEG simply by changing the position of the electrodes on the body. In each active electrode, the biopotential signal at baseband is bandwidth-expanded into a frequency-modulated (FM) band centered at a unique carrier frequency. This upconversion is performed in an application-specific integrated circuit (ASIC) and the resultant FM signals are all driven onto a single signal wire, each FM signal occupying a distinct area of spectrum for frequency domain multiplexing (FDM). The electrodes are daisy-chained in any order and connected to the PCD via a 4-wire cable (the remaining three wires being power, ground, and a reference

voltage). The aggregate signal content of the single composite FM-FDM wire is sampled by an analog-to-digital converter (ADC) in the PCD. The data can then be streamed using WiFi for off-body processing or processed locally in multi-modal signal processing applications. In either case, after demodulation, the original biopotential signals can be recovered.

The benefits of such a biopotential acquisition system strategy include: power efficiency intrinsic to the distributed FM-ADC architecture, ruggedization against inertial motion artifacts, reduced system weight due to reduced wiring burden, and frequency up-conversion which eliminates baseband coupling artifacts in the signal wire. Its high input dynamic range ensures that the acquisition hardware does not saturate and lose signal for large motion artifacts at the input; combined with the IMUs in the BTE-RICs, OSP could in the future support IMU-based motion artifact removal as demonstrated in [46].

As presented in [11], the FM modulation allows for an increased effective signal-to-noise ratio ($\mathrm{SNR_{FM}}$) compared to the SNR of the ADC at the carrier frequency, called carrier-to-noise ratio (CNR). The overall SNR of the system depends on the bandwidth expansion ratio D [47] as follows:

$$\mathrm{SNR_{FM}} = 10\log_{10}(\frac{3}{2}D^2) + \mathrm{CNR}$$

The CNR of an ideal 12-bit ADC (i.e. 12 effective number of bits or ENoB) is 72 dB, so we chose $D = 20$ to give a theoretical $28 + 72 = 100$ dB SNR for each FM channel. Assuming EEG signals have a maximum frequency of 500 Hz, $D = 20$ leads to a 10 kHz FM frequency deviation. The actual FM bandwidth may be computed two different ways: by the empirical Carson's Rule, giving $2 \times (10\mathrm{kHz} + 500\mathrm{Hz}) = 21$ kHz, or by including all side tones with greater than 1% of the unmodulated carrier amplitude, giving $3.2 \times 10\mathrm{kHz} = 32$ kHz [47]. Based on these two estimates of the bandwidth and the desire for ≈ 10 kHz guard bands between channels, we space the channels 40 kHz apart. With a sampling frequency of 1 MHz, 12 ExG channels can be supported.

An overview of the hardware included on the PDC to realize this is shown in Fig. 1.8.

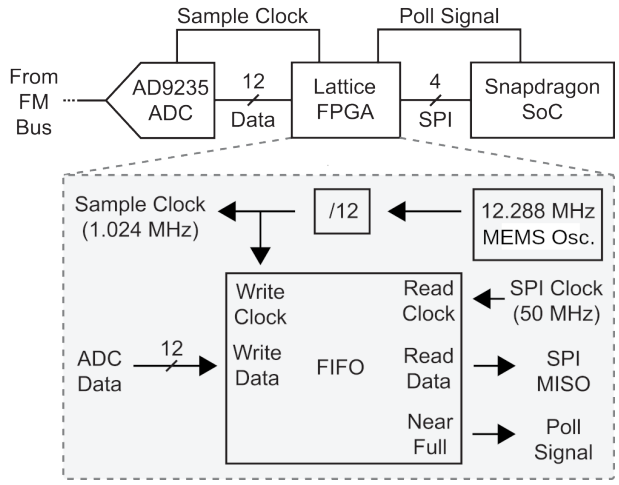

Figure 1.8. Block diagram of FM-ExG hardware in the OSP PCD. The FPGA converts between parallel and SPI data formats and stores samples in a FIFO queue for batched access by the SoC. Note that the FM sample clock is derived from the same MEMS oscillator as the I2S audio is, so the ExG and audio streams remain permanently synchronized.

The Analog Devices AD9235 [48] was chosen for its parallel interface, 12-bit resolution, and supported sampling rates up to 60 MHz. The ADC is clocked by the FPGA with a 1.024 MHz clock signal generated by dividing the 12.288 MHz clock from the MEMS oscillator driving the I2S by 12. The ADC's parallel data interface connects to the FPGA, which contains a simple FIFO queue to store the samples until they are ready to be retrieved by the SoC via SPI. A level-based signal is sent to the SoC when more than 1024 samples (1 ms of data) are available; the SoC polls this signal and then performs an SPI transfer of 1536 bytes, which covers the 1024

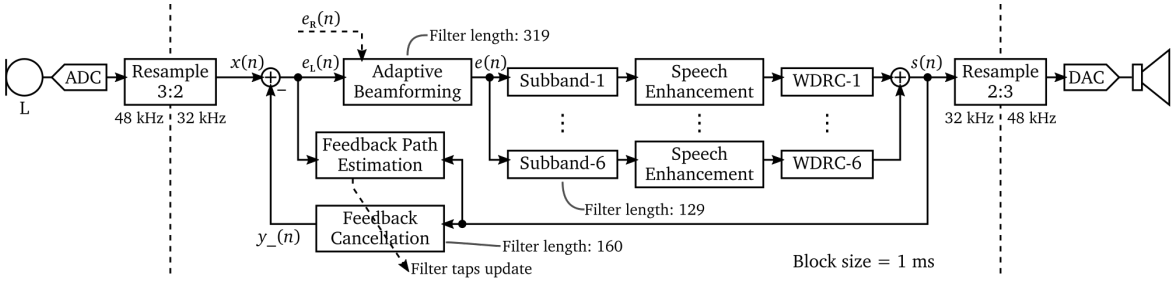

Figure 1.9. RT-MHA software block diagram with signal flows. Audio I/O operates at 48 kHz and all HA processing is carried out at 32 kHz. The baseline HA functions provided include adaptive beamforming (BF), subband decomposition, speech enhancement (SE), wide dynamic range compression (WDRC), and adaptive feedback cancellation (AFC). See Fig. 1.10 for an enlarged picture of the beamforming block.

12-bit samples. Since the SPI clock runs at 50 MHz—which could theoretically transfer 6250 bytes per ms if the clock ran continuously—there is sufficient timing slack for transfers to be stable.

When FM-ExG streaming is running, CPU core 3 is dedicated to the FM-ExG thread. It runs at the highest real-time priority and is the only thread permitted to run on this core. It polls the "data ready" signal described above, performs the SPI transfers, and executes a callback to user code for each 1 ms (1024 samples) of data received. Any processing or transmission of the data for any research application would occur during this callback. We created two programs which implement this callback to collect results as described in Sec. 1.6.3: one which measures the time between rising edges of a pulse wave for the sync measurements, and one which saves 10 seconds of data to RAM and then to disk. In the latter case, we performed digital demodulation offline using MATLAB.

1.3.3 Future Work

Our first goal for future work with FM-ExG is to enable demodulated data to be streamed via WiFi from the PCD. This will require creating a real-time implementation of the demodulator, ensuring its performance is high enough to run in the callback without disrupting the data capture, and implementing both the local and remote side of the WiFi streaming system. Once this is accomplished, we are excited to begin exploring clinical uses of FM-ExG, particularly in pediatric hearing loss research.

1.4 Real-Time Master Hearing Aid (RT-MHA)

1.4.1 Baseline Algorithms

We provide a full set of baseline implementations of common HA algorithms in the RT-MHA, to facilitate basic HA research with the platform and to provide a reference implementation for engineers to build from. An overview of the RT-MHA signal flow is shown in Fig. 1.9.

These algorithms are essential components of any HA, and can be categorized into "basic" and "advanced" functions. The basic HA functions necessary for amplification are:

1. Subband decomposition
2. Wide dynamic range compression (WDRC)
3. Adaptive feedback cancellation (AFC)

Many commercial HAs include advanced features to improve speech perception in realistic situations such as in a noisy environment. The RT-MHA implements two advanced functions for improving conversation in noise:

4. Speech enhancement (SE)
5. Microphone array processing (or beamforming)

In the following we briefly describe the role and our baseline implementation of each of these five algorithms.

Subband decomposition

Hearing loss is typically highly frequency dependent; it is common for loss to be worse at high frequencies, but loss curves vary widely among individuals. Hence, gain and other processing must be applied differently at different frequencies, motivating the decomposition of the input signal into frequency bands. In the RT-MHA, this decomposition is implemented as a bank of 6 finite impulse response (FIR) filters, where the bandwidths and upper and lower cutoff frequencies of these filters are based on Kates's MATLAB master hearing aid implementation [49].

WDRC

Both healthy hearing and hearing loss are known to be nonlinear in amplitude, with these nonlinearities varying over frequency. Therefore, a gain control mechanism that enables a

frequency-dependent, nonlinear gain adjustment is needed for modern HAs. This is carried out by the wide dynamic range compressor (WDRC), which is one of the essential building blocks of a HA [50]. The WDRC amplifies soft sounds while limiting the gain of loud sounds, with the aim of improving audibility without introducing discomfort. Typically, WDRC amplifies quiet sounds (40-50 dB SPL), attenuates loud sounds (85-100 dB SPL), and applies a variable gain for everything in between. The basic WDRC system described in [51] comprises an envelope detector for estimating the input signal power and a compression rule to realize nonlineaer amplification based on the estimated power level. Primary control parameters of the basic WDRC system are: attack time (AT), release time (RT), compression ratio (CR), gain at 65 dB input (G65), and upper and lower kneepoints (K_{up} and K_{low}) [51]. The AT or RT is the time the envelope detector takes to recover the output signal level to its steady state when a sudden rise or drop takes place in the input signal level, respectively. The amount of gain to apply will then be determined based on a compression rule as a function of the estimated input power level given by the envelope detector. The CR, G65, AT, RT, K_{up}, and K_{low} are the control parameters for characterizing the compression rule. In the RT-MHA, the above WDRC is implemented in a 6-channel system [51], where gain control is realized independently in each subband, enabled by selecting different parameters to specify the compression rule. The outputs of all the subbands after applying the WDRC are combined together to produce the HA output signal.

AFC

Feedback due to acoustic coupling between the microphone and receiver is a very well-known problem in HAs [51]. There are many methods to alleviate this phenomenon [52]. Among them, adaptive feedback cancellation (AFC) has become the most common technique because of its ability to track the variations in the acoustic feedback path and cancel the feedback signal accordingly. The AFC generates an estimate of the feedback path using an adaptive finite impulse response (FIR) filter that continuously adjusts its filter coefficients to emulate the feedback path impulse response. Typically the AFC can provide 5-12 dB added stable gain (ASG) [14]

depending on the adaptive filtering algorithms used. The RT-MHA implements the least mean square (LMS) based algorithms and features the sparsity promoting LMS (SLMS) [13] which is an advanced adaptive filtering algorithm developed by the OSP team and discussed below (Sec. 1.4.2).

SE

In a quiet environment, the above features of HAs are enough to help the user better understand speech. However, in a noisy environment such as a cafeteria or a restaurant, the HA might not be able to improve conversations without any noise reduction mechanism—for example, WDRC may amplify noise components along with soft sounds. It is therefore essential to have reliable and robust speech enhancement (SE) systems implemented in the HA. A baseline SE module, based on a version of the SE systems investigated in [53], has been added to the RT-MHA. The SE module performs denoising in the subband domain, between the subband decomposition and the WDRC blocks.

Microphone array processing

To improve speech intelligibility in noisy environments, RT-MHA implements a baseline left/right two-microphone adaptive beamforming (BF) system. This baseline system described in [54] realizes the generalized sidelobe canceller (GSC) implementation [55] of the linearly constrained minimum variance (LCMV) beamformer [56]. Fig. 1.10 depicts the BF block diagram. For the adaptation, an adaptive filter using the (modified) LMS [57] is used to continuously estimate the interference signal components. In addition, adaptation-mode-control and norm-constrained adaptation schemes have also been incorporated to improve robustness [58], i.e., to mitigate misadjustment of the BF due to array misalignment, head movement and shadow effect, room reverberation, etc. Based on simulation with one target and one interference speech signal, the baseline 2-mic beamformer improves the Signal-to-Interference Ratio (SIR) from 1.6 dB to 15.8 dB, and the Hearing-Aid Speech Quality Index (HASQI) from 0.21 to 0.43 over the

system with only one microphone (i.e., no beamformer). In informal subjective assessments, the listeners were given a web app for turning the beamforming on/off. All listeners reported a perceived reduction in the interfering speech and background noise with beamforming enabled.

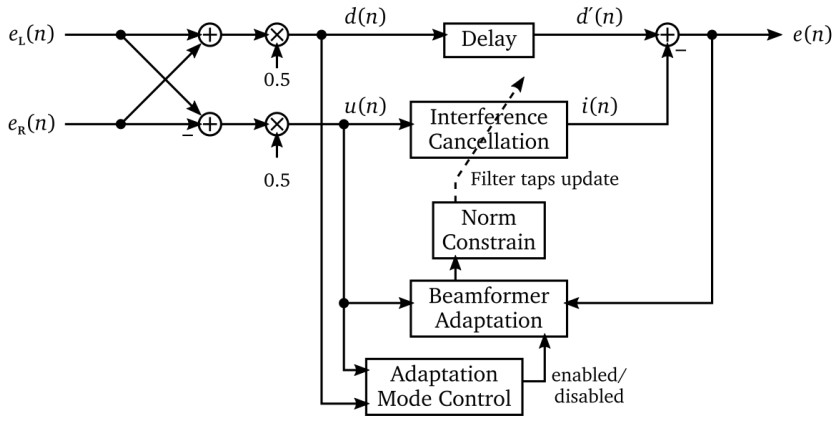

Figure 1.10. The two-microphone adaptive beamforming system in the RT-MHA. Adaptive filtering algorithms are utilized to generate interference estimates based on the left and right channel inputs, which are used to enhance the target signal.

1.4.2 Case Study: SLMS

One of the purposes of OSP is to provide a platform for academic research in DSP with easy prototyping, high-quality real-time I/O, and a strong connection to the clinical research community. As an example of such research already performed with this platform, we briefly describe the sparsity promoting LMS (SLMS) algorithm [13] used in several of the adaptive filters on the platform. The SLMS is an adaptive filtering algorithm that takes advantage of the sparsity of the underlying system response—which is present in many HA DSP applications—for improved convergence behavior when adapting the filter coefficients. In testing on early versions of OSP, we have found the SLMS to be useful in the AFC and the adaptive beamforming subsystems. In the AFC, typical feedback path impulse responses are (quasi) sparse in nature, which means they contain many zero or near-zero coefficients and few large ones. It has been shown in [13] that a proper p value of the SLMS parameter leads to a performance improvement. We reported 5 dB improvement in added stable gain with a p of 1.5 for the SLMS over the

conventional methods. For adaptive beamforming, the two-microphone GSC system of [54] also benefits from using the SLMS for the filter coefficient adaptation. We have found that improvement in signal-to-interference ratio (SIR) can be achieved for a p of $1.3 \sim 1.5$. For reference, $p = 1$ in SLMS results in the ℓ_1 norm similarly used in the well-known proportionate normalized LMS (PNLMS) [59] and $p = 2$ results in the ℓ_2 norm which yields the standard LMS.

1.5 Embedded Web Server

Most commercial HAs provide smartphone apps for the user to control various aspects of their HA. Recent evidence suggests that adults with hearing loss who have access to smartphone-based tools feel more empowered, autonomous, and in control of their hearing loss [60]. While smartphone apps hold much promise for both professionals and patients, a significant amount of research is needed in terms of assessment and guidance for informed, aware, and safe adoption of such apps by the community [61]. In order to fulfill the visions of the NIH workshop [8], we undertook development of multiple classes of such apps aimed at users (people with HL controlling their HAs), researchers (clinicians engaged in hearing healthcare research and translation), and engineers (those contributing to OSP and the open source initiative).

Most modern mobile-oriented applications fall into two categories: native apps and web apps. Web apps would typically require a remote server and guaranteed availability of an Internet connection, and thus be unsuitable for a wearable system to be used in the field. However, due to the processing power and wireless connectivity of the Snapdragon 410c SoC and the well-developed web software infrastructure on Linux, we are able to host a WiFi hotspot and a web server directly on the PCD. Thus, any browser-enabled device (such as a smartphone or a tablet) can connect to the PCD without the need for any external hardware or connection. As a result, the design decision of native apps versus web apps remained. Native apps can have better hardware integration and certain aspects of user experience, while web apps have the benefits that they do not require installation, they are operating system and form-factor agnostic, and they are

easier for programmers to modify and extend [62]. For these reasons and especially due to the ability to rapidly prototype with web apps, we adopted web apps and developed the Embedded Web Server (EWS) subsystem of OSP to support them. All together, the EWS comprises (i) a WiFi hotspot for browser-enabled devices to connect to, (ii) a web server running on the PCD, (iii) bidirectional communication between the web server and the RT-MHA for monitoring and control, and (iv) a suite of web apps hosted on the web server. Researchers can customize these apps to enable a broader range of research in hearing healthcare.

1.5.1 EWS Architecture / Software Stack

The EWS on OSP is implemented using the LAMP stack (Linux OS, Apache web server, MySQL database, and PHP scripting language) [63]. The web apps themselves are coded using HTML, CSS, and JavaScript. We have chosen SQLite as the database and a test server provided by the PHP framework as the web server. The choice of SQLite and PHP test server were guided by the fact that they do not require complex configuration steps like Apache and MySQL do. In addition, they are very lightweight from processing load and memory footprint perspectives. In the context of realtime monitoring and control of RT-MHA from a browser enabled device, we have a limited number of connections and many of the features of Apache and MySQL are not relevant.

The RT-MHA serializes OSP parameters between a binary representation in memory and a JSON string format for communication with EWS over a TCP/IP socket. All the RT-MHA parameter states are stored in the SQLite database for persistency and use by the web apps.

1.5.2 Web Apps

In order to expedite web app development, OSP provides Laravel and Node.js frameworks. Web apps in OSP present a graphical user interface (GUI) to the user via their device's browser. Based on the user's interactions with the GUI, the apps' control logic may modify the RT-MHA parameters, play back audio to the user through the BTE-RICs, record audio from the

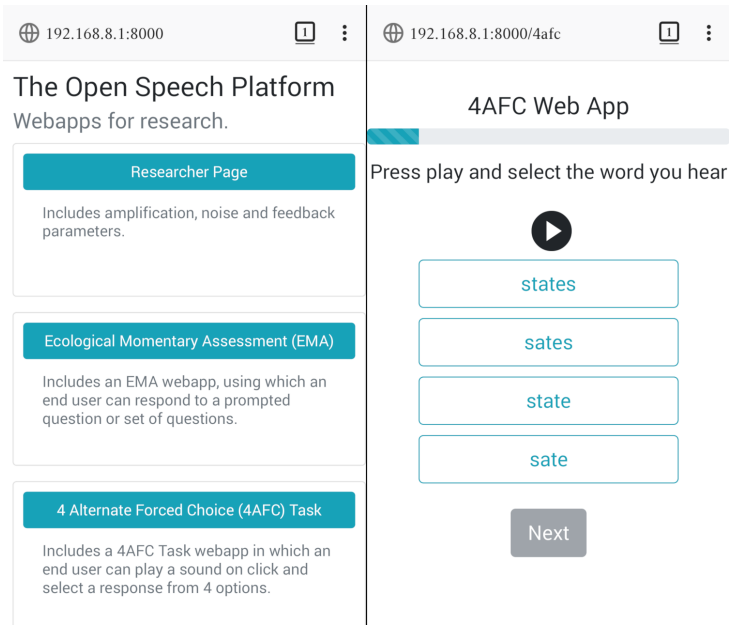

Figure 1.11. Screenshots of the main EWS page and the 4AFC task, taken from a smartphone connected to the WiFi hotspot of an OSP PCD. After powering on the PCD and connecting the smartphone to the new "ospboard-*" WiFi hotspot, the user simply enters "ospboard.local" or "192.168.8.1:8000" into a web browser and receives the page on the left. Clicking on the "4AFC" button and logging in returns the page on the right, which is a fully-functional web app that interfaces with the RT-MHA state in real time.

microphones, store information in the SQLite database or in logs, or take other actions. In this section, we describe the current suite of web apps, which showcase the functionality of the EWS and OSP as a whole and which serve as templates to be modified and extended for specific investigations.

Researcher App

The "Researcher App" is used to manipulate any of the exposed RT-MHA parameters. The main tab of this app includes all the WDRC parameters in each subband. Researchers can save different configurations in named files and load them from the GUI. A Transmit button sets the RT-MHA to the parameters displayed in the GUI. The researcher can individually control the right ear channel or the left ear channel, or both at the same time. The Noise Management tab has the parameters associated with noise management algorithms described in

Secs. 1.4.1 and 1.4.1. It enables researchers to experiment with various parameters and provide configurations such as aggressive, mild and no noise suppression in studies with human subjects. Similarly, the Feedback Management tab allows the researcher to optimize AFC parameters for specific investigations. This app is suitable for "audiologist fit" research by entering the user's initial prescription from a fitting such as NAL-NL2, DSL, etc. [64] and then optimising the HA for user comfort. The researcher app, like the other apps, requires a researcher ID and user ID to access, allowing user profiles to be easily loaded for clinical studies in which one system is used sequentially by multiple users.

Self-Fitting Apps

There has been a lot of interest in self-fitting research, wherein the user is able to choose the HA parameters with the help of apps. The recent passage of the Over-the-Counter (OTC) Hearing Aid Act of 2016 was aimed at easing the financial burden of owning HAs, at least for some users with mild to moderate hearing loss. The use of OTC HAs will require users to be able to independently control the HAs in multiple listening environments without professional assistance.

We have implemented baseline web apps for two self-fitting paradigms. First, for the lab based OSP system [21], we initially implemented a native Android version of the Goldilocks explore-and-select self fitting protocol proposed in [65] [66]. For the wearable system [10] aimed at field studies, we transitioned to web apps for the ease of rapid prototyping and ported Goldilocks as a web app.

Second, we created an AB app in which the user can switch between hearing an A or B set of RT-MHA parameters for the same stimulus, and then select the one that they prefer. For the baseline implementation, the app performs a binary search over the overall gain parameter, allowing the user to narrow in on the gain they most prefer. This is intended as a proof-of-concept for researchers to incorporate other HA parameters in their self-fitting research.

This AB app, like several others described below, relies on the audio file I/O module

included in OSP. This module, under control of the EWS, can play audio files (typically speech content) stored on the PCD to the user, with or without the RT-MHA processing. This capability allows researchers to provide stimuli to the user in a repeatable and reproducible manner. The file I/O module can also record the raw or processed microphone audio to audio files on the PCD, as described below.

Monitoring user and environment state

We have created an Ecological Momentary Assessment (EMA) web app, which is designed to help researchers understand more about the user's actions in the context of an experiment or a self-fitting adjustment. It does so by collecting information about the environmental state that elicited the user's behavior along with the user's behavior itself. The EMA web app has two components. First, it displays a brief survey through the GUI which asks the user qualitative questions about their experience and environment. Researchers can edit the survey questions by changing the contents of the JSON file associated with the EMA web app. Second, the app records microphone audio in order to characterize the user's auditory environment. This works with a circular buffer that temporarily keeps the last few seconds of microphone audio. When the EMA is started, the previous buffer is saved, and the audio continues to be saved while the user completes the survey and for a certain time after leaving the app. In the future, the information gathered from the EMA web app could be used to create machine learning models that dynamically update their parameters depending on environmental factors.

Outcomes assessment

This class of apps is aimed at assessing the benefits to the user of a proposed hearing loss intervention (such as a particular fitting or an entire self-fitting paradigm). In these apps, researchers define a series of questions in which the user hears pre-recorded sound stimuli (typically speech) and indicates their preference among them or attempts to distinguish between them. The stimuli are processed through specific HA parameter sets during playback, so they

can be used to assess the effectiveness of these fitting parameters for the user. The environment audio recording described above may also optionally be enabled in these apps.

In the 4-Alternative Forced Choice (4AFC) app (Fig. 1.11), each question has a playable prompt stimulus and four written words, one of which matches the stimulus. The words are themselves also playable, and any errors in the user's choices can inform the researcher about what improvements may be needed in the user's HA fitting. The app can easily be modified to create *N*-alternative forced choice tests.

In the outcomes assessment AB app, the user hears two different stimuli A and B, and rates their preference for B relative to A on a Likert scale. At the researcher's option, A and B may be different audio files played through the same set of RT-MHA parameters, or the same audio played through different parameter sets. In the latter case, the audio may be from a file or it may be the live real-world sound from the user's environment.

Finally, in the ABX app, the user is presented with a target stimulus X, and then two stimuli A and B where one is identical to X and the other is typically very similar. The user selects the one they believe is identical; errors imply that the user could not hear the difference between A and B. This approach has strong discriminative power; its uses include optimizing signal processing (for example, whether the user can detect distortions introduced by approximate computations to save battery power), determining just noticeable differences between parameter settings, etc.

1.5.3 Web App Customization

The current suite of web apps are meant to function as baseline, reference implementations for the development of new web apps. Some web apps can be reconfigured for new users and new trials by the researchers without modifying the software. For example, in the case of the outcomes assessment web apps (Sec. 1.5.2), the researchers can specify the contents of the questions which will be shown to the user. In the case of 4AFC, for one question, the researcher needs to specify the audio file for the prompt, as well as the text and audio files of

the four choices. The researcher can encode these choices by editing the text-based JSON file that accompanies the app. The audio files themselves are stored in a specific hierarchical file structure, so that a researcher can easily track which files are associated with which question, and have a consistent scheme to document the files referenced in the JSON file. Similarly, the AB and ABX web apps also have JSON files that are used to specify which sound files should be played for which question, which can also be edited with a text editor.

It is also possible to combine aspects of different web apps to create new apps for novel investigations. This requires familiarity with HTML, JavaScript, and PHP. When new HA parameters are exposed by the RT-MHA signal processing, they can also be easily integrated in the web apps with appropriate changes to the HTML and JavaScript (for the modified GUI) and the PHP (for the HA parameter control logic).

1.6 Results

Initial results about the performance of the wearable OSP system were reported in [10]. This section summarizes those results and includes updated results based on the current internal development versions of the OSP hardware and software (a version of which will become Release 2019b). In addition, the results relating to the FM-ExG are reported here for the first time.

1.6.1 HA Performance

Latency

Latency plays an important role in users' comfort with their devices [3] [4], and most commercial HAs have under 10 ms latency [67]. In the OSP wearable system, with the RT-MHA algorithms disabled and the software set to simply pass through an amplified copy of the front microphone input signal to each receiver, the microphone-to-loudspeaker latency is about 2.4 ms. This delay is caused by input and output buffers of 1 ms each allocated by the audio subsystem (ALSA / PortAudio), plus additional delays due to resampling filters within the codec. With the RT-MHA enabled without beamforming, it measures at about 4.6 ms, with this 2.2 ms difference

Table 1.1. ANSI 3.22 test results for OSP system configurations as measured by Audioscan Verifit 2, as compared to results from four commercial HAs.

Metric	Units	Commercial HAs				OSP Lab Sys.		OSP Wearable	
		A	B	C	D	X	Y	X	Y
Average Gain	dB	40	40	25	35	40	40	35	38
Max OSPL90	dB SPL	107	112	110	111	121	130	119	129
Average OSPL90	dB SPL	106	109	108	106	112	126	111	125
Average Gain @ 50 dB	dB	37	39	25	35	35	41	34	38
Low Cutoff	kHz	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
High Cutoff	kHz	5	6	5	6.73	8	6.3	8	6.73
Equivalent Input Noise	dB SPL	27	26	30	27	29	28	28	27
Distortion @ 500 Hz	% THD	1	1	0	0	2	1	5	1
Distortion @ 800 Hz	% THD	1	1	0	0	3	2	5	1
Distortion @ 1600 Hz	% THD	0	0	0	0	1	1	2	0

X: with high-bandwidth Knowles receiver [70]
Y: with high-power Knowles receiver [71]

being due to the FIR filters within the audio processing. With beamforming enabled and set to 5 ms delay, the latency is measured to be 9.6 ms as expected. Thus, our full-featured baseline implementation meets the 10 ms target maximum latency for a HA system. While the latency due to hardware and firmware (2.4 ms) is not user-adjustable, the latencies due to the steps of HA processing are determined by the parameters of that processing (e.g. the length of the FIR filters), which will vary as researchers tweak the baseline algorithms and implement their own. The latency "budget" of 7.6 ms for all HA processing allows for a wide range of experimentation and research.

ANSI 3.22 Test Results

ANSI 3.22 [68] is a standard test protocol for HAs, the results of which are available for commercial HAs. We measured the OSP wearable system, as well as the previous OSP laptop-based system, with the Audioscan Verifit 2 test unit [69], for comparison with four anonymous commercial HAs.

The OSP wearable system meets or exceeds the performance of the commercial HAs on

most metrics. With the high-power (bandwidth-limited) receiver, it provides higher OSPL90 (loudness) with gain, bandwidth, noise, and distortion figures which are comparable to the best of the commercial HAs. With the high-bandwidth receiver, it has similar performance with slightly reduced gain, but with higher distortion. Reducing the gain from 35 to 25 dB (not shown in the table) did reduce the distortion to 1% or less in all bands. We believe this distortion is due to impedance differences between the two receivers: the codec's output voltage swing is limited by its 3.3V supply rail, which will lead to distortion at a lower power with the higher-impedance (high-bandwidth) [70] receiver than with the lower-impedance (high-power) [71] receiver. Future BTE-RIC designs could add a boost regulator and additional power amplifier to increase the gain with the high-bandwidth receiver; nevertheless, the OSP wearable system meets its performance goals with the high-power receiver.

1.6.2 Embedded Software Performance

CPU Usage

Each audio channel (left and right ear) is processed by a separate thread so most of the computation can be done simultaneously on two CPU cores. Three of the four cores are assigned to the RT-MHA process at the OS level with the remaining core left for all OS functions and other non-realtime processes. The RT-MHA process is also given maximum CPU and I/O priority. The RT-MHA processes audio in 1 ms frames (48 samples), which means the system has less than 1 ms of real time to complete the processing of each frame. Thus, we report the real time required for each step of the RT-MHA processing.

As shown in Table 1.2, on average the processing completes with some time to spare. In addition, while most of the processing is being done by two cores (one per ear), there are three cores available for the RT-MHA as long as the FM-ExG is not in use. In this case, a substantial amount of additional processing could be added on a third thread provided that it could be done in parallel. Between 2018c and the current version, the subband and AFC filter lengths were reduced somewhat to free up CPU and latency budget for the addition of beamforming. In

Table 1.2. RT-MHA real-time processing performance statistics for Release 2018c and the current test version. Wall-clock time taken to perform each processing step on 1 ms audio buffers.

	Average Time (µs)		Maximum Time (µs)	
Operation	2018c	Current	2018c	Current
Downsampling	32	25	91	52
Beamforming	N/A	81	N/A	121
Subband filtering	282	193	822	408
Speech Enhancement	12	12	408	120
Peak detection	30	33	384	228
WDRC	18	14	384	180
AFC	190	142	3661	168
Global MPO	N/A	11	N/A	35
Upsampling	33	25	91	54
Total L or R*	597	536	5841	1366
Overall HA†	694	580	4635	953

*The sum of the above rows; for maximum time, this does not necessarily represent a single real frame, since it is the sum of the maximum values ever recorded for each operation
†The measured total time of the audio processing callback, including sending work to the threads for the left and right channels and waiting for them to complete

addition, the "maximum time" values, which effectively measure the algorithms' stability in terms of CPU usage, have dramatically reduced. This is partly due to stability and performance improvements in the OS, and partly due to improved initialization in the RT-MHA. However, it is also partly an artefact of measurement changes: in 2018c, we measured average and maximum times beginning as soon as the RT-MHA started, whereas now we begin timing measurements after the RT-MHA has initialized and run for about a second. Thus, previously the "maximum times" included initialization, whereas now they only measure timing variation in steady state.

Battery Life

We measured the current draw of the wearable system in several conditions, and computed the battery life from these assuming a 2000 mAh battery:

As seen in Table 1.3, both system and RT-MHA efficiency have been improved since last reported. Note that due to battery capacity not being fully exhaustible and other factors that may

Table 1.3. Current draw (at nominal 3.7 VDC) and battery life (assuming 2000 mAh Li-ion battery) for common system use cases.

	Avg. Current (mA) (±5)		Battery Life (h)	
Test Conditions	2018c	Current	2018c	Current
Idle (WiFi off)	243	220	8.2	9.1
Idle	284	259	7.0	7.7
HA audio loopback	317	289	6.3	6.9
RT-MHA, no BF	440	370	4.5	5.4
RT-MHA + BF	N/A	390	N/A	5.1

make the usable energy of a battery less than its rated capacity, actual usage times may be lower than reported here. Still, these results indicate that the system should provide at least 4 hours of full-featured operation per charge.

1.6.3 FM-ExG Performance

Simultaneous Acquisition

Several metrics are important in characterizing the performance of simultaneous audio and FM-ExG capture on the OSP hardware/software platform. First is the stability of the simultaneous capture—how frequently data is lost in either stream while the system is streaming them both. We tested this by running simultaneous capture from both streams into a simple utility which validated whether and when samples were lost. Over a 90-minute test, no samples were lost on FM-ExG (about 5.5 billion consecutive samples received correctly), and only one incident occurred where a few ms of audio samples were lost. Second is the long-term drift or relative inaccuracy in the sample rate of both streams. This is guaranteed to be zero by design: both the 1.024 MHz FM sample clock and all the audio clocks are derived from the same 12.288 MHz MEMS oscillator which drives the FPGA, so any drift or inaccuracy in this oscillator will be reflected uniformly in the two data streams.

Finally, a key metric for simultaneous capture is how closely the two streams can be synchronized in time. We use the term *skew* to refer to the time difference between the audio and EEG sampled data streams. Since any known skew can be corrected for by simply re-aligning

the two data streams, the metric of interest is the variability of the skew over different runs. To help synchronize the two streams, we created a "sync" feature in the OSP FM-ExG API, which signals the FPGA to insert about 1 ms worth of zeros into both the FM-ExG data stream and all microphone audio data streams. Then, a utility detects this period of exactly zero data (which is virtually impossible to occur naturally due to system noise) and marks the end of this period as corresponding to the same time in both streams. To determine the remaining skew and the skew variability after this offset was corrected for, we used a signal generator to input a pulse wave into both the FM-ExG and microphone inputs, and in software measured the timing of the rising edges relative to the sync zeros period. Fig. 1.12 shows the resulting skew over 32 trials.

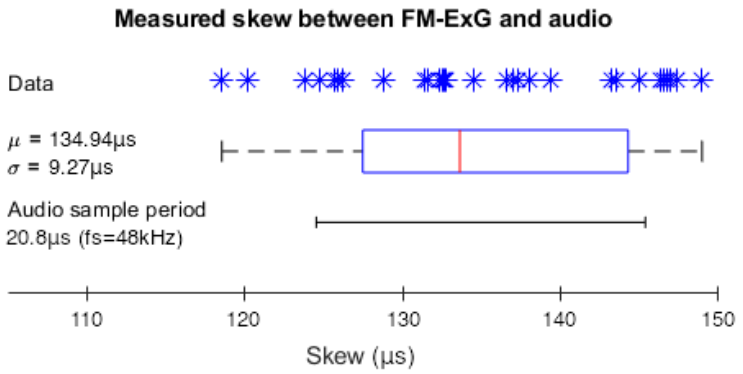

Figure 1.12. Comparing 32 trials of the measured skew between OSP's FM-ExG and audio streams with the audio sample period. Since the measurements only vary over about two audio sample periods, OSP can perform simultaneous FM-ExG and audio streaming synchronized to within about 2 audio sample periods, or about 40 µs.

The FM-ExG signal path should theoretically have a delay of about 4-5 samples due to the pipelined ADC, which accounts for about 4-5 µs; the audio signal path should have a delay of about 4-8 samples due to the resampling filters in the codec, which accounts for 80-160 µs. The measured average skew of 135 µs meets our expectations. More importantly, the standard deviation of the skew is about half the audio sample period; of course, it is not possible to identify the timing of a step signal from a sampled representation with better precision than the sample period. Since this uncertainty holds for both the sync zeros period and the pulse wave edges, and the subsample positioning of each of these are presumably uncorrelated, we expect a spread

of about $\sqrt{2}$ times the audio sample period, which closely matches the data. Hence we claim that OSP allows for simultaneous FM-ExG and audio streaming synchronized to within about 2 audio sample periods, or about 40 µs.

Analog Performance

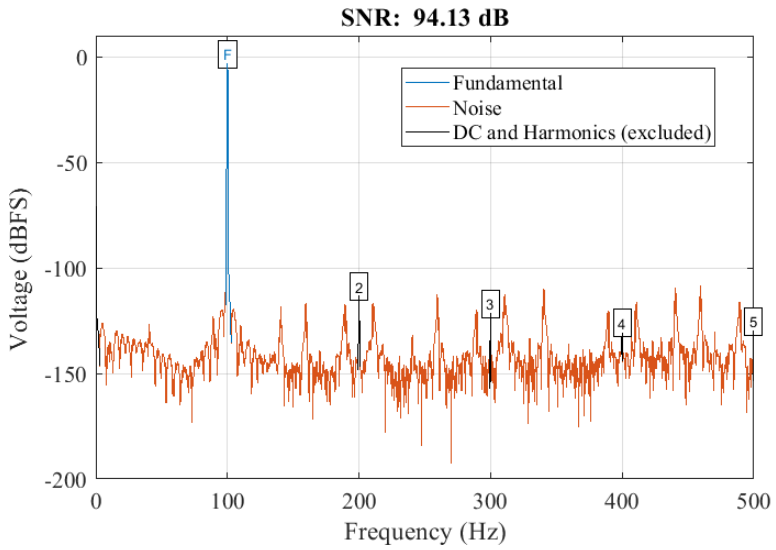

Figure 1.13. Example spectrum of demodulated FM-ExG output, for a 100 Hz sinusoidal signal as data. The FM waveform (250 kHz carrier, 20× bandwidth expansion) was generated in software and played into the OSP PCD's analog FM-ExG input via a NI DAQ test device. The sampled signal was recorded on the PCD and demodulated in MATLAB.

To evaluate the analog performance of the FM-ExG signal acquisition system, a 100Hz test sine wave modulating a 250 kHz center frequency FM carrier with bandwidth expansion of 20 (to fit the FM bandplan outlined in Sec. 1.3.2) was generated by MATLAB's fmmod() function and driven into the FM-ExG ADC introduced in Section III.B. by a National Instruments USB-6361 DAQ Multifunction Analog/Digital I/O Device. After being sampled and recorded by the PCD, the data was copied to a computer where it was demodulated using MATLAB's fmdemod() to recover the original test signal. Fig. 1.13 depicts the frequency domain representation of the result of this process, which demonstrates 94dB of signal-to-noise ratio (SNR). Compare this to the theoretical 100dB described in Sec. 1.3.2. While this is more than enough SNR for the target application, we believe this measurement may have been partially limited by the test

equipment. The DAQ test device mentioned above has a timing resolution of 10 ns, which limits the precision with which the FM carrier wave's instantaneous frequency could be generated, and thus the resolution with which the message signal could be encoded on the carrier wave.

1.6.4 Results Summary

OSP meets or exceeds the performance of four representative commercial HAs on the ANSI 3.22 test protocol with an appropriate receiver. OSP also matches the latency of commercial systems with its baseline algorithms ($<$ 10 ms), although its latency will vary as researchers reconfigure it with optimized or additional algorithms. Its capabilities for wireless control, monitoring, and user interaction via the EWS enable rapid prototyping for clinical investigations that may not be possible with most commercial systems. The CPU occupancy reported in Table 1.2 and current draw reported in Table 1.3 are only partially optimized, and may be improved further by the open-source community. The addition of 6 DOF IMUs at ear level and the capability of acquiring multi-channel EEG synchronized with auditory stimuli with about 40 µs are expected to facilitate phychophysical investigations beyond what is currently possible. In conclusion, OSP meets the requirements of the community as a HA research platform; it is not a form-factor-accurate HA, in the sense of commercial HAs.

1.7 Conclusion

Open Speech Platform (OSP) is a comprehensive hardware and software platform for research in hearing healthcare and related fields. It is designed to facilitate lab and field studies in speech processing algorithms, human sound perception, HA fitting procedures, and much more, while also enabling new kinds of research which were never before possible.

The OSP PCD hardware contains the quad-core Snapdragon 410c smartphone chipset running a custom-optimized Debian Linux OS. The PCD software comprises basic and baseline advanced binaural HA audio processing algorithms, which run in real time with CPU resources to spare. The total microphone-to-loudspeaker latency due to hardware and OS is about 2.4 ms.

Currently, basic HA processing adds 2.2 ms of latency and beamforming adds an additional 5 ms, for a total latency of 9.6 ms. The PCD is packaged in a small, light plastic case, roughly $73 \times 55 \times 20$ mm with a mass of roughly 83 grams. It contains enough battery power for at least 4 hours of operation with all features enabled.

OSP includes custom ear-level transducers in BTE-RIC form factor. They support up to four microphones per ear, including special-purpose in-ear and VPU microphones, and sample all inputs and outputs at 48 kHz 24 bit with hardware support for 96 kHz. They also contain an six-axis IMU for measuring look direction, assessing balance, and other physical activity research. The BTE-RICs communicate with the PCD via a custom packetized protocol over LVDS facilitated by FPGAs at either end, which transmits high-speed audio, control, and clock information over a single differential pair in a thin four-wire cable.

The OSP PCD is also the gateway for FM-ExG, a low-power wearable biopotential signal acquisition system for collecting EEG, ECG/EKG and EMG signals. The PCD includes a high-speed ADC and interface logic in the FPGA, to enable acquisition of 12 channels of biopotential signals with a measured SNR of 94 dB. FM-ExG can run while the HA processing is occurring, for simultaneous acquisition of audio and EEG synchronized to within 40 µs and with no long-term drift.

Finally, the PCD hosts a WiFi hotspot and web server which users and researchers can connect to with any browser-enabled device. The OSP software framework serves web apps from the PCD which allow users to interact with the parameters of the HA processing in real time. The web apps provided with the current release of OSP include apps for direct monitoring and control of all HA parameters, self-fitting, collecting data about the user's environment, and assessing HA performance. The web apps use a popular software stack and are easy to modify and extend, so that researchers can adapt them or design new web apps to conduct novel studies and field trials.

OSP has been architected to fulfill the vision set out by the NIH workshop [8] for an open, extensible research tool for hearing healthcare and related fields. OSP meets all of the

basic requirements presented there—portable hardware, real-time signal processing, advanced processing power, wireless controllability, a reference HA implementation, and open-source hardware and software releases. It further meets many of the advanced or optional suggestions: wearability, use of an FPGA in the signal chain, binaural processing, and incorporation of sensing paradigms not traditionally associated with hearing aids, such as FM-ExG and the IMUs. OSP is a powerful set of tools which promote the open initiative for collaborative work on research hardware and software, towards new discoveries in hearing-related healthcare research.

Acknowledgments

This work is supported by the National Institute of Health, NIH / NIDCD grants R21DC015046, R33DC015046, "Self-fitting of Amplification: Methodology and Candidacy," and R01DC015436, "A Real-time, Open, Portable, Extensible Speech Lab," awards to University of California, San Diego; by the U.S. Army Research Laboratory under Contract W911QX-16-C-0003; and by the National Science Foundation Graduate Research Fellowship under Grant DGE-1144086, and grant IIS-1838830 from the Division of Information & Intelligent Systems, "A Framework for Optimizing Hearing Aids In Situ Based on Patient Feedback, Auditory Context, and Audiologist Input."

Support from Sonion for providing emerging ear level transducers (in-ear microphone integrated in the speaker module and "VPU" bone conduction microphone), traditional BTE-RIC transducers, and other electromechanical components is greatly appreciated.

Chapter 1, in full, is a reprint of the material as it appears in IEEE Access. Pisha, Louis; Warchall, Julian; Zubatiy, Tamara; Hamilton, Sean; Lee, Ching-Hua; Chockalingam, Ganz; Mercier, Patrick P; Gupta, Rajesh; Rao, Bhaskar D; Garudadri, Harinath, IEEE, November 2019.

The dissertation author was the primary investigator and author of this paper.

Bibliography

[1] Rebecca J Bennett, Ariane Laplante-Lévesque, Carly J Meyer, and Robert H Eikelboom. "Exploring hearing aid problems: perspectives of hearing aid owners and clinicians". In: *Ear and hearing* 39.1 (2018), pp. 172–187.

[2] American Speech-Language-Hearing Association et al. *Guidelines for manual pure-tone threshold audiometry*. Tech. rep. ASHA, 2005.

[3] Jeremy Agnew and Jeffrey M Thornton. "Just noticeable and objectionable group delays in digital hearing aids". In: *Journal of the American Academy of Audiology* 11.6 (2000), pp. 330–336.

[4] Michael A Stone and Brian CJ Moore. "Tolerable hearing aid delays. I. Estimation of limits imposed by the auditory path alone using simulated hearing losses". In: *Ear and Hearing* 20.3 (1999), pp. 182–192.

[5] Annie N Simpson, Lois J Matthews, Christy Cassarly, and Judy R Dubno. "Time from hearing aid candidacy to hearing aid adoption: a Longitudinal Cohort Study". In: *Ear and hearing* 40.3 (2019), pp. 468–476.

[6] SERGEI Kochkin. "MarkeTrak VIII: utilization of PSAPs and direct-mail hearing aids by people with hearing impairment". In: *Hearing Review* 17.6 (2010), pp. 12–16.

[7] Lisa Brody, Yu-Hsiang Wu, and Elizabeth Stangl. "A Comparison of Personal Sound Amplification Products and Hearing Aids in Ecologically Relevant Test Environments". In: *American journal of audiology* 27.4 (2018), pp. 581–593.

[8] Roger L. Miller and Amy Donahue. *Open Speech Signal Processing Platform Workshop*. Tech. rep. Bethesda, MD: National Institutes of Health, Oct. 2014.

[9] THE_Lab at UC San Diego. *Open Speech Platform*. http://openspeechplatform.ucsd.edu/. 2019.

[10] Louis Pisha, Sean Hamilton, Dhiman Sengupta, Ching-Hua Lee, Krishna Chaithanya Vastare, et al. "A Wearable Platform for Research in Augmented Hearing". In: *2018 52nd Asilomar Conference on Signals, Systems, and Computers* 52 (2018).

[11] Julian Warchall, Shiva Kaleru, Nidhi Jayapalan, Bijoor Nayak, Harinath Garudadri, and Patrick P Mercier. "A 678 uW Frequency-Modulation-Based ADC With 104 dB Dynamic Range in 44 kHz Bandwidth". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 65.10 (2018), pp. 1370–1374.

[12] Julian Warchall, Paul Theilmann, Yuxuan Ouyang, Harinath Garudadri, and Patrick P. Mercier. “A Rugged Wearable Modular ExG Platform Employing a Distributed Scalable Multi-Channel FM-ADC Achieving 101dB Input Dynamic Range and Motion-Artifact Resilience”. In: *2019 IEEE International Solid- State Circuits Conference - (ISSCC)* (2019), pp. 362–363.

[13] C.-H. Lee, B. D. Rao, and H. Garudadri. “Sparsity promoting LMS for adaptive feedback cancellation”. In: *Proc. Europ. Signal Process. Conf. (EUSIPCO)*. 2017, pp. 226–230.

[14] C.-H. Lee, J. M. Kates, B. D. Rao, and H. Garudadri. “Speech quality and stable gain trade-offs in adaptive feedback cancellation for hearing aids”. In: *J. Acoust. Soc. Am.* 142.4 (2017), EL388–EL394.

[15] Roger L Miller. *Open Design Tools for Speech Signal Processing (R01)*. Tech. rep. National Institutes of Health, National Institute on Deafness, and Other Communication Disorders, 2015.

[16] Roger L Miller. *Open Design Tools for Speech Signal Processing (R43/R44)*. Tech. rep. National Institutes of Health, National Institute on Deafness, and Other Communication Disorders, 2015.

[17] Odile Clavier, Chip Audette, Daniel Rasetshwane, Stephen Neely, et al. *Tympan*. https://tympan.org/. 2019.

[18] Paul Stoffregen. *Teensy USB Development Board*. https://www.pjrc.com/teensy/. 2019.

[19] Christopher Obbard, Daniel James, Tobias Herzke, Hendrik Kayser, et al. *Open community platform for hearing aid algorithm research*. http://www.openmha.org/. 2019.

[20] Issa Panahi, Nasser Kehtarnavaz, Linda Thibodeau, et al. *Smartphone-Based Open Research Platform for Hearing Improvement Studies*. https://www.utdallas.edu/ssprl/hearing-aid-project/. 2019.

[21] Harinath Garudadri, Arthur Boothroyd, Ching-Hua Lee, Swaroop Gadiyaram, Justyn Bell, Dhiman Sengupta, Sean Hamilton, Krishna Chaithanya Vastare, Rajesh Gupta, and Bhaskar D Rao. “A realtime, open-source speech-processing platform for research in hearing loss compensation”. In: *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2018, pp. 1900–1904.

[22] Variscite. *DART-SD410: Qualcomm Snapdragon 410*. https://www.variscite.com/product/system-on-module-som/cortex-a53-krait/dart-sd410-qualcomm-snapdragon-410/. 2019.

[23] Louis Pisha, Sean Hamilton, Dhiman Sengupta, Ching-Hua Lee, Krishna Chaithanya Vastare, Sergio Luna, Tamara Zubatiy, Cagri Yalcin, Alex Grant, Mark Stambaugh, Arthur Boothroyd, Ganz Chockalingam, Rajesh Gupta, Bhaskar Rao, and Harinath Garudadri. “A Wearable Platform for Hearing Aids Research”. In: *International Hearing Aid Research Conference (IHCON)*. 2018.

[24] Analog Devices, Inc. *ADAU1372 Data Sheet*. https://www.analog.com/media/en/technical-documentation/data-sheet/ADAU1372.pdf. 2014.

[25] Ruth Bentler, H Gustav Mueller, and Todd A Ricketts. *Modern hearing aids: verification, outcome measures, and follow-up*. Plural Publishing, 2016.

[26] Francis Kuk, Denise Keenan, and Chi-chuen Lau. "Vent configurations on subjective and objective occlusion effect". In: *Journal of the American Academy of Audiology* 16.9 (2005), pp. 747–762.

[27] David T Kemp. "Stimulated acoustic emissions from within the human auditory system". In: *The Journal of the Acoustical Society of America* 64.5 (1978), pp. 1386–1391.

[28] P Sergi, G Pastorino, P Ravazzani, G Tognola, and F Grandori. "A hospital based universal neonatal hearing screening programme using click-evoked otoacoustic emissions". In: *Scandinavian Audiology* 30.1 (2001), pp. 18–20.

[29] Sonion. *Sonion*. https://www.sonion.com/. 2019.

[30] Sonion. *VPU14AA01 Tentative Data Sheet. Private communication*. 2018.

[31] Ching-Hua Lee, Bhaskar D. Rao, and Harinath Garudadri. "Bone-Conduction Sensor Assisted Noise Estimation for Improved Speech Enhancement". In: *Proc. Interspeech 2018*. 2018, pp. 1180–1184. DOI: 10.21437/Interspeech.2018-1046. URL: http://dx.doi.org/10.21437/Interspeech.2018-1046.

[32] Volodymyr Kuleshov, S Zayd Enam, and Stefano Ermon. "Audio super resolution using neural networks". In: *arXiv preprint arXiv:1708.00853* (2017).

[33] Lattice Semiconductor. *MachXO3: futureproof your control PLD and bridging designs*. https://www.latticesemi.com/Products/FPGAandCPLD/MachXO3. 2019.

[34] John Goldie. *The Many Flavors of LVDS (SNLA184)*. Tech. rep. Texas Instruments, 2011.

[35] Michael Peffers. *Introduction to M-LVDS (TIA/EIA-899)*. Tech. rep. Texas Instruments, Feb. 2002.

[36] Syed B. Huq and John Goldie. *Application Note 971: an Overview of LVDS Technology*. Tech. rep. National Semiconductor, July 1998.

[37] Susan E Lord, Mark Weatherall, and Lynn Rochester. "Community ambulation in older adults: which internal characteristics are important?" In: *Archives of physical medicine and rehabilitation* 91.3 (2010), pp. 378–383.

[38] Thomas R Prohaska, Lynda A Anderson, Steven P Hooker, Susan L Hughes, and Basia Belza. "Mobility and aging: transference to transportation". In: *Journal of Aging Research* 2011 (2011).

[39] Nancye M Peel, Suzanne S Kuys, and Kerenaftali Klein. "Gait speed as a measure in geriatric assessment in clinical settings: a systematic review". In: *The Journals of Gerontology Series A: Biological Sciences and Medical Sciences* 68.1 (2013), pp. 39–46.

[40] Jareen Meinzen-Derr, Lynne H.Y. Lim, Daniel I. Choo, Samantha Buyniski, and Susan Wiley. "Pediatric hearing impairment caregiver experience: impact of duration of hearing loss on parental stress". In: *International Journal of Pediatric Otorhinolaryngology* 72.11 (2008), pp. 1693–1703. ISSN: 0165-5876. DOI: https://doi.org/10.1016/j.ijporl.2008.08.005. URL: http://www.sciencedirect.com/science/article/pii/S0165587608003819.

[41] Christian Bech Christensen, Renskje K Hietkamp, James M Harte, Thomas Lunner, and Preben Kidmose. "Toward EEG-Assisted Hearing Aids: objective Threshold Estimation Based on Ear-EEG in Subjects With Sensorineural Hearing Loss". In: *Trends in hearing* 22 (2018), p. 2331216518816203.

[42] V. Mihajlović, B. Grundlehner, R. Vullers, and J. Penders. "Wearable, Wireless EEG Solutions in Daily Life Applications: what are we Missing?" In: *IEEE Journal of Biomedical and Health Informatics* 19.1 (Jan. 2015), pp. 6–21. ISSN: 2168-2194. DOI: 10.1109/JBHI.2014.2328317.

[43] Karli Kondo, Katherine M Noonan, Michele Freeman, Chelsea Ayers, Benjamin J Morasco, and Devan Kansagara. "Efficacy of Biofeedback for Medical Conditions: an Evidence Map". In: *Journal of general internal medicine* (2019), pp. 1–11.

[44] M. Barry Sterman and Tobias Egner. "Foundation and Practice of Neurofeedback for the Treatment of Epilepsy". In: *Applied Psychophysiology and Biofeedback* 31.1 (Apr. 2006), p. 21. ISSN: 1573-3270. DOI: 10.1007/s10484-006-9002-x. URL: https://doi.org/10.1007/s10484-006-9002-x.

[45] Stefanie Enriquez-Geppert, Diede Smit, Miguel Garcia Pimenta, and Martijn Arns. "Neurofeedback as a Treatment Intervention in ADHD: current Evidence and Practice". In: *Current psychiatry reports* 21.6 (2019), p. 46.

[46] B. H. Kim, J. Chun, and S. Jo. "Dynamic motion artifact removal using inertial sensors for mobile BCI". In: *2015 7th International IEEE/EMBS Conference on Neural Engineering (NER)* (Apr. 2015), pp. 37–40. ISSN: 1948-3546. DOI: 10.1109/NER.2015.7146554.

[47] Simon Haykin. *Communication systems*. John Wiley & Sons, 2008.

[48] Analog Devices, Inc. *AD9235 12-Bit, 20/40/65 MSPS 3 V A/D converter*. https://www.analog.com/media/en/technical-documentation/data-sheets/AD9235.pdf. 2012.

[49] J. M. Kates. *Master hearing aid implementation in MATLAB*. Private communication. 2016.

[50] J. M. Kates. "Principles of digital dynamic-range compression". In: *Trends in Amplification* 9.2 (2005), pp. 45–76.

[51] J. M. Kates. *Digital hearing aids*. Plural publishing, 2008.

[52] T. van Waterschoot and M. Moonen. "Fifty years of acoustic feedback control: atate of the art and future challenges". In: *Proc. IEEE* 99.2 (2011), pp. 288–327.

[53] J. M. Kates. "Modeling the effects of single-microphone noise-suppression". In: *Speech Commun.* 90 (2017), pp. 15–25.

[54] J. E. Greenberg and P. M. Zurek. "Evaluation of an adaptive beamforming method for hearing aids". In: *J. Acoust. Soc. Am.* 91.3 (1992), pp. 1662–1676.

[55] L. J. Griffiths and C. W. Jim. "An alternative approach to linearly constrained adaptive beamforming". In: *IEEE Trans. Antennas Propag.* 30.1 (1982), pp. 27–34.

[56] O. L. Frost. "An algorithm for linearly constrained adaptive array processing". In: *Proc. the IEEE* 60.8 (1972), pp. 926–935.

[57] J. E. Greenberg. “Modified LMS algorithms for speech processing with an adaptive noise canceller”. In: *IEEE Trans. Speech Audio Process.* 6.4 (1998), pp. 338–351.

[58] Osamu Hoshuyama and Akihiko Sugiyama. “Robust adaptive beamforming”. In: *Microphone arrays: signal processing techniques and applications*. Springer, 2001, pp. 87–109.

[59] D. L. Duttweiler. “Proportionate normalized least-mean-squares adaptation in echo cancelers”. In: *IEEE Trans. Speech Audio Process.* 8.5 (2000), pp. 508–518.

[60] David W Maidment, Yasmin HK Ali, and Melanie A Ferguson. “Applying the COM-B Model to Assess the Usability of Smartphone-Connected Listening Devices in Adults with Hearing Loss”. In: *Journal of the American Academy of Audiology* 30.5 (2019), pp. 417–430.

[61] Alessia Paglialonga, Gabriella Tognola, and Francesco Pinciroli. “Apps for hearing science and care”. In: *American Journal of Audiology* 24.3 (2015), pp. 293–298.

[62] A Charl and B LeRoux. *Web apps are cheaper to develop and deploy than native apps, but can they match the native user experience?, communications of the acm, 54 (5), 49 Gartner Inc.,(2013)*. 2011.

[63] Lee and Brent Ware. *Open Source Development with LAMP: Using Linux, Apache, MySQL and PHP*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 020177061X.

[64] Gitte Keidser, Harvey Dillon, Matthew Flax, Teresa Ching, and Scott Brewer. “The NAL-NL2 prescription procedure”. In: *Audiology research* 1.1 (2011).

[65] Arthur Boothroyd and Carol Mackersie. “A “Goldilocks” approach to hearing-aid self-fitting: user interactions”. In: *American journal of audiology* 26.3S (2017), pp. 430–435.

[66] Carol Mackersie, Arthur Boothroyd, and Alexandra Lithgow. “A “Goldilocks” approach to hearing aid self-fitting: ear-canal output and speech intelligibility index”. In: *Ear and hearing* 40.1 (2019), pp. 107–115.

[67] Harvey Dillon, Gitte Keidser, Anna O’Brien, and Heidi Silberstein. “Sound quality comparisons of advanced hearing aids”. In: *The hearing journal* 56.4 (2003), pp. 30–32.

[68] American National Standards Institute. *Specification of Hearing Aid Characteristics (ANSI 3.22-2014)*. ANSI New York, 2014.

[69] Audioscan Verifit 2. *https://www.audioscan.com/verifit2*. Audioscan / Etymonic Design Inc., 2014.

[70] Knowles Electronics. *RVA-90020-NXX Datasheet*. 2009.

[71] Knowles Electronics. *RVA-90080-NXX Datasheet*. 2014.

Chapter 2

Accelerating non-power-of-2 size Fourier transforms with GPU Tensor Cores

Reprinted from 2021 IEEE International Parallel and Distributed Processing Symposium, May 2021, DOI 10.1109/IPDPS49936.2021.00059

2.0 Abstract

Fourier transforms whose sizes are powers of two or have only small prime factors have been extensively studied, and optimized implementations are typically memory-bound. However, handling arbitrary transform sizes—which may be prime or have large prime factors—is difficult. Direct discrete Fourier transform (DFT) implementations involve extra computation, while fast Fourier transform (FFT)-style factorized decompositions introduce additional overheads in register use, multiprocessor occupancy, and memory traffic. Tensor Cores are hardware units included in modern GPUs which perform matrix multiply-adds at a much higher throughput than normal GPU floating-point instructions. Because of their higher throughput and better uniformity across sizes, DFT/FFT implementations using Tensor Cores can surpass the performance of existing DFT/FFT implementations for difficult sizes. We present key insights in this approach, including complex number representation, efficient mapping of odd sizes to Tensor Cores (whose dimensions are all powers of 2), and adding a size 2 or size 4 epilogue transform at very low cost. Furthermore, we describe a method for emulating FP32 precision while using lower-precision

Tensor Cores to accelerate the computation. For large batch sizes, our fastest Tensor Core implementation per size is at least 10% faster than the state-of-the-art cuFFT library in 49% of supported sizes for FP64 (double) precision and 42% of supported sizes for FP32 precision. The numerical accuracy of the results matches that of cuFFT for FP64 and is degraded by only about 0.3 bits on average for emulated FP32. To our knowledge, this is the first application of Tensor Cores to FFT computation which meets the accuracy and exceeds the speed of the state of the art.

2.1 Introduction

Power-of-2 size Fourier transforms are extremely popular, in large part because of the availability of the $O(n \log n)$-complexity Cooley-Tukey fast Fourier transform (FFT) algorithm [1]. However, FFT libraries must provide implementations for all transform sizes, including sizes which are prime or have large prime factors. While sub-$O(n^2)$ algorithms such as Bluestein [2] do exist for these cases, these approaches have a large overhead, and direct computation of the discrete Fourier transform (DFT) is often faster despite being $O(n^2)$. However, the latter is heavily reliant on compute performance, unlike Cooley-Tukey which is typically memory-bound for sizes which can be factorized into small prime numbers. As a result, we looked to Tensor Cores to improve performance in these cases.

Tensor Cores (TCs) are hardware arithmetic units in the GPU designed to perform accelerated matrix multiply-add operations for deep learning and linear algebra applications. They were introduced by NVIDIA in the Volta GPU architecture in 2017 [3] and were enhanced with the subsequent Turing [4] and Ampere [5] GPU architectures. Other GPU hardware manufacturers have indicated plans for similar hardware in their GPUs [6].

A previous paper by Sorna *et. al.* [7] introduced the notion of using TCs to accelerate FFT computation, and provided the basic approach to mixed-precision computation which we build on. However, their implementation was unable to reach either the speed or the numerical accuracy of cuFFT in FP32, and was limited to a small number of power-of-2 sizes for which

cuFFT was already memory-bound. In contrast, we use TCs for sizes on which they are likely to provide the most benefit, and we exploit the capabilities of the new third-generation TCs in the NVIDIA Ampere A100 GPU. As a result, we are able to match the accuracy and exceed the performance of cuFFT.

The remainder of this paper is organized as follows. Section 2.2 introduces our strategies for making efficient use of the data layouts imposed by the TCs, including representation of complex numbers, computing odd size transforms with power-of-two size TC operations, and efficiently performing size 2 or size 4 transforms after the main odd size transform. Section 2.3 describes how we use mixed-precision arithmetic to produce FP32 results with very little loss of accuracy. Section 2.4 discusses two implementations of these strategies with different I/O and resource usage patterns. Finally, section 2.5 presents the speed and accuracy results of these implementations, and compares them to the cuFFT library (state-of-the-art baseline) and to memory copies (ideal baseline).

2.2 Leveraging Tensor Core Data Layouts

The TCs can be directly accessed via special inline assembly (PTX) instructions within a CUDA program [8], as well as indirectly by CUDA C++ functions. They are warp-wide instructions, in that each thread within the warp supplies input data in registers for different elements of the input matrices, and receives output data corresponding to different elements of the output matrix. For example, for FP64 (double precision), the DMMA (double-precision matrix multiply-accumulate) instruction performs the operation

$$D = AB + C \tag{2.1}$$

where $A \in \mathbb{D}^{8\times4}$, $B \in \mathbb{D}^{4\times8}$, $C, D \in \mathbb{D}^{8\times8}$, and $\mathbb{D}$ is the space of IEEE FP64 floating point values. The layout of the input and output operands in the threads within the warp is shown in Figure 2.1.

The main advantage of using TCs as opposed to traditional fused multiply-add (FMA)

				0	4	8	12	16	20	24	28	Matrix B (4x8)
				1	5	9	13	17	21	25	29	
				2	6	10	14	18	22	26	30	
Matrix A (8x4)				3	7	11	15	19	23	27	31	
0	1	2	3	0	0	1	1	2	2	3	3	Matrices C/D (8x8)
4	5	6	7	4	4	5	5	6	6	7	7	
8	9	10	11	8	8	9	9	10	10	11	11	
12	13	14	15	12	12	13	13	14	14	15	15	
16	17	18	19	16	16	17	17	18	18	19	19	
20	21	22	23	20	20	21	21	22	22	23	23	
24	25	26	27	24	24	25	25	26	26	27	27	
28	29	30	31	28	28	29	29	30	30	31	31	

Figure 2.1. TC instructions are warp-wide, meaning that the elements of the input and output matrices are distributed over the threads of a warp (thread index shown by number). The layout of the DMMA (double-precision matrix multiply-accumulate) instruction from NVIDIA Ampere A100 is shown here. © 2021 IEEE

instructions is increased throughput, meaning increased total number of floating-point operations completed in a given time. On A100, TCs for FP64 have twice the throughput compared to traditional FP64 arithmetic, and TCs for TF32 have eight times the throughput compared to traditional FP32 arithmetic [5]. As described below in section 2.3, four operations are required to emulate FP32 precision with TF32 arithmetic, so the overall throughput of TC emulated FP32 is up to twice that of traditional FP32.

2.2.1 Complex Number Representation

The DFT is fundamentally the multiplication of a complex signal vector $\vec{x} \in \mathbb{C}^N$ by the fixed complex DFT matrix $W \in \mathbb{C}^{N \times N}$, $[W]_{k,n} = \exp\left(-2\pi j \frac{kn}{N}\right)$,

$$\vec{y} = W\vec{x} \tag{2.2}$$

(In this paper, we focus only on the forward transform; the inverse transform may be easily obtained by replacing W with $W^H = W^*$ and, depending on the definition, optionally multiplying by $1/N$.) As such, the only arithmetic operations involved are complex multiply operations and complex add operations. One complex multiply-add $s = pq + r$ requires four real multiplies and

four real adds:

$$\begin{aligned} p &= a + jb,\ \ q = c + jd,\ \ r = e + jf \\ s &= pq + r = (ac - bd + e) + j(ad + bc + f) \\ &= \Re(s) + j\Im(s) \end{aligned} \tag{2.3}$$

One approach to mapping these operations to the TCs is to simply perform four DMMA operations, on inputs that are the same size as the DMMA, as:

$$\begin{aligned} \Re(S) &= (-B)D + (AC + E) \\ \Im(S) &= BC + (AD + F) \end{aligned} \tag{2.4}$$

where $A, B \in \mathbb{D}^{8\times4}$, $C, D \in \mathbb{D}^{4\times8}$, $E, F, \Re(S), \Im(S) \in \mathbb{D}^{8\times8}$. This approach has the advantage that the real and imaginary components of a number are always held by the same thread within the warp.

However, we instead map the elements of the complex multiply to subsets of the elements of the DMMA instruction, performing only one DMMA operation and reducing the effective size of that instruction by four times. The complex multiply can be rewritten as

$$\begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} c & d \\ -d & c \end{bmatrix} + \begin{bmatrix} e & f \end{bmatrix} = \begin{bmatrix} \Re(s) & \Im(s) \end{bmatrix} \tag{2.5}$$

These four submatrices are tiled in 8×2, 2×4, 8×4, and 8×4 patterns respectively to fill up the DMMA. This effectively results in a $8 \times 2 \times 4$ "complex DMMA instruction" (Figure 2.2). This smaller size is critical to the algorithm described below in subsection 2.2.2. A further benefit of this approach is that each thread holds exactly one complex r and s value, although the data layout of the p and q values is more complicated.

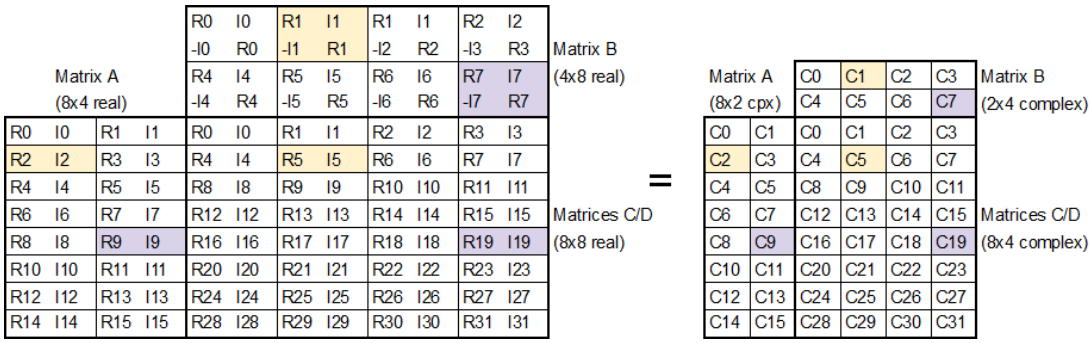

Figure 2.2. The layout of the components of complex numbers within a DMMA instruction, shown in terms of the real components (left) and as the equivalent complex numbers with a smaller tile size (right). The numbers represent the index within each matrix. The colors highlight two out of the 64 complex multiply-adds performed. © 2021 IEEE

2.2.2 The Accordion Algorithm: Odd Size DFTs with Tensor Cores

The basic concept for using TCs to accelerate Fourier transforms is to take a batch of M signals $X \in \mathbb{D}^{N\times M}$ and multiply this matrix by the DFT matrix W to obtain $Y \in \mathbb{D}^{N\times M}$, a batch of transformed signals. If this matrix multiply can be efficiently broken up into tiles that match the dimensions of the DMMA instruction, the TCs can be used. As explained in the Introduction, we are interested in sizes that are prime or have large prime factors, so we first focus on odd-size transforms (which includes all prime numbers besides 2).

Unfortunately, all three dimensions of all TC operations are power-of-two sizes; this remains the case for our $8 \times 2 \times 4$ complex DMMA introduced above in subsection 2.2.1. Fortunately, the DFT matrix has structure that can be exploited to solve this problem: the first row and first column of the matrix are all equal to 1. The fact that the first row is all 1 means that the first output element is the sum of all input elements:

$$y_0 = \sum_{n=0}^{N-1} x_n \tag{2.6}$$

Conversely, the fact that the first column is all 1 means that each of the output elements is the

sum of the first input element, plus complex multiples of the other input elements:

$$y_k = x_0 + \sum_{n=1}^{N-1} \exp\left(-2\pi j \frac{kn}{N}\right) x_n,\ k \in [1, N-1] \tag{2.7}$$

Each of these equations contains $N-1$ pure complex adds, i.e. with complex multiplies by 1 omitted. Thus, the TC can be used for the nontrivial complex multiply-adds making up the bulk of the DFT matrix—which, with the first row and first column removed, is now even in size—and these extra sums can be computed separately. We proceed to detail this process for each case.

To remove the first column of the DFT matrix (Figure 2.3 top), we observe that the operation of the TC is always a matrix multiply-add, while the DFT operation is merely a matrix multiply (the "add" input is zero). Thus, there is an extra add available at no extra cost. We simply initialize each element of the output vector to the first element of the input vector, $y_k = x_0, k \in [0, N-1]$, and then perform the multiply-add using the truncated DFT matrix with this pre-initialized $\vec{y}$ as the initial "add" input. For odd sizes, this reduces one dimension of the DFT matrix $N = 2k+1, k \in \mathbb{Z}$ to a multiple of 2. As a result, this dimension of the DFT matrix always fits in the inner dimension of the B matrix of the complex DMMA.

To remove the first row of the DFT matrix (Figure 2.3 bottom), we simply compute the sum of the input elements according to Equation 2.6 with normal floating-point add instructions. In implementations where the input data is loaded incrementally or streamed (section 2.4), whenever input data is loaded to registers for use with the TCs, it is also added to a separate per-thread accumulator, and the accumulators are added across the threads as appropriate at the end of the computation. However, this process is only peformed for FFT sizes $N = 4n+1,\ n \in \mathbb{Z}$, i.e. for odd sizes 5, 9, etc. This is because in these cases, removing the first row causes the DFT matrix to be a multiple of 4 size which matches the TCs. Conversely, in the other odd cases $N = 4n-1$, removing the first row would cause the size to be $4n-2$, which does not match the TCs. In these cases, we instead leave the first row of the DFT matrix intact and add an extra row at the bottom containing zero values which do not contribute to the result (Figure 2.4 bottom

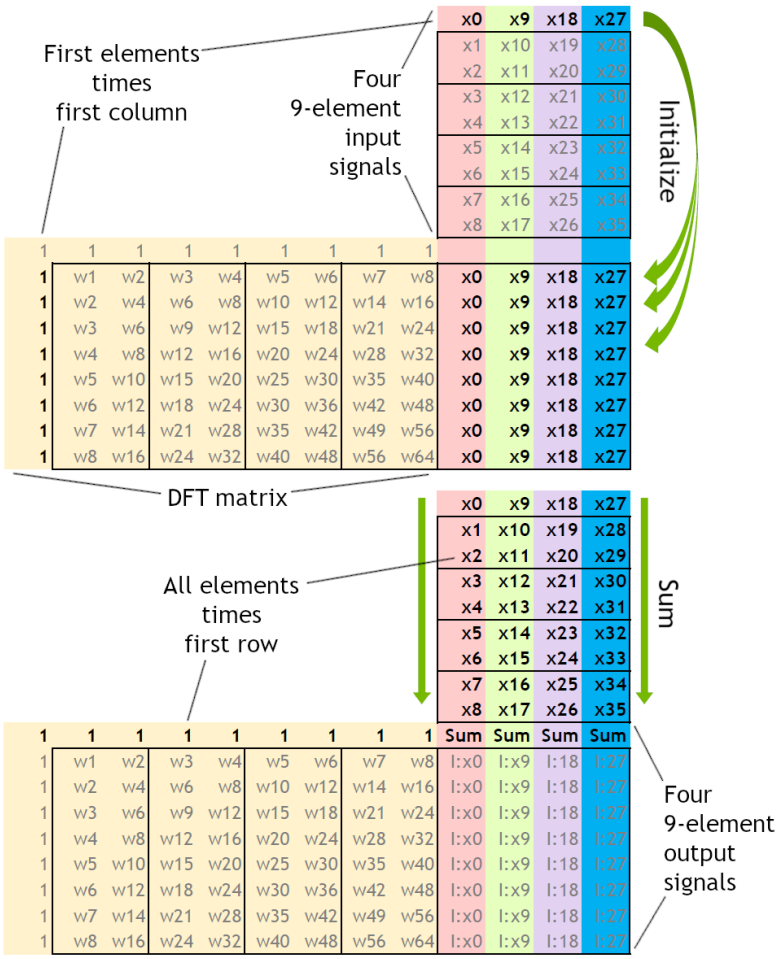

Figure 2.3. The strategies for removing the first column and first row of the DFT matrix, which are all 1s. For the first column, the accumulators are initialized with the first input signal sample. For the first row, all the samples are added. In the case shown here, the DFT matrix is thus reduced from size 9×9 to 8×8, which fits perfectly into four DMMA instructions. © 2021 IEEE

left). This has the cost of discarding one quarter of the TC's computation in the last tile, but it brings the size to a multiple of 4 in all odd cases. Thus, in both of these cases, this dimension of the DFT matrix always fits in the column dimension of the *B* matrix of the complex DMMA. It also fits in the row dimension when the size is a multiple of 8, i.e. for transform sizes 7, 9, 15, 17, etc., as shown in Figure 2.3 for size 9.

The final dimension of the complex DMMA, 4 or 8 depending on which is occupied by the DFT matrix, signifies the minimum batch size (i.e. the minimum number of signals). The

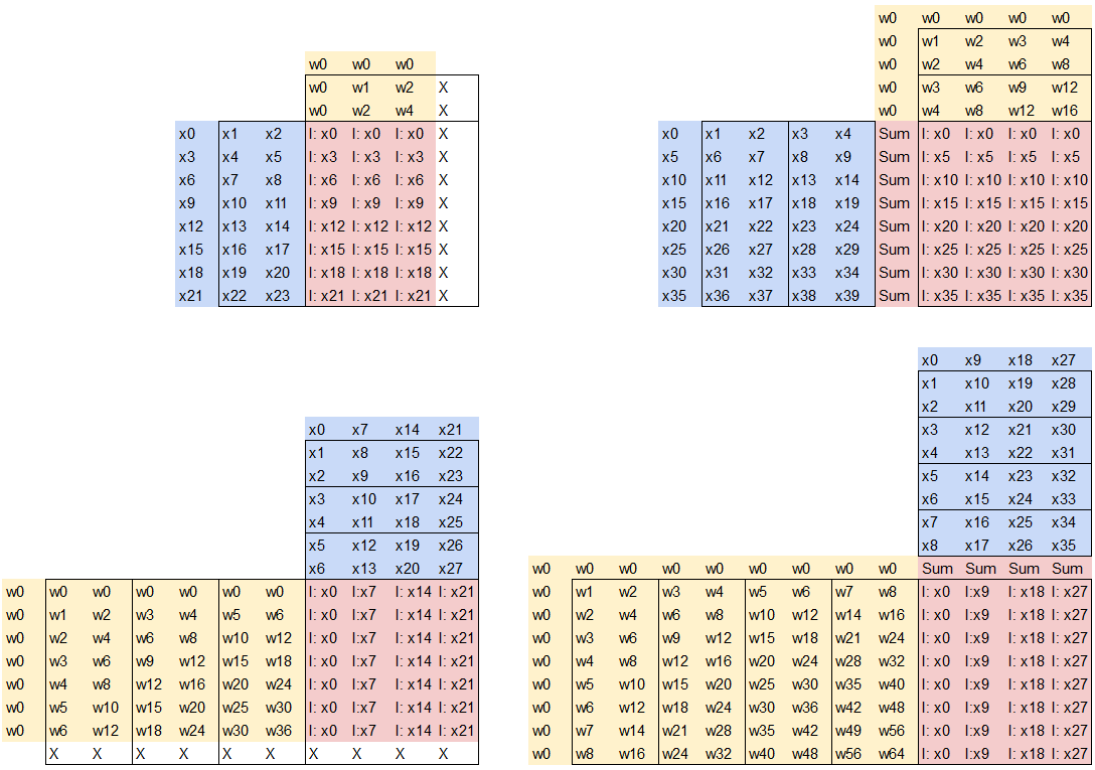

Figure 2.4. The layouts of size 3, 5, 7, and 9 transforms, which take 1, 2, 3, and 4 DMMA instructions respectively. Blue denotes input signals, yellow is DFT matrices, and pink is output signals. Note the initialization in all cases and the extra adds for size 5 and 9. © 2021 IEEE

matrix can be tiled in that direction to accommodate multiples of 4 or 8 signals.

As a result of these operations, for example, two $8 \times 2 \times 4$ complex DMMAs, tiled as 2 inner $\times$ 1 column for a total size of $8 \times 4 \times 4$, would normally accommodate a 4-element DFT of eight signals. However, this layout can be expanded to perform a 5-element DFT of eight signals, or contracted to perform a 3-element DFT of eight signals (using one fewer DMMA) (Figure 2.4 top). Similarly, four complex DMMAs, tiled as 4 inner $\times$ 1 row for a total size of $8 \times 8 \times 8$, would normally perform an 8-element DFT of eight signals, but this layout can be expanded to perform 9-element DFTs or contracted to perform 7-element DFTs (requiring only three complex DMMAs) (Figure 2.4 bottom). With further tiling, any odd-number size can be covered. Because each odd size alternately expands and contracts the effective DMMA size, this approach was nicknamed *"the accordion algorithm"*, after the instrument that similarly expands and contracts.

2.2.3 Epilogue Size 2 or 4 Transforms

With the above approach, transforms of four or eight signals are always computed in parallel by the threads of a warp. Regardless of whether the implementation computes and stores each segment of the output signals before moving on to the next or whether it has the entire batch of output signals in registers at the end of the process, at any given time the kth element of all four or eight output signals is present in registers at once. Furthermore, due to the data layout of the $8 \times 2 \times 4$ complex MMA, each thread holds exactly one complex output signal sample. These conditions are ideal for the addition of an epilogue Fourier transform of size 2 or 4 "across" the signals. This allows the overall Fourier transform size to be that of an odd number times 2 or times 4, on half or one quarter as many signals. (An epilogue transform of size 8 is also possible, but due to the increased complexity of the size 8 coefficients and the incompatibility of this approach with the case of only four signals per DMMA, it was not implemented.)

These epilogue transforms are accomplished as follows:

1. The signals are initially loaded in an interleaved pattern, such that consecutive samples are distributed among what would otherwise be two or four signal slots.

2. The main (odd-size) transform is performed using TCs as described in subsection 2.2.2.

3. The samples are pointwise multiplied by a twiddle matrix, according to the Cooley-Tukey algorithm [1].

4. The size 2 DFT or size 4 FFT is computed by exchanging values among the threads in the warp (with "shuffle" instructions) and performing complex adds using normal floating-point instructions. For both size 2 and size 4, all coefficients are ± 1 or $\pm j$, so only exchanges and adds are needed.

5. The output signals are stored to memory in the same layout as if the epilogue transform was not performed. The size 2 or 4 transform being "across" the signals effectively accomplishes the interleaving needed both before and after that transform.

This process only requires a small number of floating-point add operations and a few shuffles, and introduces no new memory operations. (The twiddle matrix can be preloaded and used for many batches, and the interleaved signal loading represents the same set of loads, just to different threads, which is handled efficiently by the GPU memory subsystem.) As a result, the cost of these epilogue transforms is very low. Conversely, if cuFFT encounters a size which is a prime number times 2 or times 4, in many cases it launches one kernel to perform the prime size and then another to perform the size 2 or 4 transform. The intermediate results must be written out to memory, doubling the needed bandwidth and requiring an additional intermediate data array the same size as the input and output.

2.3 Emulating FP32 with TF32 Tensor Cores

2.3.1 Mixed-Precision Arithmetic

The use of lower-precision arithmetic in computation of higher-precision results is a popular approach to accelerating high-performance computing and linear algebra applications [9] [10]. For example, FP32 computations can be used to initialize iterative solvers which eventually produce FP64 results [11]. The aforementioned paper by Sorna *et. al.* [7] introduced the mixed-precision approach of representing a FP32 value as the sum of two FP16 values, performing the linear FFT operation on the FP16 values, and then adding the results to FP32 at the end. This 2$\times$ FP16 representation provides maximum precision when the first, "big", value is the closest FP16 number to the true FP32 value, and the second "small" value is the closest FP16 representation of the error in the "big" value. However, due to the limited range of FP16 values, this approach required rescaling throughout the computations.

With the Ampere GPU architecture, NVIDIA introduced a new floating-point format for the TCs: TF32 (TensorFloat-32) [12]. This format is stored and processed as IEEE FP32 everywhere except at the input to the multipliers in the TC, where the mantissa is truncated to 10 bits (discarding 13 bits with no rounding). Once the inputs have been reduced to this size,

the multipliers and adders within the TC have no loss of precision, and the result is rounded to FP32 after each multiply-add step [8]. Thus, TF32 can be thought of as a 19-bit floating point format with 1 sign bit, 8-bit exponent, and 10-bit mantissa—the same range as FP32, and the same precision as FP16—which is used just at the input to the TCs. Since true FP32 TCs are not available, and the TF32 TCs have eight times the throughput of traditional FP32 fused multiply-add (FFMA) instructions, emulating FP32 arithmetic using TF32 TCs is an attractive option.

The conversion from FP32 to 2× TF32 is performed as:

1. Round the input value v to TF32 as b, the resulting "big" value.
2. Subtract b from v to get s'.
3. Round s' to TF32 as s, the resulting "small" value.

We follow the FP32-to-TF32 rounding scheme introduced by the open-source CUDA linear algebra library CUTLASS [13], except that we assume input values are finite to save instructions on this critical path. As a result, the rounding is merely an integer add of 0x1000 to the 32-bit register containing the floating-point value, and then a logical-and mask setting the lower 13 bits to zero. (This mask is omitted in the case of s, as the TC itself truncates the value.)

2.3.2 Retaining 21-22 Bits from FP32

While it might seem that the representation of a FP32 value (23 bits of precision) by two TF32 values (each with 10 bits of precision) would produce a total of 20 bits, leading to 3 bits being lost, in fact more precision than that is retained. Consider the case when the floating-point exponent of v is 10, i.e. when the least-significant bit in the TF32 representation (the tenth most significant bit in FP32) represents a value of 1. In this case, b, rounded to this position, has a maximum error of 0.5, since if it had a larger error than that the next largest or next smallest value could be chosen and the error would be reduced. The error $v - b$ is encoded by s, which

therefore must represent a value between -0.5 and 0.5. If just the sign bit of s was provided, that range would be halved again (either $[-0.5, 0]$ or $[0, 0.5]$), meaning one bit of precision is encoded in the sign bit of s. Put another way, the exponent of s is at most -1, which is at least 11 less than the exponent of b, so the mantissa bits of s are shifted by at least 11 bits from v.

However, we need not stop there. Assuming the mantissa bits of v are independent and uniformly distributed, there is a 50% chance $s \in [-0.5, -0.25]$ or $[0.25, 0.5]$, and a 50% chance $s \in [-0.25, 0.25]$. In the latter case, the exponent of s is now at least 12 less than the exponent of b—another bit has been gained. Similarly, there is a 25% chance $|s| < 0.125$ and two extra bits are gained, and so on. These extra bits are being encoded in the difference between the exponent of b and s: in the base case it is -11, but it may be -12 and so on. The expected value E of the number of extra bits can be written as the sum of the probability of each case, times the number of extra bits in each case, or:

$$E = \sum 2^{-(n+1)} n \tag{2.8}$$

If this series is summed to infinity, the result is 1 extra bit on average [14]. In reality, the sum stops at 12, as extra bits are only meaningful when they correspond to bits present in the original FP32 value. However, whenever there are at least 2 extra bits, the 20 mantissa bits between b and s plus the sign bit of s plus the ≥ 2 extra bits gives ≥ 23 bits, which is sufficient to perfectly represent the original FP32 value. So, the situation is actually slightly better than in the infinite-bits case.

In summary, the representation of an FP32 value by two TF32s results in 21 bits of precision deterministically plus about 1 additional bit stochastically under certain assumptions. Note that this discussion is about the precision with which typical FP32 values can be represented; certain very large or very small values cannot be represented by this method. For instance, if the exponent of v is the maximum finite value and the mantissa is at least 0x7FF000, rounding it to TF32 will round up to infinity, causing information to be lost. Similarly, if the exponent of v is so small that s with an exponent 11 less than it would become denormalized, precision is

again lost. However, given the relatively large range of FP32, these cases are rare in practice; furthermore, designers should avoid numbers close to the bounds of the floating-point range in any algorithm due to the risk of overflow and underflow. This algorithm merely requires wider safety margins due to the low-precision representations.

2.3.3 TF32 Data Layout

As discussed above in subsection 2.2.1, the only arithmetic operations needed for Fourier transform computation are complex multiplies and adds, which are expressed in terms of real multiply-adds performed in the TCs and extra real adds performed with normal floating-point operations. Only the multiply portion of the TC operations are done with reduced TF32 precision; the adds in the TCs and the extra adds are all done at FP32 precision. Thus, the only operation which we need to emulate with TF32 precision is a multiply.

If the two values $v_1, v_2 \in \mathbb{F}$ are decomposed as $v_1 \approx b_1 + s_1$, $v_2 \approx b_2 + s_2$, $b_1, b_2, s_1, s_2 \in \mathbb{T}$, where $\mathbb{F}$ and $\mathbb{T}$ are the space of FP32 and TF32 values respectively, and b_i and s_i are "big" and "small" values respectively, then multiplication is performed as:

$$v_1 \times v_2 \approx (b_1 + s_1) \times (b_2 + s_2) = b_1 b_2 + b_1 s_2 + s_1 b_2 + s_1 s_2 \tag{2.9}$$

That is, the product of FP32 values can be approximated as the sum of the four products of the TF32 numbers making up their representations.

For TF32 TC operations, there are two tile sizes available: $16 \times 4 \times 8$ and $16 \times 8 \times 8$. These two forms have the same floating-point throughput (the twice-as-large size is issued half as often), but the smaller size has greater pressure on register file communication, which can become a bottleneck in some cases. To avoid this, we use the $16 \times 8 \times 8$ size. This tile size is twice as large in the "rows" and "inner" dimensions compared to DMMA, so writing the

$16 \times 8 \times 8$ operation in terms of $8 \times 4 \times 8$ tiles looks like:

$$\begin{bmatrix} A_0 & A_2 \\ A_1 & A_3 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \end{bmatrix} + \begin{bmatrix} C_0 \\ C_1 \end{bmatrix} = \begin{bmatrix} D_0 \\ D_1 \end{bmatrix} \tag{2.10}$$

where $A \in \mathbb{T}^{4\times8}$, $B \in \mathbb{T}^{8\times4}$, $C, D \in \mathbb{F}^{8\times8}$, i.e. A, B, C, D are tiles the same size as in the DMMA instruction.

Similar to how we mapped complex numbers to groups of values within the DMMA layout in subsection 2.2.1, we map these subtiles to the decomposed values in the mixed-precision representation:

$$\begin{bmatrix} b_1 & b_1 \\ s_1 & s_1 \end{bmatrix} \begin{bmatrix} b_2 \\ s_2 \end{bmatrix} = \begin{bmatrix} b_1 b_2 + b_1 s_2 \\ s_1 b_2 + s_1 s_2 \end{bmatrix} \tag{2.11}$$

An extra addition step at the end is needed to add the D_0 and D_1 values to give $v_1 \times v_2 \approx b_1 b_2 + b_1 s_2 + s_1 b_2 + s_1 s_2$; this requires two floating-point add instructions, as each thread holds two values each from D_0 and D_1.

Thus, the process for emulating an FP32 TC operation is:

1. Convert the input A and B values to $2\times$ TF32 as described in subsection 2.3.1.

2. Perform one $16 \times 8 \times 8$ TF32 TC instruction with the big and small values laid out as shown in Equation 2.11.

3. Perform two extra floating-point adds per thread.

This results in a pseudo-"FMMA" (FP32 matrix multiply-accumulate) with the same tile size and data layout as the DMMA instruction. Crucially, this means that the complex number format described in subsection 2.2.1—and therefore the algorithms dependent on the $8 \times 2 \times 4$ complex MMA tile size in subsection 2.2.2—remain the same, as does the code for loading the input data and so on. Furthermore, since the TF32 TC operation has eight times the throughput of normal FP32 multiply-add instructions, but four multiplies within this instruction are needed to emulate

each FP32, the overall throughput is up to twice that of normal FP32 operations. The extra add instructions and the instructions needed for the conversion are performed by floating-point and integer hardware units, which run in parallel with the TCs, so their presence may or may not reduce the overall throughput. The results in section 2.5 indicate slightly lower performance gains for emulated FP32 as compared to FP64, but this may be for a variety of reasons besides the conversion, so it is not clear the conversion is to blame. Furthermore, as discussed in section 2.4, some of our implementations perform the conversions less often than indicated in this section, reducing their potential performance impact.

2.3.4 Recovering Additional Precision with Separated Accumulation

There remains one problem with this pseudo-FMMA operation: one to two bits are lost in the conversion from FP32 to 2× TF32. If one of the inputs is the identity matrix, for instance, it is clear that the lost bits in the other input translate to the same lost bits at the output. When both matrices have lost bits, such as in the Fourier transform case where one matrix contains DFT coefficients and the other contains arbitrary input data, the error in the product can be another one to two bits larger.

However, floating-point error accumulates in any floating-point computations, not just those with TF32. Each Fourier transform output sample y_k can be viewed as the dot product of a row of the DFT matrix W with the input signal $\vec{x}$. The length of this dot product is the transform size N. If this dot product is implemented as a loop over the pairs of elements, with fused multiply-add instructions used for each step, the results are still rounded to FP32 after each fused multiply-add. Thus, assuming the input signals are on average uniformly distributed, the sum gets larger and larger each step, while the product remains the same size. For every power of two the sum becomes larger than the product by, one bit from the product is effectively lost. So, even with a straightforward FP32 DFT implementation, two or three bits will be lost in most of the products for any transform sizes larger than 8 or 16.

This type of error does not add to the error due to the 2× TF32 representation; instead, it

obscures that error. Suppose the "add" input is four times the size of the product, so that two bits of the product are effectively lost; and suppose the last two bits of the product are incorrect due to the TF32 representation. It is those incorrect bits which will be lost due to the adding; with the exception of effects due to rounding, the result will have the same accuracy as if the product was completely correct. To maximize the chance that these rounding effects are favorable, and to halve the accuracy penalty for the dot product accumulation, we employ the following strategy.

All TC operations are matrix multiply-accumulate operations, $D = AB + C$. However, the operation is not performed as written, with a multiply first and then an add. Instead, each element of D is computed as the dot product of a row of A with a column of B, with fused multiply-add operations used for each of the elements of that dot product. The corresponding element of C is used as the "add" input to the first fused multiply-add. Put another way, C contains the initial state of the accumulator, which is repeatedly added to. When performing multiple TC operations to produce longer dot products, this corresponds to the situation described above, where precision is lost in the products when the "add" input value is larger.

To improve accuracy, we instead use the TC to perform the operation $D = AB + \mathbf{0}$, and add C to D afterwards with normal floating-point instructions (two instructions per thread, as each thread holds two values in these matrices). This improves accuracy in two ways. First, since the $8 \times 2 \times 4$ complex matrix multiply tile being performed as a result of this whole process has an inner dimension of 2, that means we are grouping the overall dot product sum in pairs. Each pair of values is added independently, and the results of these pairs are added serially, for half the number of serial additions and hence roughly half the error (again with appropriate assumptions about uniformity of inputs). Second, the sums within the TC are adding "big$\times$big", "big$\times$small", and "small$\times$small" values, which are all at different size scales. For the rounding to be most accurate, these values should be added in order from smallest to largest. While the order of the adds within the TC cannot be guaranteed, we can at least guarantee that the addition to the overall dot product accumulator—which is most likely the largest value of them all—is done last. As a result, the intermediate rounding is more favorable with this approach than if we had used

the C input to the TC operation.

2.4 Implementations

Several implementations with a variety of hyperparameter settings were tested. We present the two implementations with the best overall performance for both FP64 and emulated FP32.

In all our implementations, due to the TCs' warp-wide nature, each warp performs the transform of a batch of signals at once. The total number of signals being transformed must be a multiple of the batch size. However, the strategies for loading the DFT coefficients and signal data, and the organization of the warps, differ between the implementations.

2.4.1 Cache- and Register-Based Implementation

Our first implementation closely follows the design described in subsection 2.2.2. The DFT matrix and signals swap positions between the A and B matrices of the TC after every two odd sizes, and there is an optional batch size multiplier for how many signals are processed at once (i.e. the original batch size is 4 or 8 depending on transform size, but this can be optionally multiplied by 2 or 4 for a total batch size b). Besides the $\times 2$ and $\times 4$ epilogue transforms, we implement a two-way factorized transform: composite odd FFT sizes are broken once into a pair of factors p and q, as close in magnitude to each other as possible. The $b \times q$ size-p transforms and the $b \times p$ size-q transforms are performed sequentially by the same kernel, with intermediate results stored in shared memory.

For small transform sizes (up to around 33), the entire DFT matrix is loaded into registers at the beginning, and then each warp sequentially processes many batches of transforms without having to load any DFT coefficients again. For larger transform sizes where there are not enough registers, instead DFT coefficients are loaded from memory as they are used. This approach relies on the L1 cache to keep the DFT coefficients close to the execution units.

2.4.2 Streaming- and Shared Memory-Based Implementation

Our second implementation is simpler in its use of the TCs: a batch of 8 signals is always in the A matrix, and the DFT coefficients are always in the B matrix. The $\times 2$ and $\times 4$ epilogue transforms are implemented, but the two-way factorized transforms are not, due to complexity and shared memory use.

The DFT matrix is loaded to shared memory, and for FP32 also converted to $2\times$ TF32, at the beginning of the kernel. For DFT sizes that are too large to fit in shared memory (even with the extended shared memory up to 160 KiB available in the A100), the transforms are split up among multiple thread blocks, where each computes a subset of the output samples (and therefore requires only a portion of the DFT matrix). Multiple warps are run per block, which share the same DFT matrix in shared memory, and these warps iterate over the signal batches to further increase the reuse of the DFT coefficients. Finally, signals are loaded either normally from global memory via the L1 cache, or using the new pipelined asynchronous memory copies introduced in Ampere [15]. As the latter copies large chunks of data directly from the L2 cache to shared memory via a dedicated, high-bandwidth pipeline, it is typically faster for sizes where it is used. However, it also consumes a large shared memory footprint when many warps are running per block, so it is only an efficient approach for smaller sizes.

2.5 Results

All results are measured on one NVIDIA Ampere A100-PCIE-40GB (250W) GPU. The GPU graphics clocks are locked to 1410 MHz while the memory clocks are locked to 1215 MHz.

2.5.1 Numerical Accuracy

All approaches are compared to ground truth results computed on the CPU in FP128 (quad) precision and then rounded to the target datatype. In each case, we compute the root-mean-square (RMS) error between the output of the algorithm in question and the ground truth.

Relative bit errors are computed as $\varepsilon = \log_2(\mathrm{RMS}_{\mathrm{TC}}/\mathrm{RMS}_{\mathrm{cuFFT}})$, where $\mathrm{RMS}_{\mathrm{TC}}$ and $\mathrm{RMS}_{\mathrm{cuFFT}}$ are the error in the present TC implementation and in cuFFT respectively. Except where stated below, signal samples are independent, uniformly distributed random floating-point numbers in the range $[-0.5, 0.5)$. The reference algorithms used for comparison are: cuFFT [16] version 10.3 (CUDA toolkit 11.1), FFTW [17] version 3.4, and two basic DFT implementations in the target datatype on the CPU: one using DFT coefficients computed in the target precision, and one using DFT coefficients computed in quad precision and rounded to the target precision. CPU math for the reference computations is performed on the SSE path only (not x87) with all compiler optimizations that may affect floating-point precision disabled.

The accuracy results do not differ significantly between the implementations described in section 2.4, with two exceptions. First, a factorized FFT typically has lower error than a DFT of the same size. Because cuFFT uses two-way or three-way factorized transforms whenever possible, for fair comparison we use the two-way TC implementation for those sizes. Second, by default we use separated accumulation (subsection 2.3.4) for both FP64 and emulated FP32; we show an example below (Figure 2.6) with it disabled to demonstrate the accuracy impact.

FP64 precision

Figure 2.5 shows FP64 accuracy results for the present TC implementation and the four reference implementations discussed above. For the uniformly distributed input data pictured, the present implementation matches the accuracy of cuFFT, beating it by 0.07 bits. For the two other types of input data described in the caption of Figure 2.8, the average differences are 0.10 and 0.02 bits again in slight favor of the present implementation (not pictured). However, as discussed above, the DFT has higher (and more consistent) error than the factorized transforms, and separated accumulation has a substantial impact on the error (Figure 2.6).

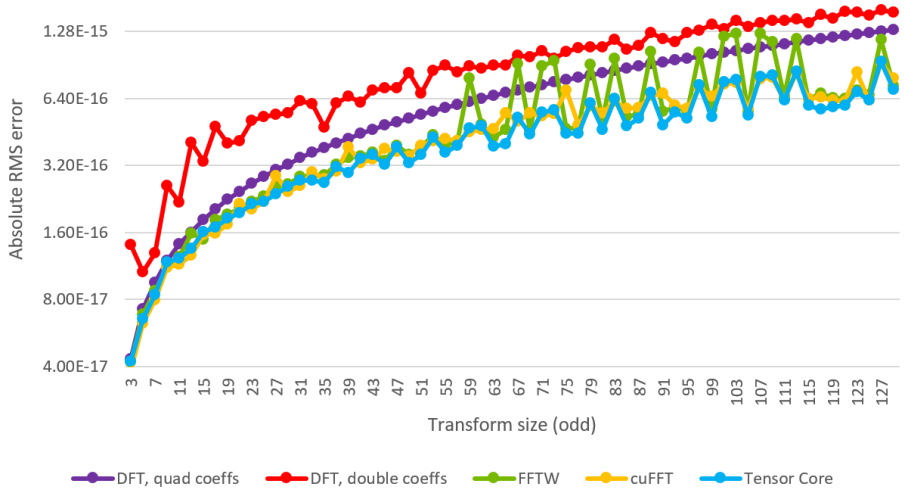

Figure 2.5. FP64 numerical accuracy of five Fourier transform implementations for uniformly distributed input data. The present TC implementation slightly exceeds the accuracy of cuFFT, by an average of 0.07 bits. © 2021 IEEE

Emulated FP32 precision

As a result of the techniques described in section 2.3, most of the precision of FP32 is retained when emulated by 2× TF32 with TCs. The precision is only degraded by an average of 0.26 bits for uniformly distributed data (Figure 2.7) and 0.31 and 0.33 bits for other data distributions (Figure 2.8). Note also that the present implementation matches or exceeds the accuracy (except for certain small sizes) of a CPU-based DFT implementation using high-precision DFT coefficients, and substantially exceeds the accuracy of a purely-FP32 DFT implementation.

2.5.2 Performance

We compare the performance of the present TC implementations to cuFFT and to a memory copy of the input data to the output. For each FFT size N, we measure the execution time for 32 Mi input signal samples rounded up to the batch size, or $S = b\left\lceil \frac{32\times1024\times1024}{b\times N} \right\rceil$ signals. Execution time is measured with CUDA events and averaged over 100 runs in each case. Caches are not forcibly cleared between runs for any of the implementations (including cuFFT and memcpy); due to the repeated runs, all measurements are effectively in "hot cache" state.

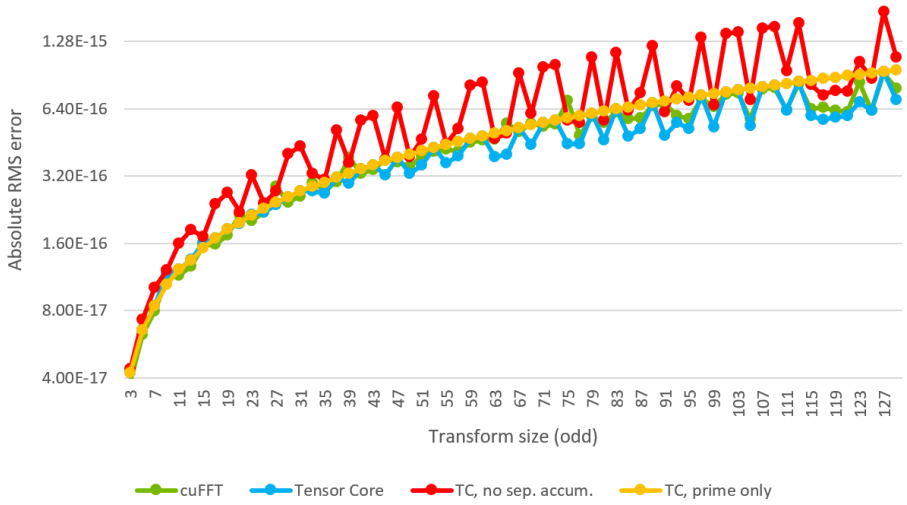

Figure 2.6. (FP64) Both the present TC implementation and cuFFT have lower error for composite sizes where two-way or three-way factorized transforms are used (compare blue and green to yellow). When separated accumulation is disabled, the error in the present implementation rises (red). © 2021 IEEE

Execution times are converted into equivalent floating-point operations per second (equivalent FLOPS) with the simplified definition of $5N\log_2 N$ FLOPS needed for an FFT of size N, as:

$$f = \frac{S \times 5N\log_2 N}{t \times 10^6} \tag{2.12}$$

where f is in equivalent GFLOPS and t is in milliseconds. Note that this metric does not represent the actual number of floating-point operations performed in any of the implementations, and it is even applied to memcpy which of course does not perform any floating-point operations at all.

Performance is measured for various TC implementations with different settings at each transform size, and the best result per-size is presented. However, settings regarding accuracy remain the same as presented in subsection 2.5.1 (except when otherwise specified).

Compared to memcpy

Figure 2.9 shows the absolute performance of the fastest TC implementation and cuFFT compared to memory copies for FP64 and FP32. The TC implementation exceeds the performance of cuFFT in a similar set of sizes in both cases. This appears to be for two main

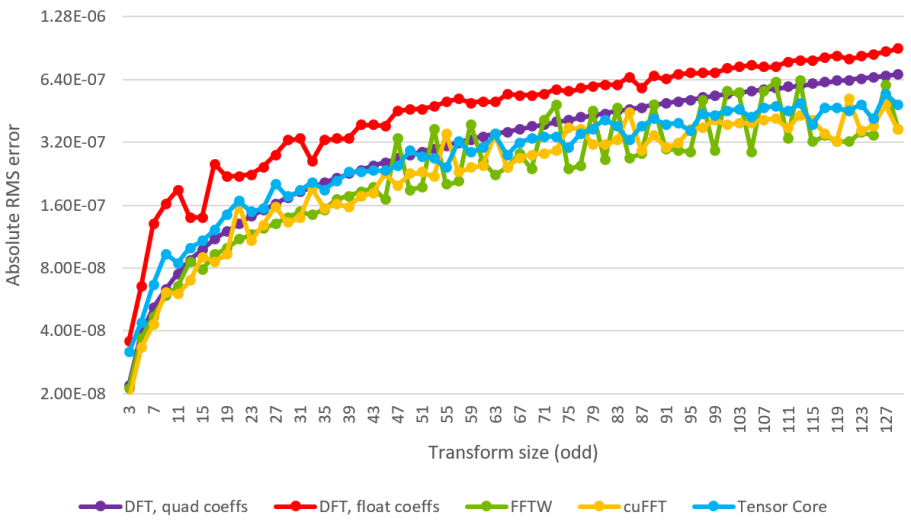

Figure 2.7. Numerical accuracy of five Fourier transform implementations, for FP32 precision computation and with uniformly distributed input data. In this case, the present TC implementation using emulated FP32 with TF32 (blue) is on average only 0.26 bits worse than cuFFT (green). It also meets or exceeds the precision of a basic DFT implementation with ideal DFT coefficients (purple) for all but small sizes. © 2021 IEEE

reasons. First, prime sizes are computed with the DFT by both the present implementation and cuFFT, which is a highly compute-dependent problem. Both the FP64 TCs and the emulated "pseudo-FMMA" (subsection 2.3.3) have twice the throughput on A100 compared to normal arithmetic, so these cases can be accelerated up to 2× or until some other bottleneck sets in. Second, as is visible in Figure 2.9, cuFFT runs at about half the performance of memcpy for a number of sizes. These are presumably the sizes where it runs two kernels for a factorized transform, and therefore requires twice the memory traffic. In contrast, the TC implementations are all single kernel, and therefore do not have this bottleneck.

Compared to cuFFT

Figure 2.10 shows the relative performance of the present TC implementation compared to cuFFT. The fastest TC implementation is at least 10% faster than cuFFT in 49% of the supported Fourier transform sizes for FP64 and 42% of these sizes for emulated FP32. For FP64, gains reach and exceed 2× in a few cases; for emulated FP32, gains even reach 2.5×.

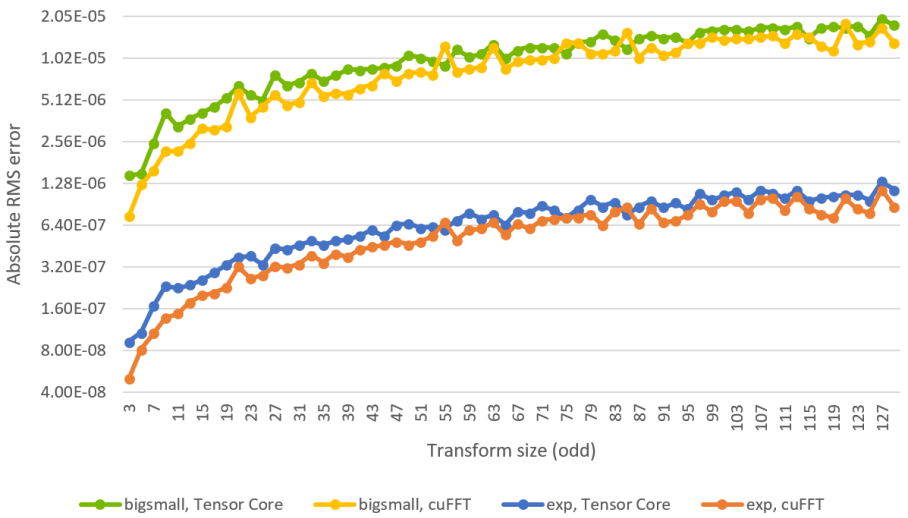

Figure 2.8. Average absolute RMS error in cuFFT and the present TC implementation, for two other types of IID input data. Top ("bigsmall"): 1 in 8 chance of uniform distribution $[-50, 50)$, otherwise uniform distribution $[-0.5, 0.5)$. Bottom ("exp"): Exponential distribution $p(x) = 2e^{-2x}$, $x > 0$. The average degredation of the emulated FP32 results compared to cuFFT is only 0.31 and 0.33 bits respectively for these two cases. © 2021 IEEE

Over batch size

Figure 2.11 shows performance results as a function of the batch size (i.e. number of transforms), from 32 to as many as will fit in global memory, for fixed transform size 11. The sharp drop in the middle is where the input and output data no longer fit entirely within the L2 cache. Note that the relative performance of the different implementations is stable over larger batch sizes.

Impact of separated accumulation

When separated accumulation is disabled, as shown above in Figure 2.6, accuracy is reduced. However, some applications may not be sensitive to this, and prefer to disable separated accumulation to increase performance. For FP64, the average performance gain for the fastest TC implementation per-size when disabling separated accumulation is 10.9%. In this configuration, the fastest TC implementations beat cuFFT by at least 10% in 59% of supported sizes, up from 49% with separated accumulation enabled. However, disabling separated accumulation for

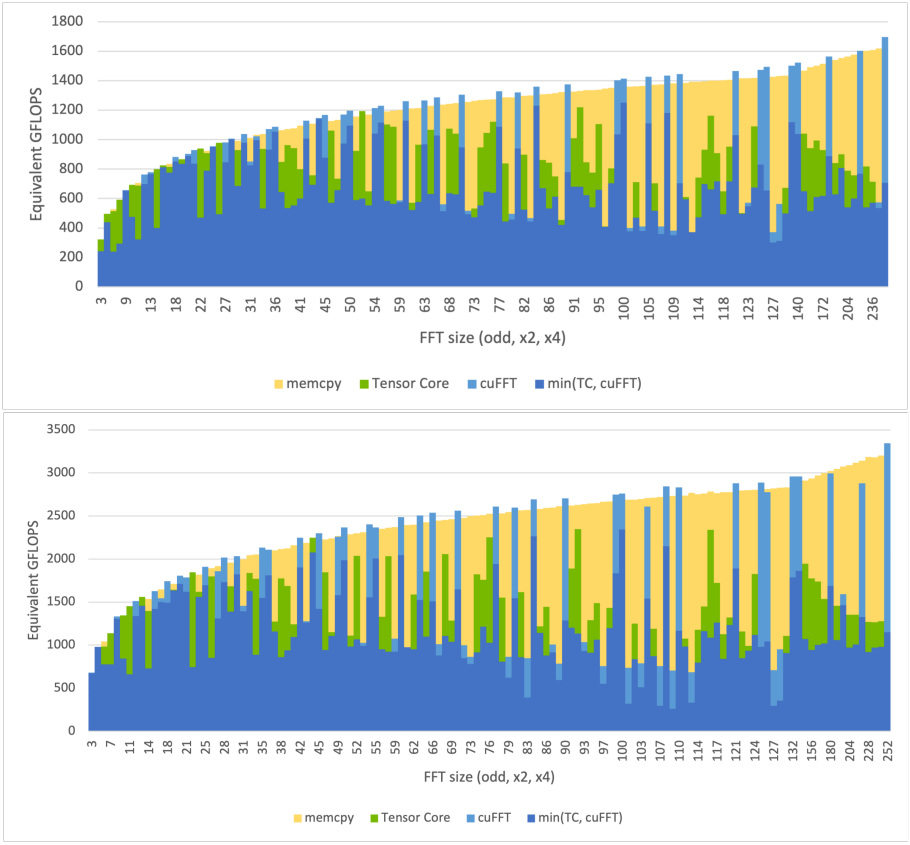

Figure 2.9. FP64 (top) and FP32 (bottom) performance of cuFFT and the fastest TC implementations compared to the ideal baseline of memory copies. The visible green is where the TC implementation exceeds the performance of cuFFT. The visible light blue is where cuFFT exceeds the performance of the TC implementation. The fact that the FFT implementations sometimes slightly exceed the performance of memcpy is not a misprint. © 2021 IEEE

emulated FP32 precision provides only 1.7% extra performance on average, and does not change the set of sizes which exceed the performance of cuFFT.

2.6 Conclusion

We have introduced a set of algorithms for computing discrete Fourier transforms on Tensor Core GPU hardware. Our approach has the same or nearly the same accuracy as the state-of-the-art cuFFT and FFTW libraries, and exceeds the performance of cuFFT by at least 10% on 49% and 42% of supported sizes for FP64 and FP32 respectively. The supported sizes are odd numbers or odd numbers times 2 or times 4, which includes prime numbers

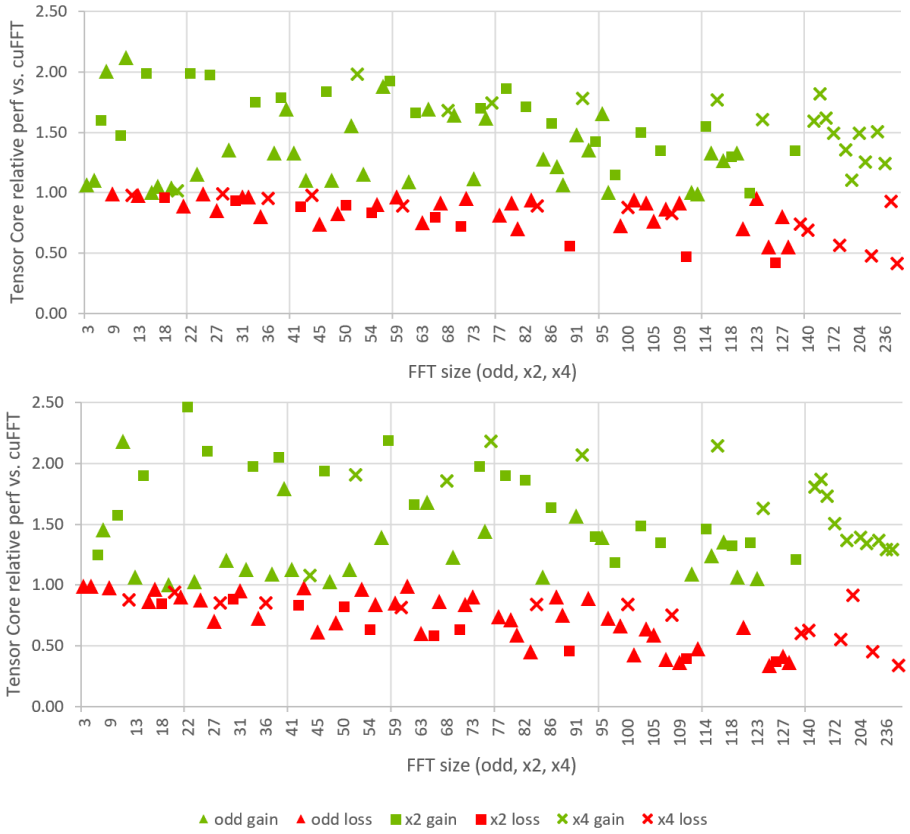

Figure 2.10. FP64 (top) and FP32 (bottom) performance of the fastest TC implementation relative to cuFFT, with gains in green and losses in red. The TC implementation exceeds the performance of cuFFT by at least 10% in 49% (FP64, top) and 42% (FP32, bottom) of these cases (odd, $\times 2$, and $\times 4$ sizes). © 2021 IEEE

and numbers with large prime factors. The computation of Fourier transforms of these sizes is more compute-dependent than that of powers of 2 or other sizes with small prime factors, so existing implementations are often slower. The insights behind these algorithms include methods for mapping complex numbers and odd sizes to the Tensor Cores, and strategies for using lower-precision TF32 Tensor Cores to emulate FP32 arithmetic.

The implementations described in this paper are expected to be integrated into a future release of cuFFT. We also plan to extend the TC approach to cover additional FFT sizes and types in the future.

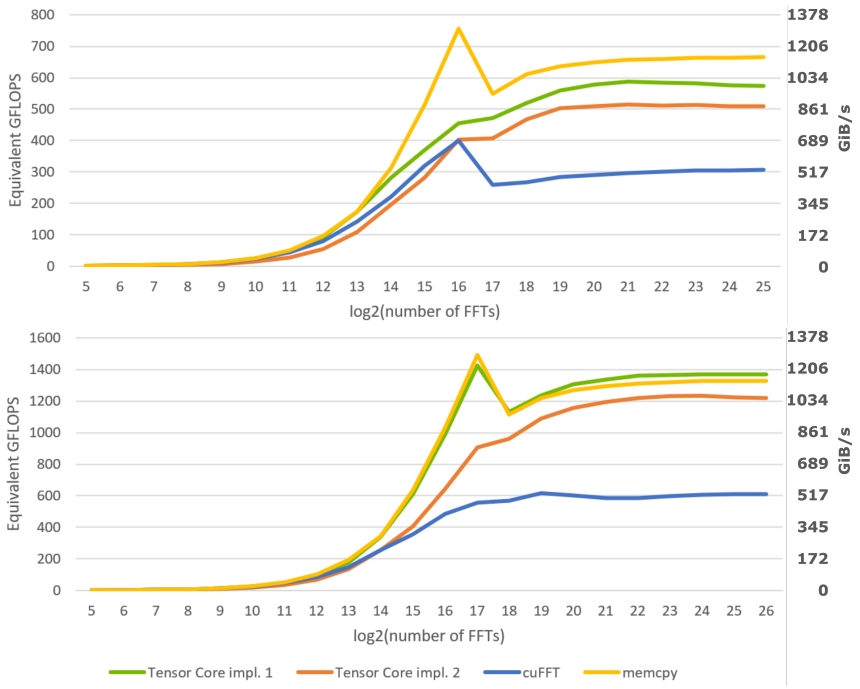

Figure 2.11. FP64 (top) and FP32 (bottom) performance versus batch size, for fixed size 11. © 2021 IEEE

Acknowledgments

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of UC San Diego's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Chapter 2, in full, is a reprint of the material as it appears in the 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Pisha, Louis; Ligowski, Łukasz, IEEE, May 2021. The dissertation author was the primary investigator and author of this paper.

Bibliography

[1] James W Cooley and John W Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[2] Leo Bluestein. "A linear filtering approach to the computation of discrete Fourier transform". In: *IEEE Transactions on Audio and Electroacoustics* 18.4 (1970), pp. 451–455.

[3] *NVIDIA Tesla V100 GPU accelerator*. Tech. rep. NVIDIA Corporation, 2017. URL: https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf.

[4] *NVIDIA T4 tensor core GPU*. Tech. rep. NVIDIA Corporation, 2018. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf.

[5] *NVIDIA A100 tensor core GPU architecture*. Tech. rep. NVIDIA Corporation, 2020. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf.

[6] Intel Corporation. *Architecture Day 2020*. Aug. 2020. URL: https://newsroom.intel.com/press-kits/architecture-day-2020/.

[7] Anumeena Sorna, Xiaohe Cheng, Eduardo D'azevedo, Kwai Won, and Stanimire Tomov. "Optimizing the fast fourier transform using mixed precision on tensor core hardware". In: *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*. IEEE. 2018, pp. 3–7.

[8] *CUDA Parallel Thread Execution ISA Version 8.0: Warp Level Matrix Multiply-Accumulate Instructions*. NVIDIA Corporation. URL: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#warp-level-matrix-instructions.

[9] Ahmad Abdelfattah, Hartwig Anzt, Erik G Boman, Erin Carson, Terry Cojean, Jack Dongarra, Mark Gates, Thomas Grützmacher, Nicholas J Higham, Sherry Li, et al. "A Survey of Numerical Methods Utilizing Mixed Precision Arithmetic". In: *arXiv preprint arXiv:2007.06674* (2020).

[10] Geetika Gupta. *What's the Difference Between Single-, Double-, Multi- and Mixed-Precision Computing?* Tech. rep. NVIDIA Corporation, Nov. 2019. URL: https://blogs.nvidia.com/blog/2019/11/15/whats-the-difference-between-single-double-multi-and-mixed-precision-computing/.

[11] Alfredo Buttari, Jack Dongarra, Julie Langou, Julien Langou, Piotr Luszczek, and Jakub Kurzak. “Mixed precision iterative refinement techniques for the solution of dense linear systems”. In: *The International Journal of High Performance Computing Applications* 21.4 (2007), pp. 457–466.

[12] Paresh Kharya and NVIDIA Corporation. *TensorFloat-32 in the A100 GPU accelerates AI training, HPC up to 20x*. May 2020. URL: https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/.

[13] NVIDIA Corporation. *CUTLASS: CUDA Templates for Linear Algebra Subroutines*. 2020. URL: https://github.com/NVIDIA/cutlass.

[14] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd. USA: Addison-Wesley Longman Publishing Co., Inc., 1994. ISBN: 0201558025.

[15] *Asynchronously Copy Data from Global to Shared Memory*. NVIDIA Corporation. 2020. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#async-copy.

[16] NVIDIA Corporation. *cuFFT: Fast Fourier Transforms for NVIDIA GPUs*. 2020. URL: https://developer.nvidia.com/cufft.

[17] Matteo Frigo and Steven G Johnson. “The design and implementation of FFTW3”. In: *Proceedings of the IEEE* 93.2 (2005), pp. 216–231.

Chapter 3

Approximate Diffraction Modeling for Real-Time Sound Propagation Simulation

Reprinted from the Journal of the Acoustical Society of America (JASA) 148 (4), pp. 1922-1933 (2020), DOI 10.1121/10.0002115

3.0 Abstract

Convincing simulation of diffraction around obstacles is critical in modeling sound propagation in virtual environments. Due to the computational complexity of large-scale wave-field simulations, ray-based models of diffraction are used in real-time interactive multimedia applications. Among popular diffraction models, the Biot-Tolstoy-Medwin (BTM) edge diffraction model is the most accurate, but it suffers from high computational complexity and hence is difficult to apply in real time. This paper introduces an alternative ray-based approach to approximating diffraction, called *Volumetric Diffraction and Transmission* (*VDaT*). VDaT is a volumetric diffraction model, meaning it performs spatial sampling of paths along which sound can traverse the scene around obstacles. VDaT uses the spatial sampling results to estimate the BTM edge-diffraction amplitude response and path length, with a much lower computational cost than computing BTM directly. On average, VDaT matches BTM results within 1–3 dB over a wide range of size scales and frequencies in basic cases, and VDaT can handle small objects and gaps better than comparable state-of-the-art real-time diffraction implementations.

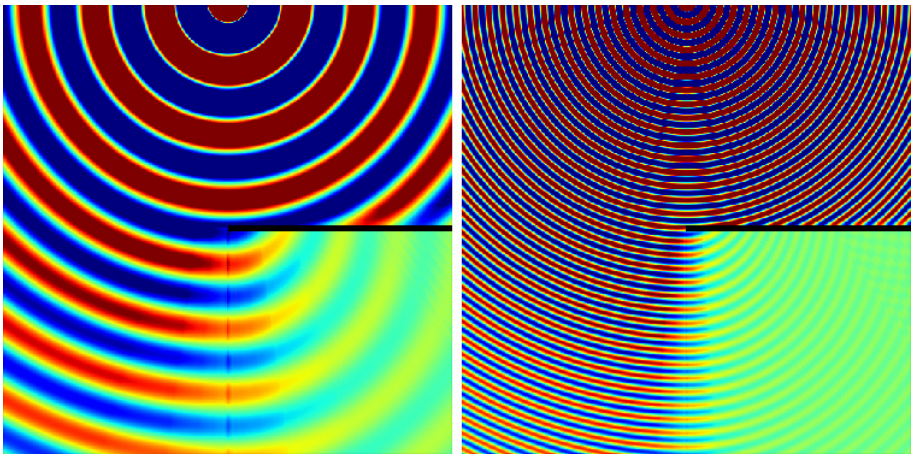

Figure 3.1. Real part of the VDaT-simulated soundfield showing diffraction around a large planar occluder (black line), at ~ 600 and ~ 2850 Hz respectively. For clarity, reflections from the occluder are disabled. Note how the diffracted waves emanate from the diffracting edge, despite the VDaT algorithm involving neither waves nor edges.

A GPU-parallelized implementation of VDaT is shown to be capable of simulating diffraction on thousands of direct and specular reflection path segments in small-to-medium-size scenes, within strict real-time constraints and without any precomputed scene information.

3.1 Introduction

Diffraction of sound is a readily perceptible phenomenon in any environment that includes objects which can occlude sound. As such, it is a key element of any acoustical simulation. Diffraction can be modeled more-or-less exactly in a large-scale wavefield simulation of the acoustical space [1, 2]. However, since human perception of sound spans roughly 10 octaves, and the shortest wavelengths of interest (about 2 cm) are orders of magnitude smaller than the scale of typical environments (2-20 m or possibly more), wavefield simulations require a very large number of points and therefore a vast amount of computing power. As a result, these methods are typically unsuitable for real-time applications.

Instead, we consider methods from geometrical acoustics (GA), in which the propagation of sound is simulated by ray tracing techniques, and effects such as reflection and diffraction are modeled through transformation of these rays. GA methods are efficient and have been

used in real-time audio systems for decades, from experimental systems in the '90s [3] to major commercial software packages today [4–6]. The two most popular GA-based diffraction models are the Uniform Theory of Diffraction (UTD) [7] and the Biot-Tolstoy-Medwin model (BTM) [8–10]. UTD is derived from the leading terms of an expansion of the wavefield result for an infinite wedge [7]; it is reasonably fast to compute for each diffraction path, and it can be evaluated at a coarse set of frequencies to approximate the amplitude response with reduced computation. However, it has some error at low frequencies when the source or receiver are close to the edge [10, 11], and even more error is introduced when it is used on practical scenes with small edges, violating its assumption of infinite edges (subsection 3.4.2).

BTM is a theoretically superior model which handles finite edges correctly. It has been shown to satisfy the wave equation for the infinite wedge, and it is conjectured to do so for all scenes if diffraction is simulated to infinite order [10]. While computation of the discrete-time impulse response for BTM [10, 12, 13] involves finite sample rates and numerical integration, the sample rate and integration quality can be raised arbitrarily to (presumably) approximate the wavefield result as closely as desired. However, due to the numerical integration, BTM suffers from high computational complexity even with the minimum parameter settings, so its utility in real-time applications has been limited to small-scale examples [14].

Both UTD and BTM are edge diffraction models: they model the filtering of sound on a path that goes around one or more edges of objects in the scene. Exhaustively considering all sets of edges for diffraction is of polynomial complexity in the number of edges:

$$C_{\text{edge}} \propto \sum_{o=1}^{N_o} [s \cdot \eta^o \cdot ((o+1)I(t) + C_D(o))] \tag{3.1}$$

where N_o is the maximum order, s is the number of source-receiver pairs (assuming no reflections), η is the number of edges (typically about $(3/2)t$), t is the number of triangles, $I(t)$ is the cost of determining whether a given segment intersects any triangle in the scene or not ($O(t)$, using hierarchical structures $O(\log t)$ or better [15]), and $C_D(o)$ is the cost of the o-order diffraction

computation itself. Assuming that higher-order diffraction has the same computational cost as repeated first-order diffraction, and dropping smaller terms, the complexity is at least

$$C_{\text{edge}}^{\text{approx}} > s \cdot t^{N_o} \cdot N_o \cdot (\log t + C_D(N_o)) \tag{3.2}$$

As a result, this approach cannot be used in real-time simulation of scenes of nontrivial complexity. Published approaches to circumventing this complexity (subsection 3.2.1) include pre-computing edge visibility information [16], which restricts live changes to the scene, and Monte Carlo sampling techniques [17], which raises questions about consistency of quality. In the course of creating our own real-time GA audio spatialization system, we needed a diffraction model that was suitable for fully dynamic, real-time scenes, while being as accurate as possible over the wide range of scenes that are likely to be encountered in interactive multimedia applications.

Thus, as an alternative to edge diffraction, we have experimented with volumetric diffraction modeling—a ray-based sampling of the empty space around obstacles, relative to the occluded direct path or path segment through the obstacles. This approach has significant advantages: its computational complexity is largely decoupled from the scene complexity (subsection 3.5.1); it can simulate non-shadowed diffraction, where the direct path is not occluded, with little additional cost (subsection 3.4.3); and it natively incorporates modeling of sound transmission through obstacles, including on higher-order reflection and diffraction path segments (subsection 3.3.3). The challenge was to use this volumetric sampling approach to create reasonably accurate diffraction amplitude response and path length results over a wide range of scene configurations. With a combination of theoretical analysis, numerical simulation, and heuristic experimentation, we have developed a volumetric diffraction model, called *VDaT (Volumetric Diffraction and Transmission)*, which approximates the behavior of the BTM model but has a much lower computational complexity. In typical scene configurations, there is a small reduction in accuracy, as a trade-off for the large reduction in computation. However, like BTM, VDaT does not exhibit the errors of UTD for small edge lengths (subsection 3.4.2), and in certain

cases VDaT can produce results which are objectively superior to those of comparable real-time implementations of UTD or BTM (subsection 3.4.3). Thus, we present VDaT as an alternative method for approximating diffraction in acoustical simulations, and argue that it has substantial advantages over the existing models in many real-time applications.

The remainder of this paper is organized as follows. In section 3.2, we review existing approaches to diffraction modeling, with a focus on approaches suitable for real-time applications. In section 3.3, we derive VDaT and discuss how its spatial sampling results are used to approximate BTM. In section 3.4, we present simulated results for many scene configurations at several size scales. Finally, in section 3.5, we discuss the computational complexity and real-time performance of VDaT as integrated into a real-time audio spatialization system.

3.2 Past Work

3.2.1 Reducing Complexity of Edge Diffraction

Pre-computing edge visibility; computing diffraction separately from reflections

Reference [16] precomputes which edges in the scene are visible to each other, i.e. not occluded by other objects, and stores this information in a graph structure. Traversing this graph at run time substantially reduces the number of sets of edges which need to be considered for higher-order diffraction. However, any precomputation on the scene requires the precomputed elements—the diffracting edges—to be in fixed relative positions at runtime. Reference [16] partly avoids this problem by separately computing edge visibility for objects which are internally static but may move relative to each other, such as buildings and vehicles; unfortunately, this approach omits any diffraction paths which would involve more than one of these objects. Reference [16] also processes diffraction paths separately from specular reflection paths, to reduce the number of path segments needing diffraction computation. Of course, not allowing reflection paths to experience diffraction means these reflection paths cut in and out as the scene elements move, often causing audible discontinuities in the output.

Monte Carlo or beam tracing

Reference [17] traces rays through the scene from the receiver on Monte Carlo trajectories, computing UTD edge diffraction around the triangles they intersect. This approach successfully decouples the computational complexity from the scene complexity, and simulates diffraction on higher-order reflection paths. However, because Monte Carlo does not guarantee that important reflection or diffraction paths are found, there may be quality issues. Reference [17] ameliorates this problem by introducing a caching scheme, which allows the ray tracing complexity to be effectively amortized over many frames, improving the quality of long, slow-moving sounds every frame. Reference [18] performs a similar technique, except tracing adaptively-subdivided frusta (polygonal convex conical beams) through the scene instead of individual rays. This is a promising general approach, as it retains the advantages of the above approach while eliminating the quality issues. However, the reported performance was barely real-time on the simplest scenes, due to the higher complexity of the frustum computation.

Culling low-amplitude diffraction paths

Reference [19] presents a procedure for culling diffraction paths which are likely to be low in amplitude and hence only make small contributions to the overall output. This approach appears to be successful at reducing the computational burden of simulating audio along insignificant paths, but it does not reduce the complexity of generating the diffraction paths in the first place.

3.2.2 Other Non-Edge Diffraction Models

Fresnel zones

Fresnel zones are ellipsoidal, wavelength-dependent spatial volumes around the direct path which represent the region in which most of the sound at each frequency propagates. As an early real-time diffraction system, Reference [20] rasterizes the triangles of occluding objects from a camera behind the source, to approximate what portion of each Fresnel zone around the path segment is blocked. Reference [21] uses Fresnel zones in reflection computations, plus

a basic approximate diffraction attenuation factor, for estimating environmental noise in large, outdoor scenes.

Neural networks

Reference [22] trains a neural network to estimate filter parameters which approximate the edge-diffraction results for a basic occluding object. The results are reasonably accurate and are shown to be perceptually acceptable to listeners; however, it is not clear how this approach generalizes to arbitrary geometries.

Uncertainty relation

Reference [23] investigates a way of incorporating diffraction into purely ray-traced sound propagation, in which rays that pass by close to edges contribute to the diffraction total. This model was extended to 3D [24] and produced good results when compared to real measurements [25]. However, due to the large number of Monte Carlo rays needed to achieve good accuracy, simulation times were measured in minutes, not milliseconds.

3.3 VDaT: Approximating BTM

3.3.1 Volumetric Diffraction

The notion of modeling diffraction by examining the empty space around obstacles, as opposed to examining the obstacles' edges, arose from the combination of a simple acoustical observation and an implementation consideration. The observation is as follows: Consider any real-world situation where diffraction has a noticeable effect, such as listening to someone speak while walking around a corner. It is immediately apparent that the high frequencies are attenuated by the obstacle more quickly than the low frequencies when the obstacle begins to occlude the direct path. It is as if the obstacle acts as a low-pass filter, with the cutoff dependent on how far the obstacle has cut through the direct path. (In fact, some rudimentary real-time diffraction simulators simply apply a generic low-pass filter when the direct path is occluded [26].) Geometrically, it is as if the high-frequency sound mostly travels in a small region around

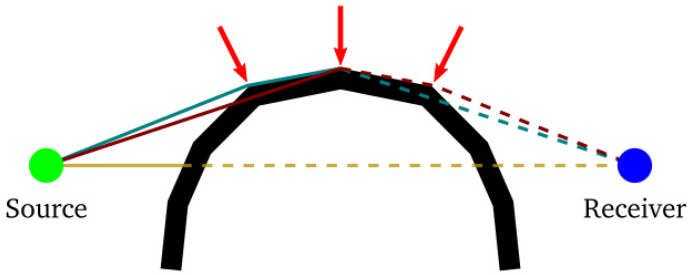

Figure 3.2. For this object and source/receiver positions, edge diffraction must be simulated to at least third order, or the diffraction path will be completely absent. If the object's curvature was approximated by more polygons, the edge diffraction order would have to be raised even further to avoid dropouts.

the direct path, and thus is blocked more fully by a small amount of occlusion, whereas the low-frequency sound occupies a larger volume around the direct path and therefore requires more occlusion to be affected. The concept of Fresnel zones [20] is one formalization of this notion; we take the notion in a somewhat different direction below.

Along with these observations, VDaT was also inspired by an implementation consideration: it is desirable, both computationally and theoretically, for the diffraction modeling to be decoupled from the amount of detail in the object meshes. On the one hand, the amount of mesh detail has a huge impact on the computational performance of an edge diffraction model. Not only is the computational complexity polynomial in the number of edges for a given diffraction order (section 3.1), the diffraction order must be raised as the meshes become more complex. If the algorithm is not able to traverse around the outside of the obstacles in the limited steps available, important diffraction paths will be simply omitted (Figure 3.2). On the other hand, as 3D meshes become more detailed, the acoustical role of each edge typically diminishes, as most edges are contributing to the approximation of smooth surfaces or adding small-scale detail. Only when many edges are considered together does the acoustical behavior of the whole object emerge. This can still be simulated with high-order edge diffraction, but it is no longer clear that the edges *per se* play a privileged role in determining the acoustical behavior of the object.

As a result of these considerations, we developed the volumetric diffraction approach

which became VDaT. At a high level, it operates as follows:

- Spatially sample the scene around the occluded direct path at many size scales, using ray tracing.
- Use the results to estimate the predominant shape of the object meshes near the direct path.
- Compute the diffraction amplitude response and path length, based on how BTM typically behaves for the estimated objects.

The spatial sampling is the only place the algorithm interacts with the scene objects, and it does so in a highly efficient, parallelizable way (subsection 3.5.1). The remainder of the algorithm uses the spatial sampling results to approximate BTM edge-diffraction results, without needing any numerical integration as in BTM.

Note that while in our discussions below we typically use the example where the direct path is occluded, we also apply VDaT to non-occluded paths, to simulate non-shadowed edge diffraction (subsection 3.4.3). Also note that we apply VDaT to every segment of every high-order specular reflection path.

3.3.2 Spatial Sampling

VDaT samples the space around an original direct or specular reflection path segment, according to the following hierarchy.

- Several "rings" (quantity N_r) are constructed around the original path segment, each with a different radius r. One of these rings is shown in Figure 3.3.
- For each ring, a number of angles ψ (quantity N_a) are taken, uniformly spaced in $[0, 2\pi)$. Eight angles are shown in Figure 3.3.
- At each angle, a number of "subpaths" (quantity N_{sp}) are constructed. Each subpath begins at one end of the original path segment and ends at the other end, representing a discretized

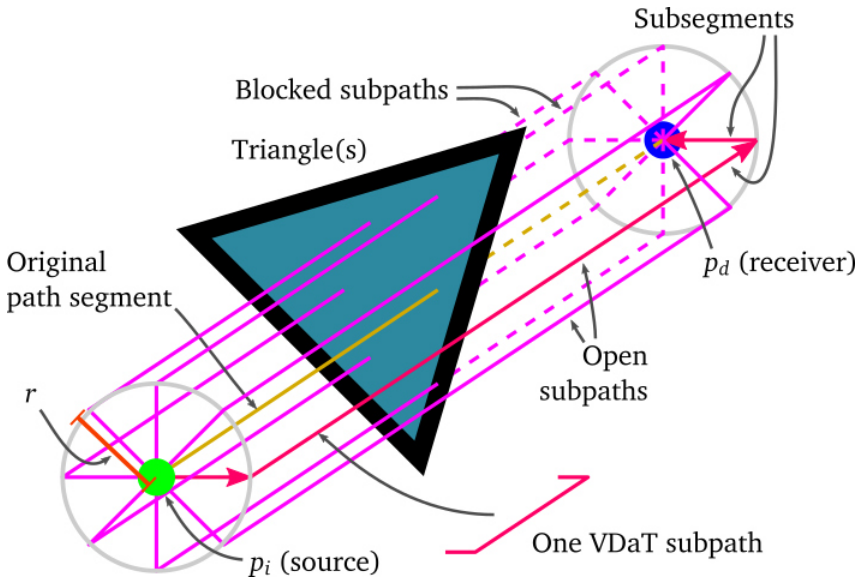

Figure 3.3. The main features of VDaT spatial sampling.

approximation of a path sound could travel around obstacles. Only one subpath per angle is shown in Figure 3.3; additional subpaths are shown in Figure 3.4.

- Each subpath is composed of a number of straight "subsegments". For example, each subpath shown in Figure 3.3 has three subsegments. Each subsegment is checked for intersection with all the triangles in the scene, to determine whether the subpath it belongs to is blocked by obstacles or not. Some of the subpaths share subsegments for a reduction in computation, so N_{ss} refers to the total number of unique subsegments traced per angle, across all the subpaths.

The equations describing this process are as follows:

$$\Xi(r) = \frac{1}{N_a} \sum_{a=0}^{N_a-1} \Xi_a(p_i, p_d, r, \frac{2\pi a}{N_a}) \tag{3.3}$$

$$\Xi_a(p_i, p_d, \lambda, \psi) = \frac{1}{N_{\text{sp}}} \sum_{\text{subpaths}} \Xi_{\text{sp}}(p_i, q_i, q_d, p_d) \tag{3.4}$$

$$\Xi_{\text{sp}} = \prod_{\text{subsegments}} \Xi_{\text{ss}} \tag{3.5}$$

where Ξ represents "transmission" (Ξ_a, Ξ_{sp}, and Ξ_{ss} are the transmission for one angle, one

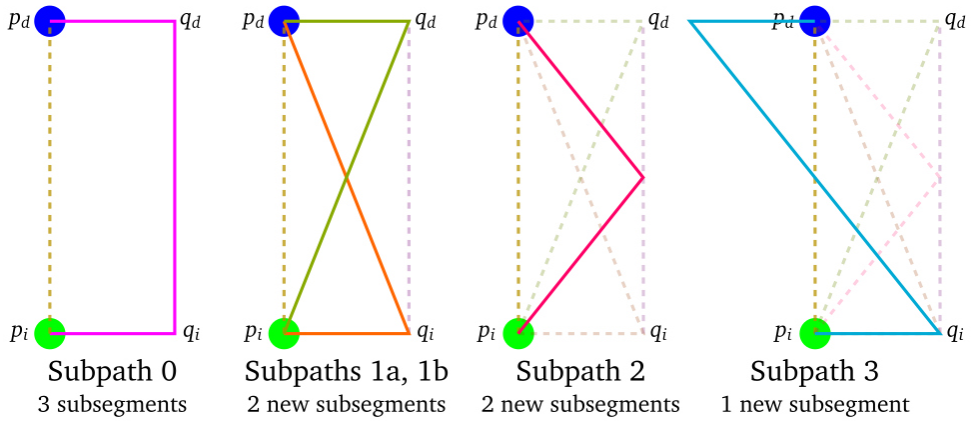

Figure 3.4. Example VDaT subpaths. The choice of subpaths balances performance (subsegments can be reused in multiple subpaths) with robustness (more subpaths means more ways sound waves could traverse the scene around obstacles).

subpath, and one subsegment respectively), p_i and p_d are the two ends of the original path segment, and q_i and q_d are the outer points at each angle as indicated in Figure 3.4. These latter two points are constructed as follows. Planes are constructed through p_i and p_d perpendicular to $\overline{p_i p_d}$. Orthonormal vectors b_1, b_2 are constructed in these planes, in a way that is consistent between frames and that changes smoothly when p_i and p_d are moved. The points u of the ring are constructed from these vectors, and if p_i or p_d are reflections, they are projected onto the reflecting triangles parallel to the original path segment:

$$b_1, b_2 : \|b_1\|_2 = \|b_2\|_2 = 1,\ \{b_1 \perp b_2\} \perp \overline{p_i p_d} \tag{3.6}$$

$$u(p, r, \psi) = p + b_1 r\cos(\psi) + b_2 r\sin(\psi) \tag{3.7}$$

$$q(p, r, \psi) = \begin{cases} \text{proj}_{t_p}^{\|\overline{p_i p_d}}(u) & \text{if } p \text{ on triangle } t_p \\ u & \text{if } p \text{ source/receiver} \end{cases} \tag{3.8}$$

$$q_i = q(p_i, r, \psi),\ q_d = q(p_d, r, \psi) \tag{3.9}$$

Note that as a type of spatial sampling, VDaT is bound to the sampling theorem: it can only consistently resolve physical features of similar or larger size than the sampling. However,

the multiscale approach of VDaT ensures the scene is sampled more finely near the direct path, where small features have more acoustical impact. In other words, unlike UTD, VDaT has no trouble handling a scene with a 5 cm square object where the source and receiver are close to it on both sides. Meanwhile, unlike BTM, VDaT will completely ignore that small object when processing other path segments which are several meters away from the object.

3.3.3 Scene transmission

In the simplest case, a subsegment which intersects at least one triangle has a transmission $\Xi_{\text{ss}} = 0$, and a subsegment which does not intersect any triangle has $\Xi_{\text{ss}} = 1$. However, VDaT supports optional modeling of sound transmission through obstacles. Triangles have transmission coefficients $X(t) \in [0,1]$, and the subsegment's transmission is the product of the transmission coefficients of each triangle the subsegment intersects.

$$\Xi_{\text{ss}}(a,b) = \prod_{t \in T_i(a,b)} X(t) \tag{3.10}$$

$$T_i = \left\{ t \mid t \text{ intersects } \overline{ab} \right\} \tag{3.11}$$

3.3.4 BTM: Filtering by Interference

Before we can introduce how VDaT approximates BTM, we must develop one point about the operation of BTM itself.

BTM is a "discrete Huygens"[27] / "secondary source" [10] model: it models diffraction as if there are an infinite number of secondary sources on the edge, which re-emit an impulse as soon as they receive it. In other words, there are an infinite number of diffraction paths around each edge. Each of these paths contributes one impulse to the overall impulse response, with different amplitudes due to the diffracting angles and different delays due to the path lengths. This implies that the filtering which is the hallmark of diffraction is actually a result of *interference between these infinite diffracting paths around the edge*. If all of these paths had

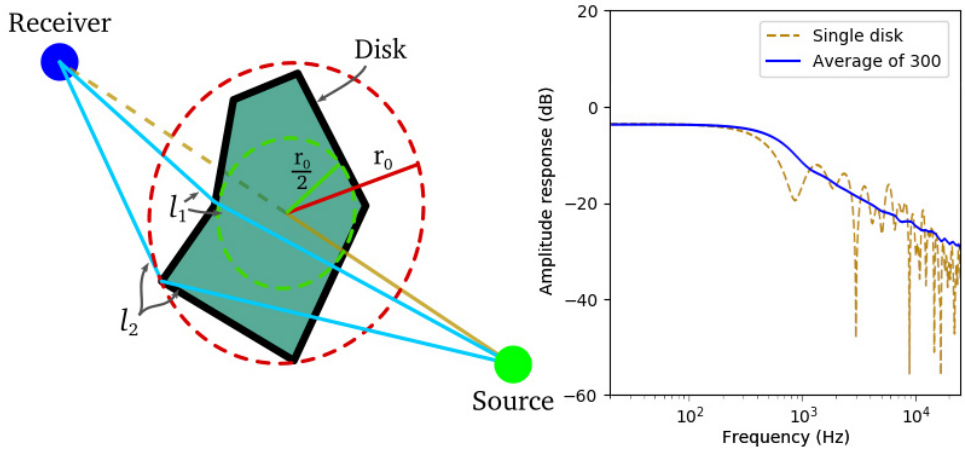

Figure 3.5. (Left) A random disk with linear edges, between the sizes $r_0/2$ and r_0. (Right) The BTM amplitude response of that particular disk, and the average amplitude response of 300 random disks constructed similarly.

the same length—for instance, in the case of a thin circular disk with the source and receiver on-axis—the impulse response would be a single scaled impulse, i.e. a flat frequency response, since all the impulses would arrive at exactly the same time. However, this case is a singularity. In most cases, the diffracting paths around the edge are of different lengths—in fact, the use of point sources and straight edges of finite triangles guarantees it. That is, in scenes composed of triangle-based meshes, diffraction filtering is caused by the existence of secondary-source diffraction paths with lengths ranging continuously within certain bounds. The operations below are effectively using the spatial sampling to estimate those bounds, and fill in the amplitude response based on typical BTM behavior.

3.3.5 Approximating BTM

Consider the case where $\Xi(r) = 0,\ \forall r \leq r_0/2$ and $\Xi(r) = 1,\ \forall r \geq r_0$. That is, we have performed spatial sampling at radii separated by powers of two, and found that all the subpaths were blocked up to a certain radius, but they are all open starting at the next largest radius. This means that there must be one or more obstacles in the scene, blocking the direct path, whose size is between those two radii. We do not know their exact shape, nor how close to the source or receiver they are, but we do know that they are no smaller than $r_0/2$ and no larger than r_0 (at least,

we are sure of that if N_a approaches infinity). If we project these obstacles onto a normal plane midway between the source and receiver, which corresponds to how subpath 0 performs spatial sampling, they will become a disk with jagged, linear edges (Figure 3.5 left). Crucially, the lengths of the BTM secondary source diffraction paths around these edges are bounded by ℓ_1 and ℓ_2, which are directly determined by r_0. We construct hundreds of arbitrary jagged disks between $r_0/2$ and r_0, compute the true BTM amplitude response for each, and average them (Figure 3.5 right). While each individual amplitude response displays severe interference effects due to the summing of the main BTM diffraction paths around the edges, their average is the familiar lowpass shape which we expect from diffraction around an obstacle. This filter is effectively constant up to a certain knee value k, and then has a rolloff of 10 dB/decade, corresponding to $a_2/a_1 = 1/\sqrt{f_2/f_1}$.

Of course, the obstacles are unlikely to be such a disk shape—on average they would be uniformly distributed along the length of the direct path. Thus, to effectively sample the other extreme of their possible distribution, we also project the obstacles onto a normal plane which is 99% along the length of the direct path, right next to either the source or the receiver. (Both BTM and VDaT are symmetrical—the source and receiver can be swapped with identical results.) This results in changes to ℓ_1 and ℓ_2, as well as to the angle-dependent terms in BTM. However, the results differ only modestly; for example for $l_s = 2$ m and $r_0 = 0.8$ m (Figure 3.6), they differ by 1.7 dB FW-LSD (error metric given in Equation 3.25).

There is one more case we consider at this stage: that of non-shadowed diffraction through a hole or gap between obstacles. This is the case where the spatial sampling results are the reverse of the above: fully open ($\Xi(r) = 1$) at small radii ($\forall r \leq r_0/2$), and fully blocked above that ($\Xi(r) = 0$, $\forall r \geq r_0$). We construct hundreds of jagged holes in infinite planes midway between the source and receiver, and compute and average the BTM amplitude responses to first order as above. In this case, the BTM results (Figure 3.6 and Figure 3.7 bottom) are that of an attenuating low shelving filter; the sloped section is now 20 dB/decade.

Since these averaged BTM amplitude responses depend on the length of the original path

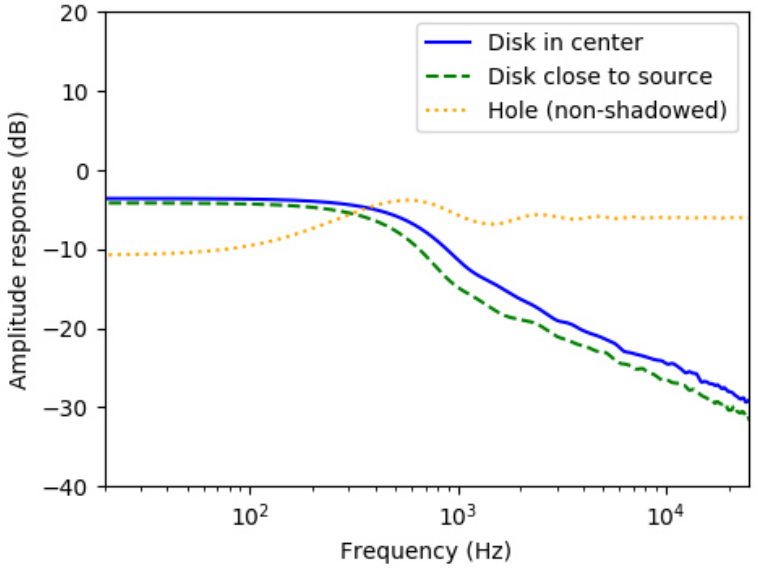

Figure 3.6. Average BTM amplitude responses over 300 random geometries in each of the three cases considered, for $l_s = 2$ m and $r_0 = 0.8$ m.

segment $\|\overline{p_i p_d}\| = l_s$, which may be any continuous value, it is not feasable to precompute and store them. We instead model them (Eqns. 3.12–3.14) by the shapes described above: a flat response and then a 10 dB/decade rolloff for the two disk cases, and two flat segments with a 20 dB/decade sloped section in between for the hole case.

$$D_{\text{disk}}(\omega) = a_d \min\left(1, \frac{1}{2\sqrt{\omega/k}}\right) \tag{3.12}$$

$$D_{\text{hole}}(\omega) = \max\left(\min\left(D_d\omega/k, D_d\right), a_h\right) \tag{3.13}$$

$$D_d = 1/l_s \tag{3.14}$$

where D_{disk} and D_{hole} are diffraction amplitude responses in frequency ω, and D_d is the amplitude response for the original direct path segment with no diffraction. We observe that the knee for the two filter shapes in each case are in roughly the same location, so we use a single knee value k for both. The other two needed parameters are a_d, the low-frequency amplitude for the disk cases, and a_h, the low-frequency amplitude for the hole. The three parameters $\{k, a_d, a_h\}$ can

only be functions of the radius r_0 and l_s; they cannot depend on the position of the obstacles between the source and receiver, as the spatial sampling does not provide that information. Of course, the BTM results do depend on that information; so the best we can hope for is an estimate of $\{k, a_d, a_h\}$ given $\{r_0, l_s\}$, that produces as little error as possible over the range of relevant conditions.

To approach this optimization problem, we began by plotting and observing the values of each of these parameters as r_0 and l_s were varied, and attempting to write equations that matched their behavior. Once we had developed a general form for these equations, we created a parameter optimization system which uses a random walk to jointly fine-tune their parameters. We considered the three cases discussed above, each for three size scales ($l_s = \{20\text{ cm},\ 2\text{ m},\ 20\text{ m}\}$), and for ten radii spaced in powers of two from 16.384 m to 3.2 cm, to roughly cover the range of wavelengths of human hearing. The optimization objective was the ℓ_1 distance between the averaged BTM amplitude response and the VDaT amplitude response, in the range 20 Hz–20 kHz. This process produced the following model, where $\{k_1, \ldots, k_6, d_1, \ldots, d_5, h_1\}$ are the optimized parameters given in Table 3.1.

$$\hat{k} = \exp_{10}\left(k_1(b_k - k_5)^{k_2} + k_3 b_k + k_4\right) \tag{3.15}$$

$$b_k = \log_{10}\frac{c(l_s/2)^{k_6}}{r_0} \tag{3.16}$$

$$\hat{a}_d = \frac{1 + (d_4 l_s/r_0)^{d_5}}{\sqrt{l_s^2 + r_0^2}}\left(d_2 + \frac{d_3}{x + 1/x}\right) \tag{3.17}$$

$$x = d_1 l_s / r_0 \tag{3.18}$$

$$\hat{a}_h = \min\left(\frac{h_1 r_0}{l_s^2},\ D_d\right) \tag{3.19}$$

From Figure 3.7 top, it is evident that much of the resulting error is due to the "bump" near the knee in the BTM amplitude response, which seems to be caused by the first sidelobe

Table 3.1. Numerically optimized parameter results for Eqns. 3.15–3.19.

Param	Value	Param	Value	Param	Value
k_1	0.7735	k_2	1.0446	k_3	0.7991
k_4	-1.0501	k_5	0.6817	k_6	0.3453
d_1	1.4430	d_2	0.9190	d_3	-0.3280
d_4	1.3076	d_5	-0.3378	h_1	1.5829

lining up at the same frequency over all realizations rather than being randomly distributed. We experimented with modeling this "bump", but while doing so led to lower errors in this step, it worsened the performance in the next section. Future work may investigate alternative approaches to modeling the average BTM amplitude response, from a purely analytical standpoint or with a fully machine-learning implementation (see Reference [22] in subsection 3.2.2).

3.3.6 Combining Results Across Rings

We now consider how the estimated BTM amplitude responses above are combined for the general case of partial transmission on multiple rings. First, we initialize the amplitude response to zero if the original direct path segment is blocked, or to the direct-path amplitude response if it is open:

$$D_0(\omega) = \begin{cases} D_d & \text{if } \Xi(0) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

Next, we iterate over the rings, and use the difference in transmission between the current ring and the next smallest ring as a weighting on the estimated BTM amplitude response between those two radii. This ensures the responses computed above are used without modification if the spatial sampling results are the special cases considered above, and it provides an interpolation between the responses in all other cases. If the difference is negative, we use the "hole" amplitude

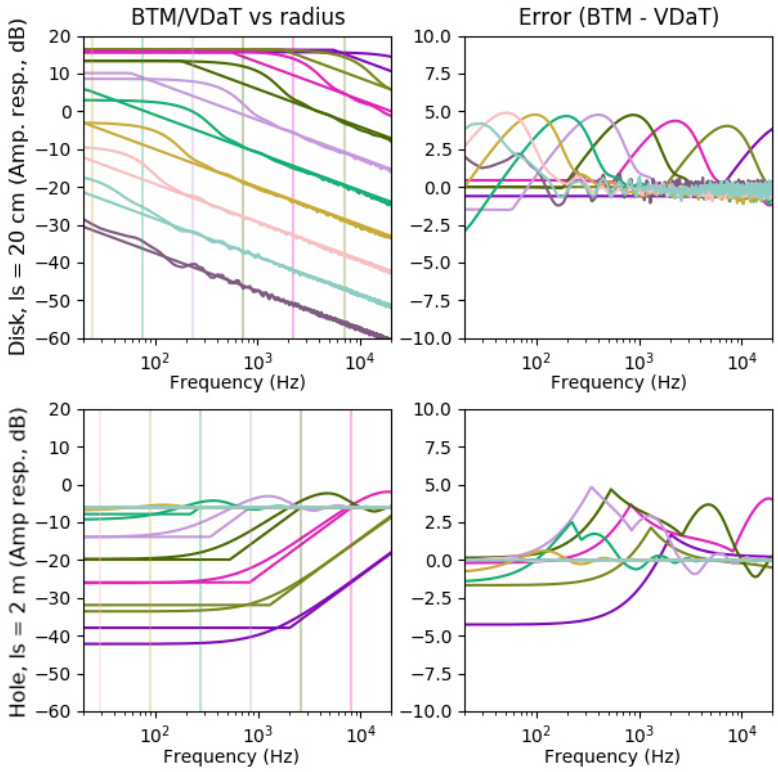

Figure 3.7. Two of the nine cases used in the parameter optimization: disks with small l_s, and holes with medium l_s. On the left, the curved lines are BTM amplitude responses, and the piecewise linear lines are the VDaT approximations. The ten cases shown in each subplot are ten radii, $r_0 = 16.384$ m to 3.2 cm by powers of 2 from left to right. The light vertical lines are the values of k.

response, as this corresponds to the larger ring being more blocked than the smaller ring.

$$D_i(\omega) = D_{i-1}(\omega) + (\Xi(r_i) - \Xi(r_{i-1}))\, D_\Delta \tag{3.21}$$

$$D_\Delta = \begin{cases} D_{\text{disk}}(\omega) & \text{if } \Xi(r_i) \geq \Xi(r_{i-1}) \\ D_d - D_{\text{hole}}(\omega) & \text{otherwise} \end{cases} \tag{3.22}$$

The results for the large occluder in subsection 3.4.1 demonstrate the effectiveness of this accumulation / interpolation scheme.

Finally, we apply one more heuristic to improve the results. Often, the set of angles which are blocked on consecutive rings are the same or similar, for instance when there is a large object

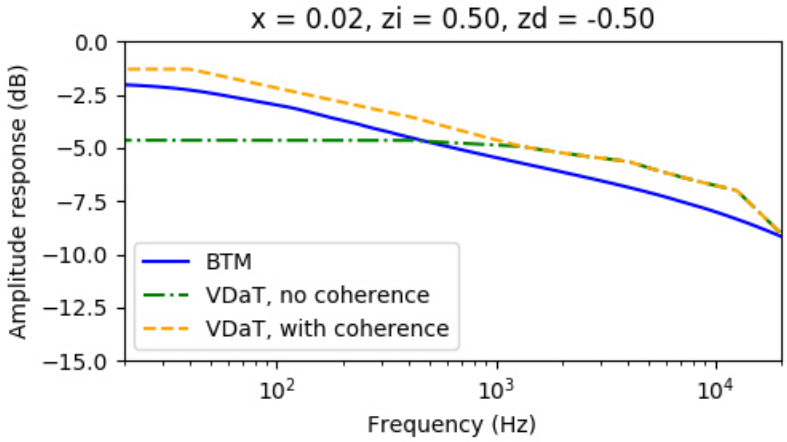

Figure 3.8. Example VDaT results with and without the "coherence" heuristic.

occluding the direct path only slightly. We observed that in these cases, the BTM amplitude response has a higher value at low frequencies than expected (Figure 3.8). We conjecture that this is because in these cases, the low frequencies that diffract around the portions of the edge in one range (say $[r_1, 2r_1]$) are coherent with the low frequencies diffracting around the portions of the edge in the next range ($[2r_1, 4r_1]$). The two ranges have low-frequency amplitude responses that are quite similar, but in reality they interfere constructively. Because we are effectively averaging the amplitude responses instead of adding them, we correct for this by creating a "coherence" metric such that coherence in all rings together would result in doubling the amplitude response at low frequencies. For each consecutive pair of partially-blocked rings with radii r_1 and $r_1/2$, there is coherence if $|\Xi(r_1) - \Xi(r_1/2)| < 1/N_a$, meaning that the number of angles blocked for each is the same. The frequencies corresponding to wavelengths of these radii are computed, $\omega_1 = c/r_1$ and $2\omega_1$ respectively, to form a frequency band. In each frequency band which has coherence, the gain is interpolated down from $2^{1/N_r}$ at ω_1 to 1 at $2\omega_1$, linearly in the log-log amplitude-frequency domain. The use of base-2 logarithms and power-of-2 radii allows this expression to be greatly simplified (Equation 3.24). Note that D_{N_r} is the result of combining N_r rings from Equation 3.21, and D_{VDaT} is the final VDaT amplitude response.

$$D_{\mathrm{VDaT}}(\omega) = D_{N_r} \prod_{\omega_1 \text{ coher.}} \kappa(\omega_1) \tag{3.23}$$

$$\kappa(\omega_1) = \mathrm{clip}\left(1, \left(\frac{2\omega_1}{\omega}\right)^{1/N_r}, 2^{1/N_r}\right) \tag{3.24}$$

3.3.7 Path Length

In addition to amplitude response estimation, any diffraction model must estimate the delay of the diffracted sound, which is represented by the diffraction path length. As VDaT does not compute edge diffraction paths, VDaT modifies the path length of the original direct path segment if it is occluded. In simple cases with only one edge diffraction path, VDaT produces path length results that roughly match the existing models over a wide range of positions (Figure 3.10), as will be explained presently. In more complicated scenes, in which the existing models would produce many diffraction paths, VDaT estimates the shortest secondary-source edge diffraction path through the scene, to represent the first arrival of sound.

Each VDaT subpath that is unblocked represents a way, albeit an angular one, that sound can get through the scene from the source to the receiver. Since unblocked subpaths are effectively coarse overestimates of the true shortest secondary-source diffraction path, the shortest unblocked subpaths—i.e. at the minimum radius—will provide the most accurate information. For example, if only one single subpath is unblocked on the smallest ring r_1 that has any unblocked subpaths, that subpath is very likely to traverse the scene near the true shortest secondary-source edge diffraction path. The length of the true path can be easily estimated from the subpath (Figure 3.9 left). If using subpath 0, an assumption must be made about where the occluder is along the direct path segment, but the worst-case error—when the occluder is assumed in the middle of the direct path segment, while it is actually at one end, and when $l_s = 2r_1$—is only 14.4%. For other geometries, the error is lower, and the use of multiple subpaths can impose tighter bounds on the length estimate, reducing the error further.

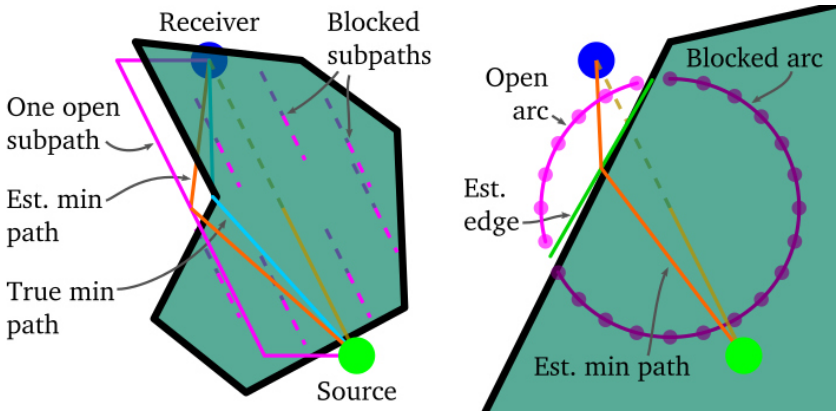

Figure 3.9. Estimating the minimum edge diffraction path length from a single unblocked subpath (left) or a range of consecutive unblocked subpaths (right).

When there are multiple individual unblocked subpaths at isolated angles on a given ring, there is no additional information beyond the case of a single subpath above. However, when there are unblocked subpaths at consecutive angles, this implies the edges of the obstacles are somewhere inside the ring in that region. Assuming that the edges of the obstacles are straight on average, we connect the ends of the unblocked arc with a straight line, and estimate that the shortest edge diffraction path goes around the center of this line (Figure 3.9 right). When this assumption of straight edges is correct, such as for convex objects whose edge lengths are $\gtrsim r_1$, this approach is quite accurate (Figure 3.10). When it is not, such as for concave objects in particular orientations, the error can be larger, but it is bounded by the estimated length around the next smallest ring.

The VDaT path length estimation system is essential to accurately model delay and phase in static simulations such as sound field intensity plots and room impulse responses. However, because its estimates change discretely as subpaths are blocked or unblocked, we disable diffraction path length estimation in dynamic scenes; future work may instead smooth these results over time. While VDaT does not model the true direction diffracted sound arrives to the listener from (instead using the original direct path segment), this typically does not introduce much inaccuracy in the applications we are interested in. For a small obstacle, there are typically multiple edge-diffraction paths of similar lengths around the obstacle, so the sum

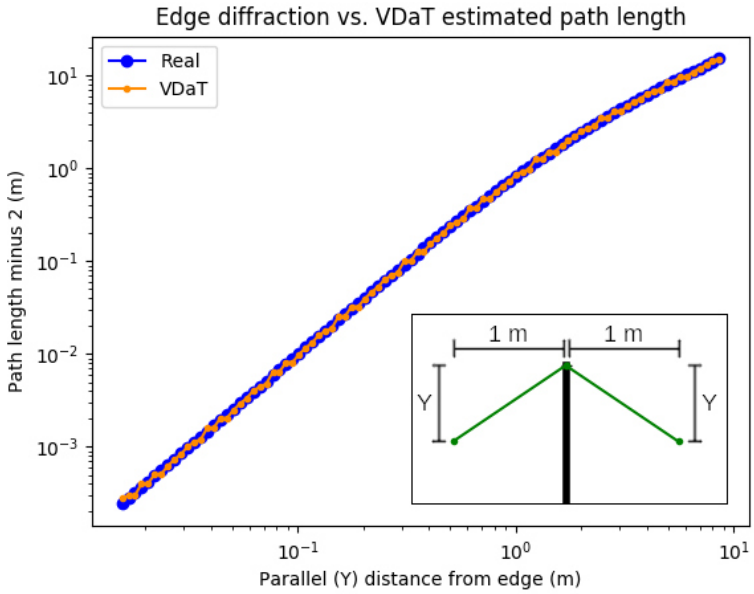

Figure 3.10. BTM edge diffraction path length vs. VDaT path length estimates, for the symmetrical case, over a wide range of distances to the edge.

of them is usually perceived as sound coming from behind the obstacle (which matches how VDaT models it). Conversely, large objects such as room walls often have a non-negligible transmission component, and due to the precedence effect [28], the perception of direction is usually dominated by the sound with the shortest path delay (which is the transmitted sound).

3.4 VDaT vs. UTD/BTM Results

The results in this section are from a set of Python scripts (File 3.2) which compute VDaT, BTM, and other diffraction models for definable 2D and 3D scenes. Note that our implementation of the edge diffraction models is limited to first-order diffraction; the second-order diffraction results for the thick object in Figure 3.15 are computed with Reference [29]. In all examples in this section, VDaT uses nine rings, 64 angles, and subpath 0 only ($N_r = 9$, $N_a = 64$, $N_{\text{sp}} = 1$, $N_{\text{ss}} = 3$). Our error metric is frequency-weighted log spectral distance (FW-LSD) (Equation 3.25) in the 20 Hz–20 kHz range, as compared to the BTM amplitude response. This metric ensures that error is weighted the same over each octave, rather than over linear frequency as in standard

LSD.

$$\text{FW-LSD}(a_1(\omega), a_2(\omega)) = \sqrt{\frac{\sum_\omega \frac{\left(20\log_{10}\frac{a_1(\omega)}{a_2(\omega)}\right)^2}{\omega}}{\sum_\omega \frac{1}{\omega}}} \tag{3.25}$$

3.4.1 Half-Plane

We first consider an infinite half-plane, which stands in for any thin object with long edges. Figs. 3.11 and 3.12 show results for normal and non-shadowed diffraction over two orders of magnitude each of l_s, the direct path length, and x, the distance from the diffracting edge to the direct path.

For shadowed diffraction, the average error in the VDaT amplitude response as compared to BTM is 1.8 dB FW-LSD (Figure 3.11). For non-shadowed diffraction, the results are even better (Figure 3.12). The worst cases, with up to 3.4 dB FW-LSD, are for high frequencies in the large, asymmetrical case (upper right), which VDaT does not distinguish from the symmetrical case (upper left).

3.4.2 Small Objects

As discussed in section 3.1, the UTD diffraction model, which is the most popular for real-time systems, assumes all edges are of infinite length. As a result, its amplitude response results have substantial error for small objects. Figure 3.13 shows two such cases, in which the direct path is obliquely and non-symmetrically occluded by small thin plates of different sizes and shapes. In both, VDaT approximates the BTM results (FW-LSD 2.6 dB top / 2.5 dB bottom) much more closely than UTD does (4.3 dB top / 10.0 dB bottom), while retaining a much lower computational complexity (subsection 3.5.1). Furthermore, if the mesh moves such that the apex point of one of the UTD edge diffraction paths moves outside the bounds of its edge, UTD will abruptly delete that path, and there will be a discontinuity in the output. Neither BTM nor VDaT exhibit such discontinuities with moving small objects.

As mentioned in subsection 3.3.5, BTM produces sharp interference effects as a result of

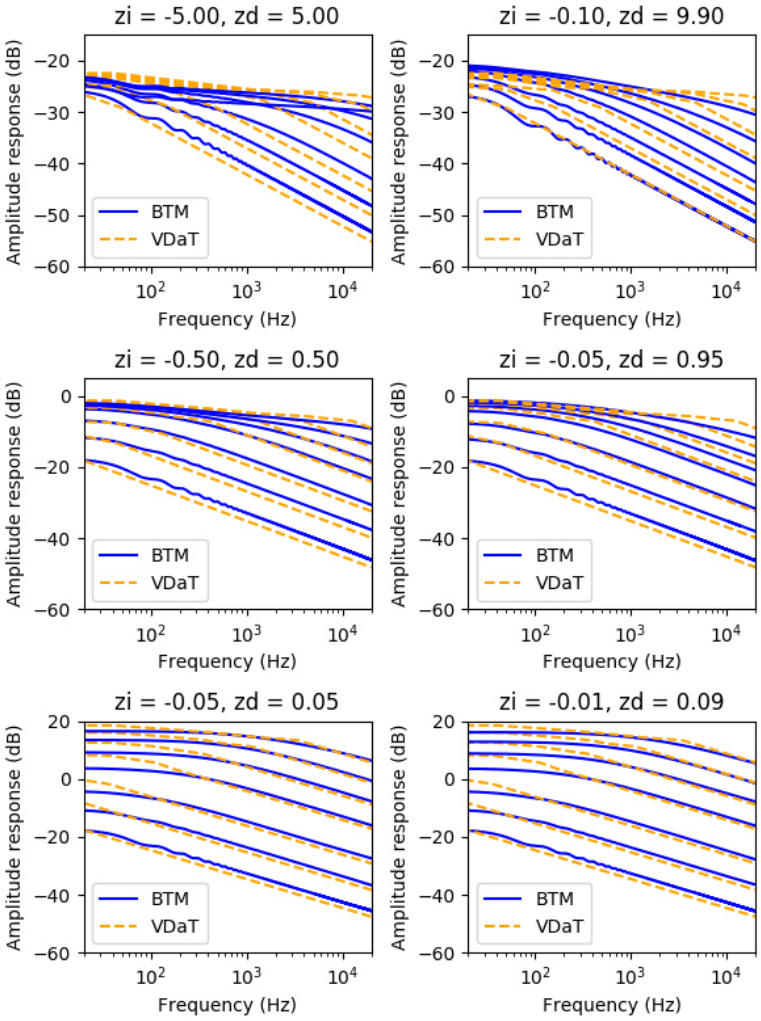

Figure 3.11. BTM and VDaT amplitude responses of diffraction by an infinite half-plane. The three rows are three different size scales; the two columns are the symmetrical and asymmetrical cases of placement of the half-plane along the direct path (subsection 3.3.5). The seven cases shown in each plot are for the half-plane cutting the direct path by [0.02, 0.05, 0.1, 0.2, 0.5, 1.0, and 2.0] m, from top to bottom.

the diffraction paths around the different edges being summed. VDaT does not attempt to model this interference, instead producing a smooth response designed to approximate the average BTM result. This characteristic can be considered both a shortcoming and an advantage of VDaT. If the simulated occluding object was constructed in the real world—infinitely sharp edges, completely rigid, in a perfectly anechoic environment—the measured diffraction amplitude response would indeed include interference effects, matching BTM. However, most real-world environments

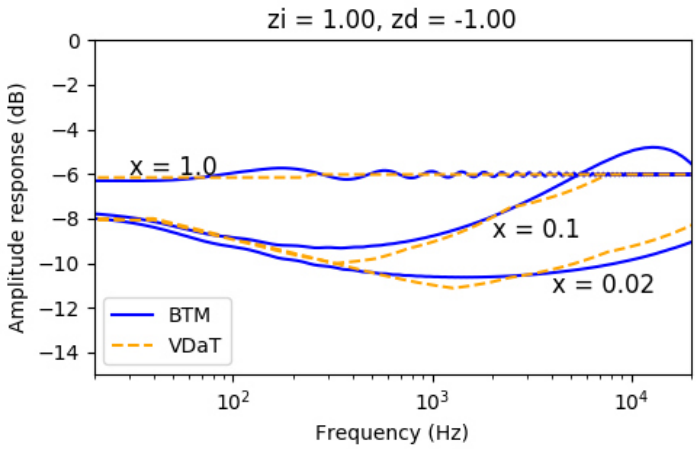

Figure 3.12. BTM and VDaT amplitude responses of non-shadowed diffraction by an infinite half-plane, for the symmetrical case with $l_s = 2$ m. x is the distance from the direct path to the diffracting edge. This is a case VDaT handles well; the average FW-LSD for these three cases is 0.34 dB.

are much more complex, and this complexity tends to perform an averaging effect like VDaT. Since real situations where sharp interference effects are audible are very rare, these effects in an acoustical simulation may sound "wrong", especially when objects are moving and the peaks sweep across frequency. VDaT avoids this situation and may better match users' perceptual expectations.

3.4.3 Non-Shadowed Diffraction

In GA edge diffraction models, as long as an edge is visible to the source and receiver, diffraction is occurring, regardless of the position of the source and receiver. The GA diffraction term additively "corrects for" the discontinuity when the direct path becomes occluded on one side (the shadow boundary), and the discontinuity when the specular reflection path becomes invalid on the other side. VDaT is designed to handle cases on both sides of the shadow boundary—both shadowed and non-shadowed diffraction—but it does not handle the effects of diffraction near the specular reflection boundary. (The uncertainty model [23, 24] in subsection 3.2.2 is another model with these same characteristics.) Future work [chapter 4] is planned to validate and characterize a spatial sampling model similar to VDaT to handle

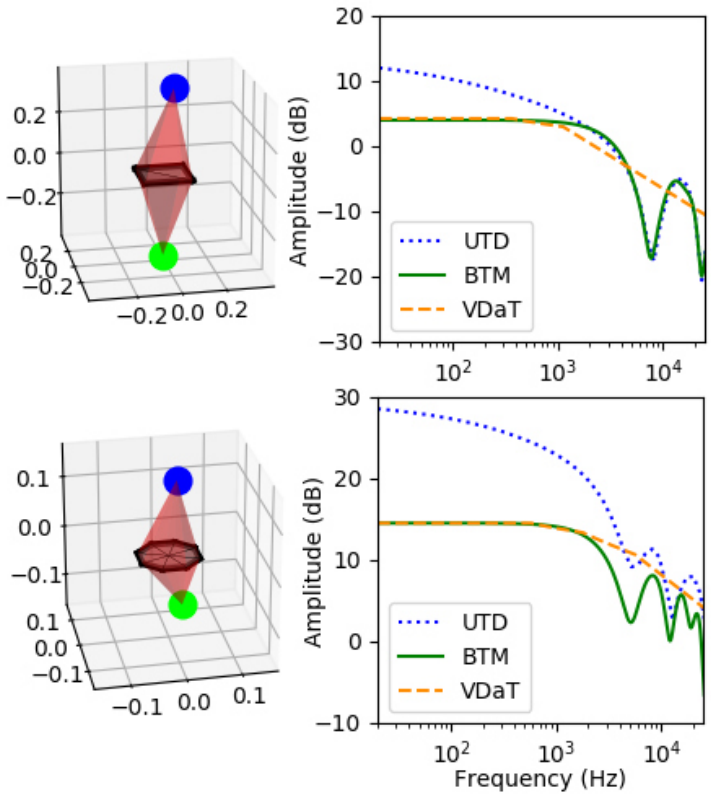

Figure 3.13. Diffraction amplitude responses around two small objects (3D plots in meters), as simulated by UTD, BTM, and VDaT. VDaT matches the overall shape of the BTM response, without the interference effects. UTD assumes the objects' edges are infinite, resulting in large error.

near-reflection cases.

However, in addition to omitting diffraction effects near reflections, most real-time edge diffraction implementations ignore all non-shadowed diffraction [16–18], as non-shadowed diffraction vastly expands the number of edges which must be considered at each step of higher-order edge diffraction. Instead they use a heuristic [30] which adjusts the level of the diffracted field in the shadow region so that its amplitude is continuous with the direct sound at the shadow boundary. However, non-shadowed diffraction—especially non-shadowed higher-order diffraction—plays an important role when there is a small gap or hole in a large occluder. If non-shadowed diffraction is ignored, the sound will always be fully open (unfiltered) when the direct path is open, or receive diffraction filtering based on the closer edge, regardless of

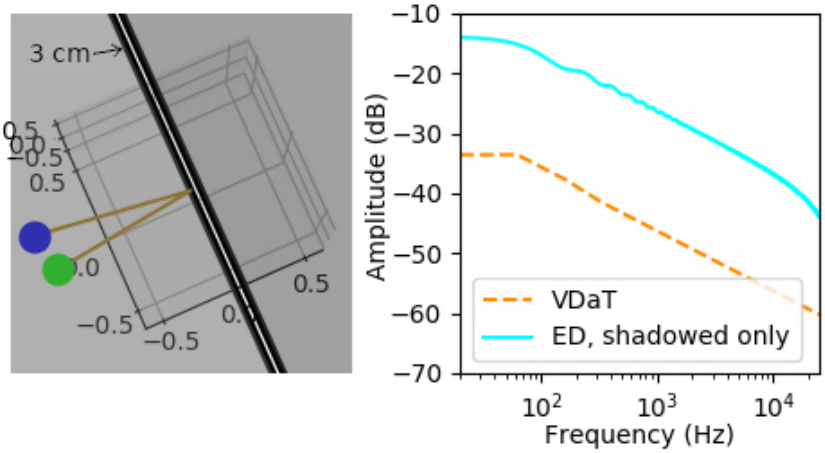

Figure 3.14. Most real-time edge diffraction (ED) systems, based on UTD or BTM, ignore non-shadowed diffraction edges. For this slit, they ignore the right edge, meaning the amplitude response does not change as the slit shrinks to zero. In contrast, the VDaT amplitude response correctly falls to zero as the slit shrinks to zero.

the size of the gap or hole. This leads to the absurd result of the sound remaining constant while the gap or hole is shrinking to zero (Figure 3.14). In contrast, as VDaT does handle non-shadowed diffraction near the shadow boundary, VDaT can produce sensible results in these cases. (Note that even BTM does not compute correct results for the small gap or hole without additional considerations [31]; to the authors' knowledge no general edge-diffraction simulation system exists which gives theoretically verifiable results for arbitrary non-convex geometry. Consequently, instead of comparing the VDaT results to a reference, we merely point out that its asymptotic behavior is qualitatively correct, unlike the existing real-time implementations discussed above.)

3.4.4 Other Occluding Objects

Figure 3.15 shows three examples of other types of objects occluding the direct path: a plane A at 45° to the direct path, a wedge W with a 90° inner angle, and a 2 m thick object T undergoing second-order diffraction. When using only subpath 0, the same set of subpaths is blocked in each of these cases, so VDaT cannot distinguish among them. Nevertheless, the BTM results for these cases are similar, and the VDaT results fall among them, resulting in errors

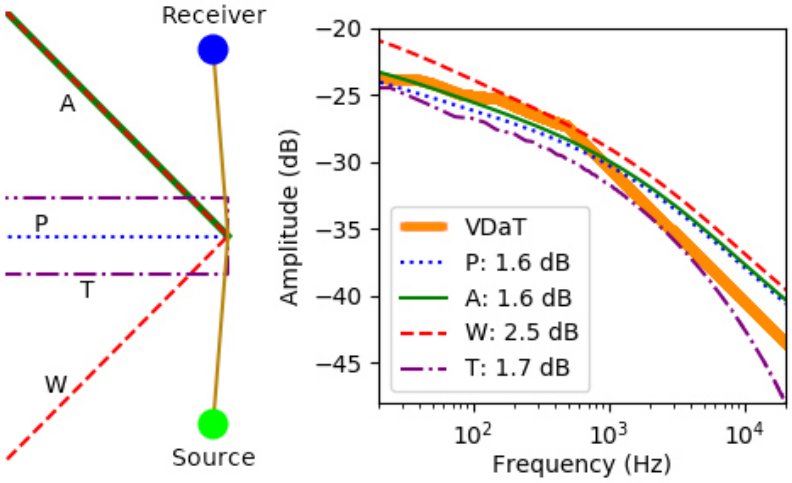

Figure 3.15. The direct path ($l_s = 10$ m) is occluded by $x = 0.4$ m by four different objects: plane P, angled plane A, wedge W, and thick object T. VDaT produces the same result for all of these cases, as its spatial sampling does not distinguish among them. Nevertheless, the BTM results for each case are similar, so VDaT is only in error by 1.6–2.5 dB FW-LSD in these cases.

below 3 dB FW-LSD. However, if the wedge angle or the object thickness increases so far that the source and receiver approach the surfaces, the BTM amplitude response changes substantially especially at low frequencies and VDaT no longer approximates it well. This is because the wedge approaches a plane, and the edge diffraction path approaches a specular reflection path, which VDaT does not directly model (see subsection 3.4.3). Future work may use information about where triangle intersections occurred along subsegments, as well as comparisons between transmissions over different subpaths, to provide more refined estimates of the shape of the objects near the direct path.

3.5 VDaT Implementation

3.5.1 Complexity

As a result of the spatial sampling hierarchy in subsection 3.3.2, it is immediately clear that the cost of computing the VDaT spatial sampling for one original path segment is $N_r \cdot N_a \cdot N_{ss} \cdot I(t)$, where $I(t)$ is the cost of determining which triangles in the scene intersect a given line segment (which can be $O(\log t)$ or better [15]). The remaining operations in VDaT

are all of constant complexity per original path segment, regardless of the scene complexity, so the overall complexity of VDaT is

$$C_{\text{VDaT}} \propto s \cdot I(t) \cdot N_r N_a N_{\text{ss}} \tag{3.26}$$

where s is the number of original path segments (direct paths plus segments of specular reflection paths). Compare this to Eqns. 3.1 and 3.2; in VDaT the power-law term η^o is missing, and the "quality" parameters N_r, N_a, and N_{ss} in VDaT only affect the performance linearly as opposed to exponentially in N_o. Note also that since each of the subsegments in VDaT is independent of the others, the ray tracing can be parallelized across all of them.

The only manner in which VDaT performance has a disadvantage compared to the existing models is that the generation of direct and/or reflected paths as the input to VDaT must always generate transmission paths through obstacles, effectively treating all objects as partially transparent to sound even if they are supposed to be fully opaque. This is because VDaT operates on existing path segments, particularly ones which are occluded by scene objects. However, in reality many solid objects (such as building walls) do perceptibly transmit sound, especially at low frequencies. As a result, transmission paths will have to be computed for many objects for realistic results using any model, so the penalty of computing transmission paths for all objects for VDaT may not be very high.

3.5.2 Real-Time Implementation: VDaT in Space3D

VDaT was created as the model for diffraction and transmission in the real-time audio spatialization and acoustical simulation system *Space3D*. This system is being developed (based on previous work [32, 33]) to deliver high-quality spatialization results over speaker arrays or headphones, and simulate high-order reflections, diffraction and transmission, material properties, directional sound sources, Doppler, and more in fully dynamic scenes. Space3D performs all geometry and audio processing on GPUs using NVIDIA CUDA [34]. Our implementation

Table 3.2. VDaT timing on real-time scenes.

Scene				Complexity		Time (ms)	
Name	t	N_o	S/K	P	s	VDaT	Total
Shoebox	12	6	1/1	377	2183	4.7	8.2
Cathedral2	34	4	2/1	635	2797	6.8	10.1
Hall1	50	4	2/1	220	941	3.4	10.2
Cathedral1	190	3	1/1	158	570	5.2	8.6
BasicCity	1024	2	2/1	100	278	9.9	11.2

S/K = number of sources/receivers; P = number of paths;
s = number of direct + reflection path segments
Conditions: $N_r = 9$, $N_a = 32$, $N_{\text{sp}} = 1$ ($N_{\text{ss}} = 3$), 512 spl buffers @ 44.1 kHz $\rightarrow$ 11.6 ms/frame, NVIDIA GTX 1080

recomputes all paths, and performs all audio processing, in 11.6 ms by default (frame length of 512 spls @ 44.1 kHz); much longer frames than this start leading to perceptible delays between audio and visuals. Approaches such as caching schemes [17] or lower scene update frame rates [3] can be very effective at amortizing the computational load across multiple frames, but these approaches limit how dynamic the scene can be, so we do not use them.

We assume that a low-complexity mesh, like the type typically used for computing collisions in interactive multimedia applications, is used for the GA audio processing. This mesh may be created by the artist, or automatically generated in a preprocessing step by simplification of a visual mesh [16] for static portions of the scene. Note that unlike UTD, we do not require that the simplified mesh have no small edges or small objects; the goal is to eliminate any detail that is not acoustically relevant.

Table 3.2 demonstrates the efficiency of VDaT on small-to-medium-size scenes[1]. Note that P represents the number of actual, valid paths having audio simulated along them, which in our example real-time scenes ranges from 100 to 635. In contrast, Monte Carlo implementations [16, 17] typically trace on the order of 1000 visibility paths per frame, and can handle tens or

[1]The demonstrated scene complexity is primarily limited by our reflection path generation system, which uses the exponential-complexity image source method, as well as by our use of a parallelized but not hierarchical implementation for checking segment-triangle intersections. Future work is planned to improve both of these elements.

hundreds of thousands of triangles, but often produce only a handful of resulting acoustical paths (between 4.4 and 21.4 in Reference [17]). Furthermore, VDaT simulates diffraction and transmission on every one of the hundreds or thousands of segments of these paths, still in real time and with no precomputation. In the Cathedral2 case, this represents a total of $s \cdot N_r \cdot N_a \cdot N_{ss} = 2.41$ million VDaT subsegments and 82.2×10^6 segment-triangle intersections—all completed in 6.8 ms.

Figure 3.1 shows the real part of the soundfield simulated with Space3D and VDaT for diffraction around a large object, so the simulated sound waves are visible. The fact that they appear to emanate from the diffracting edge is evidence of the success of VDaT's path length estimation algorithm. File 3.1[2] shows a demo of Space3D and VDaT in action in a prototypical real-time scene, designed for comparison purposes to be similar to the city scene used to demonstrate the diffraction model in Reference [16].

3.6 Conclusion

Edge-diffraction models, especially the more accurate BTM model, suffer from high computational complexity, severely limiting their use in real-time applications. Volumetric Diffraction and Transmission (VDaT) is proposed as an alternative model for approximating diffraction. It operates by spatially sampling the scene around the direct or reflected path segment, and using the results to estimate the BTM edge-diffraction amplitude response and path length for that scene. Its results match BTM to within 1–3 dB over a wide range of scales in basic cases, and it can handle small objects and gaps in obstacles better than existing real-time diffraction systems. Furthermore, its performance is high enough that it can simulate diffraction for thousands of higher-order reflection path segments in a handful of milliseconds on consumer-grade GPU hardware, without needing any precomputed information about the scene. As a result, VDaT is a strong choice for diffraction simulation in hard real-time applications with arbitrary, fully

[2] Video caption for File 3.1: A demo of Space3D and VDaT in an example real-time scene. Compare to the video in Reference [16].

dynamic scenes, such as virtual reality and other interactive multimedia.

Acknowledgements

Reproduced from Louis Pisha, Siddharth Atre, John Burnett, Shahrokh Yadegari, “Approximate Diffraction Modeling for Real-Time Sound Propagation Simulation”, Journal of the Acoustical Society of America (JASA) 148 (4), pp. 1922-1933, The Acoustical Society of America, October 2020, with the permission of the Acoustical Society of America.

Chapter 3, in full, is a reprint of the material as it appears in the Journal of the Acoustical Society of America (JASA) 148 (4). Pisha, Louis; Atre, Siddharth; Burnett, John; Yadegari, Shahrokh, The Acoustical Society of America, October 2020. The dissertation author was the primary investigator and author of this paper.

Bibliography

[1] Damian Murphy, Antti Kelloniemi, Jack Mullen, and Simon Shelley. "Acoustic modeling using the digital waveguide mesh". In: *IEEE Signal Processing Magazine* 24.2 (2007), pp. 55–66.

[2] Dick Botteldooren. "Finite-difference time-domain simulation of low-frequency room acoustic problems". In: *J. Acoust. Soc. Am.* 98.6 (1995), pp. 3302–3308.

[3] Thomas Funkhouser, Patrick Min, and Ingrid Carlbom. "Real-time acoustic modeling for distributed virtual environments". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1999, pp. 365–374.

[4] NVIDIA Corporation. "NVIDIA VRWorks – Audio". In: (2019). (Last viewed Feb. 19, 2019). URL: https://developer.nvidia.com/vrworks/vrworks-audio.

[5] Valve Corporation. "A benchmark in immersive audio solutions for games and VR". In: (2019). (Last viewed May 23, 2019). URL: https://valvesoftware.github.io/steam-audio/.

[6] Dear Reality GmbH. "DearVR: Ultimate tools for immersive 3D audio production". In: (2019). (Last viewed May 14, 2019). URL: https://www.dearvr.com.

[7] Robert G Kouyoumjian and Prabhakar H Pathak. "A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface". In: *Proceedings of the IEEE* 62.11 (1974), pp. 1448–1461.

[8] Maurice Anthony Biot and Ivan Tolstoy. "Formulation of wave propagation in infinite media by normal coordinates with an application to diffraction". In: *J. Acoust. Soc. Am.* 29.3 (1957), pp. 381–391.

[9] Herman Medwin. "Shadowing by finite noise barriers". In: *J. Acoust. Soc. Am.* 69.4 (1981), pp. 1060–1064.

[10] U Peter Svensson, Roger I Fred, and John Vanderkooy. "An analytic secondary source model of edge diffraction impulse responses". In: *J. Acoust. Soc. Am.* 106.5 (1999), pp. 2331–2344.

[11] T Kawai. "Sound diffraction by a many-sided barrier or pillar". In: *J. Sound and Vibration* 79.2 (1981), pp. 229–242.

[12] U Peter Svensson and Paul T Calamia. "Edge-diffraction impulse responses near specular-zone and shadow-zone boundaries". In: *Acta acustica united with acustica* 92.4 (2006), pp. 501–512.

[13] Paul T Calamia and U Peter Svensson. “Fast time-domain edge-diffraction calculations for interactive acoustic simulations”. In: *EURASIP J. Applied Signal Proc.* 2007.1 (2007), pp. 186–186.

[14] Lakulish Antani, Anish Chandak, Micah Taylor, and Dinesh Manocha. “Fast geometric sound propagation with finite edge diffraction”. In: *Technical Report TR10-011, University of North Carolina at Chapel Hill* (2010).

[15] László Szirmay-Kalos and Gábor Márton. “Worst-case versus average case complexity of ray-shooting”. In: *Computing* 61.2 (1998), pp. 103–131.

[16] Carl Schissler, Ravish Mehra, and Dinesh Manocha. “High-order diffraction and diffuse reflections for interactive sound propagation in large environments”. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 39.

[17] Carl Schissler and Dinesh Manocha. “Gsound: Interactive sound propagation for games”. In: *Audio Engineering Society Conference: 41st International Conference: Audio for Games*. Audio Engineering Society. 2011.

[18] Anish Chandak, Christian Lauterbach, Micah Taylor, Zhimin Ren, and Dinesh Manocha. “Ad-frustum: Adaptive frustum tracing for interactive sound propagation”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1707–1722.

[19] Paul T Calamia, Benjamin E Markham, and U Peter Svensson. “Diffraction culling for virtual-acoustic simulations”. In: *Acta Acustica United with Acustica* 94.6 (2008), pp. 907–920.

[20] Nicolas Tsingos and Jean-Dominique Gascuel. “Soundtracks for Computer Animation : Sound Rendering in Dynamic Environments with Occlusions”. In: *Graphics Interface ’97*. Kelowna, Canada, 1997. URL: https://hal.inria.fr/inria-00510105.

[21] Erik Salomons, Dirk Van Maercke, Jérôme Defrance, and Foort de Roo. “The Harmonoise sound propagation model”. In: *Acta acustica united with acustica* 97.1 (2011), pp. 62–74.

[22] Ville Pulkki and U Peter Svensson. “Machine-learning-based estimation and rendering of scattering in virtual reality”. In: *J. Acoust. Soc. Am.* 145.4 (2019), pp. 2664–2676.

[23] Uwe M Stephenson. “An energetic approach for the simulation of diffraction within ray tracing based on the uncertainty relation”. In: *Acta Acustica united with Acustica* 96.3 (2010), pp. 516–535.

[24] Uwe M Stephenson. “Simulation of multiple Sound Particle Diffraction based on the Uncertainty Relation-a revolution in noise im-mission prognosis; Part I: Principle and Method”. In: *Proc. of Euronoise* (2018).

[25] Stefan Weigand, Uwe M Stephenson, and Jochen Schaal. “Simulation of multiple Sound Particle Diffraction based on the Uncertainty Relation-a revolution in noise immission prognosis; Part II: Evaluation by Measurements”. In: *Proc. of Euronoise* (2018).

[26] Google. “Resonance Audio: fundamental concepts”. In: (2017). (Last viewed June 4, 2020). URL: https://resonance-audio.github.io/resonance-audio/discover/concepts.html.

[27] Herman Medwin, Emily Childs, and Gary M Jebsen. “Impulse studies of double diffraction: A discrete Huygens interpretation”. In: *J. Acoust. Soc. Am.* 72.3 (1982), pp. 1005–1013.

[28] Ruth Y Litovsky, H Steven Colburn, William A Yost, and Sandra J Guzman. “The precedence effect”. In: *J. Acoust. Soc. Am.* 106.4 (1999), pp. 1633–1654.

[29] Peter Svensson. “EDtoolbox: edge diffraction Matlab toolbox”. In: (1999). (Last viewed June 11, 2020). URL: https://github.com/upsvensson/Edge-diffraction-Matlab-toolbox.

[30] Nicolas Tsingos, Thomas Funkhouser, Addy Ngan, and Ingrid Carlbom. “Modeling acoustics in virtual environments using the uniform theory of diffraction”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, pp. 545–552.

[31] Jason E Summers. “Inaccuracy in the treatment of multiple-order diffraction by secondary-edge-source methods”. In: *J. Acoust. Soc. Am.* 133.6 (2013), pp. 3673–3676.

[32] F Richard Moore. “A general model for spatial processing of sounds”. In: *Computer Music Journal* 7.3 (1983), pp. 6–15.

[33] S. Yadegari. “Inner Room Extension of a General Model for Spatial Processing of Sounds”. In: *Proc. International Computer Music Conference*. Barcelona, Spain, Sept. 2005, pp. 1–4.

[34] NVIDIA Corporation. *CUDA Zone*. 2022. URL: https://developer.nvidia.com/cuda-zone.

Chapter 4

Specular Path Generation and Near-Reflective Diffraction in Interactive Acoustical Simulations

Submitted to IEEE Transactions on Visualization and Computer Graphics, June 2022; returned for revisions August 2022; resubmitted with revisions November 2022

4.0 Abstract

Most systems for simulating sound propagation in a virtual environment for interactive applications use ray- or path-based models of sound. With these models, the "early" (low-order) specular reflection paths play a key role in defining the "sound" of the environment. However, the wave nature of sound, and the fact that smooth objects are approximated by triangle meshes, pose challenges for creating realistic approximations of the reflection results. Existing methods which produce accurate results are too slow to be used in most interactive applications with dynamic scenes. This paper presents a method for reflections modeling called spatially sampled near-reflective diffraction (SSNRD), based on an existing approximate diffraction model, Volumetric Diffraction and Transmission (VDaT). The SSNRD model addresses the challenges mentioned above, produces results accurate to within 1–2 dB on average compared to edge diffraction, and is fast enough to generate thousands of paths in a few milliseconds in large scenes. This method encompasses scene geometry processing, path trajectory generation, spatial sampling for

diffraction modeling, and a small deep neural network (DNN) to produce the final response of each path. All steps of the method are GPU-accelerated, and NVIDIA RTX real-time ray tracing hardware is used for spatial computing tasks beyond just traditional ray tracing.

4.1 Introduction

Interactive simulation of acoustics in virtual environments is a central component of audio systems in applications such as games and VR. In particular, the accurate simulation of "early reflections"—sound bouncing once or a small number of times off of objects in the environment—is crucial in giving the listener a perceptual experience of presence in that specific environment. Wavefield simulations of sound propagation [1] [2] can be very accurate; effects such as diffraction and reflections arise naturally from the wave equation at the boundaries. In applications where the scene is fixed and only sources and receivers move, detailed acoustical profiles precomputed with wavefield methods can be used [3] [4] [5]; and in applications where the scene can be sufficiently approximated in 2D, real-time wavefield methods are on the horizon [6]. However, for large-scale, non-trivial 3D acoustical environments which may change dynamically, wavefield methods require far too much computation to run in real time [7].

To practically simulate acoustics interactively, ray- or path-based models of sound propagation are used, and algorithms are developed to approximate the results of wave effects using these models. These approximations help produce accurate frequency-dependent results in static configurations, but more importantly, if they are absent, acoustical paths will appear and disappear as the sources, receivers, and objects move. This leads to discontinuities in the output audio, which are often audible and perceptually distracting. There are two types of discontinuities, corresponding to the appearance or disappearance of direct paths and specular reflection paths respectively. In the first case, a direct path becomes occluded by an object, or conversely the occlusion ceases; this is called shadowed diffraction while the occlusion is present, or near-shadowed diffraction when the occluding object is near the direct path but not blocking it.

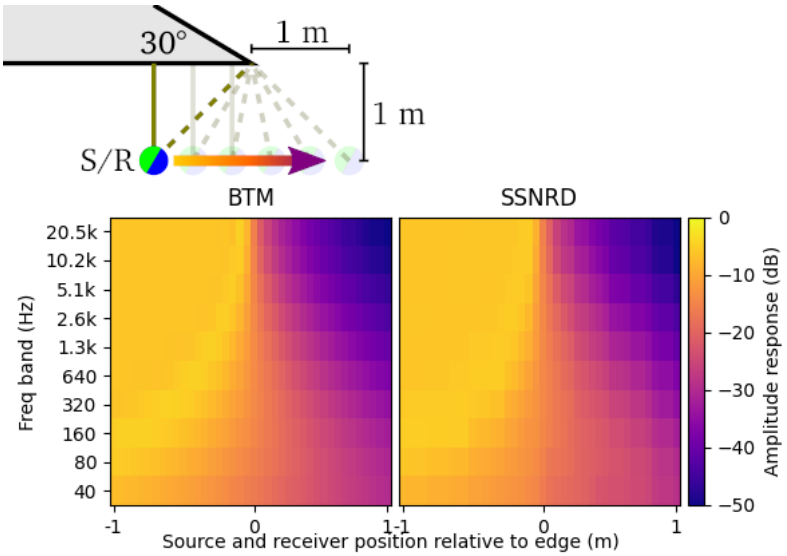

Figure 4.1. Changing magnitude response of a reflection path as the reflection point moves past the edge of the object. The sound source and receiver are colocated and move as shown in the upper diagram. SSNRD (right), the method described in this paper, closely matches the results of the BTM edge-diffraction model (left). © 2022 IEEE

Similarly, the second case, which will be called *near-reflective diffraction* (Figure 4.1), happens when an object moves so that a specular reflection path off of it "falls off" its edge and ceases to exist, or conversely moves in the opposite direction so that the specular reflection path comes into existence. In reality, not only does the sound smoothly fade at these boundaries, but this behavior is frequency-dependent, changing more quickly near the edge at high frequencies than at low frequencies.

The need for a model for shadowed diffraction is widely recognized; even basic implementations apply a generic low-pass filter when the direct path is occluded [8]. More sophisticated implementations based on edge diffraction (ED) models, typically the Uniform Theory of Diffraction (UTD) [9], have been in use for two decades [10] [11]. Unfortunately, UTD suffers from accuracy issues on detailed meshes with edges which are short compared to the wavelength, in contrast to the accurate but much more computationally intensive Biot-Tolstoy-Medwin (BTM) [12] [13] [14] edge diffraction model. Furthermore, a major challenge in these approaches is finding a set of relevant ED paths out of what is sometimes an astronomical

number of possible paths, with many approaches relying on precomputation and/or path caching [15]. Recently, Schissler *et. al.* developed an efficient ray tracing based algorithm for finding shadowed edge-diffraction paths in dynamic scenes [16]. For performance reasons, most ED implementations ignore near-shadowed diffraction. One non-ED model which supports shadowed and near-shadowed diffraction is the "uncertainty relation" model [17] [18], which traces rays around diffracting edges according to a stochastic distribution. In 2020, the present authors introduced the Volumetric Diffraction and Transmission (VDaT) model [19], which also handles shadowed and near-shadowed diffraction. This model spatially samples the scene around path segments with ray tracing, to determine where sound can traverse around obstacles, and then uses this data to approximate the BTM results with a numerical model.

In contrast to shadowed diffraction (and to a lesser extent, near-shadowed diffraction), near-reflective diffraction has received little attention. There are two additional challenges in simulating near-reflective diffraction besides avoiding discontinuities at edges. First, the size and shape of the meshes around the reflection affects the power and frequency content of the reflection. This information cannot be merely "baked" into the material properties if the meshes might move; for example, the reflection from a brick should sound very different based on whether it is alone in space or is part of a wall (which might be built dynamically at runtime). Second, smooth surfaces are approximated by triangle meshes, and geometric acoustics (GA) specular reflection paths may not exist for angles between adjacent triangle planes. The reflection normals can be interpolated between the triangles, like in Gouraud shading [20] in computer graphics, but there are some subtleties in applying this to acoustics which will be discussed below.

ED models can address all of these challenges, but only if near-reflective ED paths are generated. Because the overall reflection sound is produced in ED models from the sum of all the ED paths involving all the edges comprising a surface, a great number of these paths may be necessary if the meshes are detailed. Presumably because of the performance implications of this, it is not clear that there is any published acoustical simulation system designed for

interactive use which actually generates near-reflective edge diffraction paths. Furthermore, even if these paths were generated, the responses would have to be computed with the UTD model, which can be highly inaccurate for fine meshes, or the BTM model, which is very computationally intensive. Models based on stochastic diffuse reflections may also be able to handle near-reflective diffraction (see subsection 4.1.1), but again it is not clear that any implementation actually operates in this way.

This paper applies the principles from VDaT—spatial sampling and approximating BTM results—to near-reflective diffraction, creating a model called *Spatially Sampled Near-Reflective Diffraction (SSNRD)*. SSNRD also expands upon VDaT in three respects. First, SSNRD includes a set of algorithms for efficiently generating specular reflection paths with the necessary properties for near-reflective diffraction modeling. Second, while VDaT uses a heuristic-based numerical model for approximating BTM responses from the spatial sampling results, SSNRD uses a small deep neural network (DNN) for the corresponding task. Finally, SSNRD is designed to take advantage of the real-time ray tracing hardware called "RT cores" in NVIDIA RTX GPUs [21], and uses this hardware for spatial computing tasks beyond traditional ray tracing. All other processing is also done on the GPU with CUDA [22]. Together, SSNRD and VDaT form a complete system for interactively simulating the wave properties of acoustics by means of spatial sampling instead of edge diffraction.

4.1.1 Stochastic Methods

Another possible approach to the challenges of near-reflective diffraction is to ignore specular paths entirely and instead trace large numbers of diffuse paths with Monte Carlo sampling [23]. The distribution of reflection angles (BRDF) could give more weight to near-specular reflections, and this distribution could be frequency-dependent so that low frequencies are scattered more widely and high frequencies are more focused, potentially producing realistic behavior near edges. Generating diffuse paths by Monte Carlo has been widely discussed [24] [25] [15], but it is not clear that this technique has been used in order to address the challenges

with near-reflective diffraction described above. In particular, because scattering behavior would need to be frequency-dependent, the diffuse path tracing would have to be done separately for each frequency band, which does not appear to be typically done in practice.

This approach was not pursued for the present work for two main reasons. First, it is not clear *a priori* whether enough Monte Carlo samples could be made in real time to achieve sufficient accuracy without any perceptible artefacts from constantly varying sets of paths [26], especially for the critical early reflections. Dealing with this sort of noise in the output is a well-known challenge in Monte Carlo ray tracing for computer graphics. Path caching [15] is a viable option when the scene does not change quickly, but having the quality drop when things move quickly or when sound sources are first spawned is not ideal. Second, even given an arbitrarily large number of samples, this method would merely produce a room impulse response for the whole environment, every time the acoustics are simulated (typically once every 10–30 ms). A system could interpolate between these impulse responses between each pair of frames to produce continuous audio output, but this is not equivalent to simulation of individual continuously changing acoustical paths. The latter is needed for realistic phases and Doppler, especially on early reflections, and identifying paths uniquely across frames is not generally possible with Monte Carlo diffuse reflections. With that said, stochastic methods can be complementary to SSNRD, and used to accumulate late reflections for simulation of reverberation.

4.1.2 Acoustics on RTX

RT cores were introduced in 2018 in the Turing generation of NVIDIA RTX GPUs [27], primarily to bring real-time ray tracing to graphical rendering for games. NVIDIA developed VRWorks Audio [28] [29], which included RTX-based real-time acoustic modeling of virtual scenes. However, this software quickly stopped receiving support, and a full description of the algorithms involved was never published. To the authors' knowledge, the present paper is the first complete published method for interactive acoustical simulation using dedicated ray

tracing hardware in GPUs. Note that in many interactive applications, GPU resources are mostly occupied by graphics, and there are technical challenges in sharing GPU resources between graphics and non-graphics compute tasks. However, GPU compute, especially for AI tasks, is becoming increasingly important in gaming, and real-time ray tracing performance is always growing, so the authors expect these challenges will be solved in the coming years.

4.1.3 Overview of SSNRD algorithms

The algorithms comprising SSNRD are interdependent, and their design was influenced by the capabilities of the GPU and RT cores they were designed for. Nevertheless, leaving the implementation details aside, SSNRD operates as follows.

First, reflection normals are modeled to smoothly transition around convex edges of meshes, even if those edges represent real, sharp edges in the object being modeled. This enables reflection paths to be found at angles between the flat faces' typical normals. Conversely, concave edges are modeled as disconnected. Computing these normals involves information extracted from the meshes in fast, real-time compatible preprocessing (section 4.2).

Second (section 4.3), candidate reflection paths are traced from audio receivers, reflecting off objects according to the normals above, until they hit large virtual objects around audio sources. Then, these paths are iteratively refined to actually hit their point source. Path candidates are merged into discrete specular reflection paths, and linked to the corresponding paths which existed in the previous frame for continuity. These algorithms are able to generate thousands of acoustical paths in a few milliseconds in scenes with millions of triangles (section 4.6).

Third, to sample the space around each reflection point, rays are traced "into" the reflecting object in a pattern of concentric cylinders (section 4.4). The distance each of these rays travels before hitting the reflecting object (or any nearby meshes) provides information about those meshes' local size, shape, and the distance from any edges.

Finally, a small DNN is trained (section 4.5), whose input is the array of spatial sampling distances at a particular reflection point, and whose output is an estimate of the frequency-domain

magnitude response of the acoustical path at that reflection point. The ground truth which the network is trained to approximate is the magnitude response for a reflection off of the same object as simulated by the BTM edge diffraction model, which is often the sum of many near-reflective edge diffraction paths. The DNN approximation of BTM is typically accurate to within 1–2 dB (subsection 4.5.3).

4.2 Mesh Preprocessing

The method described in this paper, SSNRD, is the path generation system for *Space3D*, a real-time acoustic modeling and audio spatialization system for interactive applications such as VR and games. Space3D is designed to be used in game engines as an audio plugin, so it is essential to be able to quickly handle typical dynamic scene data from these applications. Furthermore, the game engine may arbitrarily deform objects, such as for skeletal animation, and submit the updated vertex positions to Space3D. In these cases, any preprocessing which is dependent on vertex positions must be repeated every frame, which excludes many types of processing-intensive mesh simplification techniques [15]. Fortunately, mesh simplification is not needed, as NVIDIA RTX real-time ray tracing is extremely fast even in very large scenes [30], and all the algorithms in the present method which interact with unknown parts of meshes do so via RTX. Whenever the game engine modifies the mesh vertices, two sets of data are calculated about each mesh for the use of the SSNRD path generation: mesh connectivity (subsection 4.2.1) and specialized vertex normals (subsection 4.2.2).

4.2.1 Connectivity

Connectivity is an integer, for each edge of each triangle, stating which triangle it is connected to, or -1 if it is not connected to any triangle. Triangles are assumed to be single-sided; mesh topologies where more than two triangles share an edge, or an edge of one triangle is coincident with only part of an edge or the face of another triangle, are not properly supported and will be considered not connected.

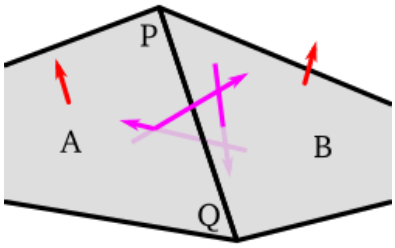

Figure 4.2. To identify which triangle, if any, is connected to triangle A along edge PQ, a triangular pattern of three rays (magenta) around the center of that edge is traced as shown here. Rays are directional and the mesh triangles are set as one-sided (red arrows), so at least one of these three rays will intersect triangle B, and none of them will intersect triangle A. The three rays slightly overlap at the ends (exaggerated here) to avoid possibly missing a triangle due to floating-point precision issues.

Connectivity information is internally used by some 3D editor programs [31], but is not present in most mesh file formats or either Unreal Engine or Unity. Therefore, it must be extracted from the mesh when instantiated. Connectivity not only depends on the mesh indices (an array stating which number vertices compose each triangle); many applications use "split vertices", where there are multiple vertices with the same position but different normals or texture coordinates. SSNRD must detect connected triangles regardless of whether they use the same or split vertices, meaning it must take into account the vertex positions. It must also detect whether the edges are intended to be "smooth" or "sharp" based on the normals if present (although these graphical normals are not used for the acoustic reflections, see subsection 4.2.2).

Since the OptiX / RTX acceleration structures [32] for the mesh are created anyway for the later ray tracing, a method was developed which uses the RT core to compute the mesh connectivity. (See subsection 4.3.3 for more background on similar topics, and section 4.6 for a performance comparison to a more traditional approach.) A small pattern of three rays (Figure 4.2) is traced around every edge of every triangle. Because triangles are one-sided, these rays are drawn in directions such that they will not hit the current triangle, but will hit any triangle connected to it with the normals pointed in the appropriate direction. Triangles hit with this method are checked to make sure they share two vertex positions with the current triangle.

The "radius" of this pattern of rays should be upper bounded by the minimum size of any detail in the mesh, and lower bounded by floating point precision considerations; the default radius is about 1 mm. Traversal by one ray of a triangle mesh in a bounding volume hierarchy (BVH) is typically $O(\log t)$, but due to the efficiency of the RT core and the extremely small length of the rays, the time taken for launching and processing results from rays is almost always longer than the actual acceleration structure traversal. Thus, this algorithm in practice appears to run in $O(1)$ time per ray, or $O(t)$ time for the whole mesh, which is extremely desirable for mesh preprocessing.

4.2.2 Reflection Normals

Vertex normals are widely used in computer graphics, and simply represent a unit normal vector to the mesh at each vertex. These normals can be interpolated at any point on a triangle to produce smooth (Gouraud) shading [20]. SSNRD precomputes the vertex normals of each mesh, but the normal at each vertex is stored per-triangle, as the normal at a given vertex may be different for each triangle sharing the same vertex. This is because, in SSNRD, concave edges are treated like disconnected edges (subsubsection 4.2.2). To compute the vertex normal for a particular triangle, the triangles sharing that vertex are iterated around in both directions until the iteration arrives at a concave or disconnected edge (or returns to the original triangle). The resulting convex group of triangles all contribute to a vertex normal according to their area and angle. If there are concave or disconnected edges, the tangent at these edges also contributes to the vertex normal. For details, see Appendix A or the source code in File 4.3.

Concave edges

Disconnected edges of a mesh are treated as if their normals are pointing outward, tangent to the triangle. (The final reflection normals used are not this extreme, see subsubsection 4.2.2.) This is so that reflection paths can be found pointing outwards near the edge, which is needed so that reflection paths do not sharply appear or disappear at the edge. Concave edges are treated

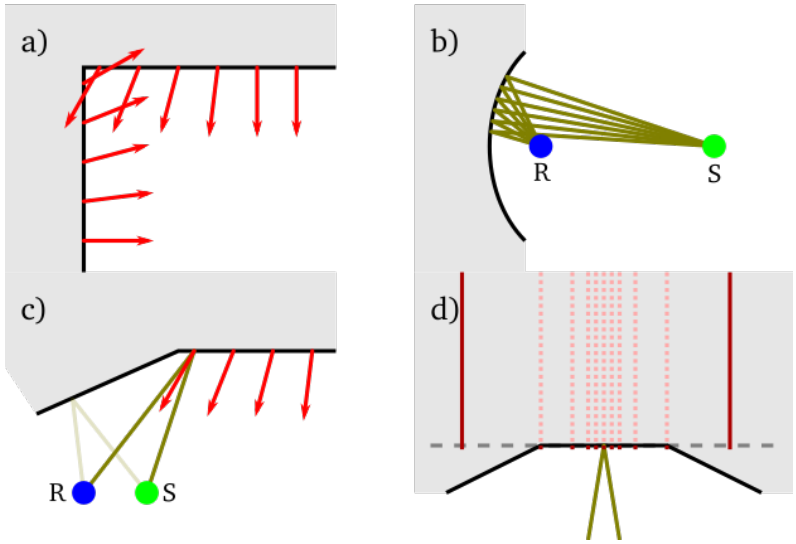

Figure 4.3. (a) Edge-on view of normals at a concave edge. (b) A continuous range of specular reflection paths may exist off a smooth, concave surface. (c) A reflection path is generated off the top face in this configuration because the normals curve outwards at the concave edge. If the normals were perpendicular to the face, this reflection path would disappear at the edge. (d) The spatial sampling (section 4.4) for one face of a concave surface only "sees" that face, not the neighboring ones.

the same way (Figure 4.3a). This is because on a surface with smooth, concave normals, specular reflection paths are not unique (Figure 4.3b), and therefore candidate paths cannot be refined into unique specular paths to be linked to "the same" paths in the next frame for continuity. With this method, there may only be one specular reflection path per convex portion of the mesh (e.g. per triangle), separated by concave edges. This produces expected results in concave spaces such as rooms, and also allows obtuse corners in these spaces to have smooth edge behavior (Figure 4.3c). Furthermore, as the triangles of a concave mesh get smaller, more specular paths may be found, but each one will have a frequency response reduced in magnitude at lower frequencies, as the triangles will appear to be smaller independent reflectors to the spatial sampling algorithm (Figure 4.3d). This avoids the total reflection energy growing as the mesh gets more detailed.

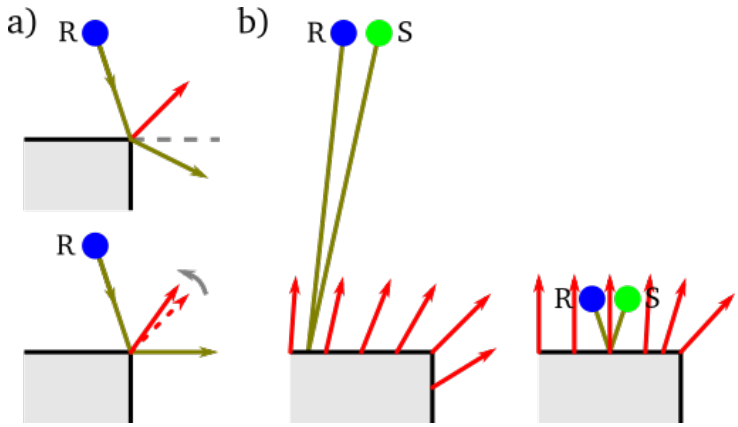

Figure 4.4. (a) The edge normal is adjusted—made more perpendicular—so that a reflection path cannot continue into the plane of the triangle. (b) The "curvature" of the normals around an edge is locally reduced as the receiver approaches the triangle. © 2022 IEEE

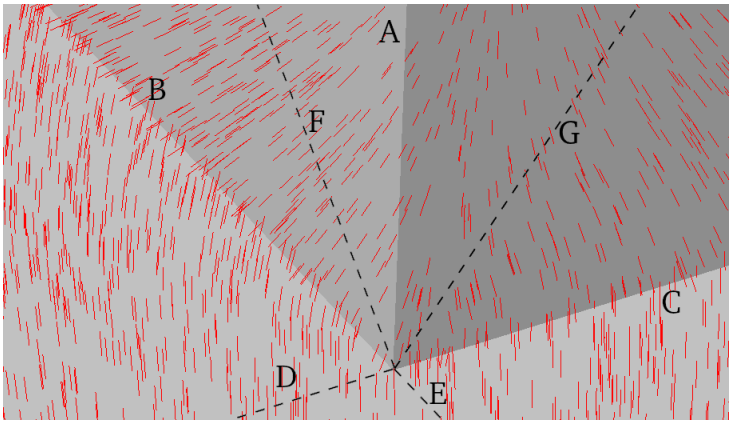

Figure 4.5. Visualization of normals computed at various points on the surface of a mesh. Note that the normals smoothly curve around the convex edge A, angle slightly outwards from each face at the concave edges B-C, and are continuous at the flat edges D-G, even though all these edges share the same vertex. © 2022 IEEE

Computing final reflection normals

When a ray intersects a triangle of a mesh, the normal is computed at that point. The algorithm for computing the normals is a set of heuristics, designed to achieve the following goals. First, the normals computed on two triangles sharing a convex edge must approach the same values from both sides of the edge, so that the normals are continuous around the edge (Figure 4.5). This is accomplished by computing an "edge normal", at the point on each edge closest to the reflection point, and in such a way that each triangle computes the same edge

normals for a shared edge. The normal at the original reflection point within the triangle is then interpolated from the three edge normals. Second, for rays hitting the triangle on the edge, the reflecting ray must not continue into the plane of the triangle. If it would, the edge normal is rotated toward the incoming ray until the reflecting ray is in the plane of the triangle (Figure 4.4a). Finally, as the ray origin approaches a triangle, the curvature of the triangle is reduced locally near the ray origin, but the triangle still remains curved near its edge (Figure 4.4b). This allows meshes to appear more flat for sources and receivers near them, and simultaneously appear more curved for paths traveling larger distances. See Appendix A or the source code in File 4.3 for the exact set of operations performed to compute the normals.

4.3 Path Generation

Generating specular reflection paths may seem to be a trivial process: trace some rays from[1] a sound receiver (e.g. microphone), reflect them off scene geometry, and see if they hit the sound source(s). Yet path generation is far from simple in SSNRD; at a high level, this is mainly for three reasons. First, in order for specular paths not to appear or disappear at edges of objects as described in section 4.1, these paths must be generated even "past" the edges of objects. Second, also as described in section 4.1, the spatialization algorithm in *Space3D* requires that paths exist over multiple consecutive frames, with their parameters possibly changing. So for example, if the scene does not change, the same set of specular paths must be found in this frame as in the previous frame; and if the scene changes slightly, some of the paths might move or change slightly but must still be identifiable as the corresponding paths in the previous frame. Finally, as many specular paths as possible should be found within the tight real-time constraints, including paths reflecting near the receiver and paths far from it.

There are four main steps of path generation: ray tracing (subsection 4.3.1), refining

[1]The propagation of sound can be simulated from source to receiver or receiver to source with theoretically identical results. However, in most VR and games applications, there are fewer receivers than sources. In this case, tracing rays from receivers allows those rays to be shared among the sources, and provides more detailed information about the scene acoustics near the receiver(s).

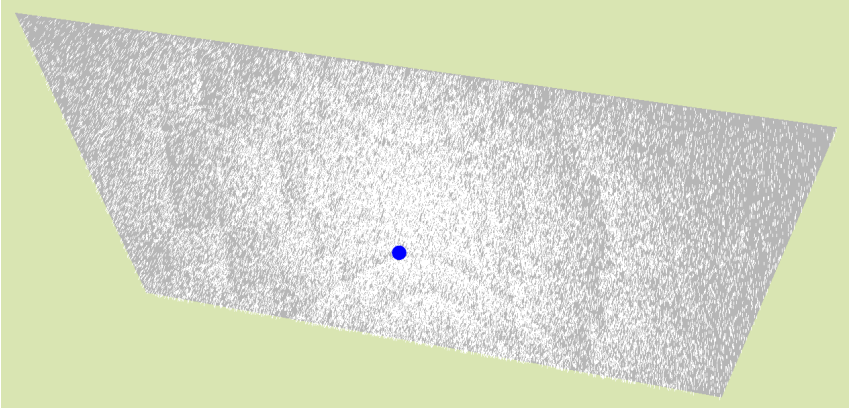

Figure 4.6. A white dot is drawn everywhere a ray intersects the mesh. The density of these rays on the mesh per unit surface area is roughly uniform, despite some parts of the mesh being much farther from the receiver (blue) than others. The visible pattern of nonuniformities in this density is due to the spherical pattern of ray distribution bins around the receiver. © 2022 IEEE

candidate paths (subsection 4.3.2), merging candidate paths with others nearby (subsection 4.3.4), and linking each path with the corresponding path in the previous frame for continuity (subsection 4.3.5). Path merging and path continuity both utilize a radius search algorithm (subsection 4.3.3).

4.3.1 Ray Tracing

First, the scene data structures required by the RT core are built or updated. NVIDIA OptiX 7 [33] accessed via OWL [34] provides the interface to the RT core and data structure setup. Meshes have BVH acceleration structures built, and are instantiated with their transformation matrices (as determined by the game engine or other program hosting Space3D). There are two top-level scenes created: one containing just meshes, for all the auxiliary ray tracing steps, and one containing both meshes and sources (described below) for the main path generation ray tracing. Also, a temporary top-level scene is created for each individual mesh when computing its connectivity (subsection 4.2.1).

Next, ray distribution sampling is performed. The goal of this step is to make the irradiance (density of rays per unit surface area) on scene geometry, by rays from the receivers during the main path generation ray tracing step below, roughly uniform (Figure 4.6). For

example, if a receiver is close to one wall in a room, and the rays from the receiver were distributed uniformly, many more rays will reflect off the nearby wall than the far wall, and so reflection paths involving the near wall will be much more likely to be found. In contrast, if the irradiance on all walls is roughly uniform, reflection paths anywhere in the room would have a more even chance of being found. To accomplish this, rays are traced from the receivers according to a uniform spherical distribution, and the average distance to a mesh intersection is computed in each of 384 bins of roughly equal angular surface area. Then, the inverse of the irradiance over these bins is used as the distribution of rays from that source for the ray tracing below. This step is computationally cheap and helps avoid missed paths in areas farther from the receiver as well as wasted computation on duplicated paths near the receiver. The RT core has been used for adaptive spatial sampling in other applications [35] [36].

Finally, the main path generation ray tracing is performed. Rays are emitted from receivers, with the direction of each ray determined by hashing its index within its bin and its bin number. This hash and therefore the ray direction does not depend on the number of rays per bin, so that the ray distribution sampling adds or removes some rays but does not change the trajectories of most of the rays every frame. Rays may intersect meshes or sources; in either case the OptiX anyhit (AH) program for that type of geometry is run.

Each ray emitted from the receiver is actually a "bundle" of n_B (e.g. 8 or 16) rays which are initially identical. When these rays intersect a mesh, half of them according to their index continue through (are transmitted), and the other half reflect according to the local mesh normal (subsection 4.2.2). If the bundle only contains one ray, because several reflections or transmissions have already occurred, the ray continues to do whatever it did more often in its history: a ray which has mostly reflected before will continue to reflect, and a ray which has mostly transmitted before will continue to transmit. Even if the meshes are not intended to have significant acoustical transmission, these transmission rays are essential for the VDaT diffraction algorithm [19]. This algorithm approximates diffraction around obstacles by spatially sampling the scene around an existing transmission path through the obstacle, and then applying

appropriate filtering to the path.

Sources are represented by OptiX "custom primitives" [32], which are effectively axis-aligned bounding boxes (AABBs) which the RT core reports ray intersections with. When an intersection occurs, the minimum distance the ray approaches the point source is computed, and the intersection is ignored if it is greater than a radius r, effectively modeling the source as a sphere. This source radius is computed on a per-ray basis as $r(d) = \frac{1}{2}\sqrt{n_T/\rho}$, $\rho = n_R/(4\pi d^2)$, where n_T is a target number of rays to hit each source, n_R is the total number of rays traced from the receiver, d is the total distance (including reflections) along the ray, and ρ is the effective ray density at this distance. The goal is to have roughly n_T (e.g. 25) rays hit each source, with this parameter adjustable to trade off between computation and probability of finding paths. ρ will be an accurate estimate of the ray density if all reflections are planar; practically when most reflections are from curved surfaces, the true ray density is lower. While r differs for every ray, the boxes must be set up in advance, and may be intersected by rays of various lengths d and orders o. Because of this, box sizes are computed based on estimated maximum path lengths of each order $r_{\text{box}}(o) = r(\hat{d}_{\text{max}}(o))$, and one box for each reflection order is instantiated at each source. Ray visibility masks are used to ensure that only rays of the appropriate order intersect each sized source box. When a ray intersects the source box and successfully passes the distance check, the path including this ray is saved to memory as a candidate path, and the ray continues through the source.

4.3.2 Path Refinement

The path refinement step converts candidate paths from the ray tracing, which approach a source within a certain minimum distance described above, to paths which directly hit the point source. If all reflections were off planes whose normals were all parallel, the exact reflection points needed to hit the source could be computed via the image source method [37]. However, since reflecting objects are usually "curved" based on their normals (and not curved according to a simple analytical function), direct computation is infeasible and an iterative method is used.

The ray from the receiver is perturbed slightly in two perpendicular angular directions. In each of these new directions, a path is traced, intersecting the same triangles at slightly different positions (with slightly different normals). These paths eventually approach the point source again, and intersect a plane through the point source which is normal to the original path segment that approached the point source. The offsets of these two new intersection points are computed and converted to multipliers on the original two perturbations, to compute a new perturbation which "should" exactly hit the point source according to this first-order approximation. A new path is traced in this direction, and its reflection points are now allowed to "slide" to connected triangles if they exit the bounds of the original triangles. Then, a new perturbation is computed and the process is repeated a small number of times. If the resulting position does not converge to within a distance δ within this number of steps, or if a valid path fails to be generated at any step, the path candidate is discarded. δ is set to half the path merging distance (see below) for the current order, ensuring that paths which do converge will be properly merged.

4.3.3 Radius Search

Both the path merging subsection 4.3.4 and path continuity subsection 4.3.5 steps below require performing a *radius search*: for each point in set A, find all points in set B which are within a certain distance according to a certain norm. In path merging, sets A and B are the same, and the goal is to find clusters of nearby points. In path continuity, A and B are paths in the current and previous frame respectively, and the goal is to find the nearest path in the previous frame to each path in the current frame. In both of these cases, the radius search must be done independently for every combination of source, receiver, and path order. The data being compared for each path is the coordinates of all of the reflection points of that path, which can be viewed as a single point in $3o$-dimensional space where o is the reflection order. Three algorithms to perform these radius searches are implemented, tested, and compared.

First, a brute force comparison of all pairs of inputs is implemented. This is the least efficient theoretically, but has almost no overhead, unlike the other approaches. Furthermore, this

can be parallelized over both A and B, whereas both of the approaches below are only parallelized over A and each thread does $O(\log(|B|))$ operations. If $|A| \ll N_t$ where N_t is the number of threads the GPU can execute in parallel, the approaches only parallelized over A are unlikely to be efficient.

Second, a traditional radius search based on Morton numbers is implemented [38] [39]. A Morton number is a single large integer, here up to 96 bits, which encodes integer coordinates of a point in two or more dimensions by interleaving the bits of each coordinate. First, a bounding box containing all the paths is computed. Next, each path is assigned its own Morton number, and these numbers are sorted. Finally, the radius search is performed, which searches the list and returns all points within a specified ℓ_∞ norm ball (hypercube) of the queried location. (Since the ℓ_2 norm is desired, all of the returned points are checked again based on their ℓ_2 distance.) Due to the structure of the Morton numbers, this search is possible in $O(\log(|B|))$ time per search point in A. However, the overhead is large because multiple GPU kernels are run to set up the data (though each one is $O(|B|)$ or $O(|B|\log(|B|))$. Also, the search itself involves a large number of bit manipulation operations, which the GPU has lower throughput for compared to floating-point operations.

Finally, the RT core was leveraged to implement a faster radius search. Since the Turing GPU architecture introduced RT cores in 2018 [27], the RT core's ability to traverse a BVH very efficiently in hardware has been applied to several types of problems outside traditional ray tracing [40] [35] [36]. The algorithm discussed here is an extension of the radius search introduced in [41] and [42] to higher-dimensional data. In the basic radius search, points B are converted into bounding boxes in a scene, centered at their point locations and with a radius equal to the search radius. Extremely short rays at each query point A are traced, and the RT core returns all boxes from B which are within the radius according to the ℓ_∞ norm. Like with the Morton numbers approach above, the actual paths are then checked for ℓ_2 distance.

The RT core only handles 3D data, but as discussed above the points here may be of dimension 3, 6, 9, etc. Furthermore, independent searches must be done for paths of each order,

source, and receiver, but it is not desirable to set up and ray trace independent scenes for each of these combinations, as this would lead to them all being processed serially. Instead, for each path p, the order $o(p)$, source index $s(p)$, receiver index $r(p)$, and all reflection points $x_i(p)$ are encoded into a single 3D point v:

$$v(p) = v_o o(p) + v_r r(p) + v_s s(p) + \sum_{i=1}^{o} M_i x_i(p) \tag{4.1}$$

v_o, v_s, v_r, and M_i are arbitrary, constant values used for all paths. v_o, v_s, and v_r are large vectors used to move the paths for each order, source, and receiver spatially away from each other. Their magnitude should be larger than the typical expected scene size but not so large that the floating-point precision is significantly reduced when farther from the origin. M_i are orthonormal (rotation) matrices, and one of them (e.g. M_1) can be the identity matrix. Their role is to break symmetries which may exist in the scene geometry, and are more likely to exist along the coordinate axes than at random rotations. They must be orthonormal to avoid changing the scale of the paths relative to the radius being searched. The search radius $\rho(o)$ for each order is slightly increased:

$$\rho'(o) = \sqrt{o}\rho(o) \tag{4.2}$$

as up to o orthogonal components of the input $3o$-dimensional point are additively combined into one component of v. In the worst case, these components are all equal to some value a, so their norm is $a\sqrt{o}$ but the length of their sum is ao. Therefore the ℓ_∞ search radius must be increased by a factor of $\sqrt{o}$ in order to ensure the point is found.

This projection from $3o$ to o dimensions can be considered a linear, scale-preserving spatial hash. Hash collisions—two widely separated paths which map to nearby 3D points—are rare due to the relatively small radius search distance compared to typical scene size. When these collisions do occur, these pairs of paths are discarded when their ℓ_2 distance is checked, so the final results are still correct.

4.3.4 Path Merging

The path merging step has the simple function of combining nearby path candidates into single paths. From a given receiver, reflecting off any ordered set of convex submeshes, to a given source, there will only be at most one possible specular reflection path. Therefore, any candidates traversing this set of geometry should coincide with each other after refinement, to a degree of accuracy which can be as high as desired at the cost of more computation. Since the merge distance and refinement tolerance are related, the chances of incorrect merging of paths can be theoretically reduced as far as desired. Also, for incorrect merging to occur, not only do separate convex submeshes need to be very close to each other, but their normals also need to be very similar, because otherwise the paths would diverge substantially after reflecting from the different normals.

The three radius search algorithms in subsection 4.3.3 are used to find nearby paths. For the brute force approach, the paths are still assigned Morton numbers, and this list is sorted and culled, so that paths which are so similar that they get the same Morton number are culled. The brute force search replaces only the expensive radius search step. With any of the three algorithms, a disjoint set data structure representing path "adjacency"—sets of paths which are all near enough to each other to be merged—is formed. Since this data structure is being generated by thousands of threads in parallel, it must be built exclusively using atomic operations. The implementation is similar to [43], but somewhat simpler in that the "union" and "find" operations do not overlap, and that the "find" operation need not simplify the graph due to the expected small sizes of disjoint sets. The reflection positions of each path which is not the representative element of any disjoint set are averaged into those of its representative element. The representative elements then become the final set of paths for the current frame.

4.3.5 Path Continuity

The role of the "path continuity" step is to link paths generated in the previous frame to "the same" paths in the current frame. Formally, given a method for generating specular paths which is continuous almost everywhere with respect to movement of the scene contents (sources, meshes, and receivers), the scene is assumed to move continuously from its state in one frame to its state in the next frame, and as such almost all the paths will morph continuously from one frame to the next. When changes which are inherently not continuous occur, such as adding or removing sources or meshes, then the paths involving these items do not have corresponding paths in the other frame, and the path continuity system must flag this for those paths.

Both the Morton number radius search and the RT core search (subsection 4.3.3) were implemented. Here also the identity of the receivers and sources are taken into account: for example, if source 0 was removed, then paths involving source 1 last frame will involve source 0 this frame. As a query path from the current frame finds nearby paths from the previous frame, it keeps track of which not-yet-linked path is the closest in ℓ_2 distance. When the query is finished, it atomically tries to link to this path, and if the path has already been "claimed" by this time, the query is repeated. This method is not guaranteed to produce consistent results when the distance paths move between frames is similar to or greater than the separation between different paths in the same frame, in which case the assignments may be random depending on GPU thread scheduling. Fortunately, these cases are not very common in practice, and the details of reflections off small objects of sound from fast-moving sources may not matter in many applications. In principle, this problem can be solved by reducing the audio buffer size and simulating the scene more frequently, until the rate of change of path reflection positions per frame is as small as needed.

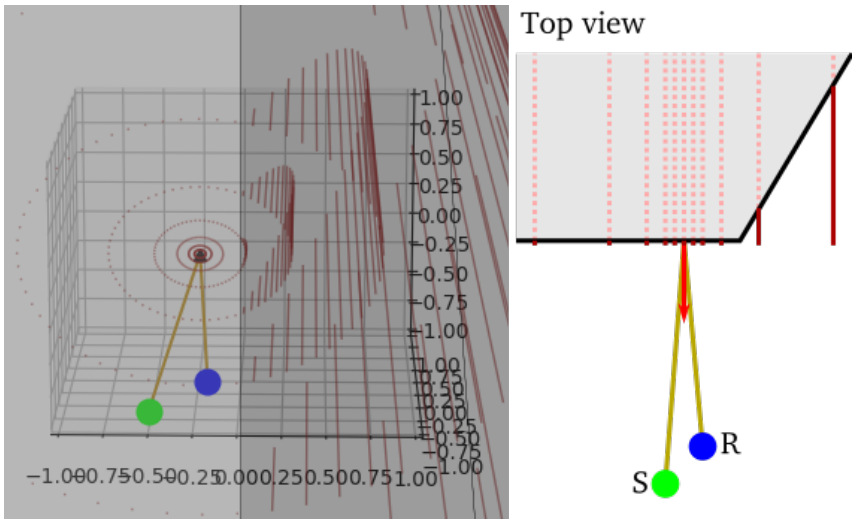

Figure 4.7. SSNRD spatial sampling. Rays are traced into the scene, in a pattern of concentric cylinders around the reflection point. The distance each of these rays travels until it hits an object is measured. © 2022 IEEE

4.4 Spatial Sampling

Once reflection paths are generated, the frequency response for each path must be computed. There are several factors which contribute to this in Space3D, including shadowed or near-shadowed diffraction as modeled by VDaT [19], but in this paper only the reflection response due to the geometry shape is considered. This is the equivalent of the sum of all near-reflective BTM edge diffraction paths involving the reflecting obstacle, plus the geometrical acoustics reflection path if it exists. For simplicity, this response is modeled separately at each reflection point and combined additively in dB. Space3D also applies material-dependent filtering at each reflection point, with the materials defined at each vertex and interpolated between them according to the reflection point. These filters can be measured from real examples of the material, or synthesized typically as low-pass filters. This filtering can be seen as converting the response of a single specular reflection path into the sum of one specular plus many diffuse reflections from the same object, though this is not strictly accurate for reflections above first order.

The concept of modeling diffraction by spatial sampling was introduced in the Volumetric

Diffraction and Transmission (VDaT) model [19], which was designed to approximate BTM results for shadowed and near-shadowed diffraction. Briefly, the motivation behind spatial sampling is that high frequencies traverse the scene in a narrow volumetric region around the ray path, and low frequencies spread out over a larger region. These regions can be sampled by tracing rays in a pattern of concentric cylinders, where each cylinder radius roughly corresponds to a frequency band. In VDaT, which rays are blocked provides information about what frequencies of sound are able to diffract around obstacles and reach the receiver. In SSNRD, the distance each ray travels before it hits (and therefore reflects from) an obstacle provides information about the local size and shape of the meshes, as discussed below.

If the occluding / reflecting object is planar, the BTM edge diffraction response is odd symmetric over the plane (see Appendix B for more details). As a result, the possibility was investigated of modeling near-reflective diffraction by running the VDaT algorithm on paths through the reflecting object and inverting the results. This was partially successful in some cases, but the inversion of the results often amplified small errors in the VDaT modeling. For example, if the (linear) amplitude response at some frequency for a shadowed diffraction case should be 0.99 and VDaT returns 1.0, that is a very accurate result; but if the amplitude in the corresponding near-reflective case should be 0.01 and VDaT returns 0.0, the error in dB is infinite. Furthermore, this model was not able to handle reflective geometry with obtuse angles like the example in Figure 4.7, as all VDaT paths are blocked and it returns a result of perfect reflections (which is incorrect).

To better model reflections, the spatial sampling approach has been modified from VDaT, and the processing of the spatial sampling results to estimate the reflection response is now accomplished with a small DNN (section 4.5) instead of a heuristic-based model. A set of concentric cylinders of rays, usually centered at the reflection point (see subsubsection 4.4), is traced into the scene, in the opposite direction of the reflection normal at the reflection point (Figure 4.7). The distance each ray travels before intersecting any triangle is computed by the RT core hardware and stored to memory, and this set of distances along all the rays is provided as

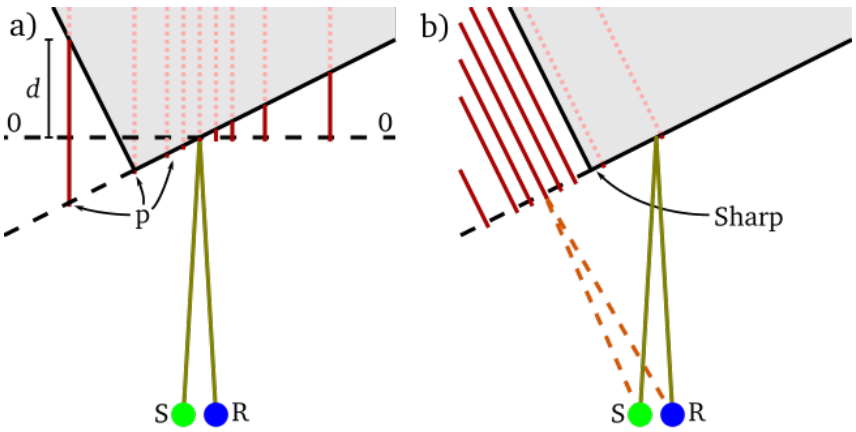

Figure 4.8. (a) The start points of spatial sampling rays are moved back to slightly behind the plane of the reflecting triangle (p), so that they will start outside the object. However, distance is measured with zero at the plane perpendicular to the reflection normal (horizontal dashed line), so for example the length of the leftmost ray is returned as d. (b) If the edge is sharp, the SSNRD rays are traced with a center and direction as if the intersecting triangle was an infinite, flat plane.

the main input to the DNN. The radii of the cylinders are chosen to span the sound wavelengths of interest; here, nine cylinders are traced with power-of-2 spacing, plus there is one central ray. Each radius is

$$r_i = c/f_i \tag{4.3}$$

where c is the speed of sound in air and $f_i = 40 \cdot 2^i$ Hz; the central ray corresponds to 20480 Hz. The number of rays traced around each cylinder is an arbitrary quality parameter, here set to 64. Both the number and spacing of the radii and the number of rays per cylinder could be changed, but this would require changing the input size of the DNN and retraining it.

The starting point of each ray is moved backwards until it is slightly behind the plane of the triangle with the intersection, or the plane perpendicular to the reflection normal, whichever is farther "back" (Figure 4.8a). This is to ensure that the rays do not miss the reflecting object when the reflection normal is not perpendicular to the triangle. However, the distance along the ray before it intersects an object is considered zero where the ray crosses the plane perpendicular to the reflection normal, and this reflection distance is clamped to zero at minimum, so any intersections behind that plane are also considered at zero distance.

SSNRD center point

For triangles with smooth edges, the center point of the cylinder of rays is the reflection point, and the normal is the reflection normal (Figure 4.7 and Figure 4.8a). However, for triangles with sharp (due to split vertices), concave, or disconnected edges, the center point is the reflection point off the triangle's flat plane, even if this point is outside of the triangle itself (Figure 4.8b). Similarly, the normal (which the rays are traced in the opposite direction of) is the triangle's face normal vector. If a triangle has a mixture of smooth and "sharp" edges, these two cases are interpolated based on the distance of the flat reflection point to each vertex. This is done so that the spatial sampling can distinguish between reflections near smooth edges, which are supposed to approximate underlying smooth objects, and sharp edges, which are actually present in the acoustical scene. Note that in the sharp case, the path reflection point is not moved to this new position; it is only used for the spatial sampling. This is because the length (delay) of the reflection path should be based on the sound reflecting from the actual triangle, not a virtual extension of it.

4.5 DNN for Reflection Response

A small (about 38k parameters) feed-forward deep neural network (DNN) is used to estimate the magnitude response of each path at each reflection point based on the spatial sampling results (section 4.4). More precisely, given a source, receiver, and a single convex mesh in a position where it is reflecting sound from the source to the receiver, the DNN is trying to estimate the total magnitude response in dB from source to receiver, excluding the direct path. In this paper, the DNN is trained based on the BTM edge-diffraction results—that is, the sum of all the edge-diffraction paths involving all the edges of the convex mesh, plus the geometrical acoustics specular reflection path if it exists. Alternatively, the DNN could be trained with data obtained from offline wavefield simulations or physical measurements. There are three reasons the mesh is assumed convex for the DNN. First, separate reflection paths are generated involving

each convex mesh or convex section thereof, and the response of each of these paths must be modeled separately. Second, higher-order reflection paths often exist between separate convex (sub)meshes, and for simplicity each reflection point on a path is modeled separately. Finally, BTM is not fully accurate on non-convex geometry [44], so it could not be used for generating ground truth training results in these cases.

The output format of the network is ten scalar values, representing magnitudes in dB at each of ten frequencies spanning the audible spectrum (the same ten frequencies used for the radii of the spatial sampling cylinders in section 4.4). The final magnitude response of the reflection is piecewise linearly interpolated between these points. BTM magnitude responses of individual edges are smooth and can be approximated this way with minimal error. Sums of BTM responses for multiple edges usually contain interference, but it is infeasible to model the exact interference pattern (other than by computing the BTM results), and for interactive applications a smooth response may be desirable, as interference patterns can lead to audible comb filtering when objects move.

4.5.1 Network Architecture

The primary input data to the network—the distance along each spatial sampling ray $d(r,\theta)$—is organized in two dimensions, where one (angle θ) is a circular dimension and the other (radius r) is a linear dimension. Thus, a circular convolutional network architecture [45] was selected. This is similar to a traditional convolutional neural network, except that in any circular dimensions in the input space, the input is circularly padded before the convolution, enabling the convolutional kernel to wrap around from the end to the start and vice versa.

The range of these input distances is $[0,\infty)$, so they are converted to a more usable range $(-1,1]$ for the network:

$$d'(r,\theta) = 2^{-d(r,\theta)/r} \cdot 2 - 1 \tag{4.4}$$

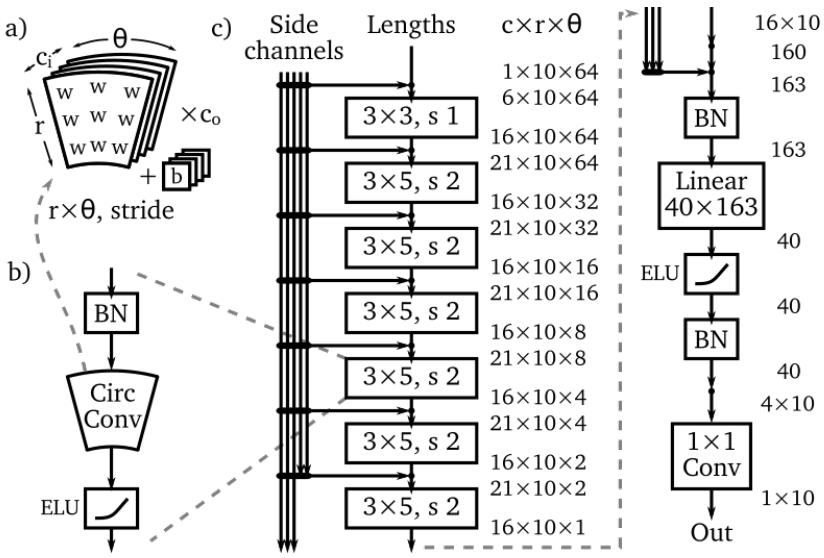

Figure 4.9. SSNRD DNN architecture. (a) The convolutional kernel of a circular convolutional layer. (b) One layer of the network. (c) The overall network structure, showing the data tensor sizes after each operation. © 2022 IEEE

Five side channel inputs are also provided:

$$s(r,\theta) = \begin{bmatrix} \ln(l_s + l_r) \\ \Delta\ell = \frac{l_s}{l_s+l_r} \cdot 2 - 1 \\ \cos(\phi) \\ \sin(\theta) \\ \cos(\theta) \end{bmatrix} \tag{4.5}$$

where l_s and l_r are the length of the source and receiver segments respectively and ϕ is the angle between the reflection normal and the source segment. The side channel inputs are concatenated along the channel dimension with the output of the previous layer (or d' for the first layer) as an input to each convolutional layer. Each convolutional layer has 16 output channels and a kernel size of 3 in r with "replicate" padding. The first convolutional layer has a kernel size of 3 in θ and a stride of 1, and all subsequent convolutional layers have a kernel size of 5 and a stride of 2, so that the size in the θ dimension is halved by these layers. These layers are repeated until the θ dimension is 1 (Figure 4.9).

After the convolutional portion of the network, the tensor dimensionality is $m \times 16 \times n_r$. This is put through a linear layer which maps $16 \times n_r$ to $4 \times n_r$, across all the radii (frequency bands). Finally, the tensor is put through a 1×1 convolutional layer which maps 4 channels to 1 using the same weights for each band. All of the convolutional and linear layers are preceded by batch normalization [46] and followed by the exponential linear unit (ELU) activation function [47], except that ELU is omitted at the very end of the network.

The output $y(r)$ is mapped to a change in magnitude response by the function

$$\Delta\hat{m}(r) = \begin{cases} \ln(y+1), & y \geq 0 \\ y, & y < 0 \end{cases} \tag{4.6}$$

which is intended to permit small increases but large decreases compared to the pure GA reflection response. The changes in magnitude at each reflection point are accumulated into the path's final estimated magnitude response

$$\hat{m}(r) = 20\log\left(\frac{1}{\sum_{i=0}^{o} l_o}\right) + \sum_{i=1}^{o} \Delta\hat{m}_i(r) \tag{4.7}$$

for order o and segment lengths l_o. Again, these ten r (radius) values map to octave frequency bands according to Equation 4.3.

4.5.2 Training Methodology

The training and test data for the DNN was generated by a set of Python scripts (File 4.3) which simulate SSNRD spatial sampling and BTM edge diffraction in randomly generated scenes of certain types. Four scene configurations were used (Figure 4.10):

1. A sharp wedge, of a random angle between 1 and 179 degrees, with the edge length much larger than the longest wavelength of interest. The reflection point is uniformly distributed between 2 meters before and 2 meters after the edge.

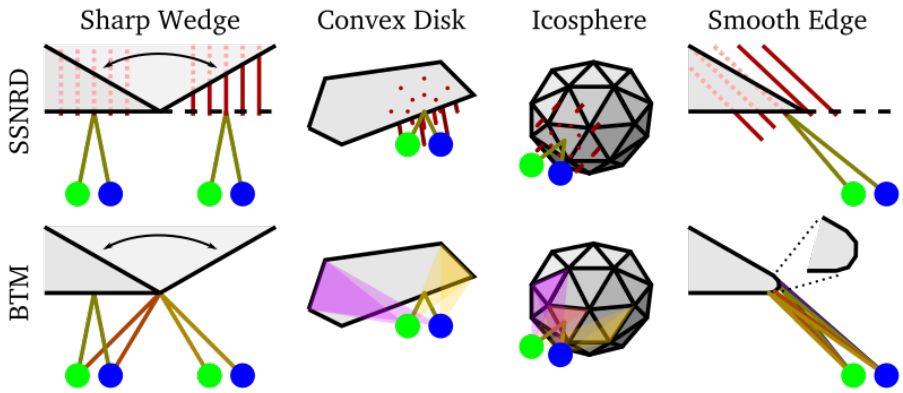

Figure 4.10. Example cases of the four scenarios used for training the SSNRD DNN. The diagrams for Convex Disk and Icosphere are 3D front views; the others are 2D "edge-on" top views. The SSNRD sampling locations and 3D BTM edge diffraction paths are simplified for clarity. © 2022 IEEE

2. A random, convex polygon disk, with a rough "radius" between 0.1 and 5 meters. The reflection point is a random distance (dependent on the radius) in a random direction from one of the edges.

3. An "icosphere" [48] (polygonal approximation to a sphere), with the reflection point on one of its faces.

4. A "smooth edge", see below.

In all cases the path length is uniformly distributed between 0.1 and 10 meters, and $\Delta\ell \in [-0.8, 0.8]$ (see Equation 4.5) except in case 4 where $\Delta\ell = 0$. Also, the direction from the intersection point to the source is random, with a weighting so that closer to perpendicular to the surface is more common.

The "smooth edge" scenario is intended to reflect the way that SSNRD path generation handles reflections smoothly interpolating around edges (subsection 4.2.2). A smooth edge in SSNRD is intended to approximate a curved surface, so it should be trained to match BTM results for the curved surface, not BTM results for a single edge. Thus, the spatial sampling results in this scenario are generated from a wedge mesh with a single smooth edge (section 4.4), but the BTM results are generated from a mesh with a polygonal "curved" surface consisting of

Table 4.1. SSNRD Network Scene Configurations and Test Set Error

Scene	Training Set Size	MAE (dB) over 1000 Test			
		Mean	Q1	Med	Q3
Sharp Wedge	4000	1.50	0.26	0.54	1.26
Convex Disk	16000	1.94	0.75	1.18	2.05
Icosphere	1600	1.75	0.51	0.88	2.27
Smooth Edge	4000	1.21	0.21	0.77	1.85

a few segments where the original edge was (Figure 4.10 lower right).

Training, validation, and test data were all generated from these scripts, with the quantities of each type in the training set shown in Table 4.1. Because all the examples were randomly generated, the training and test sets are disjoint. Furthermore, validation sets were used when adjusting the network architecture and hyperparameters; only once the final network was settled on and trained was the test set generated and evaluated, with no parameters adjusted after that time.

The loss function is mean absolute error (MAE) between the estimated path magnitude response Equation 4.7 at the ten frequencies f and the BTM magnitude response "near" those frequencies, obtained by convolving the BTM magnitude response $a_{\mathrm{BTM}}(\omega)$ with Gaussians as

$$a_{\mathrm{BTM}}(f) = \sum_{\omega=0}^{N_\omega - 1} a_{\mathrm{BTM}}(\omega)\, e^{-\frac{1}{2}(\log_2 \omega - \log_2 f)^2} \tag{4.8}$$

For the BTM ground truth, the sum of the geometrical reflection path if present plus all first-order BTM paths was used. Second-order BTM was implemented and tried, but it typically made a negligible contribution to the overall results while taking much more time to generate, so it was not used.

The network was trained with a minibatch size of 128, learning rate of $l(e) = 0.01 \cdot 0.97^e$ for epoch e for 300 epochs, and the Adam optimizer with weight decay of 10^{-5}. Training took roughly an hour on one NVIDIA GeForce RTX 3080 GPU, using TensorFloat-32 computations [49] for the convolutional and linear layers. For inference, the network was hand-implemented

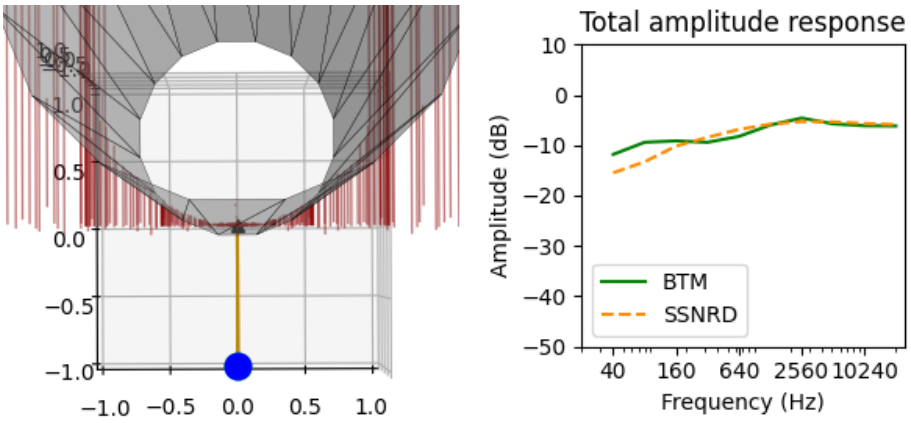

Figure 4.11. Example SSNRD network results for a reflection off a 1 m radius cylinder, with 1.3 dB mean absolute error compared to the BTM result. The training set did not include any cylinders. © 2022 IEEE

in CUDA as a single kernel, to avoid the performance penalty of copying data to memory between kernels performing individual operations in the network, and to avoid introducing a dependency on the very large libtorch library [50]. Unfortunately, this implementation does not take advantage of the Tensor Cores present in recent NVIDIA GPU architectures [27], so when there are more than a few hundred reflection points, the DNN inference becomes a bottleneck for the system's real-time performance. Integrating optimized Tensor Core matrix multiply routines into a single-kernel DNN implementation is out of scope of this paper and is planned as future work.

4.5.3 Results

Test set results, averaged over 1000 examples of each of the four trained scene configurations, are shown in Table 4.1. The mean absolute error (MAE) between the SSNRD and BTM results is less than 2 dB on average for all four types of scenes. The error tends to be largest when the source or receiver is close to the reflection plane, for example in the cases shown in Figure 4.12. The right plot in Figure 4.1 shows the SSNRD results as the source and receiver are moved past the edge, showing good agreement with the BTM results as well as smooth behavior as the inputs change. Figure 4.11 shows example results from SSNRD and BTM for a reflection

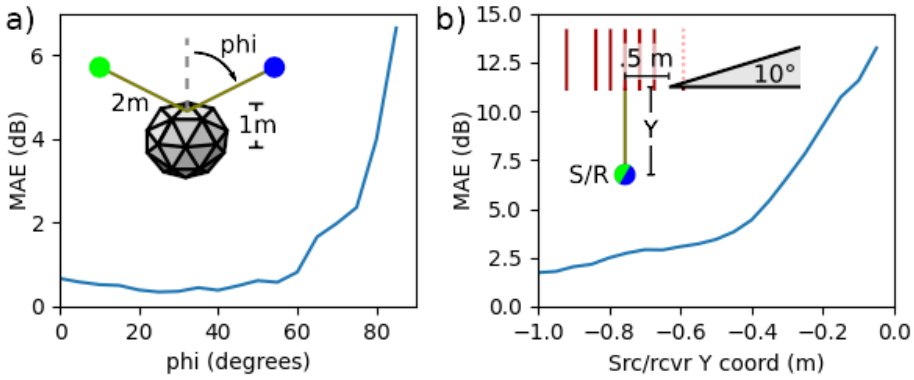

Figure 4.12. Example cases with larger errors as compared to BTM results. (a) As ϕ increases, the geometry transitions from a near-reflective diffraction case to more like a near-shadowed diffraction case. The latter would be handled better by VDaT than SSNRD. (b) For a wedge with a small angle and edge set to sharp, as the source/receiver approach the plane, the BTM response drops substantially but the SSNRD response does not. The error can be substantially reduced by setting the edge to smooth, though SSNRD still tends to overestimate the response. © 2022 IEEE

Table 4.2. Mesh Preprocessing Performance Examples

Scene	# Tris	Connectivity		Vertex
		RT Core	Edge hash	Normals
Kitchen 2	135k	830 µs	1.1 ms	681 µs
Bedroom 3	945k	2.5 ms	4.7 ms	1.3 ms
Kitchen 3	1.62M	5.7 ms	8.8 ms	3.7 ms
Office 8	3.08M	9.6 ms	16.7 ms	5.7 ms

from a cylinder, which is a type of scene the network never saw during training. Nevertheless, the error in the SSNRD results is only 1.3 dB.

4.6 System Results

All timing results are measured on one NVIDIA GeForce RTX 3080 GPU. Table 4.2 shows the time taken to preprocess some large meshes (section 4.2). The algorithms are fast enough to be run every frame when meshes are deformed, and the time appears to scale linearly with the scene size after an initial overhead. The RT core based connectivity method outperforms a method based on hashing triangle edges, similar to an existing approach [51]. The edge hash implementation is parallelized over all edges; multiple sets of hash table parameters were tested

Table 4.3. Radius Search Performance Examples

Scenario	Time for Algorithm, ±20 µs / ±3%		
	Morton cull + Brute force	Morton cull + Morton search	RT Core
Path Merge, # candidates → # paths			
577 → 172	235 µs	957 µs	773 µs
1956 → 667	395 µs	1108 µs	835 µs
4741 → 1941	638 µs	1202 µs	842 µs
Path Continuity, # cur frame to last frame links			
161	–	322 µs	568 µs
673	–	843 µs	613 µs
1848	–	2108 µs	689 µs

and the fastest were used. Note that any hashing implementation pays the memory and time cost of allocating a hash table, whereas the acceleration structures for the RT core implementation were already created for the main ray tracing and thus are not counted as a cost.

Table 4.3 shows example results for the three radius search algorithms in subsection 4.3.3. Except for small problem sizes, the RT Core implementation outperforms the Morton number based radius search, because the traversal of the scene structure is done in hardware. Table 4.4 shows example timing results for various processing steps discussed above in three medium-to-large scenes. The timings can change substantially depending on the scene geometry, source and receiver placement, and quality parameters. Nevertheless, these results are representative of the performance of each of these algorithms in real-world use: they are fast enough to be used in interactive applications, and only mildly dependent on the scene size.

Figure 4.13 shows three cases of Space3D system output: passing the edge of a plane, passing a cube, and in front of a rotating icosphere. In the first two cases, note that the response smoothly fades at the edges, with the high frequencies changing over a shorter distance than the low frequencies. In the icosphere case, note that the high frequency response changes depending on how close the reflection is to the center of a face versus an edge. Nevertheless, these changes are smooth as the mesh rotates, and at low frequencies where the wavelength is large compared to the size of the sphere, there is relatively little reflection energy. See Files 4.1 and 4.2 for videos

Table 4.4. Path Generation Performance Examples

Scene	Sibenik	Sponza	Living Room 3
# Tris	76.6k	262k	1.42M
Order, # Srcs	5, 1	3, 8	6, 2
n_R, n_B, $\hat{d}_{\max}(1)$	40k, 16, 50	80k, 8, 13.1	40k, 32, 13.1
Path Cand.	3269	10569	8020
Paths	316	2655	1389
BVH / setup	185 µs	217 µs	216 µs
Ray Tracing	1665 µs	1447 µs	3630 µs
Refinement	487 µs	325 µs	570 µs
Path Merge	270 µs	495 µs	438 µs
Spat. Sampl.	196 µs	693 µs	602 µs
VDaT	585 µs	2931 µs	2192 µs
Path Contin.	470 µs	552 µs	550 µs

All cases: $\hat{d}_{\max}(\{2,3,4,5,6\}) = \{1.5,2,2.5,3,3.5\} \cdot \hat{d}_{\max}(1)$, $n_T = 49$. See subsection 4.3.1 for explanations of params.

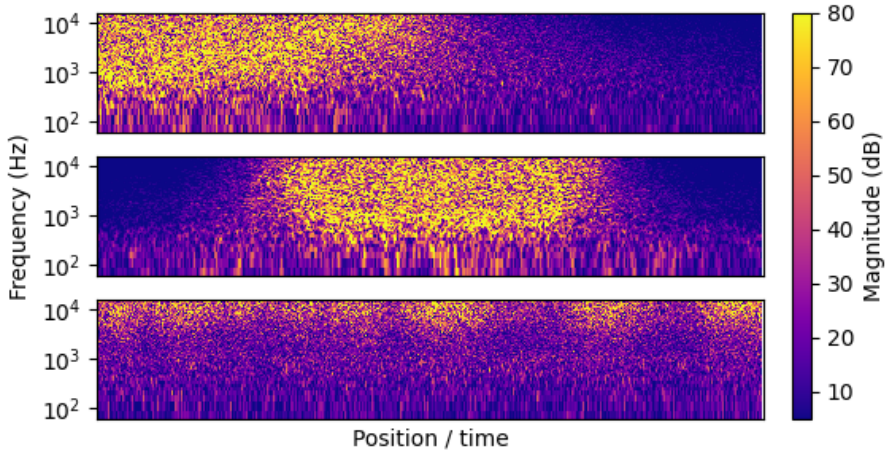

Figure 4.13. Spectrograms of Space3D output for white noise input, colocated source and receiver, and a single first-order reflection path off three different meshes. Top: Source and receiver move past the edge of a large plane (same as Figure 4.1). Middle: Source and receiver move past a cube. Bottom: A 1 m radius icosphere rotates in front of the source and receiver.

showing additional examples of real-time system output.

4.7 Conclusion

A method for generating and simulating the response of specular reflection paths in acoustical scenes is presented. This method is able to smoothly fade the magnitude response of reflection paths as they cross edges, take into account the size and shape of nearby geometry at a reflection point, and distinguish polygonal meshes representing smooth surfaces from those representing real edges. It achieves these goals through a combination of specific path generation methods, spatially sampling the scene with rays around reflection points, and approximating ground-truth edge diffraction results to within 1-2 dB with a DNN. The algorithms are fast enough to be used in interactive applications, partly thanks to NVIDIA's RT core, which is used for radius search and computing mesh connectivity in addition to traditional ray tracing.

Acknowledgements

The authors would like to acknowledge Valen Chang for assistance with creating the demo videos (Files 4.1 and 4.2).

Chapter 4, in full, has been submitted for publication of the material as it may appear in IEEE Transactions on Visualization and Computer Graphics. Pisha, Louis; Yadegari, Shahrokh, IEEE, 2022. The dissertation author was the primary investigator and author of this paper.

Appendix A: SSNRD Reflection Normals Equations

The code for the computation of normals as described in this Appendix is provided in File 4.3. The variable names shown here are chosen to generally match the code.

There are two main steps to computing normals in SSNRD. First, the mesh vertex normals are computed as a preprocessing step. These are computed and stored per-triangle: for a mesh with n_t triangles, there are $3n_t$ vertex normals. For a fully connected convex mesh, the vertex normals for each vertex shared by multiple triangles will be the same in each triangle; but for a mesh with concave or disconnected edges, a vertex may have different normals in different triangles. Note that vertex normals are often used in 3D graphics, and graphical vertex normals may already exist for meshes being processed by SSNRD. Nevertheless, due to differences in how disconnected, sharp, and concave edges are handled in SSNRD, graphical normals are not directly used by SSNRD, and the vertex normals are always computed as described here. Second, when a ray intersects the mesh during ray tracing, the reflection normal at that intersection point is computed, based on the triangle's vertex normals and other information. Both of these steps use precomputed mesh connectivity information, which is an integer for each edge of each triangle, stating what triangle index it is connected to or -1 if it is not connected.

4.8.1 Vertex Normals

The process described here is performed (on the GPU, in parallel) for each vertex index $v \in [0,2]$ for each triangle $t \in [0,n_t)$. First, the triangle vertices v_0, v_1, v_2 are loaded and "rotated" (circularly swapped) so that v_0 holds vertex v, v_1 holds vertex $(v+1) \mod 3$, and so on. Two vectors which will be accumulated into are initialized: the vertex normal $n_v = C(v_0, v_1, v_2)$, and the tangent sum $\sigma = 0$. The contribution $C(\cdot)$ of any vertex to the current vertex normal is

directly proportional to the angle at the vertex, and inversely proportional to the triangle area:

$$C(v_0, v_1, v_2) = n_f C_{\text{angle}} / C_{\text{area}} \tag{4.9}$$

$$C_{\text{angle}} = \arccos \frac{(v_1 - v_0) \cdot (v_2 - v_0)}{\|v_1 - v_0\| \|v_2 - v_0\|} \tag{4.10}$$

$$C_{\text{area}} = \|n_{\text{raw}}\| + 10^{-6} \tag{4.11}$$

$$n_f = \text{Nr}(n_{\text{raw}}) \tag{4.12}$$

$$n_{\text{raw}} = (v_1 - v_0) \times (v_2 - v_0) \tag{4.13}$$

where n_f is the "face normal" and $\text{Nr}(x) = x/\|x\|$ is the normalize function. Because n_v will be normalized at the end after the contributions from all triangles are added, this process is computing the weighted average of the face normals of each of the triangles, based on the angle and area weightings in Equation 4.9. Weighting by angle is unremarkable—triangles which occupy a larger angular range around the vertex should influence the normal more—but weighting by the inverse of area may be surprising. The reason is that when a mesh represents an underlying smooth surface, the face normals of small triangles are more likely to accurately represent the normal of the surface at the vertex than the face normals of large triangles, because the small triangles themselves represent the surface better. From another perspective, assuming the local surface curvature is roughly uniform, more of that curvature will occur where there are larger triangles, so the normals should more closely match those of the smaller triangles.

Two iterations are performed around v_0, one in each direction—one starting over each of the two edges of t connected to v_0. Each iteration traverses over an edge connected to v_0 to the next triangle, based on the connectivity. Each iteration terminates when the edge is not connected to any triangle, when it is a concave edge, or when the iteration has reached the original triangle t again. (If the first iteration around v_0 in one direction reaches the original triangle, the second iteration is skipped, as all the triangles have been taken into account.) At each step of the iteration, the new triangle is loaded as v_{0b}, v_{1b}, v_{2b}, and rotated as above so that

v_{0b} is v_0 and so that the correct edge is in place for the next step of the iteration. The contribution from this triangle $C(v_{0b}, v_{1b}, v_{2b})$ is added to n_v.

If the next edge is disconnected or concave, besides the iteration terminating, the edge's tangent vector τ is added to the tangent sum σ:

$$\tau_{\text{pre}}(v_0, v_1, v_2) = \text{Nr}(n_f \times (v_1 - v_0)) \tag{4.14}$$

$$\tau(v_0, v_1, v_2) = \begin{cases} -\tau_{\text{pre}} & \text{if} \tau_{\text{pre}} \cdot (v_2 - v_0) > 0 \\ \tau_{\text{pre}} & \text{otherwise} \end{cases} \tag{4.15}$$

The edge is considered concave if, for triangles a and b and edge e, $(\tau(e|a) \cdot n_f(b)) < -0.001$, i.e. the face normal of the next triangle is partly pointing in the opposite direction of the edge tangent. It is also considered concave if $\|n_f(a) + n_f(b)\| < 0.003$, i.e. the face normals are pointing in nearly opposite directions, which occurs for connected triangles forming opposite sides of the same flat surface. It is indeterminate whether the triangles are both facing outwards or both facing inwards, so it is important that this relatively common edge case be handled in a consistent manner rather than be subject to floating-point precision problems. Note that edges being smooth versus sharp is not taken into account; sharp edges do not affect the normals or path generation in SSNRD at all, only the spatial sampling step.

Once both iterations are complete, all the triangles surrounding v_0 via convex edges have made their contributions to n_v, and if there are disconnected edges, the tangents of the two disconnected edges have been added together in σ. The final vertex normal is computed as:

$$n_{v;\text{final}} = \text{Nr}\left(\text{Nr}(n_v) + 5\sigma\right) \tag{4.16}$$

The constant 5 was chosen heuristically; the goal is to have the vertex normal almost in the "outward" direction as indicated by the tangents of the disconnected or concave edges, if present. However, some component of the regular vertex normal n_v must be included, or else the reflection

normal algorithm below may not be able to determine which way to interpolate the normals along the edges. If there are no disconnected or concave edges, Equation 4.16 reduces to the weighted average of the face normals as discussed above.

4.8.2 Reflection Normals

When a ray from origin o hits a mesh triangle at intersection point i, the reflection normal n_r must be computed at that point so that the direction of the reflecting ray can be computed. The barycentric coordinates $b = (b_x, b_y)$ of the intersection point are returned by the RT core. Barycentric coordinates are a coordinate system in the plane of the triangle, not generally orthonormal, such that vertex 0 is at $(0,0)$, vertex 1 is at $(1,0)$, and vertex 2 is at $(0,1)$.

The triangle vertices v_0, v_1, v_2, the vertex normals n_{v0}, n_{v1}, n_{v2}, and connectivity are loaded, and the face normal n_f, edge vectors $e_0 = \mathrm{Nr}(v_1 - v_0)$, $e_1 = \mathrm{Nr}(v_2 - v_1)$, $e_2 = \mathrm{Nr}(v_0 - v_2)$, and tangent vectors $\tau_0 = e_0 \times n_f$, $\tau_1 = e_1 \times n_f$, $\tau_2 = e_2 \times n_f$ are computed. Next, for each edge k, the "edge reference point" η_k and "ratio" r_k are computed, representing the closest point on that edge to the reflection and how far along the edge it is:

$$n_y = e_0 \cdot (i - v_0) \tag{4.17}$$

$$\eta_0 = v_0 + n_y e_0 \tag{4.18}$$

$$r_0 = \mathrm{clamp}\left(\frac{n_y}{\|v_1 - v_0\|}, 0, 1\right) \tag{4.19}$$

and similarly for η_1, r_1 and η_2, r_2. The clamping is needed both because triangles may contain obtuse angles and η may be outside the triangle, and because during the path refinement step (which this code is also used for), the intersection point i itself may move outside the triangle.

Edge Normals

Next, the edge normal n_e for each edge at its respective reference point is computed. The equations are only shown for edge 0 but the corresponding operations are done for the other

edges as well. This is a multi-step process.

Interpolation between vertex normals

The edge normal is "circularly" interpolated between the vertex normals at each end of the edge, subject to some additional checks. First, each vertex normal is clamped to being "outwards" relative to the edge's extent, i.e. any component along the edge towards the opposite end is removed:

$$n'_{v0} = \begin{cases} \mathrm{Nr}(n_{v0} - e_0(n_{v0} \cdot e_0)) & \text{if} n_{v0} \cdot e_0 > 0 \\ n_{v0} & \text{otherwise} \end{cases} \tag{4.20}$$

$$n'_{v1} = \begin{cases} \mathrm{Nr}(n_{v1} - e_0(n_{v1} \cdot e_1)) & \text{if} n_{v1} \cdot e_1 < 0 \\ n_{v1} & \text{otherwise} \end{cases} \tag{4.21}$$

Next, the axis of rotation α is computed. If n'_{v0} and n'_{v1} point in nearly the same direction ($\|n'_{v0} - n'_{v1}\| < 10^{-4}$), it doesn't matter which way the interpolation is done, so set $\alpha = \tau_0$. If n'_{v0} and n'_{v1} point in nearly opposite directions ($\|n'_{v0} + n'_{v1}\| < 10^{-4}$), then an estimate of the edge normal in the middle of the edge m is needed, and $\alpha = e_0 \times m$. If the edge is concave, the interpolation should happen in the plane of the triangle outwards from it, so $m = \tau_0$; otherwise, the interpolation should happen towards the face normal, so $m = n_f$. Finally, if neither of the above conditions on n'_{v0} and n'_{v1} hold, the interpolation occurs in the plane defined by the two vertex normals, so $\alpha = n'_{v0} \times n'_{v1}$.

Next, the actual circular interpolation is performed. First, a third orthonormal basis vector

z, besides n'_{v0} and α, is constructed:

$$z_{\text{pre}} = \text{Nr}(n'_{v0} \times \alpha) \tag{4.22}$$

$$z = \begin{cases} -z_{\text{pre}} & \text{if} z \cdot n'_{v1} < 0 \\ z_{\text{pre}} & \text{otherwise} \end{cases} \tag{4.23}$$

Next, the total angle between the normals is computed:

$$\theta_{\text{total}} = \arccos\left(\text{clamp}(n'_{v0} \cdot n'_{v1}, -1, 1)\right) \tag{4.24}$$

The angle based on the ratio along the edge is computed:

$$\theta = \theta_{\text{total}} r_0 \tag{4.25}$$

Finally, the edge normal is computed:

$$n_{e0;A} = \text{Nr}(n'_{v0} \cos\theta + z \sin\theta) \tag{4.26}$$

Finally, if this normal is "backwards" relative to the plane of the triangle, it is clamped to the plane of the triangle:

$$n_{e0;B} = \begin{cases} \text{Nr}(n_{e0;A} - n_f(n_{e0;A} \cdot n_f)) & \text{if} n_{e0;A} \cdot n_f < 0 \\ n_{e0;A} & \text{otherwise} \end{cases} \tag{4.27}$$

For clarity, the notation $n_{e0;B}$ means "edge normal for edge 0, step B in the process of computing it".

Adjustment for outgoing ray direction

Next, the angle of the normal is reduced so that a reflecting ray does not continue into the plane of the triangle. This part of the algorithm assumes that the intersection point is η_0. Effectively, rays from a given origin which hit the triangle along its edges may reflect outward, in the plane of the triangle; and because of the interpolation between edge normals described below, rays which hit within the area of the triangle reflect at smaller angles and may cover the whole space above the triangle. Of course, if the original vertex normals are closer to the face normal than this, the range of possible reflecting angles will be smaller.

The incoming and outgoing rays are computed as:

$$v_{\text{in}} = o - i \tag{4.28}$$

$$v_{\text{out}} = 2(n_{e0;B} \cdot v_{\text{in}})n_{e0;B} - v_{\text{in}} \tag{4.29}$$

If $v_{\text{out}} \cdot n_f \geq 0$, the outgoing ray is in or above the plane of the triangle, so this step is skipped and $n_{e0;C} = n_{e0;B}$. Otherwise, the new outgoing ray is brought up to be in the plane of the triangle, and then the new normal is computed to be halfway between the incoming and outgoing rays:

$$x = \text{Nr}(v_{\text{in}} \times n_{e0;B}) \tag{4.30}$$

$$v_{\text{out;new}} = x \times n_f \tag{4.31}$$

$$n_{e0;C} = \text{Nr}(\text{Nr}(v_{\text{out;new}}) + \text{Nr}(v_{\text{in}})) \tag{4.32}$$

Adjustment for distance to plane of triangle

The final adjustment to the edge normal causes the triangle to appear more flat near its middle as the ray origin is closer to it. This adjustment is performed in order to take into account the scale of the reflecting object relative to the acoustical paths. For example, consider a cube with 30 cm edges; by default its reflection normals stick out in all directions, as if the normals of a sphere had been mapped onto its faces. For an acoustical path starting several meters away, this

is likely the desired behavior: it reflects sound in all directions (with the exact amount in each direction determined by the spatial sampling). However, if the source and receiver are both near each other, 5 cm from the middle of one of the cube's faces, that face should be considered flat near the source and receiver, as it is indeed flat when considered on the spatial scale of the paths.

This step is skipped, $n_{e0;\text{final}} = n_{e0;C}$, if the intersection point is on the outside side of the edge, $\tau_0 \cdot (i - \eta_0) \geq 0$. Otherwise:

$$d_i = \|o - i\| \tag{4.33}$$

$$d_e = \|i - \eta_0\| \tag{4.34}$$

$$c = \begin{cases} 1 & \text{if} d_i < 0.001 \\ \min(d_e/d_i, 1) & \text{otherwise} \end{cases} \tag{4.35}$$

$$n_{e0;\text{final}} = \text{Nr}(cn_f + (1-c)n_{e0;C}) \tag{4.36}$$

The "curve factor" c varies between 0, when the edge normal is as computed above and thus the curvature is maximum, and 1, when the face normal is used instead. When the distance from the origin to the intersection point, d_i, is less than or equal to the distance between the reflection point and the edge, d_e, the face normal is used. As the distance to the intersection point increases, i.e. as the origin gets farther away from the triangle, the result will approach the original edge normal. Note also that as the intersection point approaches the edge for a fixed d_i, the result also approaches the original edge normal. In the example above with the cube, this means that rays hitting near the cube's edges still reflect at larger angles, even though the cube is now "flat" near the source and receiver.

Combining Edge Normal Results

Once the edge normals $n_{e0} = n_{e0;\text{final}}$, n_{e1}, n_{e2} are computed, they are interpolated between based on the inverse of the distance from the reflection point to each edge w_0, w_1, w_2, which is

derived from the barycentric coordinates b:

$$b_w = 1 - b_x - b_y \tag{4.37}$$

$$w_0 = 1/b_y \tag{4.38}$$

$$w_1 = 1/b_w \tag{4.39}$$

$$w_2 = 1/b_x \tag{4.40}$$

$$n_{r;\text{pre}} = \text{Nr}\left(\frac{n_{e0}w_0 + n_{e1}w_1 + n_{e2}w_2}{w_0 + w_1 + w_2}\right) \tag{4.41}$$

$$n_r = \begin{cases} n_{e2} & \text{if} b_x \leq 10^{-5} \\ n_{e0} & \text{if} b_y \leq 10^{-5} \\ n_{e1} & \text{if} b_w \leq 10^{-5} \\ n_{r;\text{pre}} & \text{otherwise} \end{cases} \tag{4.42}$$

The clamping here avoids divide-by-zero and supports cases when the intersection point is outside the triangle during path refinement. n_r is then returned as the final reflection normal at the given intersection point.

Appendix B: Proof of Edge Diffraction Symmetry for Convex Planar Geometry

In general terms, according to the Biot-Tolstoy-Medwin (BTM) edge diffraction model [12] [13] [14], the total impulse response (IR) $\bar{h}(\tau)$ is equal to the geometrical acoustics (GA)

components plus the BTM diffraction component:

$$(1)\ \bar{h}_{\text{shd}} = h_{\text{BTM,shd}} \tag{4.43}$$

$$(2)\ \bar{h}_{\text{nsd}} = h_{\text{dir}} + h_{\text{BTM,nsd}} \tag{4.44}$$

$$(3)\ \bar{h}_{\text{nrd}} = h_{\text{BTM,nrd}} \tag{4.45}$$

$$(4)\ \bar{h}_{\text{rfd}} = h_{\text{refl}} + h_{\text{BTM,rfd}} \tag{4.46}$$

corresponding to the numbered cases in Figure 4.14 for shadowed diffraction, near-shadowed diffraction, near-reflective diffraction without a reflection path, and near-reflective diffraction with a reflection path. It is assumed that the object has rigid surfaces and does not have any absorption or transmission.

The claim is, roughly, that the pattern of energy removed from the field behind the occluder—i.e. the IR without the occluder, minus the diffraction IR—is equal to the pattern of the specularly reflected energy. When the reflection receiver is at the image of the diffraction receiver, the conditions for shadowed diffraction for one edge are the same as the conditions for a valid GA specular reflection path (Figure 4.14), so there are only two cases, (1) and (4) or (2) and (3):

$$h_{\text{dir}} - \bar{h}_{\text{shd}} \stackrel{?}{=} \bar{h}_{\text{rfd}} \tag{4.47}$$

$$h_{\text{dir}} - \bar{h}_{\text{nsd}} \stackrel{?}{=} \bar{h}_{\text{nrd}} \tag{4.48}$$

h_{dir} when the occluder is not present is equal to h_{refl} when the occluder is present, as both are GA paths of the same length without any modified frequency response, and the reflection from the rigid surface does not introduce a phase inversion. Substituting Equations 4.43–4.46, Equations

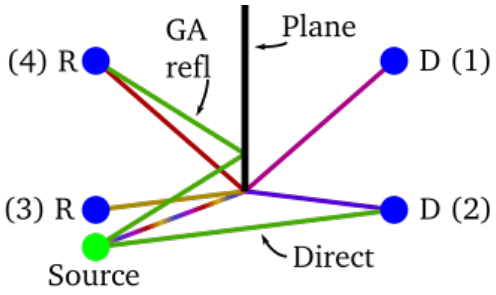

Figure 4.14. Four cases of BTM edge diffraction around a planar object. © 2022 IEEE

4.47 and 4.48 reduce to:

$$h_{\text{BTM,shd}} \stackrel{?}{=} -h_{\text{BTM,rfd}} \tag{4.49}$$

$$-h_{\text{BTM,nsd}} \stackrel{?}{=} h_{\text{BTM,nrd}} \tag{4.50}$$

or more generally

$$h_{\text{BTM,diff}} \stackrel{?}{=} -h_{\text{BTM,refl}} \tag{4.51}$$

where $h_{\text{BTM,diff}}$ is the BTM impulse response (without GA components) on the diffraction side, and $h_{\text{BTM,refl}}$ is the corresponding response for an appropriate position on the reflection side.

It can be shown that Equation 4.51 is not true for non-planar geometry, regardless of the choice of receiver positions. However, it is conjectured that:

Conjecture 1 *If the occluder is planar and the reflection receiver is at the image of the diffraction receiver (i.e. reflected across the plane), Equation 4.51 holds.*

Conjecture 1 is proved below for general first-order diffraction (subsection 4.9.1) and for second-order diffraction from convex occluders (subsection 4.9.2). It is conjectured to also hold for all higher orders of BTM edge diffraction and for non-convex planar occluders (subsection 4.9.3).

4.9.1 First-order diffraction

Infinite half-plane

The BTM IR for the infinite wedge is given in [13] as

$$h_{\mathrm{BTM}}(\tau) = -\frac{c\nu}{2\pi}\frac{\beta(\tau)}{r_S r_R \sinh\eta(\tau)}H(\tau-\tau_0) \tag{4.52}$$

$$\eta(\tau) = \cosh^{-1}\frac{c^2\tau^2-(r_S^2+r_R^2+z_R^2)}{2r_S r_R} \tag{4.53}$$

$$\beta(\tau) = \beta_{++}(\tau)+\beta_{+-}(\tau)+\beta_{-+}(\tau)+\beta_{--}(\tau) \tag{4.54}$$

$$\beta_{\pm\pm}(\tau) = \frac{\sin\left(\nu(\pi \pm_1 \theta_S \pm_2 \theta_R)\right)}{\cosh\left(\nu\eta(\tau)\right)-\cos\left(\nu(\pi \pm_1 \theta_S \pm_2 \theta_R)\right)} \tag{4.55}$$

where H is the Heaviside step function, $\nu = \pi/\theta_W$ is the "wedge index", and θ_W is the open angle of the wedge. The only term here which is dependent on θ_S and θ_R, the source and receiver angles, is $\beta(\tau)$. For clarity we write

$$\beta_{\pm\pm}(\tau) = \frac{N}{C+M} \tag{4.56}$$

$$C = \cosh\left(\nu\eta(\tau)\right) \tag{4.57}$$

$$\begin{matrix} N \\ M \end{matrix} = \begin{matrix} \sin \\ -\cos \end{matrix}\left(\pi\left(\frac{\pi}{\theta_W} \pm_1 \frac{\theta_S}{\theta_W} \pm_2 \frac{\theta_R}{\theta_W}\right)\right) \tag{4.58}$$

since most of the manipulations will be analogous on numerator and denominator.

For a planar "wedge", $\theta_W = 2\pi$ so

$$\begin{matrix} N \\ M \end{matrix} = \begin{matrix} \sin \\ -\cos \end{matrix}\left(\frac{\pi}{2} \pm_1 \frac{\theta_S}{2} \pm_2 \frac{\theta_R}{2}\right) \tag{4.59}$$

$$= \begin{matrix} \cos \\ \sin \end{matrix}\left(\pm_1\frac{\theta_S}{2} \pm_2 \frac{\theta_R}{2}\right) \tag{4.60}$$

These equations are valid for any source and receiver points. We now assume that R is the reflection receiver, whose impulse response contains β term β_{refl}, and D is the diffraction receiver with β term β_{diff}. The position D is R reflected over the plane, so $\theta_D = 2\pi - \theta_R$. Substituting:

$$\begin{aligned}
\begin{matrix} N_D \\ M_D \end{matrix} &= \begin{matrix} \cos \\ \sin \end{matrix} \left(\pm_1 \frac{\theta_S}{2} \pm_2 \frac{2\pi}{2} \mp_2 \frac{\theta_R}{2} \right) \\
&= \begin{matrix} -\cos \\ -\sin \end{matrix} \left(\pm_1 \frac{\theta_S}{2} \mp_2 \frac{\theta_R}{2} \right) \\
&= \begin{matrix} -\cos \\ -\sin \end{matrix} \left(-\left(\mp_1 \frac{\theta_S}{2} \pm_2 \frac{\theta_R}{2} \right) \right) \\
&= \begin{matrix} -\cos \\ \sin \end{matrix} \left(\mp_1 \frac{\theta_S}{2} \pm_2 \frac{\theta_R}{2} \right) \\
\beta_{\text{diff};\pm\pm} &= \frac{-\cos(\mp_1 \theta_S/2 \pm_2 \theta_R/2)}{C + \sin(\mp_1 \theta_S/2 \pm_2 \theta_R/2)} \\
&= -\beta_{\text{refl};\mp\pm}
\end{aligned}$$

Since all four combinations of the $\pm_1\pm_2$ terms are summed, swapping the first $\pm$ simply swaps the additions, giving the same result. So, $\beta_{\text{diff}} = -\beta_{\text{refl}}$. Since β contributes multiplicitavely to h_{BTM} and no other term depends on θ_R,

$$h_{\text{BTM;diff}} = -h_{\text{BTM;refl}} \tag{4.61}$$

Finite and non-straight edges

[13] gives different expressions for the diffracted pressure $p(t)$ from a finite edge which need not be straight, depending on whether integration is performed in time or along the edge. For example,

$$p(t) = \frac{-\nu}{4\pi} \int_{z_1}^{z_2} q \left[t - \frac{m(z) + l(z)}{c} \right] \frac{\beta}{m(z) l(z)} dz \tag{4.62}$$

where $q(t)$ is the source signal. This uses a different form of β, $\beta(\alpha(z), \gamma(z), \theta_S, \theta_R)$, but this change only affects the C term from Equation 4.57, not the terms containing θ_S and θ_R. So, the analysis above remains valid; $\beta_{\text{diff}} = -\beta_{\text{refl}}$, and since the negative sign can still be pulled out from β to the result, Equation 4.61 still holds.

4.9.2 Second-order diffraction

[13] gives a general equation for second-order diffraction, which is similar to Equation 4.62 but too long to reproduce here in full. This equation is valid whenever the diffraction path between the two edges lies along a surface, which is always true for convex objects (whether planar or not). This equation contains the terms

$$\beta(\alpha_1(z_1), \gamma_1(z_1, z_2), \theta_S, 0)\beta(\alpha_2(z_1, z_2), \gamma_2(z_2), 0, \theta_R) \tag{4.63}$$

which, like in Equation 4.62, multiplicitavely contribute to the result, and there are no other terms which depend on θ_S or θ_R. As an aside, this expression may be slightly incorrect. As pointed out in the caption of Fig. 4 in [13], the angles are θ_{S1}, θ_{w1} for edge 1 and $0, \theta_{w2} - \theta_{R2}$ for edge 2, so the expression in Equation 4.63 should be $\beta(..., \theta_{S1}, \theta_{w1})\beta(..., 0, \theta_{w2} - \theta_{R2})$. Compared to the published expression, this produces an extra factor of π which swaps the signs of the sin and cos terms in the numerators and denominators. The signs in the numerators cancel out, but the signs in the denominators do not. Computing the expression with one or both of the angles defined in the opposite direction leads to the same result. However, since in this appendix the reflection and diffraction results are being compared, this sign change would appear in both results, leading to the same conclusion regardless of which version of the equations is correct. The below proceeds with the seemingly corrected version of the equations.

$$\beta_{\pm\pm;n}(\tau) = \frac{N_n}{C_n + M_n} \tag{4.64}$$

$$C_1 = \cosh\left(\nu\eta\left(\alpha_1(z_1), \gamma_1(z_1, z_2)\right)\right) \tag{4.65}$$

$$C_2 = \cosh\left(\nu\eta\left(\alpha_2(z_1, z_2), \gamma_2(z_2)\right)\right) \tag{4.66}$$

$$\begin{matrix} N_1 \\ M_1 \end{matrix} = \begin{matrix} \sin \\ -\cos \end{matrix} \left(\pi\left(\frac{\pi}{\theta_{W1}} \pm_1 \frac{\theta_{S1}}{\theta_{W1}} \pm_2 \frac{\theta_{W1}}{\theta_{W1}}\right)\right) \tag{4.67}$$

$$= \begin{matrix} -\sin \\ \cos \end{matrix} \left(\pi\left(\frac{\pi}{\theta_{W1}} \pm_1 \frac{\theta_{S1}}{\theta_{W1}}\right)\right) \tag{4.68}$$

$$\begin{matrix} N_2 \\ M_2 \end{matrix} = \begin{matrix} \sin \\ -\cos \end{matrix} \left(\pi\left(\frac{\pi}{\theta_{W2}} \pm_1 \frac{0}{\theta_{W2}} \pm_2 \frac{\theta_{W2} - \theta_{R2}}{\theta_{W2}}\right)\right) \tag{4.69}$$

$$= \begin{matrix} -\sin \\ \cos \end{matrix} \left(\pi\left(\frac{\pi}{\theta_{W2}} \mp_2 \frac{\theta_{R2}}{\theta_{W1}}\right)\right) \tag{4.70}$$

Given that the obstacle is planar, $\theta_{W1} = \theta_{W2} = 2\pi$, so

$$\begin{matrix} N_1 \\ M_1 \end{matrix} = \begin{matrix} -\sin \\ \cos \end{matrix} \left(\pi\left(\frac{1}{2} \pm_1 \frac{\theta_{S1}}{2\pi}\right)\right) \tag{4.71}$$

$$= \begin{matrix} -\cos \\ -\sin \end{matrix} \left(\pm_1 \frac{\theta_{S1}}{2}\right) \tag{4.72}$$

and similarly

$$\begin{matrix} N_2 \\ M_2 \end{matrix} = \begin{matrix} -\cos \\ -\sin \end{matrix} \left(\mp_2 \frac{\theta_{R2}}{2}\right) \tag{4.73}$$

Now we assume the receiver $R2$ is at the end of a near-reflective diffraction path, i.e. on the same side of the obstacle as source $S1$. We compare the IR to that of the diffraction path with receiver $D2$, corresponding to $R2$ reflected over the plane, so $\theta_{D2} = 2\pi - \theta R2$. The N_1, M_1, and $\beta_{\pm\pm;1}$ terms do not change, so

$$\begin{matrix} N_{2;\text{diff}} \\ M_{2;\text{diff}} \end{matrix} = \begin{matrix} -\cos \\ -\sin \end{matrix} \left(\mp_2 \frac{2\pi - \theta_{R2}}{2} \right) \tag{4.74}$$

$$= \begin{matrix} \cos \\ \sin \end{matrix} \left(\pm_2 \frac{\theta_{R2}}{2} \right) \tag{4.75}$$

$$= \begin{matrix} \cos \\ -\sin \end{matrix} \left(\mp_2 \frac{\theta_{R2}}{2} \right) \tag{4.76}$$

$$N_{2;\text{diff}} = -N_{2;\text{refl}} \tag{4.77}$$

$$M_{2;\text{diff}} = M_{2;\text{refl}} \tag{4.78}$$

Since the numerators are negated and the denominators are the same for all of the $\pm$ combinations being summed, $\beta_{\pm\pm;2;\text{diff}} = -\beta_{\pm\pm;2;\text{refl}}$. Like for first-order diffraction, β contributes multiplicitavely to the final IR, and no other terms in the IR depend on θ_{R2}, so

$$p_{\text{BTM;diff;2nd}}(t) = -p_{\text{BTM;refl;2nd}}(t) \tag{4.79}$$

(second order) given the same source signal $q(t)$ and any convex planar occluder.

4.9.3 Non-convex geometry

[44] shows that, compared to the analytical solution and measurement results, the BTM secondary source formulation omits components in certain cases where the diffraction path travels from one edge to another not along a surface. This can only happen in second or higher order diffraction in a non-convex scene. More specifically, BTM predicts that for a gap in a planar occluder, the first-order diffraction from one edge has a zero magnitude at the other edge, so there is no second or higher order diffraction. However, this first-order diffracted component violates

the boundary condition on the plane forming the other edge, so there must be a higher-order component to compensate for this. This is a limitation of BTM, but this does not mean that Conjecture 1 is necessarily violated for this non-convex geometry, just that BTM cannot model this case.

Bibliography

[1] Damian Murphy, Antti Kelloniemi, Jack Mullen, and Simon Shelley. "Acoustic modeling using the digital waveguide mesh". In: *IEEE Signal Processing Magazine* 24.2 (2007), pp. 55–66.

[2] Dick Botteldooren. "Finite-difference time-domain simulation of low-frequency room acoustic problems". In: *J. Acoust. Soc. Am.* 98.6 (1995), pp. 3302–3308.

[3] Nikunj Raghuvanshi and John Snyder. "Parametric directional coding for precomputed sound propagation". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–14.

[4] Chakravarty R Alla Chaitanya, Nikunj Raghuvanshi, Keith W Godin, Zechen Zhang, Derek Nowrouzezahrai, and John M Snyder. "Directional sources and listeners in interactive sound propagation using reciprocal wave field coding". In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 44–1.

[5] Microsoft Game Dev. *Project Acoustics overview – What is Project Acoustics?* May 2022. URL: https://docs.microsoft.com/en-us/gaming/acoustics/what-is-acoustics.

[6] Matthew Rosen, Keith W Godin, and Nikunj Raghuvanshi. "Interactive sound propagation for dynamic scenes using 2D wave simulation". In: *Computer Graphics Forum*. Vol. 39. 8. Wiley Online Library. 2020, pp. 39–46.

[7] Jukka Saarelma, Jonathan Califa, and Ravish Mehra. "Challenges of Distributed Real-Time Finite-Difference Time-Domain Room Acoustic Simulation for Auralization". In: *Audio Engineering Society Conference: 2018 AES International Conference on Spatial Reproduction - Aesthetics and Science*. July 2018. URL: http://www.aes.org/e-lib/browse.cfm?elib=19609.

[8] Google. "Resonance Audio: fundamental concepts". In: (2017). (Last viewed June 4, 2020). URL: https://resonance-audio.github.io/resonance-audio/discover/concepts.html.

[9] Robert G Kouyoumjian and Prabhakar H Pathak. "A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface". In: *Proceedings of the IEEE* 62.11 (1974), pp. 1448–1461.

[10] Nicolas Tsingos, Thomas Funkhouser, Addy Ngan, and Ingrid Carlbom. "Modeling acoustics in virtual environments using the uniform theory of diffraction". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, pp. 545–552.

[11] Carl Schissler and Dinesh Manocha. “Gsound: Interactive sound propagation for games”. In: *Audio Engineering Society Conference: 41st International Conference: Audio for Games*. Audio Engineering Society. 2011.

[12] Maurice Anthony Biot and Ivan Tolstoy. “Formulation of wave propagation in infinite media by normal coordinates with an application to diffraction”. In: *J. Acoust. Soc. Am.* 29.3 (1957), pp. 381–391.

[13] U Peter Svensson, Roger I Fred, and John Vanderkooy. “An analytic secondary source model of edge diffraction impulse responses”. In: *J. Acoust. Soc. Am.* 106.5 (1999), pp. 2331–2344.

[14] Paul T Calamia and U Peter Svensson. “Fast time-domain edge-diffraction calculations for interactive acoustic simulations”. In: *EURASIP J. Applied Signal Proc.* 2007.1 (2007), pp. 186–186.

[15] Carl Schissler, Ravish Mehra, and Dinesh Manocha. “High-order diffraction and diffuse reflections for interactive sound propagation in large environments”. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 39.

[16] Carl Schissler, Gregor Mückl, and Paul Calamia. “Fast diffraction pathfinding for dynamic sound propagation”. In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–13.

[17] Uwe M Stephenson. “An energetic approach for the simulation of diffraction within ray tracing based on the uncertainty relation”. In: *Acta Acustica united with Acustica* 96.3 (2010), pp. 516–535.

[18] Alexander Pohl. “Simulation of diffraction based on the uncertainty relation”. PhD thesis. HafenCity Universität Hamburg, 2013.

[19] Louis Pisha, Siddharth Atre, John Burnett, and Shahrokh Yadegari. “Approximate diffraction modeling for real-time sound propagation simulation”. In: *J. Acoust. Soc. Am.* 148.4 (2020), pp. 1922–1933.

[20] Henri Gouraud. “Continuous shading of curved surfaces”. In: *IEEE Trans. Comput.* 100.6 (1971), pp. 623–629.

[21] NVIDIA Corporation. *NVIDIA RTX platform*. 2022. URL: https://developer.nvidia.com/rtx.

[22] NVIDIA Corporation. *CUDA Zone*. 2022. URL: https://developer.nvidia.com/cuda-zone.

[23] Lauri Savioja and U Peter Svensson. “Overview of geometrical room acoustic modeling techniques”. In: *The Journal of the Acoustical Society of America* 138.2 (2015), pp. 708–730.

[24] Jean-Jacques Embrechts. “Broad spectrum diffusion model for room acoustics ray-tracing algorithms”. In: *The Journal of the Acoustical Society of America* 107.4 (2000), pp. 2068–2081.

[25] Lakulish Antani, Anish Chandak, Lauri Savioja, and Dinesh Manocha. “Interactive sound propagation using compact acoustic transfer operators”. In: *ACM Transactions on Graphics (TOG)* 31.1 (2012), pp. 1–12.

[26] Chunxiao Cao, Zhong Ren, Carl Schissler, Dinesh Manocha, and Kun Zhou. "Interactive sound propagation with bidirectional path tracing". In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), pp. 1–11.

[27] NVIDIA Corporation. *NVIDIA Turing GPU architecture*. 2018. URL: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[28] NVIDIA Corporation. *VRWorks – audio*. 2018. URL: https://developer.nvidia.com/vrworks/vrworks-audio.

[29] NVIDIA Corporation. *VRWorks Audio SDK in-depth*. May 2017. URL: https://developer.nvidia.com/vrworks-audio-sdk-depth.

[30] Ingo Wald. *The elephant on RTX - first light, or: ray tracing Disney's Moana Island using RTX, OptiX, and OWL*. 2020. URL: https://ingowald.blog/2020/10/26/moana-on-rtx-first-light/.

[31] Blender community. *BMesh design document*. 2020. URL: https://wiki.blender.org/wiki/Source/Modeling/BMesh/Design.

[32] NVIDIA Corporation. *NVIDIA OptiX 7.4 Programming Guide: Acceleration structures*. 2022. URL: https://raytracing-docs.nvidia.com/optix7/guide/index.html#acceleration_structures.

[33] NVIDIA Corporation. *NVIDIA OptiX ray tracing engine*. 2022. URL: https://developer.nvidia.com/rtx/ray-tracing/optix.

[34] Ingo Wald, Nathan Morrical, Dylan Lacewell, Louis Pisha, Jefferson Amstutz, Stefan Zellmann, et al. *OWL: a node graph "wrapper" library for OptiX 7*. 2022. URL: https://github.com/owl-project/owl.

[35] Nate Morrical, Will Usher, Ingo Wald, and Valerio Pascucci. "Efficient space skipping and adaptive sampling of unstructured volumes using hardware accelerated ray tracing". In: *2019 IEEE Visualization Conference (VIS)*. IEEE. 2019, pp. 256–260.

[36] Nate Morrical and Stefan Zellmann. "Inverse Transform Sampling Using Ray Tracing Hardware". In: *Ray Tracing Gems II*. Springer, 2021, pp. 625–641.

[37] Jont B Allen and David A Berkley. "Image method for efficiently simulating small-room acoustics". In: *The Journal of the Acoustical Society of America* 65.4 (1979), pp. 943–950.

[38] Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. 1966.

[39] Herbert Tropf and Helmut Herzog. "Multidimensional Range Search in Dynamically Balanced Trees." In: *ANGEWANDTE INFO.* 2 (1981), pp. 71–77.

[40] Nathan Morrical, Ingo Wald, Will Usher, and Valerio Pascucci. "Accelerating unstructured mesh point location with RT cores". In: *IEEE Transactions on Visualization and Computer Graphics* (2020).

[41] Stefan Zellmann, Martin Weier, and Ingo Wald. "Accelerating force-directed graph drawing with RT cores". In: *2020 IEEE Visualization Conference (VIS)*. IEEE. 2020, pp. 96–100.

[42] I Evangelou, G Papaioannou, K Vardis, and AA Vasilakis. "Fast Radius Search Exploiting Ray-Tracing Frameworks". In: *Journal of Computer Graphics Techniques Vol* 10.1 (2021).

[43] Richard J Anderson and Heather Woll. "Wait-free parallel algorithms for the union-find problem". In: *Proceedings of the twenty-third annual ACM symposium on Theory of computing*. 1991, pp. 370–380.

[44] Jason E Summers. "Inaccuracy in the treatment of multiple-order diffraction by secondary-edge-source methods". In: *J. Acoust. Soc. Am.* 133.6 (2013), pp. 3673–3676.

[45] Stefan Schubert, Peer Neubert, Johannes Pöschmann, and Peter Protzel. "Circular convolutional neural networks for panoramic images and laser data". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 653–660.

[46] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[47] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (ELUs)". In: *arXiv preprint arXiv:1511.07289* (2015).

[48] Volodymyr Agafonkin. *Fast icosphere mesh*. 2019. URL: https://observablehq.com/@mourner/fast-icosphere-mesh.

[49] Paresh Kharya and NVIDIA Corporation. *TensorFloat-32 in the A100 GPU accelerates AI training, HPC up to 20x*. May 2020. URL: https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/.

[50] Stephen McDowell and PyTorch Contributors. *PyTorch C++ API*. 2019. URL: https://pytorch.org/cppdocs/.

[51] Jan Hradek, Martin Kuchař, and Vaclav Skala. "Hash functions and triangular mesh reconstruction". In: *Computers & geosciences* 29.6 (2003), pp. 741–751.