Real-time Auralisation of the Lower Frequency Sound Field Using Numerical Methods on the GPU

Master's Thesis

Nikolas Borrel-Jensen
mail@nikolasborrel.com

Date of submission: 07/03-12
Revised: 22/05-12

Supervisors:
Frank Wefers (RWTH University Aachen, Germany)
Jakob Grue Simonsen (University of Copenhagen, Denmark)

Institute for Technical Acoustics
RWTH Aachen University
Neustraße 50, Aachen, Germany
www.akustik.rwth-aachen.de

Department of Computer Science
University of Copenhagen
Universitetsparken 1, Copenhagen, Denmark
www.diku.dk

Abstract

We present a software system for simulating the lower frequency sound field in real-time using the FDTD method. The FDTD method is a numerical method for solving the wave equation by approximating the time and space derivatives by finite-differences. With the goal of achieving real-time performance, the FDTD method was implemented using CUDA on the GPU.

A family of 3-D non-staggered compact explicit FDTD schemes have been implemented incorporating a frequency-dependent boundary model that is consistent with locally reacting surfaces, taking the full 3-D wave field into account. The boundaries are modelled using digital impedance filters (DIFs), realised as IIR filters. Three GPU versions have been implemented using CUDA, with differences in the number of kernels used.

The physical correctness of the FDTD method has been compared with the FEM method for cubic rooms using seven FDTD schemes with frequency-dependent and -independent boundaries. In general, the results showed good correspondence between the sound fields simulated with the FEM and FDTD methods, when less than 2% of dispersion errors are allowed. The Standard Leapfrog method introduces fewest errors (less than 0.3 dB) and the Interpolated Digital Waveguide Mesh method introduces the most errors (less than 0.7 dB). The precision was also studied when oblique boundaries are present, and when discretisation errors due to meshing the scenes are introduced.

A performance test was done for three selected schemes in different environments, and it was shown that real-time simulations below the Schröder frequency is possible.

Resumé

Vi præsenterer et software system, der ved brug af FDTD metoder kan simulere det nedre frekvenslydfelt i real-time. FDTD metoden er en numerisk metode, der løser bølgeligningen ved at approksimere de spatiale og temporale afledte med endelige differenser. FDTD metoderne er blevet implementeret på GPU'en ved brug af CUDA med henblik på real-time simulering.

En familie af 3-D ikke-forskudte kompakte eksplicitte FDTD skemaer er blevet implementeret, indbefattende en frekvens-afhængig grænsemodel konsistent med lokalt reagerende overflader, og som tager højde for den fulde 3-D bølgemodel. Grænsemodellen er modelleret ved brug af digitale impedansfiltre (DIFs) formuleret som IIR filtre. Tre GPU versioner er blevet implementeret med variationer i antallet af kerner.

Den fysiske korrekthed af FDTD metoden er blevet sammenlignet med FEM simuleringsmetoden for syv FDTD skemaer i rektangulære rum for frekvensafhængig og -uafhængig grænseabsorbering. Resultaterne viste generelt en god overensstemmelse mellem lydfelterne simuleret med FDTD og FEM metoderne, når mindre end 2% dispersionsfejl tillades, hvoraf standard leapfrog metoden introducerer færrest fejl (mindre end 0.3 dB) og Interpolated Digital Waveguide Mesh metoden introducerer flest fejl (mindre end 0.7 dB). Præcisionen er blevet undersøgt, når skrå kanter er til stede, samt når diskretiseringsfejl under sammenvævningen af den originale scene er introduceret.

En hastighedstest er blevet udført i forskellige miljøer for tre udvalgte skemaer, og det er påvist at real-time simuleringer under Schroeder frekvensen er mulig.

Preface

This thesis was prepared in collaboration with the Institute for Technical Acoustics, RWTH Aachen UNiversity, Germany, and the Department of Computer Science, University of Copenhagen, in partial fulfilment of the requirements for acquiring the Master of Science degree in Computer Science. The work has been done at the Institute of Technical Acoustics, RWTH Aachen University, Germany. The main supervision was done by Frank Wefers. The work on this thesis was carried out from 1st of September to 7th of March with a workload of 30 ECTS points.

Aachen, March 2012

Nikolas Borrel-Jensen

Acknowledgements

I thank my supervisor Frank Wefers, Institute of Technical Acoustics, RWTH Aachen University, Germany, for his great commitment, helpfulness and technical assistance. Also a big thanks should go to Professor Michael Vorländer, Institute of Technical Acoustics, RWTH Aachen University, Germany, for accepting me as a student. A thanks goes to Jakob Grue Simonsen, Department of Computer Science, University of Copenhagen, Denmark, for his open-mindedness and receptiveness, as well as his readiness to give valuable advice.

A special thank goes to Konrad Kowalczyk for his help with clarifying aspects concerning the boundary model used in this work. Thanks to Jens Hugger, Department of Mathematics, University of Copenhagen, for fruitful discussions on numerical methods in general.

Finally, I wish to thank Marc Aretz for introducing me to the FEM simulation tool, and Pascal Dietrich for valuable advice concerning DSP.

Contents

Chapter 1

Introduction

Nowadays, most real-time auralisation systems are using efficient geometrical acoustics (GA) for modelling the sound field. GA has been used for precise modelling of room acoustics and is computationally inexpensive. The limitation of such methods is that they are only capable of modelling the higher frequencies above the virtual Schroeder frequency. In Virtual Reality (VR) systems, it is important to give the user a realistic perception of the environment in terms of graphics and sound, and therefore – concerning the sound – the whole frequency range should be modelled. In big rooms, such as concerts hall, the Schroeder frequency is very low making GA sufficient to modelling the audible frequency range, but in smaller environments, such as cars, the lower frequency range must be modelled for physical realistic room acoustics. To the author's knowledge, no previous work has investigated the physical correctness of simulations of the lower sound field in complex environments for numerical methods capable of performing in real-time.

In the following section, the reader will briefly be introduced to commonly used methods in the field of room acoustics and a literature review of important work in the field of numerical acoustics including Finite-Difference Time-Domain (FDTD) methods and Digital Waveguide Meshes (DWGM) will be given. The objective of the thesis is clarified, and finally a thesis overview is given.

1.1 Related Work

1.1.1 Geometrical Acoustics

The algorithms of typical programs for simulation of sound in rooms are based on geometrical acoustics and model the acoustical effects by applying ray theory. The description of the sound field is reduced to energy, transition time and the direction of rays. This approach is correct under the assumption that the sound wavelengths are significantly smaller than the dimension of the room and the size of obstacles in the environment.

These geometrical methods originate from similar methods used in computer graphics, and are used for finding significant paths from sources to receivers, along which sound can travel. Mathematical models are used to modify the emitted sound waves along each path by approximating filters corresponding to the characteristics of the room. The impulse response constructed for each of the paths can be considered in three parts (Vorländer, 2007): 1) direct sound representing the earliest arriving sound wave, 2) early reflections describing the sound waves arriving within the first milliseconds of the impulse response, and 3) late reverberation. The early reflections contain the most information about the room because of their relatively high strength, recognisable directionalities and distinct arrival times. In the late reverberation phase, the sound has reflected many surfaces in the environment to the extend the human ear is no longer able to distinguish the reflections independently.

Common practice is to use geometrical methods to find early reflections and use statistical methods for the late reverberation, since the error in geometrical approximation and computational complexity increase with larger number of reflection and diffraction. We can roughly divide the GA methods into ray-tracing, image sources and hybrid methods. In ray tracing methods, sound is radiated as a number of particles from a source to a receiver. These rays of particles are then followed through the environment until an appropriate set of rays has been found that reaches the representation of the receiver position. Image source methods compute specular reflections by introducing virtual image sources, where the source is mirrored at all wall planes recursively, creating image sources of higher order, and an "audibility test" is performed for all image sources detecting whether the image sources represent a specular reflection path to the source. Ray tracing has the advantage of being able to capture various kinds of reflections, but introduces aliasing and errors due to the discrete approximation of the continuous space of rays, whereas image sources guarantee that all reflection paths up to a given order are found, but only models specular reflections and comes with exponential computational complexity. Hybrid

image source methods combine these two approaches to obtain a finer temporal resolution in the sampling rate, a faster audibility check of sources and the ability to handle scattering.

The limitation of these methods is, that diffraction and scattering are not incorporated by the basic formulation and is hard to implement. Furthermore, only diffuse sound fields can be modelled, making it impossible to capture the modes of the waves present below the Schroeder frequency.

1.1.2 Finite-Difference Time-Domain Methods

The Finite-Difference Time-Domain (FDTD) method was originally proposed by (Yee, 1966) for electromagnetic simulations and employs finite differences as approximation to both the spatial and temporal derivatives that appears in Maxwell's equation. Botteldooren (Botteldooren, 1995) was the first to use FDTD methods in the field of acoustics, computing the sound pressure and velocity directly in discrete time steps by approximating the Laplacians and time-derivative by finite differences, making the modelling of transient acoustic properties easily feasible. He demonstrated that FDTD methods were useful for simulations of acoustical behaviour of rooms at low frequencies, though dispersion errors are introduced.

The benefits of these methods are, that diffraction and scattering are implicitly part of the wave equation incorporates, and that dynamic scenes – such as change in geometry and moving sources and receivers – can easily be handled, which is required in virtual reality systems. The drawback is that the computation increases with a factor r^4 in time and a factor r^3 in space, making the method only suitable for lower frequencies.

Recently, (Kowalczyk et al., 2011) have presented frequency-dependent boundaries for the family of 3-D compact explicit schemes based on non-staggered rectilinear grid, resulting in a digital impedance filter (DIF) boundary model derived from a 3-D perspective. Improved performance over other approaches commonly found in the FDTD and DWM literature is indicated. Because the same scheme is applied across the medium, the boundaries, the edges, and the corners, consistency of the schemes is realised, and a numerical stability proof is given.

In (Savioja, 2010), a GPU implementation using the Interpolated Wideband Scheme with frequency-dependent boundaries is presented. The results were obtained by allowing 10% of dispersion errors and using frequency-independent boundaries. For a room of size $7 \times 5 \times 2.8$ meter, the standard leapfrog scheme

was capable of simulating frequencies up to 1063 Hz, whereas the interpolated wideband scheme was capable of simulating frequencies up to 1509 Hz. For a room of size $40 \times 20 \times 15$ meter, the standard leapfrog scheme simulated frequencies up to 339 Hz, whereas the interpolated wideband scheme simulated frequencies up to 469 Hz.

In (Raghuvanshi et al., 2009) an adaptive rectangular domain decomposition method exploiting the known analytical solution of the wave equation in rectangular domains has been proposed, where FDTD methods are used across the domain boundaries. Exploiting the rectangular partitioning, the grids can be much coarser than those required in most numerical methods, reducing the order of magnitude for memory and computation by at least an order of magnitude, while still enabling high accuracy.

1.1.3 Digital Waveguides

The digital waveguide mesh (DWM) is a numerical, discrete-time simulation method based on a regular spatial sampling grid used to model wave propagation in an enclosed system. The continuous solution p to the 1-D wave equation is given by the D'Alambert equation (Murphy et al., 2007)

$$p(x,t) = p^-(x-ct) + p^+(x+ct) \tag{1.1}$$

where $p^-(x-ct)$ and $p^+(x+ct)$ represents arbitrary twice-differential fixed wave shape functions travelling at speed of sound c in the left and right directions, respectively. Sampling the continuous signal is carried out by the change of variables $x \to x_m = mX$ and $t \to t_n = nT$ leading to the discrete formulation

$$p_m(n) = p^+(n-m) + p^-(n+m) \tag{1.2}$$

where the notation $p_m(n) \equiv p(x_m, t_n)$ is used. This formulation can be implemented in an efficient and straightforward manner using two parallel delay lines representing the left- and right-going travelling wave. Scattering junctions together with the 1-D waveguide elements are the building blocks for modelling physical models of a vibrating system using digital waveguides. By interconnecting waveguides with scattering junctions, the behaviour of travelling waves in 2-D and 3-D can be modelled. Figure 1.1a shows four waveguides connected by a scattering junction S. The line segments with opposite arrows represent bi-direction delay lines, where Y_1, Y_2, Y_3 and Y_4 are the admittances of the propagation media. By locating these junctions in e.g. a rectilinear grid as depicted in Figure 1.1b, the sound field in cavities can be modelled.

The sound pressure p_m at junction m for L connected waveguides can be expressed using the *W-DWMs* formulation and the *K-DWMs* formulation. The

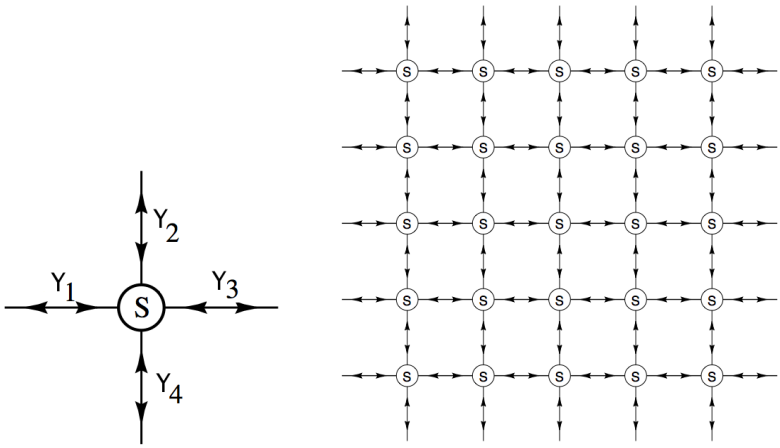


Figure 1.1: a) Lossless scattering junction S connecting four waveguides with admittances Y_1, Y_2, Y_3 and Y_4 corresponding to the propagation media, b) Lossless scattering junctions S connecting four waveguide each resulting in a rectangular 2-D grid. Source: (Duyne and Smith III, 1993)

W-DWM is formulated as (Murphy et al., 2007)

$$p_m(n) = \frac{2\sum_{l=1}^{L} Y_l \cdot p^+_{m,l}(n)}{\sum_{l=1}^{L} Y_l} \tag{1.3}$$

where $Y_l = 1/Z_l$ of each interconnection is called the admittance coefficients used to model the propagation media, and is defined as the inverse of the acoustic impedance Z. L is the number of junctions, and $p^+_{m,l}(n)$ and $p^-_{m,l}(n)$ are denoting the wave travelling towards the junction and away from the junction, respectively. For updating p_m at the next time-step $n+1$, the ingoing pressure value $p^+_{m,l}(n+1)$ must be computed. The pressure at a given location m is obtained by adding the ingoing and outgoing wave to a junction as

$$p^-_{m,l}(n) = p_m(n) - p^+_{m,l}(n) \tag{1.4}$$

and using the relation between the ingoing and outgoing wave to a junction, gives us the pressure for the ingoing pressure value at the next time step:

$$p^+_{m,l}(n+1) = p^-_{m,l}(n) \tag{1.5}$$

The above equations basically states, that if a wave enters a junction along the given directions, some portion of the wave is reflected back, and the rest is divided into the outgoing waves travelling through the junction.

The W-formulation is a generalisation of Eq. (1.2), which can be seen by setting $L = 2$ and $Y = 1$ (corresponding to a homogeneous grid in 1-D). The K-DWM formulation is given by

$$p_m(n) = \frac{2\sum_{l=1}^{L} Y_l \cdot p_l(n-1)}{\sum_{l=1}^{L} Y_l} - p_m(n-2) \tag{1.6}$$

The K-formulation is also called the *FDTD* formulation due to the fact that it is equal to the standard leapfrog scheme from the finite-difference literature (can be seen directly by setting $Y_i = 1$ for the 1-D case). In Section 2.4 we will show that the W-DWM can be regarded as a subclass of the FDTD method as well.

An advantage of W-formulation is its numerical robustness, the relative straightforward way of using fractional delays when building digital waveguides and the ability to commute losses to specific lumped points in the system (Karjalainen and Erkut (2004), Murphy et al. (2007)). In general, the W-formulation is the right choice when modelling in 1-D, but when going to higher dimensions, calculations must take place in every junction for every time-step. The advantages of the K-formulation are found when modelling mesh-like structures in 2-D and 3-D, since the overhead of calculating $p^+_{m,l}$ and $p^-_{m,l}$ is eliminated.

1.1.4 FEM/BEM

Finite Element Methods (FEM) and Boundary Element Methods (BEM) are numerical methods for solving linear partial differential equations (Hunter (2001), Vorländer (2007)).

In FEM, finite elements are created by discretising a field of volume into volume elements. In these elements, the energy formulation of the harmonic field is used and weighting functions are defined to represent the sound pressures within the elements. All elements' entries are combined into a "stiffness" matrix S, a mass matrix M and a damping matrix C, together with a matrix incorporating source contributions and boundary conditions, which is to be solved to obtain the sound pressures.

BEM solves the wave equation expressed in Green's equation describing how sound radiates from a point, and is re-arranged into the Helmoltz-Kirchoff's integral equation. The integral is discretised into a mesh and solved numerically by subdividing only the boundaries of the environment and assuming the particle velocity is a linear combination of a finite number of basis functions on the elements.

Both methods are usually used in the frequency domain and compute steady-state solutions. FEM and BEM methods are good choices when solving partial differential equations over complex domains and when the desired precision varies over the domain, making it possible to increase the precision for regions of particular interest. The drawback is, that dynamic scenes are difficult to model, making these models less suitable for virtual reality systems. The FEM method provides an accurate solution to the wave equation, but is mainly used at low frequencies since computational time and storage space increases dramatically with frequency.

1.2 Thesis Objective

This thesis will investigate whether it is possible to implement numerical methods for solving the wave equation in 3-D in real-time with focus on physical correct simulations of the lower frequency sound field.

Interactive scenes – such as moving objects around, opening a door or changing the material on the surfaces – must be supported in order to allow the user to explore the virtual environment in a realistic manner. We are only interested in modelling the lower frequency sound field, and therefore we will consider Finite-Difference Time-Domain (FDTD) for solving the 3-D wave equation in the time domain. Methods such as Finite Element Method (FEM) and Boundary Element Method (BEM) are valuable tools for simulating the lower sound field, and are widespread methods giving physically precise results. Though, interactive scenes are rather hard to incorporate, and real-time simulation is far from being realisable in the near future due to the computational load, making them only usable for offline modelling.

To accomplish the task of real-time modelling with FDTD, we will exploit that these numerical methods are suitable for parallelisation, and implement them using the massive parallel architecture of Graphic Processing Units (GPUs). Concerning the physical correctness of the simulations of the sound field, several aspects must be considered, namely dispersion errors introduced by solving the wave equation numerical, and the modelling of the boundaries. Dispersion errors implies that waves with different frequencies will travel with slightly different speeds, which does not correspond to the physical behaviour of wave dissemination in air. The boundaries should be frequency-dependent and modelled with the same characteristics as the boundaries encountered in the real-world, such that the sound is absorbed, reflected and scattered in a realistic manner.

The system should be implemented in such a way, that it can be integrated into

the virtual reality system VirKopf in the future. VirKopf is under development at the Institute of Technical Acoustics (ITA) at RWTH Aachen University and currently implements GA.

1.3 Thesis Overview

The prototyping of the system has been done in Matlab, whereas the final system has been implemented in C++. The prototyping focused primarily on understanding the methods and validating the physical correctness by comparing the results with the widely used FEM method. The prototype was implemented in C++, and afterwards a final version exploiting the GPU using CUDA was done with high performance in mind.

The structure of the thesis is given below:

In Chapter 2, the fundamentals for the work is reviewed, including the wave equation, sound wave reflections, sound fields in cavities, and finite-difference time-domain methods.

In Chapter 3, a family of non-staggered 3-D compact explicit schemes are introduced and seven specific scheme choices are given including a short discussion on dispersion errors. The formulation of transparent sources is derived and two impulses used for excitation of the system are proposed.

In Chapter 4, a boundary model consistent with locally reacting surfaces taking the 3-D wave equation into account is presented and update formulas for all point types are derived.

In Chapter 5, CUDA programming is introduced and three GPU implementations of the FDTD method are described. In Chapter 6, an overview of the software modules is given, including how grid and grid points are modelled, how the solving interface is designed and how grid point classification is done.

In Chapter 7, experiments concerning the physical correctness and performance are done.

Finally, conclusion and further work can be found in Chapter 8.

Chapter 2

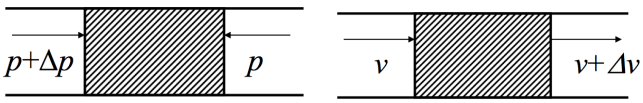
Fundamentals

In this section we will found the basics of finite differences, how reflections of sound waves can be described and how sound fields in cavities behave.

2.1 Wave Equation

In physics, waves are to be considered as disturbances in a physical medium with respect to time and space, resulting in the effect of energy transfer. To illustrate the effect of waves, we can imagine a chain of masses connected by springs. When one mass is initially moved by an external force, it transfers energy to the spring, which is compressed and transfers the energy to the next mass and so on. It is also intuitively clear, that heavier masses are moved more slowly, since more energy is needed compared to lighter masses. The intuition in this mechanic system can be directly transferred to the energy transport and the nature of wave speed in sound waves, where particles are displaced in space and time, which can be denoted by the vector $\mathbf{s} = \mathbf{s}(x, y, z, t)$, where x, y and z denotes the length and direction of the displacement and t is the time. The velocity is found by taking the derivative of the displacement vector

$$\mathbf{v} = \frac{\partial \mathbf{s}}{\partial t} \tag{2.1}$$



a) b)

Figure 2.1: Volume element in a one-dimensional fluid medium. a) Pressure variation Δp introduced by pushing from the left side, b) velocity variation Δv introduced by adding pressure form the left side.

By analogy to the mechanical system described above, the particles transfer energy from one point to another often with no permanent displacement of particles in the medium. Instead small local oscillations occur, introducing local pressure fluctuations in the medium, given by

$$p = p_{\text{tot}} - p_0 \tag{2.2}$$

where p_{tot} is the space- and time-dependent pressure resulting from the particle displacement and p_0 is the pressure of the medium at rest. In acoustics, the main quantity of interest is pressure, mainly because the human ear is sensitive to pressure and that the hearing system directly computes the sound on basis of that.

If the sound pressure is considered small $p \ll p_0$ (and the density follows the same prerequisite), the sound field can be described by only two linear sound field equations in terms of velocity and pressure (Kuttruff, 2000):

$$\nabla p = -\rho_0 \frac{\partial \mathbf{v}}{\partial t} \tag{2.3}$$

$$\nabla \cdot \mathbf{v} = -\frac{1}{\rho_0 c^2} \frac{\partial p}{\partial t} \tag{2.4}$$

where ρ_0 is the medium density ($\rho_0 = 1.42$ for air) and c is the speed of sound ($c = 344$ m/s in 21° C). The gradient and divergence in 3-dimensions are denoted by $\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}, \frac{\partial p}{\partial z}\right)$ and $\nabla \cdot \mathbf{v} = \left(\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial v}{\partial z}\right)$, respectively. These quantities are depicted in Figure 2.1. Consider a small volume element of thickness Δx in a one-dimensional fluid medium, and that the volume element is being pushed by a source with pressure $p + \Delta p$ by for example a piston mounted somewhere in the left side of the tube. A pressure difference Δp in the fluid medium at the left and right side of the volume element will lead to a force on the element, and this difference is denoted by the pressure gradient. In general, the pressure gradient will point in the direction, where the change is biggest. The divergence of the velocity is a scalar denoting the magnitude of the velocity field's source or sink at a given point. If we consider the heating of air in a

region, the velocity field will point outwards from the region and hence the divergence of the velocity of the particles in the region will be a positive value due to an expansion and hence act as a source. When cooling the air, the opposite will happen and the region will act as a sink. In b), the velocity field points outwards to the right (because of the injected pressure) and hence the velocity divergence will be a positive value.

By taking the gradient of Eq. (2.3) and inserting Eq. (2.4), we can express the sound field by only considering the pressure, which leads to the standard form of the wave equation (Kuttruff, 2000)

$$\Delta^2 p = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} \tag{2.5}$$

where $\Delta^2 p = \left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2}\right)$ is the Laplacian.

2.2 Reflections of Sound Waves

When a sound wave encounters a wall, a part is reflected and another part is absorbed by the wall material. The amount of absorption depends on the material property and determines the amount of reflected energy. Also, a phase change (delay) might occur if the wave can enter the material. The acoustic properties of a wall can be described by its wall impedance Z defined by the ratio of the sound pressure at the wall surface to the normal of the velocity at the same location:

$$Z = \frac{p}{v} \tag{2.6}$$

Another measure of the a wall's properties is given by the reflectance factor, defined in terms of the impedance

$$R = \frac{Z\cos\theta - Z_0}{Z\cos\theta + Z_0} \tag{2.7}$$

where $Z_0 = \rho_0 c$ is called the characteristic impedance of air. The specific wall impedance is defined as $\xi = Z/Z_0$, which - by rewriting the Eq. 2.7 - yields

$$R = \frac{\xi\cos\theta - 1}{\xi\cos\theta + 1} \tag{2.8}$$

and the corresponding formula for the specific wall impedances written in terms of the reflectance yields

$$\xi = \frac{1 + R\cos\theta}{1 - R\cos\theta} \tag{2.9}$$

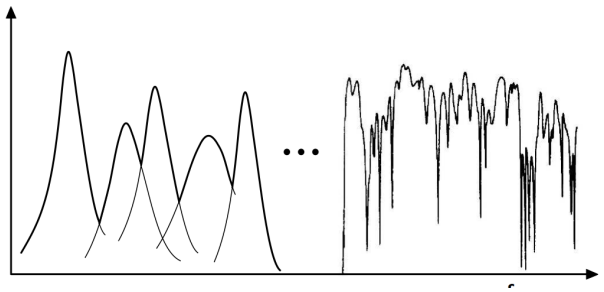


Figure 2.2: Overlapping room modes at low and high frequencies. Source: (Vorländer, 2007).

2.3 Sound Fields in Cavities

We will in the following describe the theory of normal modes leading to the so-called Schroeder frequency, for which geometrical methods can no longer yield a good approximation to the sound field. Modes can be explained as being the pattern of motion in which all parts of the system moves sinusoidally with the same frequency and with a fixed phase. Such a phenomenon occur when waves interfere at specific frequencies. In other words, the modes of a system is given when the superpositions of all the travelling waves result in a standing wave. A mode is characterised by a modal frequency and a modal shape, and is numbered according to the number of sinusoidal periods.

In 1-D, the relation between the wavelength λ, the dimension of the 1-D space L and the wave number k is given by

$$L = \frac{\lambda}{2}k, \quad k \in \mathbb{N} \tag{2.10}$$

The equation can be easily understood by for example considering the case where the standing wave only consists of one crest, giving us the wave number $k = \frac{L}{2L/2} = 1$. When 2 crests are present, $k = \frac{L}{L/2} = 2$, and so on. The wave number is nothing else than a number describing the shape and the frequency, dependent of the size of the geometry.

The Schroeder frequency is the limit, where the overlap of the modes in the cavity result in a sound field, where independent modes can no longer be studied independently and instead a diffuse sound field occur. In room acoustics, the density of the modes increases with frequency as depicted in Figure 2.2. The frequency in between small modal overlap and heavily overlapping modes is

called the Schroeder frequency and is calculated by the following formula

$$f_s \gg 1200\sqrt{\frac{T_{60}}{V}} \tag{2.11}$$

where f_s is the Schroeder frequency, T_{60} is the reverberation time in seconds and V is the volume in m^3. In a room with perfectly reflecting boundaries, each mode can be uniquely mapped to exactly one frequency since the modes do not overlap, but for higher wall impedances the modes becomes wider and their overlap regions increase accordingly. This fact is reflected in Schroeder's equation where the reverberation time is $T \to \infty$ for perfectly reflecting boundaries, and hence the Schroeder frequency $f_s \to \infty$ for fixed V. Accordingly, $f_s \to \infty$ for $V \to 0$ and T fixed. The reverberation time is traditionally described by the time required for a level decrease of 60 dB, and can be approximated using Sabine's equation (Vorländer, 2007)

$$T_{60} = 0.16\frac{V}{S\bar{\alpha}} \tag{2.12}$$

where S is the total surface area of the room in m^2, and $\bar{\alpha} = \frac{1}{S}\sum_{i=1}^{N} S_i\alpha_i$ is the average absorption coefficient of the room surfaces. The product $S\bar{\alpha}$ is denoted *equivalent absorption area* and is the total absorption of the room measured in sabins.

2.4 The Finite-Difference Time-Domain Method

The idea behind the FDTD method is to replace all derivates in the wave equation (2.5) by finite-differences. Different approximations can be used, where the simplest is the second-order centered finite-differences. The second-order centered finite-differences w.r.t. t is given as

$$\frac{\partial^2 p}{\partial t^2} = \frac{p^{n+1}(x_i, y_j, z_k) - 2p^n(x_i, y_j, z_k) + p^{n-1}(x_i, y_j, z_k)}{\Delta t^2} + O(\Delta t^2) \tag{2.13}$$

where Δt is the temporal sampling resolution and $O(\Delta t^2)$ is the temporal truncation error. The superscript notation n denotes the time index and is a shorthand for the temporal variable t_n. The approximation to the Laplacians with grid spacing Δx is given by

$$\frac{\partial^2 p}{\partial x^2} = \frac{p^n(x_{i+1}, y_j, z_k) - 2p^n(x_i, y_j, z_k) + p^n(x_{i-1}, y_j, z_k)}{\Delta x^2} + O(\Delta x^2) \tag{2.14}$$

where $O(\Delta x^2)$ is the spatial truncation error. Formulations for the y- and z-dimensions are done in a similar manner. By replacing the partial derivatives in Eq. (2.5) with the above finite-difference approximations without the truncation

terms, we get the *standard non-staggered leapfrog* method (Kowalczyk et al., 2011):

$$\begin{aligned} p^{n+1}(x_i,y_j,z_k) &= \frac{1}{3}\left(p^n(x_{i+1},y_j,z_k)+p^n(x_{i-1},y_k,z_j)\right. \\ &\quad +p^n(x_i,y_{j+1},z_k)+p^n(x_i,y_{j-1},z_k) \\ &\quad \left.+p^n(x_i,y_j,z_{k+1})+p^n(x_i,y_j,z_{k-1})\right)-p^{n-1}(x_i,y_k,z_j) \end{aligned} \tag{2.15}$$

The temporal resolution $\Delta t = \frac{\Delta x}{c\sqrt{3}}$ has been used, which is the upper limit for the so-called Courant's stability property condition for the SLF method (explained in Section 3.1). Using this limit, the constant $c^2\frac{\Delta t^2}{\Delta x^2} = 1/3$ is obtained.

A *staggered* grid formulation is obtained by approximating the coupled wave equation given in Eq. (2.3) and (2.4) with finite differences. The staggered leapfrog method is then given by (Botteldooren, 1995)

$$\begin{aligned} v_x^{n+1/2}(i+1/2,j,k) &= v_x^{n-1/2}(i+1/2,j,k)-\frac{\Delta t}{\rho_0\Delta x}\left(p^n(i+1,j,k)-p^n(i,j,k)\right) \\ v_y^{n+1/2}(i,j+1/2,k) &= v_x^{n-1/2}(i,j+1/2,k)-\frac{\Delta t}{\rho_0\Delta x}\left(p^n(i,j+1,k)-p^n(i,j,k)\right) \\ v_z^{n+1/2}(i,j,k+1/2) &= v_x^{n-1/2}(i,j,k+1/2)-\frac{\Delta t}{\rho_0\Delta x}\left(p^n(i,j,k+1)-p^n(i,j,k)\right) \\ p^{n+1}(i,j,k) &= p^n(i,j,k)-\frac{\rho_0c^2\Delta t}{\Delta x}\left(v_x^{n+1/2}(i+1/2,j,k)\right) \\ &\quad -\frac{\rho_0c^2\Delta t}{\Delta x}\left(v_y^{n+1/2}(i,j+1/2,k)\right) \\ &\quad -\frac{\rho_0c^2\Delta t}{\Delta x}\left(v_z^{n+1/2}(i,j,k+1/2)\right) \end{aligned}$$

where the nodal points for the coupled differential equations are located at different geometrical positions allowing for a natural and accurate formulation. The additional equations have been shifted a half step in both time and space due to the staggered formulation, but still yields the same truncation errors as for the non-staggered formulation, since the derivatives for the wave equation have been approximated centered differences with the same truncation errors. Using staggered or non-staggered formulation should therefore give the same level of *dispersion errors*. Dispersion errors are introduced in all numerical schemes due to the fact that the partial derivatives are approximated by finite differences, and manifest themselves as waves with different wave numbers travelling with slightly different speed, which does not correspond with the physical behaviour. By using a finer grid, i.e. by choosing a smaller Δx, the dispersion error can be minimised, but it comes with the cost of more computational power.

The schemes considered so far are all *explicit* schemes, since they calculate the

state of the system at the next time step by only considering the current and previous time steps. There are also another scheme types called *implicit* schemes, that solves the equation by considering current and previous time steps but *also* includes the next time step. In general, implicit schemes introduces fewer dispersion errors, but comes with the cost of more computations.

2.4.1 Comparison with Digital Waveguides

As already mentioned in Section 1.1, the K-DWM formulation can directly be seen as a FDTD standard leapfrog scheme. In the following, we will show that the W-DWM from Eq. (1.3) can also be seen as a subclass of the FDTD method. We will only show the relationship for the 1-D case with two junctions (L=2) using the the notion from Eq. (1.2):

$$p(m,n) = p^+(n-m) + p^-(n+m) \tag{2.16}$$

The formulation of the standard leapfrog method from Eq. (2.15) can be formulated in 1-D as

$$p_m(n+1) = p_{m+1}(n) + p_{m-1}(n) - p_m(n-1) \tag{2.17}$$

Substituting the right-hand side of Eq. (2.17) using Eq. (2.16) gives us (Smith, 2012)

$$\begin{aligned} p(m,n+1) &= p(m+1,n) + p(m-1,n) - p(m,n-1) \\ &= p^+(n-m-1) + p^-(n+m+1) \\ &+ p^+(n-m+1) + p^-(n+m-1) \\ &- p^+(n-m-1) - p^-(n+m-1) \\ &= p^-(n+m+1) + p^+(n-m+1) \end{aligned}$$

which has the same formulation as the W-DWM formulation in Eq. (2.16). Therefore, the W-DWG can be considered as a subclass of the FDTD methods, only the implementation differs.

2.4.2 Computation and Memory Consumption

The general computation and memory consumption for the DWG and FDTD method will be considered in this section. Without loss of generality, we will only consider the standard leapfrog methods.

The grid spacing Δx must be sufficient to capture the wave length, and assuming that we wish to sample the wavelength k times, we get

$$\Delta x = \frac{\lambda_{\min}}{k} \tag{2.18}$$

where $\lambda_{\min}$ is the wave length and k depends on the scheme. Typically $k = 8$ to 10 gives adequate results, with inaccuracies appearing as soon as the sampling drops below this rate[1]. Note that the Nyquist sample rate ($k = 2$) is much to less when considering FDTD simulations. The time step must be chosen to satisfy the Courant's stability property condition (more details in Section 3.1)

$$\Delta t \leq \frac{\Delta x}{\sqrt{D} \cdot c} \tag{2.19}$$

where D is the dimension of the simulation and c is the speed of sound. This limit is crucial, since larger sampling rates require denser meshes specified by Δx, resulting in more computation time. Increasing the mesh resolution leads to less dispersion errors, but memory and computation time dramatically increases: By refining the mesh by a factor r, the requirements are augmented by r^3 in memory and r^4 in computation time. To see why this is the case, let us first assume that the geometry is cubic. By refining the mesh by 2, the original cube now consists of 8 smaller cubes each with the same number of points as the original one, giving a memory increase by a factor $2^3 = 8$. Regarding the computation time, we will need a factor of r time steps more in all dimensions, yielding r^3. Moreover, using the Courant's condition from Eq. (2.19), the wave has travelled $c \cdot \Delta t \leq c\frac{\Delta x}{c\sqrt{3}} = \frac{\Delta x}{\sqrt{3}}$ in one time step and again, dividing Δx by r, we will need a factor of r time steps more, resulting in an increase in time by r^4.

In Table 2.1, the number of grid points per m^3 for the SLF and IWB schemes is given for frequencies between 100 Hz and 20,000 Hz using $k = 13.4$ samples per wavelength for the SLF scheme and $k = 5.4$ samples per wavelength for the IWB scheme. From these observations, we can forget about simulating the

	Grid nodes / m^3				
Scheme	100 Hz	1,000 Hz	5,000 Hz	10,000 Hz	20,000 Hz
SLF	59.11	59,107	7.4 mio.	59.1 mio.	472.8 mio.
IWB	3.87	3,868	483,520	3.8 mio.	30.9 mio.

Table 2.1: Number of grid points per m^3 when using the SLF and IWB for different frequencies.

whole bandwidth using FDTD, but for lower frequencies the number of grid

[1]This is true for the SLF, but fewer sample points can be used when considering other schemes. See Chapter 3.1 and Chapter 7

points are manageable. We also see a big difference in the number of points used for the two schemes due to the number of sampling points required for – in this case – a maximum of 2% of dispersion errors.

2.4.3 FDTD Variants

An overview of different schemes used for solving the acoustic wave equation is given below:

- Implicit schemes (Kowalczyk and van Walstijn, 2010a):
  - Alternating Direction Implicit (ADI).
  - Maximally Flat Isotropic (MFI)
  - Optimum Implicit (OI)
- Explicit schemes (Kowalczyk et al. (2011), Raghuvanshi et al. (2009)):
  - Staggered and non-staggered Standard Leapfrog (SLF)
  - Octahedral (OCTA)
  - Cubic Closed-Packed (CCP)
  - Interpolated Digital Waveguide (IDWG) Mesh
  - Interpolated Isotropic (IISO), Interpolated Isotropic 2 (IISO2), Interpolated Wideband (IWB).
  - Large star systems.

All the listed explicit schemes are second-order in time and space (compact in space), except the large star systems, which takes nodes located farther away into account. Large star systems can lead to less dispersion errors, but are inconvenient due to complicated treatment of boundaries (Kowalczyk, 2008a). The compact schemes differ in how the neighbouring points are chosen. The implicit schemes have different properties, with the ADI methods usually having better stability properties than explicit schemes and can even be fourth-order accurate in time and space. MFI is useful when the aim is to apply pre- and post-warping techniques and OI is an optimisation technique, that is computational most effective when the dispersion error is below 1 % (Kowalczyk and van Walstijn, 2010a).

Because real-time simulation is crucial for the project, GPU programming will be used as a main tool for obtaining this goal. The implicit schemes will not be

considered because of their computational requirements and strong dependencies between nodal points within each time steps, making parallelisation less useful. In (Kowalczyk et al., 2011), non-staggered 3-D compact explicit schemes on a rectilinear grid have been investigated for modelling acoustic systems in a specified audio bandwidth. A new boundary formulation approximating the locally reacting surfaces reflectance well is proposed, and it is indicated that the 3-D interpolated wide-band scheme and the 3-D interpolated isotropic schemes are the most efficient choices for accurate and isotropic FDTD simulations, with improved performance over other approaches commonly found in FDTD and DWM literature on room acoustics.

Because of the promising results obtained in (Kowalczyk et al., 2011), it has been decided to implement these methods with the hope of simulating the lower frequency sound field with physical correct result in real-time on the GPU.

Chapter 3

Simulation of the Sound Field using Finite-Difference Time-Domain Methods

3.1 Finite-Difference Time-Domain Schemes

Any 3-D non-staggered compact explicit scheme approximating the wave equation can be described by (Kowalczyk and van Walstijn, 2011)

$$\delta t^2 p^n_{i,j,k} = \lambda_{\mathrm{c}}^2 \Big[(\delta_x^2 + \delta_y^2 + \delta_z^2) + a(\delta_x^2\delta_y^2 + \delta_y^2\delta_z^2 + \delta_x^2\delta_z^2) + b\delta_x^2\delta_y^2\delta_z^2 \Big] p^n_{i,j,k} \quad (3.1)$$

where a and b denote two free numerical parameters, $p^n_{i,j,k}$ is the pressure value at time n at a given location indicated by the indices i, j and k in the x-, y- and z-directions, respectively, and λ_{c} is the Courant number explained in a moment.

Applying the second-order centered finite-difference operators given as[1]

$$\begin{aligned} p^n_{i,j,k} &\equiv p(x,y,z,t)\big|_{x=i\Delta x, y=j\Delta x, z=k\Delta x, t=n\Delta t} && (3.2)\\ \delta_t^2 &\equiv p^{n+1}_{i,j,k} - 2p^n_{i,j,k} + p^{n-1}_{i,j,k} && (3.3)\\ \delta_x^2 &\equiv p^n_{i+1,j,k} - 2p^n_{i,j,k} + p^n_{i-1,j,k} && (3.4)\\ \delta_y^2 &\equiv p^n_{i,j+1,k} - 2p^n_{i,j,k} + p^n_{i,j-1,k} && (3.5)\\ \delta_z^2 &\equiv p^n_{i,j,k+1} - 2p^n_{i,j,k} + p^n_{i,j,k-1} && (3.6)\\ & && (3.7) \end{aligned}$$

we get the following generalised difference equation for compact explicit schemes:

$$\begin{aligned} p^{n+1}_{i,j,k} = {} & d_1(p^n_{i+1,j,k} + p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1})\\ & + d_2(p^n_{i+1,j+1,k} + p^n_{i+1,j-1,k} + p^n_{i+1,j,k+1} + p^n_{i+1,j,k-1} + p^n_{i,j+1,k+1} + p^n_{i,j+1,k-1}\\ & + p^n_{i,j-1,k+1} + p^n_{i,j-1,k-1} + p^n_{i-1,j+1,k} + p^n_{i-1,j-1,k} + p^n_{i-1,j,k+1} + p^n_{i-1,j,k-1})\\ & + d_3(p^n_{i+1,j+1,k+1} + p^n_{i+1,j-1,k+1} + p^n_{i+1,j+1,k-1} + p^n_{i+1,j-1,k-1}\\ & + p^n_{i-1,j+1,k+1} + p^n_{i-1,j-1,k+1} + p^n_{i-1,j+1,k-1} + p^n_{i-1,j-1,k-1})\\ & + d_4 p^n_{i,j,k} - p^{n-1}_{i,j,k} \end{aligned} \quad (3.8)$$

with the coefficients given by

$$\begin{aligned} d_1 &= \lambda_c^2(1 - 4a + 4b),\\ d_2 &= \lambda_c^2(a - 2b),\\ d_3 &= \lambda_c^2 b,\\ d_4 &= 2(1 - 3\lambda_c^2 + 6\lambda_c^2 a - 4b\lambda_c^2) \end{aligned} \quad (3.9)$$

The coefficients a and b determine the characteristics of the scheme, and in Table 3.1 specific choices for a and b are listed. For each scheme the Courant number λ is listed and the bandwidth for which a maximum of 2% and 10% of dispersion errors are allowed. In Fig. 3.1, the stencils are depicted by showing the neighbour relation for the standard Leapfrog (SLF) scheme consisting of 6 neighbours, the Octahedral (OCTA) scheme consisting of 8 neighbours, Cubic Close-Packed (CCP) scheme consisting of 12 neighbours, and the interpolated schemes. Specific choices of interpolated schemes from the table are the Interpolated Digital Waveguide Mesh (IDWM), the interpolated isotropic schemes (IISO and IISO2) and the interpolated wideband scheme (IWB). The interpolated schemes can in general be seen as linear superposition of the SLF, OCTA and CCP schemes.

[1] The second-order centered finite-difference operators correspond to Eq. (2.13) and Eq. (2.14), where the denominators Δt and Δx have been collected in the λ term outside the parenthesis in Eq. (3.1).

	Standard Leapfrog (SLF)	Octa-hedral (OCTA)	Cubic Close-Packed (CCP)	Interp. DWM (IDWM)	Interp. Isotropic (IISO)	Isotropic 2 (IISO2)	Interp. Wide-band (IWB)
nr grid points	6	8	12	26	18	26	26
a	0	$\frac{1}{2}$	$\frac{1}{4}$	0.2034	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{4}$
b	0	$\frac{1}{4}$	0	0.0438	0	$\frac{1}{48}$	$\frac{1}{16}$
λ	$\sqrt{\frac{1}{3}}$	1	1	$\sqrt{\frac{1}{3}}$	$\sqrt{\frac{3}{4}}$	$\sqrt{\frac{3}{4}}$	1
d_1	$\frac{1}{3}$	0	0	0.1205	$\frac{1}{4}$	$\frac{15}{48}$	$\frac{1}{4}$
d_2	0	0	$\frac{1}{4}$	0.0386	$\frac{1}{8}$	$\frac{3}{32}$	$\frac{1}{8}$
d_3	0	$\frac{1}{4}$	0	0.0146	0	$\frac{1}{64}$	$\frac{1}{16}$
d_4	0	0	-1	0.6968	-1	$-\frac{9}{8}$	$-\frac{3}{2}$
bandwidth	0.196	0.25	0.333	0.196	0.333	0.333	0.5
accuracy (2%)	0.075	0.093	0.175	0.069	0.175	0.175	0.186
accuracy (10%)	0.151	0.185	0.301	0.144	0.301	0.301	0.371

Table 3.1: Parameters for seven chosen 3-D compact explicit schemes.

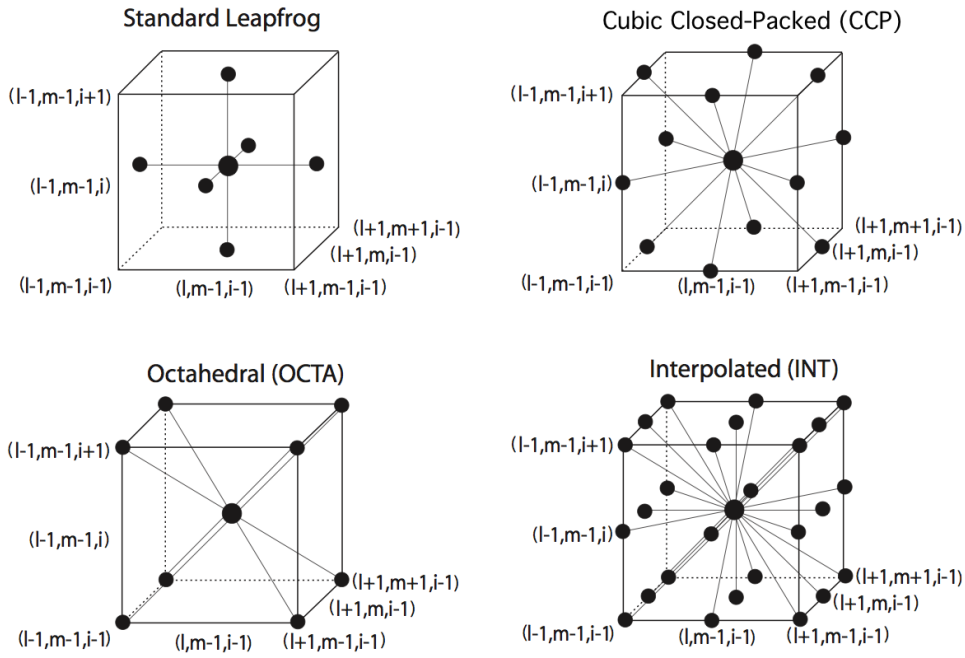


Figure 3.1: Stencils for the Standard Leapfrog scheme, Octahedral (OCTA) scheme, Cubic Close-Packed (CCP) scheme and the interpolated schemes. Source: (Kowalczyk and van Walstijn, 2011)

For explaining the Courant number, let us take a step back. When considering digital signal processing, the sampling rate must respect the Nyquist frequency theorem, which states that the sampling rate must be at least twice the maximum frequency $f_{\max}$. For FDTD methods, this limit is not enough to ensure properly sound modelling due to dispersion errors introduced in the schemes. We will denote the number of sampling points per wavelength with k, and the grid spacing is then given by

$$\Delta x = \frac{\lambda_{\min}}{k} \tag{3.10}$$

where $\lambda_{\min}$ is the wavelength given by

$$\lambda_{\min} = \frac{c}{f_{\max}} \tag{3.11}$$

The relation between the spatial and temporal resolution is given by the Courant number λ_{c} as

$$\lambda_{\mathrm{c}} \geq c\frac{\Delta t}{\Delta x} \tag{3.12}$$

λ_{c} can not be chosen freely but depends on the scheme type. For the family of 3-D compact schemes, the Courant's stability property condition is given by (Kowalczyk and van Walstijn, 2011):

$$\lambda_{\mathrm{c}}^2 \leq \min\left(1, \frac{1}{2-4a}, \frac{1}{3-12a+16b}\right) \tag{3.13}$$

Thus, stable 3-D FDTD schemes are obtained for any set of parameters (a, b) that satisfies Eq. (3.13) and choice for particular stencils have already been given in Table 3.1. We will follow (Kowalczyk and van Walstijn, 2011) and always set the stability condition to the highest value by using equality in Eq. (3.13), since it is stated that it usually gives the smallest dispersion errors. Hence, to decide on the values for our two unknown variables Δx and Δt, we use Eq. (3.10) with a suitable value k for Δx, and for Δt, we rewrite Eq. (3.12) (using equality) as

$$\Delta t = \lambda_{\mathrm{c}}\frac{\Delta x}{c} \tag{3.14}$$

3.2 Sources

Until now, we have not considered how energy is added to the system. In FDTD simulations, energy is usually injected in a single point and the energy is then distributed outwards from the point by the wave equation. Before discussing possible choices for the input signal and how to injecting energy to the system, we will derive the sampling rate for FDTD simulations.

3.2.1 Sample Resolution

As already mentioned, the spatial resolution depends on the maximum frequency and the maximum dispersion errors allowed, whereas the Courant stability condition states the relation between the spatial and temporal resolution. Let t denote the number of time steps necessary for evaluating one period $T = \frac{1}{f_{\max}}$ of the sound signal. Then the following equation has to be solved:

$$t \cdot \Delta t = \frac{1}{f_{\max}} \tag{3.15}$$

Inserting Eq. (3.14) and using $f_{\max} = \frac{c}{\lambda_{\min}}$ we get

$$t = \frac{1}{\lambda_c \frac{\Delta x}{c} \cdot \frac{c}{\lambda_{\min}}} = \frac{1}{\lambda_c \frac{\Delta x}{\lambda_{\min}}} \tag{3.16}$$

By also using the grid spacing in Eq. (3.10), we get

$$t = \frac{1}{\lambda_c \cdot \frac{\lambda_{\min}/k}{\lambda_{\min}}} = \frac{k}{\lambda_c} \tag{3.17}$$

This observation is important, since it tells us what our sample rate f_s should be, namely a factor $\frac{k}{\lambda_c}$ of the maximum frequency $f_{\max}$:

$$f_s = \frac{k}{\lambda_c} \cdot f_{\max} \tag{3.18}$$

3.2.2 Impulse Signals

When choosing an excitation signal, all input signals can be used as long as the signal is band-limited to fit the frequency range of the simulation. Gaussian impulses have been widely used in the FDTD literature, but other types can be used as well. It is for example possible to feed the source signal that should be modelled directly into the system, which would be an obvious choice when considering real-time systems. However, when the properties of a system are investigated, a band-limited wideband signal is the natural choice. The usual compromise when designing impulses is to have a function in the frequency domain that has a sharp cutoff near the lower and upper frequency limit, but at the same time having a signal as short as possible in the time domain. It is impossible in practice to have signals with perfectly sharp cutoffs in the frequency domain, and the theory tells us that approximating such a function will lead to an infinite long time-domain signal.

Two impulses will be used in this work, namely a normalised differentiated Gaussian pulse and a Kaiser-windowed sinc function. The normalised differentiated Gaussian is given as

$$U_{\text{gauss}}(t) = \frac{(t - t_0)}{t_w} e^{-\frac{(t-t_0)^2}{t_w^2}} \tag{3.19}$$

where t_0 is the time delay and $t_w = (\pi \cdot f_{\max}/2)^{-1}$ is the half width of the pulse. One should use $t_0 \geq 4t_w$ for achieving a smooth initialisation of the signal (Gedney, 2010). The Kaiser-windowed sinc impulse has been designed as

$$U_{\text{kaiser}}(t) = w_{\text{kaiser}} \cdot \frac{\sin(\pi t)}{\pi t} \tag{3.20}$$

where w_{kaiser} is the Kaiser window. The reason for using a Kaiser window is to limit the ideal sinc function of infinite duration to a finite impulse response. Since we have used the Matlab filter toolbox to design the filter, we will not go into details about how the Kaiser window is formulated. It has been observed that the DC component has to be removed, which – for the Gaussian pulse – was done by taking the derivative of the Gaussian. The two impulses are depicted in Figure 3.2 for a desired maximum frequency of $f_{\max} = 100$ Hz. The upper row shows the frequency domain, whereas the lower row shows the time domain. We see, that the Gaussian pulse drops almost 60 dB at 150 Hz, whereas the Kaiser-windowed sinc function has a cut-of frequency very near 100 Hz. The drawback of having a very step cut-off can be seen in the time-domain, where Kaiser-windowed impulse has a duration of around 0.25 seconds, whereas the Gaussian only has a duration of about 0.05 seconds.

3.2.3 Source modelling

We will need a way to inject the source energy into the system, and this can be done by embedding the sources as “hard”, “soft” or “transparent” (Schneider et al., 1998). Generally for all the source types is, that the excitation is performed in a single point in the domain, and from there the energy is distributed outward from this point due to the nature of the wave equation. In 1-D, a hard source can be implemented as

$$p^n_{i_{\text{src}}} = f^n_{i_{\text{src}}} \tag{3.21}$$

where f is the forcing term (e.g. the Gaussian impulse), depending on the iteration step n and the fixed location i_{src} of the source. Since the update equation does not apply at this source node, the source is reflecting the incoming wave when interfering with the source. That is, the source itself is being treated as a perfectly reflecting obstacle in the scene. This will not lead to physical

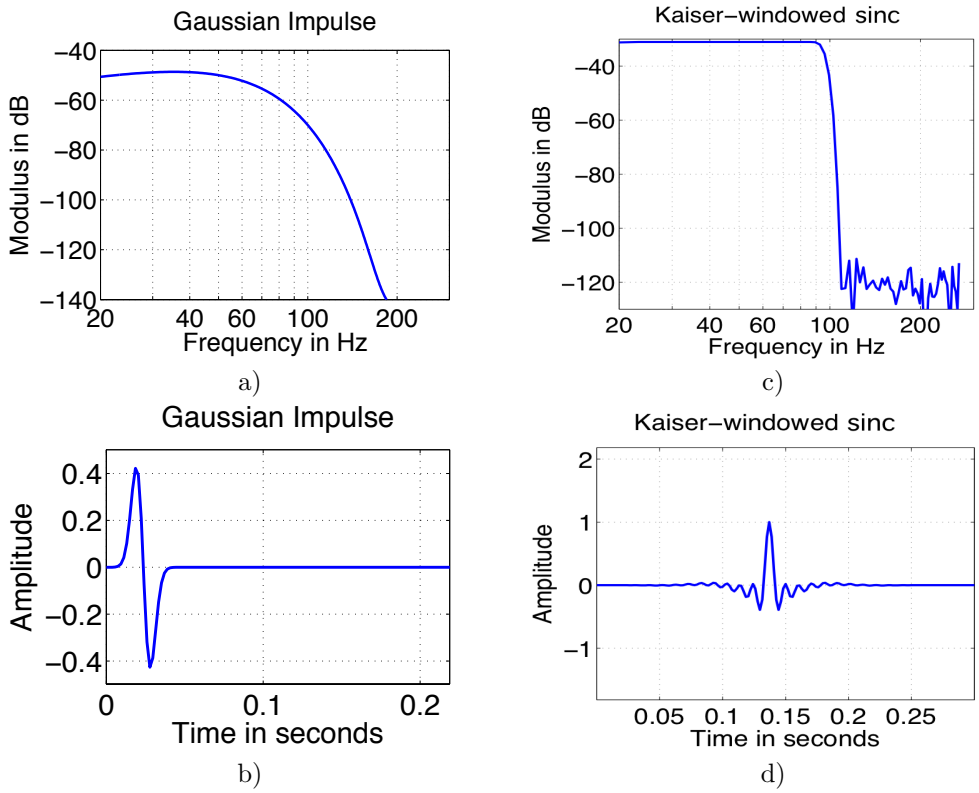


Figure 3.2: Gaussian and Kaiser-windowed impulse signal band-limited to 100 Hz. a) Spectrum of Gaussian impulse, b) Time-domain of Gaussian impulse, c) Spectrum of Kaiser-windowed impulse, d) Time-domain of Kaiser-windowed impulse.

correct behaviour for most simulations, since the size of the source will typical be small compared to the scene. A way to compensate for this is to use a soft source, where the node is set to the sum of the value returned by the update scheme and the source signal as

$$p^n_{i_{\text{src}}} = p^n_{i_{\text{src}}} + f^n_{i_{\text{src}}} \tag{3.22}$$

This yields a point that does not scatter any energy, but the sound field that is radiated may not be correct compared to the field that should be radiated by only the source function. The solution is to implement a transparent source, compensating from the medium's response.

Transparent sources were formulated in (Schneider et al., 1998) only for the staggered leapfrog method, and not explicitly for the family of 3-D compact explicit schemes. We will follow the idea presented in the paper, and show that the formulation can also be used for the family of 3-D compact explicit schemes.

First, let us investigate how the update scheme behaves in the source point i_{src} in 1-D using the Courant condition set to the upper limit $\lambda = 1$. The update scheme then takes the following form:

$$p^{n+1}_{i_{\text{src}}} = p^n_{i_{\text{src}}-1} + p^n_{i_{\text{src}}+1} - p^{n-1}_{i_{\text{src}}} \tag{3.23}$$

With the goal of creating a transparent source, let us implement a soft source in the point i modelled as the sum of the driving function and the pressure updated by the update equation in the same point i. This gives us the 1-D explicit non-staggered leapfrog update scheme

$$p^{n+1}_{i_{\text{src}}} = p^n_{i_{\text{src}}-1} + p^n_{i_{\text{src}}+1} - p^{n-1}_{i_{\text{src}}} + f^n_{i_{\text{src}}} \tag{3.24}$$

We will now investigate how the pressure in the vicinity of the source point is updated using the above scheme depicted in Table 3.2 for the first 5 time steps $n = 0, 1, 2, 3, 4$. We observe that the update scheme is returning the pressure

	$p(i-3)$	$p(i-2)$	$p(i-1)$	$p(i)$	$p(i-1)$	$p(i-2)$	$p(i-3)$
$n=0$				f^0			
$n=1$			f^0	f^1	f^0		
$n=2$		f^0	f^1	f^2+f^0	f^1	f^0	
$n=3$	f^0	f^1	f^2+f^0	f^3+f^1	f^2+f^0	f^1	f^0
$n=4$	f^1	f^2+f^0	$f^3+f^1+f^0$	$f^4+f^2+f^0$	$f^3+f^1+f^0$	f^2+f^0	f^1

Table 3.2: Values of p in the vicinity of i_{src} in a one-dimensional grid for the non-staggered leapfrog method with the Courant number set to the maximum number, $\lambda = 1$.

values f^0, f^1 and f^2 to the source node respectively at time $n = 2, 3$ and 4, which

have to be eliminated in the case of a transparent source. This is achieved by subtracting the source pressure two time-steps back in time as

$$p^{n+1}_{i_{\text{src}}} = p^{n}_{i_{\text{src}}-1} + p^{n}_{i_{\text{src}}+1} - p^{n-1}_{i_{\text{src}}} + f^{n}_{i_{\text{src}}} - f^{n-2}_{i_{\text{src}}} \tag{3.25}$$

Unfortunately, this only hold in 1-D for the Courant number set to unity.

Let us repeat the approach, but now for arbitrary Courant numbers, leading to the update scheme below:

$$p^{n}_{i_{\text{src}}} = \lambda(p^{n-1}_{i_{\text{src}}+1} + p^{n-1}_{i_{\text{src}}-1}) - p^{n-2}_{i_{\text{src}}} + f^{n}_{i_{\text{src}}} \tag{3.26}$$

The pressure node values in the vicinity of the source node for Courant numbers less that that unity are depicted in Table 3.3 for the first 5 time-steps. We will

	$p(i-1)$	$p(i_{\text{source}})$	$p(i+1)$
$n=0$		f^0	
$n=1$	λf^0	f^1	λf^0
$n=2$	λf^1	$f^2 + 2\lambda^2 f^0 - f^0$	λf^1
$n=3$	$\lambda f^2 + (2\lambda^3 - 2\lambda) f^0$	$f^3 + 2\lambda^2 f^1 - f^1$	$\lambda f^2 + (2\lambda^3 - 2\lambda) f^0$
$n=4$	$\lambda f^3 + (2\lambda^3 - 2\lambda) f^1$	$f^4 + (2\lambda^2 - 1) f^2 + (4\lambda^4 - 6\lambda^2 + 1) f^0$	$\lambda f^3 + f^1(2\lambda^3 - 2\lambda)$

Table 3.3: Values of p in the vicinity of i_{src} in a one-dimensional grid for the non-staggered leapfrog method with the Courant number less that 1.

denote the pressure values returned to the source node by the update scheme as the *grid impulse* response. The grid impulse response for an arbitrary driving function f and arbitrary Courant numbers is listed below obtained from the source column in Table 3.3 subtracting the driving function:

$$\begin{aligned}
p^0_{i_{\text{src}}} &= 0, \\
p^1_{i_{\text{src}}} &= 0, \\
p^2_{i_{\text{src}}} &= (2\lambda^2 - 1) f^0, \\
p^3_{i_{\text{src}}} &= (2\lambda^2 - 1) f^1, \\
p^4_{i_{\text{src}}} &= (2\lambda^2 - 1) f^2 + (4\lambda^4 - 6\lambda^2 + 1) f^0, \\
p^5_{i_{\text{src}}} &= (2\lambda^2 - 1) f^3 + (4\lambda^4 - 6\lambda^2 + 1) f^1 \\
p^6_{i_{\text{src}}} &= (2\lambda^2 - 1) f^4 + (4\lambda^4 - 6\lambda^2 + 1) f^2 + (8\lambda^6 - 20\lambda^4 + 12\lambda^2 - 1) f^0 \\
p^7_{i_{\text{src}}} &= (2\lambda^2 - 1) f^5 + (4\lambda^4 - 6\lambda^2 + 1) f^3 + (8\lambda^6 - 20\lambda^4 + 12\lambda^2 - 1) f^1
\end{aligned}$$

We will now define the grid impulse response I as the pressure field returned to the source node obtained by using the Kronecker delta function as a specific choice of driving function in the update scheme in Eq. (3.26):

$$f^n_{i_{\text{src}}} = \delta(n), \tag{3.27}$$

which yields

$$\begin{aligned} I^0 &= 0, \\ I^1 &= 0, \\ I^2 &= 2\lambda^2 - 1, \\ I^3 &= 0, \\ I^4 &= 4\lambda^4 - 6\lambda^2 + 1, \\ I^5 &= 0, \\ I^6 &= 8\lambda^6 - 20\lambda^4 + 12\lambda^2 - 1 \\ I^7 &= 0, \end{aligned}$$

By comparing the general grid impulse $p_{i_{\text{src}}}$ for an arbitrary forcing function and the impulse I for the forcing function defined in Eq. (3.27), we notice that the source node takes the following values for the first 8 time steps:

$$\begin{aligned} p^0 &= f^0 \\ p^1 &= f^1 \\ p^2 &= f^2 + I^2 p^0 \\ p^3 &= f^3 + I^2 p^1 \\ p^4 &= f^4 + I^2 p^2 + I^4 p^0 \\ p^5 &= f^5 + I^2 p^3 + I^4 p^1 \\ p^6 &= f^6 + I^2 p^4 + I^4 p^2 + I^6 p^0 \\ p^7 &= f^7 + I^2 p^5 + I^4 p^3 + I^6 p^1 \end{aligned}$$

Since our goal is to eliminate the pressure values returned to the source node by the update scheme such that $p^0 = f^0$, $p^1 = f^1$, $p^2 = f^2$, ..., the extra terms should be removed. This can be achieved by subtracting $I^2 f^0$ at the second update, $I^2 f^1$ at the third update, $I^2 f^2 + I^4 f^0$ at the fourth update, $I^2 f^3 + I^4 f^1$ at the fifth update, and so on. Stated another way, to implement a transparent source, the convolution of the impulse response and the driving function must be subtracted from the source node. Hence, a transparent source for an arbitrary Courant number can be obtained using

$$p_{i_{\text{src}}}^{n+1} = p_{i_{\text{src}}}^{n} + f_{i_{\text{src}}}^{n+1} - \sum_{m=0}^{n} I^{n-m+1} f_{i_{\text{src}}}^{m} \tag{3.28}$$

The formulation in Eq. (3.28) also holds in higher dimensions, the only difference is the actual values for the grid impulse I. We are interested in deriving the general formulation for the transparent source corresponding to Eq. (3.28) for

the family of compact schemes, and it turns out, that the formulation of the transparent source in Eq. 3.28 remains the same, only the grid impulse I changes (the full derivation can be found in Appendix 9.1).

The last part for obtaining a transparent source is to compute the grid impulse I. Deriving the exact polynomial form is not desirable due to the complexity of I, instead we will perform a simple simulation where a hard source is realised and the value coupled back to the update note is recorded without considering the driving function itself (Schneider et al., 1998). Thus, the impulse response is calculated from the surrounding nodes, but the impulse response does not couple back into the update scheme, since we use a hard source for injecting pressure into the system. Using this method, the impulse of any of the schemes in Eq. (3.8) can be realised and used with the formulation of the transparent source in Eq. (3.28). Because the grid size has to be big enough to ensure that no reflections are coupled back to the source node, computing the 3-D grid impulse can be quite time and memory consuming. However, by exploiting the following symmetry for a pressure node located at the origin $(0, 0, 0)$

$$\begin{aligned} p^n_{-i,j,k} &= p^n_{i,j,k} \\ p^n_{i,-j,k} &= p^n_{i,j,k} \\ p^n_{i,j,-k} &= p^n_{i,j,k} \\ p^n_{-i,-j,k} &= p^n_{i,j,k} \\ p^n_{-i,j,-k} &= p^n_{i,j,k} \\ p^n_{i,-j,-k} &= p^n_{i,j,k} \\ p^n_{-i,-j,-k} &= p^n_{i,j,k} \end{aligned}$$

the size of the computational domain can be reduced by using only half the number of points in the x, y and z-dimensions leading to a domain that is $\frac{1}{8}$ the size of the full grid[2].

For the experiments in Section 7.1, 10,000 iteration steps have been used for simulating up to 100 Hz, which means that the grid impulse should be the same length. But as we can see in Figure 3.3, the grid impulse converges towards 0 very fast and can therefore be padded with 0's when the impulse takes values near 0. Since the grid impulse only needs to be computed once for every scheme and then recalled for use in any simulation that uses the same update scheme and grid resolution, the difficulty of realising a transparent source is feasible without too much effort.

[2] An even more efficient implementation that is $\frac{1}{64}$ the size of the full grid can be done by fully exploiting symmetry.

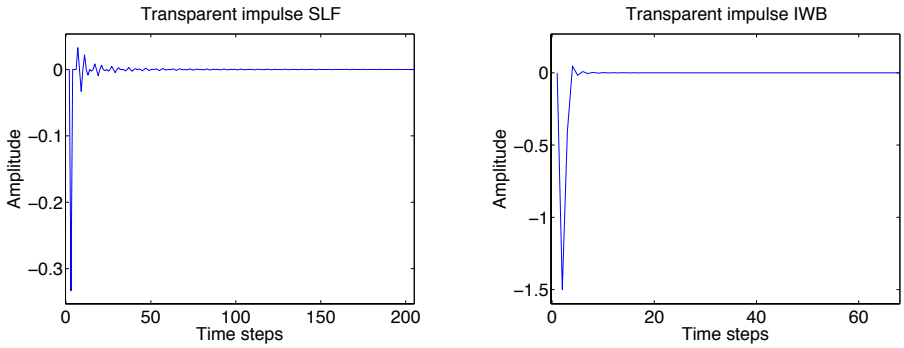


Figure 3.3: Transparent impulse I for a) SLF method using $k = 13.4$, b) IWB scheme using $k = 5.4$.

3.3 Numerical Dispersion Errors

In this section, we will briefly discuss how dispersion errors for the seven 3-D compact explicit schemes schemes in Table 3.1 can be adjusted.

Numerical dispersion is an unwanted artefact introduced due to the approximations made when discretising the continuous wave equation with finite differences. These artefacts are dependent on the direction of the travelling wave and causes high frequencies to travel with a different speed than the continuous wave equation. As a measure of dispersion error, the numerical phase velocity relative to the correct (continuous) phase velocity is used. The phase velocity is defined as

$$c = \frac{\omega}{k} \tag{3.29}$$

where k is the wave number and ω is the angular frequency. We will denote the numerical phase velocity as $\hat{c}$ with the corresponding numerical wave number $\hat{k}$. The relative dispersion error can thus be expressed by

$$d_{\text{err}} = \frac{c}{\hat{c}} \tag{3.30}$$

We can not use this relation directly, since we need a way to calculate $\hat{c}$. In (Kowalczyk and van Walstijn (2010b), van Walstijn and Kowalczyk (2008)), a formulation for the relative dispersion errors for arbitrary directions has been derived as

$$v(\hat{k}_x, \hat{k}_y, \hat{k}_z) = \frac{\omega}{\hat{k}c} = \frac{2\arcsin(\lambda\sqrt{F(s_x, s_y, s_z)})}{\lambda\sqrt{(\hat{k}_x)^2 + \hat{k}_y)^2 + \hat{k}_z)^2}} \tag{3.31}$$

The function F is the general stability condition given as

$$F(s_x, s_y, s_z) = (s_x + s_y + s_z) - 4a(s_x s_y + s_x s_z + s_y s_z) + 16 b s_x s_y s_z \quad (3.32)$$

where the following new variables are introduced:

$$s_x = \sin^2(\hat{k}_x \Delta x/2), \quad (3.33)$$
$$s_y = \sin^2(\hat{k}_y \Delta x/2), \quad (3.34)$$
$$s_z = \sin^2(\hat{k}_z \Delta x/2) \quad (3.35)$$

The formula in Eq. (3.31) can be used to obtain information about the dispersion error in all directions for each of the scheme and is useful for isotropy[3] analysis, which can be important when considering warping (Kowalczyk, 2008a). Warping is a preprocessing step that can reduce the numerical errors, but since we are concerned about dynamic scenes in real-time, warping can not be applied and hence isotropy is not an issue. Nevertheless, it is argued in (Kowalczyk and van Walstijn (2010b), van Walstijn and Kowalczyk (2008)) that the largest and smallest dispersion errors occur in the axial, side-diagonal or diagonal direction, and therefore plotting these direction will give us important knowledge about the overall dispersion error: by observing the direction where the dispersion error is biggest, we can determine the upper frequency for which a given dispersion error is reached (e.g. 2 %). In the following, we will explain how to express the relative phase velocity error (i.e. dispersion error) in the axial, side-diagonal and diagonal direction.

For the axial directions, we will consider wave propagation in one of the six directions where $\hat{k}_a$ denotes the wave number. The following cases will be present for an axial direction: $\left\{\hat{k}_x^2 \neq 0, \hat{k}_y^2 = \hat{k}_z^2 = 0\right\}$, $\left\{\hat{k}_y^2 \neq 0, \hat{k}_x^2 = \hat{k}_z^2 = 0\right\}$ or $\left\{\hat{k}_z^2 \neq 0, \hat{k}_x^2 = \hat{k}_y^2 = 0\right\}$. It can then be shown (van Walstijn and Kowalczyk, 2008) that Eq. (3.31) can be re-written as

$$v_a(\omega) = \frac{(\omega T/2)}{\lambda \arcsin\sqrt{\frac{\sin^2(\omega \Delta t/2)}{\lambda^2}}} \quad (3.36)$$

denoting the relative wave velocity in axial directions as a function of angular frequency.

For the side-diagonal directions, one of the following cases will be present: $\left\{\hat{k}_x^2 = \hat{k}_y^2 \neq 0, \hat{k}_z^2 = 0\right\}$, $\left\{\hat{k}_x^2 = \hat{k}_z^2 \neq 0, \hat{k}_y^2 = 0\right\}$ or $\left\{\hat{k}_y^2 = \hat{k}_z^2 \neq 0, \hat{k}_x^2 = 0\right\}$. Eq.

[3]Isotropy means uniformity in all orientations.

(3.31) can then be formulated as

$$v_{sd}(\omega) = \frac{\sqrt{\frac{1}{2}}(\omega \Delta t/2)}{\lambda \arcsin \sqrt{G_{sd}(\omega)}} \tag{3.37}$$

expressing the directional dispersion error by denoting the relative wave velocity in side-diagonal directions as a function of frequency.

For the diagonal directions, we have that $\left\{\hat{k}_x^2 = \hat{k}_y^2 = \hat{k}_z^2 \neq 0\right\}$. The expression for the relative wave velocity is derived in a similar manner as for the side-diagonal directions, but in this case a third-order polynomial has to be solved in contrary to a second-order polynomial for the side-diagonal case. The resulting expression is rather long and has been left out.

In Figure 3.4, the dispersion errors in the axial, diagonal and side-diagonal directions are depicted.

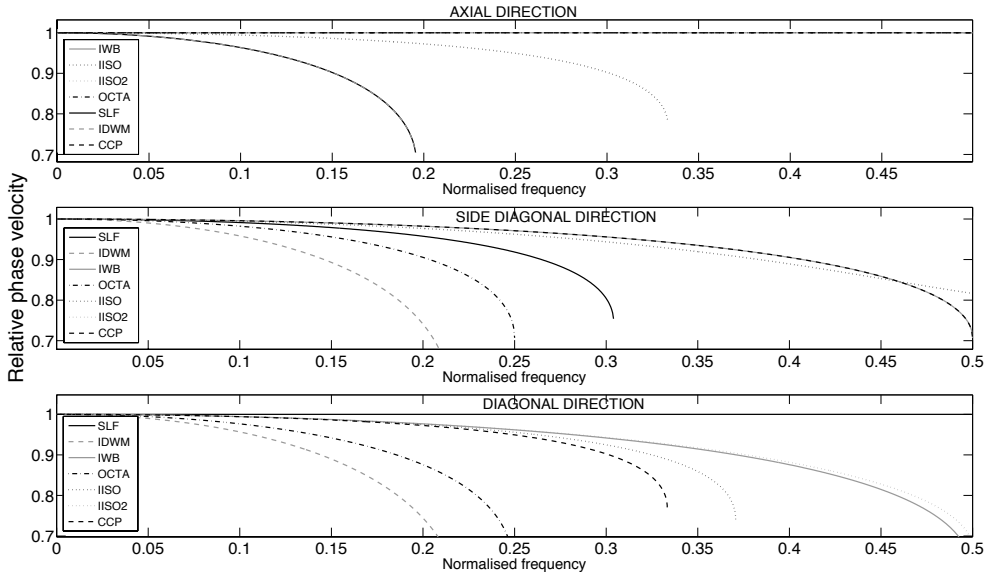


Figure 3.4: The plot denoted axial (top), diagonal (middle) and side-diagonal (bottom) dispersion errors. (Matlab figure kindly obtained from Konrad Kowalczyk).

Chapter 4

Boundary Modelling

The acoustic properties in rooms are highly determined by the geometry and the boundary material, and therefore proper modelling of boundaries is crucial for a realistic simulation of the sound field. In the DWG literature, the boundary termination is typically modelled by a frequency-independent reflection coefficient or frequency-dependent reflection filter directly attached to a single system branch (a single node), which imply that the waves are locally governed by a 1-D wave equation. Also, most methods available for non-staggered grids, including K-DWM, are also based on 1-D mesh terminations (Huopaniemi et al. (1997) Kelloniemi (2006), Murphy and Beeson (2007)). This is inconsistent with the theory and it has been shown that large reflection errors are introduced (Kowalczyk, 2008b). In the paper (Botteldooren, 1995), a 3-D boundary model has been formulated for staggered grids, but it is stated in (Kowalczyk (2008b), Kowalczyk et al. (2011)) that an additional stability bound is introduced, which in general also introduces further numerical errors.

Recently, (Kowalczyk et al., 2011) have presented frequency-dependent boundaries for the family of 3-D compact explicit schemes based on non-staggered rectilinear grid, resulting in a digital impedance filter (DIF) boundary model derived from a 3-D perspective, where the 3-D wave equation is now satisfied at the boundaries in opposite to the methods using 1-D terminating grids, and no additional stability bound is introduced. We will in the following review the theory for the boundary modelling presented in (Kowalczyk et al., 2011) and

derive the update formulas for al boundary types[1].

4.1 Locally Reacting Surfaces and Digital Impedance Filters

Considering a sound wave travelling along the x-dimension in positive direction, we can write the impedance defined in Eq. (2.6) as

$$p = Zv_x \tag{4.1}$$

where p denotes the pressure, Z is the boundary impedance for a right boundary and v_x denotes the velocity normal to this boundary. The wave equation can be expressed in two linear sound field equations in terms of pressure given by Eq. (2.3), and velocity given by Eq. (2.4), but only the first may be applied at a boundary (Kowalczyk, 2008b). Hence, a boundary normal to the x direction is given as

$$\frac{\partial p}{\partial x} = -\rho \frac{\partial v_x}{\partial t} \tag{4.2}$$

with ρ denoting the air density. Differentiating Eq. (4.1) with respect to time and inserting the result into Eq. (4.2) yields the boundary condition for the right boundary

$$\frac{\partial p}{\partial t} = -c\xi \frac{\partial p}{\partial x} \tag{4.3}$$

where $\xi = Z/\rho c$ is the specific acoustic impedance corresponding to the wall material modelled. Notice, that we have a formulation in terms of pressure only.

A surface is called locally reacting (LRS) if the velocity normal to the surface only depends on the pressure at the considered surface point and not also on the surrounding surface. In other words, no waves are transmitted along the wall surface, which is a good approximation for heavy walls, for walls with low bending stiffness and for porous absorbers (Vorländer, 2007). The LRS is captured by two conditions, namely that Z is independent of the surrounding pressure field, and that the 3-D wave equation holds at a boundary. If these conditions are met, the specific impedance in Eq. (4.3) can be obtained by rewriting Eq. (2.9) for the impedance at normal incidence ($\theta = 0°$) as

$$\xi(z) = \frac{1 + R_0(z)}{1 - R_0(z)} \tag{4.4}$$

[1] In the work by Kowalczyk, the update formulas for inner edges, inner corners, inner edge-corners and pyramid boundaries are not explicit given, and have been derived by the author of this thesis.

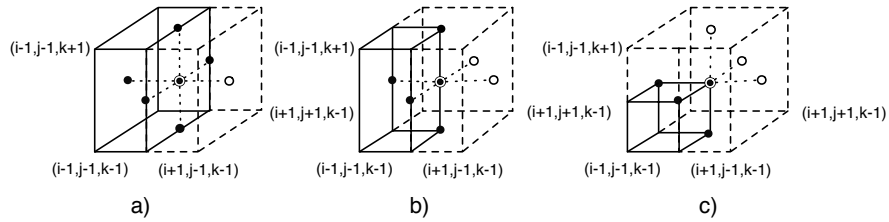


Figure 4.1: a) Plane boundary, b) Edge boundary, and c) Corner boundary. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles. The point that is being updated is indicated with a black-coloured circle with an surrounding circle.

This is considered a good approximation under the assumption of LRS. Provided that the reflectance filter represents a passive boundary ($|R_0(z)| \leq 1$), the impedance filter can be realised by IIR filters, where both the nominator and denominator are of the same order N:

$$\xi(z) = \frac{b_0 + B(z)}{a_0 + A(z)} \tag{4.5}$$

where

$$B(z) = \sum_{i=1}^{N} b_i z^{-i} \tag{4.6}$$

$$A(z) = \sum_{i=1}^{N} a_i z^{-i} \tag{4.7}$$

The above filter will be referred to as a *Digital Impedance Filter* (DIF) and how to obtain the coefficients a_i and b_i will be explained in Section 7.1.

4.2 Boundary Formulation for the SLF Scheme

In the following, we will derive the formulas for modelling frequency-dependent absorbing boundaries using 3-D compact explicit FDTD schemes realised by the use of DIFs from Eq. (4.5). We will first consider the SLF scheme and examine the cases concerning plane boundaries, boundary edges and corners depicted in Figure 4.1.

4.2.1 Plane Boundaries

The derivation of the update formula for a right plane boundary depicted in Figure 4.1a relies on combining the discrete wave equation and the boundary condition formulation in Eq. (4.3) discretised using centered differences, which yields

$$\frac{p^{n+1}_{i,j,k} - p^{n-1}_{i,j,k}}{2\Delta t} = -c\xi \left(\frac{p^{n}_{i+1,j,k} - p^{n}_{i-1,j,k}}{2\Delta x} \right) \tag{4.8}$$

Since we are now modelling a right boundary, the point with index $i + 1$ for the x-direction will be located outside of the geometry. These points are called *ghost points* and needs to be eliminated. In the following we will derive a formulation for these points. Moreover, the above equation has to combined with the frequency-dependent digital wall impedance given in Eq. (4.5). We will therefore transform Eq. (4.8) to the z-domain using the time-shifting property $p(n - k) = z^{-k}P(z)$, where $p(n)$ is the discrete-time signal and $P(z)$ is the Z-transform of $p(n)$, which gives us

$$(z - z^{-1})P_{i,j,k} = -\lambda\xi(z)(P_{i+1,j,k} - P_{i-1,j,k}) \tag{4.9}$$

The definition $P_{i,j,k} \equiv P_{i,j,k}(z)$ is used denoting the z-transform of the discrete time-domain pressure $p^n_{i,j,k}$ and λ is the Courant number. Substituting Eq. (4.5) into Eq. (4.9) yields

$$(z - z^{-1})P_{i,j,k} = \lambda Y \tag{4.10}$$

where

$$Y = \frac{b_0 + B(z)}{a_0 + A(z)}(P_{i-1,j,k} - P_{i+1,j,k}) \tag{4.11}$$

This formulation can be interpreted as an input-output relation $Y = \xi(z)X$, where the transfer function is given as the impedance $\xi(z) = \frac{b_0+B(z)}{a_0+A(z)}$, $X = P_{i-1,j,k} - P_{i+1,j,k}$ is the input and Y is the output. Multiplying in the z-domain corresponds to convolution in the time domain, making the intuition about the input-output relation even more clear. The fractional formulation of an IIR filter given in Eq. (4.5) can be formulated in the time-domain as a recursive filter

$$\begin{aligned} y(n) \quad = \quad & \frac{1}{a_0}\{b_0x(n) + b_1x(n-1) + \ldots + b_Nx(n-N)) \\ & -a_1y(n-1) - a_2y(n-2) - \ldots - a_Ny(n-N)\} \end{aligned} \tag{4.12}$$

By applying the z-transformation to this recursive formulation, Eq. (4.11) can be rewritten explicitly in terms of the input pressure values as

$$Y = \frac{1}{a_0}\{(b_0 + B(z))(P_{i-1,j,k} - P_{i+1,j,k}) - A(z)Y\} \tag{4.13}$$

Again, we have used the definition $Y(z) \equiv Y$. This explicit filter formulation can be rewritten in two parts, namely a part consisting of the present input filter values, and a part consisting of the previous values

$$Y = \overbrace{\frac{b_0}{a_0}(P_{i-1,j,k} - P_{i+1,j,k})}^{\text{present filter values}} + \overbrace{\frac{G}{a_0}}^{\text{previous filter values}} \tag{4.14}$$

where

$$G = B(z)(P_{i-1,j,k} - P_{i+1,j,k}) - A(z)Y \tag{4.15}$$

Next, substituting Eq. (4.14) into Eq. (4.11) gives us

$$P_{i,j,k}(z - z^{-1}) = \lambda \left[\frac{b_0}{a_0}(P_{i-1,j,k} - P_{i+1,j,k}) + \frac{G}{a_0} \right] \tag{4.16}$$

Since the goal is a formulation for the ghost point located at position $(i+1, j, k)$, we isolate the ghost point $P_{i+1,j,k}$

$$P_{i+1,j,k} = P_{i-1,j,k} + \frac{a_0}{\lambda b_0} P_{i,j,k}(z - z^{-1}) + \frac{G}{b_0} \tag{4.17}$$

Such a splitting is necessary in order to separate the current filter values $P_{i-1,j,k}$ and $P_{i+1,j,k}$ from the explicit filter equation. Applying the inverse z-transform of the above equation finally yields the update equation for the ghost point for a right boundary

$$p^n_{i+1,j,k} = p^n_{i-1,j,k} + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n}{b_0} \tag{4.18}$$

The intermediate value g is obtained taking the inverse z-transform of Eq. (4.15) using the unilateral Z-transform indexed with $i = 1$ and by also making use of the convolution theorem $\sum_{i=0}^{\infty} x_1(i)x_2(n-i) = X_1(z)X_2(z)$, we get

$$g^n = \sum_{i=1}^{N} [b_i x^{n-i} - a_i y^{n-i}] \tag{4.19}$$

where the filter input x^n is given by

$$x^n = p^n_{i-1,j,k} - p^n_{i+1,j,k} = \frac{a_0}{\lambda b_0}(p^{n+1}_{i,j,k} - p^{n-1}_{i,j,k}) - \frac{g^n}{b_0} \tag{4.20}$$

The last equality is obtained by substituting $p^n_{i+1,j,k}$ from Eq. (4.18) into $p^n_{i-1,j,k} - p^n_{i+1,j,k}$. Similarly, the inverse z-transform of Eq. (4.14) is

$$y^n = \frac{1}{a_0}[b_0 x^n + g^n] \tag{4.21}$$

denoting the filter output at time step n. The final update scheme is then obtained by inserting Eq. (4.18) into the update scheme in Eq. (3.8), yielding

$$\begin{aligned} p_{i,j,k}^{n+1} &= \Bigg[d_1 (2p_{i-1,j,k}^{n} + p_{i,j+1,k}^{n} + p_{i,j-1,k}^{n} + p_{i,j,k+1}^{n} + p_{i,j,k-1}^{n}) \\ &\quad + \frac{\lambda^2}{b_0} g^n + \left(\frac{\lambda a_0}{b_0} - 1 \right) p_{i,j,k}^{n-1} \Bigg] / \left(1 + \frac{\lambda a_0}{b_0} \right) \end{aligned} \tag{4.22}$$

4.2.2 Remarks

By formulating the update scheme as in Eq. (4.22), we do not need to compute the ghost points from Eq. (4.18) explicitly. The update equation takes the 3-D wave into account and the filter formulation in Eq. (4.19) assumes LRS as stated initially.

To summarise, at each time-step the boundary is updated in the following order:

- Update $p_{i,j,k}^{n+1}$ using the filter value g^n.
- Compute x^n from $p_{i,j,k}^{n+1}$, $p_{i,j,k}^{n-1}$ and g^n.
- Compute y^n from x^n and g^n.
- Update g^{n+1} from x^n and y^n.

For each boundary, the N previous values of x and y have to be kept in memory, corresponding to the filter order.

The introduction of the intermediate variable g might seems unnecessary, since $P_{i+1,j,k}$ in Eq. (4.10) could directly be isolated and a different set of update equation derived. However, it is argued in (Kowalczyk, 2008b) that doing that, instabilities can arise from numerical round-of errors.

4.2.3 Other Axial Boundaries

The other axial boundaries (left, upper, lower, front, back) proceeds in the same manner, but involves changes of relevant indexes and changing the flow normal to the boundary where required. The left boundary node for a plane boundary

is for example given by

$$p^n_{i+1,j} = \Big[d_1(2p^n_{i+1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1}) + \frac{\lambda^2}{b_0} g^n + \left(\frac{\lambda a_0}{b_0} - 1\right) p^{n-1}_{i,j,k} \Big] / \left(1 + \frac{\lambda a_0}{b_0}\right) \tag{4.23}$$

where the ghost point is given by

$$p^n_{i-1,j,k} = p^n_{i+1,j.k} + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n}{b_0} \tag{4.24}$$

The explicit filter difference equation is updated according to

$$y^n = \frac{1}{a_0}(b_0 x^n + g^n), \tag{4.25}$$

the intermediate filter value is given by

$$g^n = \sum_{i=1}^{N} (b_i x^{n-i} - a_i y^{n-i}), \tag{4.26}$$

and the filter input at time n is given by

$$x^n = p^n_{i+1,j,k} - p^n_{i-1,j,k} \tag{4.27}$$

Notice that inserting Eq. (4.24) into Eq. (4.27) yields the same formulation for the filter input given in Eq. (4.20).

4.2.4 Edges

The ghost points for an outer edge depicted in Figure 4.1b are eliminated in a similar manner as for the plane boundaries. For a right z-front edge, two ghost points are eliminated with the use of the two independent boundary conditions in the x- and y-direction as given by Eq. (4.3)

$$\frac{\partial p}{\partial t} = -c\xi_x \frac{\partial p}{\partial x}, \qquad \frac{\partial p}{\partial t} = -c\xi_y \frac{\partial p}{\partial y} \tag{4.28}$$

By using the discretisation in Eq. (4.8) for both dimensions and following the same procedure as for the plane boundaries, we get the update formulas for the two ghost points

$$p^n_{i+1,j,k} = p^n_{i-1,j,k} + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_x}{b_0} \tag{4.29}$$

$$p^n_{i,j-1,k} = p^n_{i,j+1,k} + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_y}{b_0} \tag{4.30}$$

The intermediate values g_x and g_y are updated according to Eq. (4.19) for each of the x- and y-dimensions. Substituting the ghost points in the discretised wave equation in Eq. (3.8) (using the parameters for the SLF scheme) with the above formulas, we get the update scheme for a right outmost y-edge

$$\begin{aligned} p_{i,j,k}^{n+1} &= \Bigg[d_1(2p_{i-1,j,k}^n + 2p_{i,j+1,k}^n + p_{i,j,k+1}^n + p_{i,j,k-1}^n) \\ &\quad +\lambda^2 \left(\frac{g_x^n}{b_{0,x}} + \frac{g_y^n}{b_{0,y}} \right) + \left(\frac{\lambda a_{0,x}}{b_{0,x}} + \frac{\lambda a_{0,y}}{b_{0,y}} - 1 \right) p_{i,j,k}^{n-1} \Bigg] / \\ &\quad \left(1 + \frac{\lambda a_{0,x}}{b_{0,x}} + \frac{\lambda a_{0,y}}{b_{0,y}} \right) \end{aligned} \tag{4.31}$$

4.2.5 Corners

For an outermost right-upper corner depicted in Figure 4.1c, we have three ghost points to eliminate in the x-, y- and z-direction. By discretising the independent boundary formulation for all three dimensions

$$\frac{\partial p}{\partial t} = -c\xi_x \frac{\partial p}{\partial x}, \qquad \frac{\partial p}{\partial t} = -c\xi_y \frac{\partial p}{\partial y}, \qquad \frac{\partial p}{\partial t} = -c\xi_z \frac{\partial p}{\partial z} \tag{4.32}$$

and following the same procedure as before, we get the update formula for an outermost right-upper corner as

$$\begin{aligned} p_{i,j,k}^{n+1} &= \Bigg[d_1(2p_{i-1,j,k}^n + 2p_{i,j+1,k}^n + 2p_{i,j,k-1}^n) \\ &\quad +\lambda^2 \left(\frac{g_x^n}{b_{0,x}} + \frac{g_y^n}{b_{0,y}} + \frac{g_z^n}{b_{0,z}} \right) + \left(\frac{\lambda a_{0,x}}{b_{0,x}} + \frac{\lambda a_{0,y}}{b_{0,y}} + \frac{\lambda a_{0,z}}{b_{0,z}} - 1 \right) p_{i,j,k}^{n-1} \Bigg] / \\ &\quad \left(1 + \frac{\lambda a_{0,x}}{b_{0,x}} + \frac{\lambda a_{0,y}}{b_{0,y}} + \frac{\lambda a_{0,z}}{b_{0,z}} \right) \end{aligned} \tag{4.33}$$

The intermediate value g for each of the x- y- and z-dimensions is still updated according to Eq. (4.19).

4.2.6 Final Remarks

In Figure 4.2a and 4.2b, we see the neighbour points when considering inner edges and corners. For the SLF scheme, no ghost points apply therefore the update equation for an inner grid point as given in Eq. (3.8) can be used, since no impedance boundaries occur. For inner-edge corners depicted in 4.2c, one impedance boundary occur and hence the update formula for a plane boundary can be used.

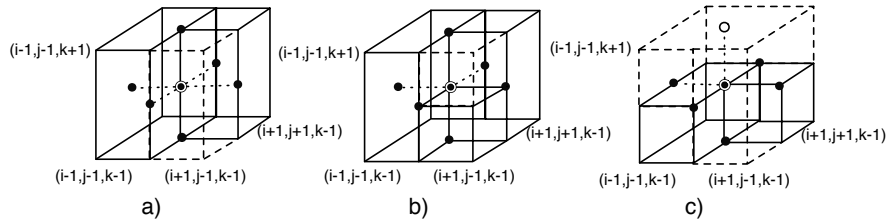


Figure 4.2: a) Inner right-front z-edge, b) Inner right-front z corner, and c) Inner right-front z edge-corner. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles. The point that is being updated is indicated with a black-coloured circle with an surrounding circle.

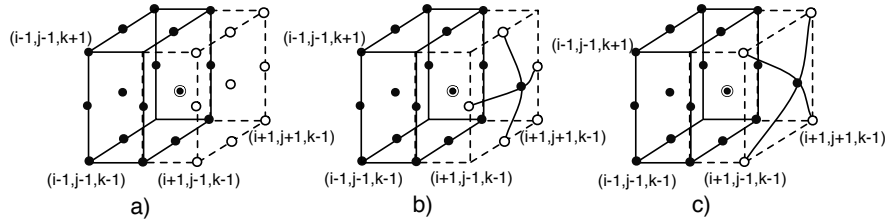


Figure 4.3: Right-plane boundary point depicted with corresponding neighbour points. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles. The point that is being updated is indicated with a black-coloured circle with an surrounding circle. a) All 9 ghost point, b) interpolated side-diagonal points, and c) interpolated diagonal points.

4.3 General Boundary Formulation for the General Family of 3-D Non-Staggered Compact Explicit Schemes

In the previous section, the general boundary formulation was derived for the standard leapfrog scheme. In this section, we will extend the formulation such that the general family of 3-D non-staggered compact explicit schemes from Eq. (3.8) is supported.

4.3.1 Plane Boundaries

Let us consider a right boundary point for a boundary plane as depicted in Figure 4.3. For such a point there are nine ghost points outside the modelled space and these points should be eliminated in such a way that a physical correct absorption occur. We will divide the elimination into three steps (Kowalczyk and van Walstijn (2010b), Kowalczyk and van Walstijn (2011)): For an axial ghost point (Fig. 4.1a), 2) for four side-diagonal ghost points (Fig. 4.3b) and 3) for four diagonal ghost points (Fig. 4.3c). The first elimination is the same as for the SLF scheme given by Eq. (4.22). For the second elimination, we apply a linear interpolation of all the side-diagonal grid points lying inside and outside the modelled space:

$$\tilde{p}^n_{i-1,j,k} = \frac{1}{4}(p^n_{i-1,j+1,k} + p^n_{i-1,j-1,k} + p^n_{i-1,j,k+1} + p^n_{i-1,j,k-1}) \quad (4.34)$$

$$\tilde{p}^n_{i+1,j,k} = \frac{1}{4}(p^n_{i+1,j+1,k} + p^n_{i+1,j-1,k} + p^n_{i+1,j,k+1} + p^n_{i+1,j,k-1}) \quad (4.35)$$

where $\tilde{p}^n_{i-1,j,k}$ and $\tilde{p}^n_{i+1,j,k}$ are the interpolated pressure values. For elimination step 3, we perform a similar interpolation of all the diagonal points lying inside and outside the modelled space:

$$\begin{aligned}\bar{p}_{i-1,j,k} = \frac{1}{4}(&p^n_{i-1,j+1,k+1} + p^n_{i-1,j-1,k+1} \\ &+p^n_{i-1,j+1,k-1} + p^n_{i-1,j-1,k-1})\end{aligned} \quad (4.36)$$

$$\begin{aligned}\bar{p}_{i+1,j,k} = \frac{1}{4}(&p^n_{i+1,j+1,k+1} + p^n_{i+1,j-1,k+1} \\ &+p^n_{i+1,j+1,k-1} + p^n_{i+1,j-1,k-1})\end{aligned} \quad (4.37)$$

where $\bar{p}_{i-1,j,k}$ and $\bar{p}_{i+1,j,k}$ are the interpolated pressure values. Let us take a closer look at these interpolated values in 2-D for simplicity. In 2-D, we have three ghost points as illustrated in Figure 4.4. The interpolated points $\tilde{p}_{i-1,j}$ and $\tilde{p}_{i+1,j}$ are lying on the circle going through the diagonal points used for the interpolation, and hence we have the same distance to the interpolated points as to the original points $p_{i-1,j-1}, p_{i-1,j+1}, p_{i+1,j+1}$ and $p_{i+1,j-1}$. This is important for the numerical stability of the scheme, since having individual points in the grid located with a different grid spacing than the rest of the point can violate the stability condition. In 3-D, the same distance property for the interpolated points is valid, since $\tilde{p}_{i-1,j,k}$ and $\tilde{p}_{i+1,j,k}$ are located on the sphere going through all eight side-diagonal ghost points, whereas $\bar{p}_{i-1,j,k}$ and $\bar{p}_{i+1,j,k}$ are located on the sphere going through all eight diagonal ghost points, as depicted in Figure 4.3 b) and c).

Now, since the interpolated boundary points are located across the boundary, we

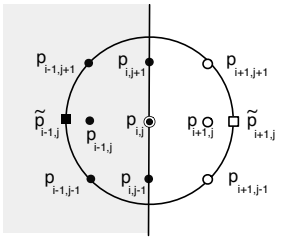


Figure 4.4: Right edge boundary in 2-D with interpolated neighbour points. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles and the point being updated is indicated with a black-coloured circle with an surrounding circle. The interpolated points are indicated with a square.

can perform the elimination of the ghost points as done in the previous section

$$p^n_{i+1,j,k} = p^n_{i-1,j,k} + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_a}{b_0} \tag{4.38}$$

$$\tilde{p}^n_{i+1,j,k} = \tilde{p}^n_{i-1,j,k} + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_{sd}}{b_0} \tag{4.39}$$

$$\bar{p}^n_{i+1,j,k} = \bar{p}^n_{i-1,j,k} + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_d}{b_0} \tag{4.40}$$

We have three intermediate filter values g^n_a, g^n_{sd} and g^n_d, where the subscripts denote the axial, side-diagonal and diagonal filter values, respectively. When formulating the filter input values x, we notice that the interpolated values are eliminated:

$$x^n_a = p^n_{i-1,j,k} - p^n_{i+1,j,k} = \frac{a_0}{\lambda b_0}(p^{n+1}_{i,j,k} - p^{n-1}_{i,j,k}) - \frac{g^n_a}{b_0} \tag{4.41}$$

$$x^n_d = \tilde{p}^n_{i-1,j,k} - \tilde{p}^n_{i+1,j,k} = \frac{a_0}{\lambda b_0}(p^{n+1}_{i,j,k} - p^{n-1}_{i,j,k}) - \frac{g^n_d}{b_0} \tag{4.42}$$

$$x^n_{sd} = \bar{p}^n_{i-1,j,k} - \bar{p}^n_{i+1,j,k} = \frac{a_0}{\lambda b_0}(p^{n+1}_{i,j,k} - p^{n-1}_{i,j,k}) - \frac{g^n_{sd}}{b_0} \tag{4.43}$$

We can formulate the interpolated pressure values in Eq. (4.39) in terms of the points lying on the original rectangular grid as

$$\begin{aligned}\frac{1}{4}(p^n_{i+1,j+1,k} + p^n_{i+1,j-1,k} + p^n_{i+1,j,k+1} + p^n_{i+1,j,k-1}) = \\ \frac{1}{4}(p^n_{i-1,j+1,k} + p^n_{i-1,j-1,k} + p^n_{i-1,j,k+1} + p^n_{i-1,j,k-1}) \\ + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_{sd}}{b_0}\end{aligned} \tag{4.44}$$

and with a similar formulation for the diagonal interpolated ghost point in Eq. (4.40)

$$\begin{aligned}\frac{1}{4}(p^n_{i+1,j+1,k+1}+p^n_{i+1,j-1,k+1}+p^n_{i+1,j+1,k-1}+p^n_{i+1,j-1,k-1}) = \\ \frac{1}{4}(p^n_{i-1,j+1,k+1}+p^n_{i-1,j-1,k+1}+p^n_{i-1,j+1,k-1}+p^n_{i-1,j-1,k-1}) \\ +\frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k}-p^{n+1}_{i,j,k})+\frac{g^n_d}{b_0}\end{aligned} \tag{4.45}$$

Isolating one of the ghost points in each of the equations (4.44) and (4.45), and inserting these in the compact scheme in Eq. (3.8) together with (4.38) yields

$$\begin{aligned}p^{n+1}_{i,j,k} \;=\; & \Big[d_1(2p^n_{i-1,j,k}+p^n_{i,j+1,k}+p^n_{i,j-1,k}+p^n_{i,j,k+1}+p^n_{i,j,k-1}+\frac{g^n_a}{b_0}) \\ & +d_2(2p^n_{i-1,j+1,k}+2p^n_{i-1,j-1,k}+2p^n_{i-1,j,k+1}+2p^n_{i-1,j,k-1} \\ & +p^n_{i,j+1,k+1}+p^n_{i,j+1,k-1}+p^n_{i,j-1,k+1}+p^n_{i,j-1,k-1}+4\frac{g^n_{sd}}{b_0}) \\ & +d_3(2p^n_{i-1,j+1,k+1}+2p^n_{i-1,j-1,k+1}+2p^n_{i-1,j+1,k-1}+2p^n_{i-1,j-1,k-1}+4\frac{g^n_d}{b_0}) \\ & +d_4p^n_{i,j,k}+\left(\frac{\lambda a_0}{b_0}-1\right)p^{n-1}_{i,j,k}\Big] \Big/ \left(1+\frac{\lambda a_0}{b_0}\right)\end{aligned} \tag{4.46}$$

This formulation would require, that three intermediate filter values should be implemented for each boundary node, but since the filter impedance coefficients b_i and a_i in the formulation of g in Eq. (4.19) are the same for all three boundaries (the boundaries belong to the same surface, and we enforce LRS), the only difference is the input signal x and, hence, the output signal y. By moving the g's outside the parenthesis, we can write the above equation as

$$\begin{aligned}p^{n+1}_{i,j,k} \;=\; & \Big[d_1(2p^n_{i-1,j,k}+p^n_{i,j+1,k}+p^n_{i,j-1,k}+p^n_{i,j,k+1}+p^n_{i,j,k-1}) \\ & +2d_2(p^n_{i-1,j+1,k}+p^n_{i-1,j-1,k}+p^n_{i-1,j,k+1}+p^n_{i-1,j,k-1}) \\ & +d_2(p^n_{i,j+1,k+1}+p^n_{i,j+1,k-1}+p^n_{i,j-1,k+1}+p^n_{i,j-1,k-1}) \\ & +2d_3(p^n_{i-1,j+1,k+1}+p^n_{i-1,j-1,k+1}+p^n_{i-1,j+1,k-1}+p^n_{i-1,j-1,k-1}) \\ & +d_4p^n_{i,j,k}+\frac{\lambda^2}{b_0}g^n+\left(\frac{\lambda a_0}{b_0}-1\right)p^{n-1}_{i,j,k}\Big] \Big/ \left(1+\frac{\lambda a_0}{b_0}\right)\end{aligned} \tag{4.47}$$

where the three intermediate filters have been grouped into one filter g given as

$$g^n=(1-4a+4b)\frac{g^n_a}{b_0}+(4a-8b)\frac{g^n_{sd}}{b_0}+4b\frac{g^n_d}{b_0} \tag{4.48}$$

The added λ^2 in Eq. (4.47) and the coefficients in front of the intermediate filters in Eq. (4.48) stems from the scheme parameter d_1, d_2 and d_3 defined in Eq. (3.9).

The intermediate filter is defined in Eq. (4.19) as

$$g^n = \sum_{i=1}^{N} [b_i x^{n-i} - a_i y^{n-i}] \tag{4.49}$$

and since the impedance filter g is a linear system (consists only of sums of scaled input and output values), the superposition principle can be applied[2], such that the weighting coefficients in front of the filters g_a^n, g_a^n and g_a^n can be moved into the sum, leading to the filter input values

$$x^n = (1 - 4a + 4b)x_a^n + (4a - 8b)x_{sd}^n + 4bx_d^n \tag{4.50}$$

Inserting the definition of x_a, x_{sd} and x_s from Eq. (4.20) into the above equation, gives

$$\begin{aligned} x^n &= (1 - 4a + 4b)\left(\frac{a_0}{\lambda b_0}(p_{i,j,k}^{n+1} - p_{i,j,k}^{n-1}) - \frac{g_a^n}{b_0}\right) \\ &\quad +(4a - 8b)\left(\frac{a_0}{\lambda b_0}(p_{i,j,k}^{n+1} - p_{i,j,k}^{n-1}) - \frac{g_d^n}{b_0}\right) \\ &\quad +4b\left(\frac{a_0}{\lambda b_0}(p_{i,j,k}^{n+1} - p_{i,j,k}^{n-1}) - \frac{g_{sd}^n}{b_0}\right) \\ &= \frac{a_0}{\lambda b_0}(p_{i,j,k}^{n+1} - p_{i,j,k}^{n-1})\Big((1 - 4a + 4b) + (4a - 8b) + 4b\Big) \\ &\quad -(1 - 4a + 4b)\frac{g_a^n}{b_0} - (4a - 8b)\frac{g_d^n}{b_0} - 4b\frac{g_{sd}^n}{b_0} \\ &= \frac{a_0}{\lambda b_0}(p_{i,j,k}^{n+1} - p_{i,j,k}^{n-1}) - \frac{g^n}{b_0} \end{aligned}$$

The formulation after applying the superposition principle is the same as for the individual filters, hence applying the superposition principle for y_a, y_{sd} and y_d will also take the same form as Eq. (4.21). We can therefore update the filter values g and the corresponding x and y values using the usual formulas.

4.3.2 Outer Corner

For an outer corner, 19 ghost points have to be eliminated as depicted in Figure 4.5. Three boundary conditions apply as given in Eq. (4.32), and for eliminating

[2]Stating that the net response of two or more stimuli to a system is the sum of each of the stimuli observed individually

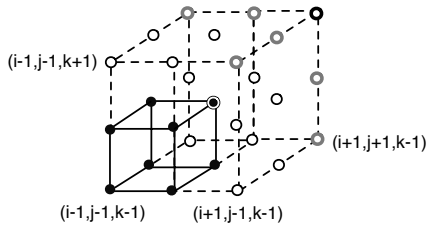


Figure 4.5: Outer outmost upper-right corner. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles with the corner ghost point depicted with a thicker surrounding and edge ghost points with a grey surrounding. The point that is being updated is indicated with a black-coloured circle with an surrounding circle.

the diagonal and side-diagonal boundary points, the same procedure as for the boundary plane is applied, but now for all three boundaries. The additional condition

$$\frac{\partial p^3}{\partial x \partial y \partial z} = 0 \tag{4.51}$$

holds at the corner (indicated by ghost points with a black bold surrounding), and

$$\frac{\partial p^2}{\partial x \partial y} = 0, \quad \frac{\partial p^2}{\partial x \partial z} = 0, \quad \frac{\partial p^2}{\partial y \partial z} = 0 \tag{4.52}$$

holds at boundary edges, which guarantees local coherence between two meeting boundaries. Discretising Eq. (4.51) using centered differences yields

$$\begin{aligned} p^n_{i+1,j+1,k+1} &= p^n_{i+1,j+1,k-1} + p^n_{i+1,j-1,k+1} + p^n_{i-1,j+1,k+1} + p^n_{i-1,j-1,k-1} \\ &- p^n_{i+1,j-1,k-1} - p^n_{i-1,j+1,k-1} - p^n_{i-1,j-1,k+1} \end{aligned} \tag{4.53}$$

and discretising the edge ghost points in (4.52) (indicated by ghost points with a grey bold surrounding) for an x,y (z-direction), x,z (y-direction) and y,z-boundary (x-direction) using centered differences, we get

$$p^n_{i+1,j+1,k} = p^n_{i+1,j-1,k} + p^n_{i-1,j+1,k} - p^n_{i-1,j-1,k} \tag{4.54}$$

$$p^n_{i+1,j,k+1} = p^n_{i+1,j,k-1} + p^n_{i-1,j,k+1} - p^n_{i-1,j,k-1} \tag{4.55}$$

$$p^n_{i,j+1,k+1} = p^n_{i,j+1,k-1} + p^n_{i,j-1,k+1} - p^n_{i,j-1,k-1} \tag{4.56}$$

For the outermost right-upper corner we would then use Eq. (4.53) for eliminating the corner ghost point, Eq. (4.54)-(4.56) for edge ghost points and Eq. (4.38), (4.44) and (4.45) with indexes corresponding to the x, y and z-dimension

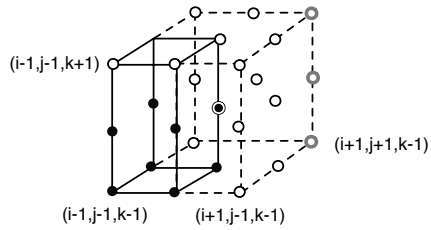


Figure 4.6: Outmost right z-edge. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles, with edge ghost points depicted with grey surroundings. The point that is being updated is indicated with a black-coloured circle with an surrounding circle.

for the impedance boundaries. This yields the update formula for the upper-outmost corner:

$$\begin{aligned} p_{i,j,k}^{n+1} \;=\; & \Big[2d_1(p_{i-1,j,k}^n + p_{i,j-1,k}^n + p_{i,j1,k-1}^n) \\ & +4d_2(p_{i-1,j-1,k}^n + p_{i-1,j,k-1}^n + p_{i,j-1,k-1}^n) + 8d_3(p_{i-1,j-1,k-1}^n) \\ & +d_4 p_{i,j,k}^n + \lambda^2 \left(\frac{g_x^n}{b_{x,0}} + \frac{g_y^n}{b_{y,0}} + \frac{g_z^n}{b_{z,0}} \right) \\ & + \left(\frac{\lambda a_{x,0}}{b_{x,0}} + \frac{\lambda a_{y,0}}{b_{y,0}} + \frac{\lambda a_{z,0}}{b_{z,0}} - 1 \right) p_{i,j,k}^{n-1} \Big] \Big/ \left(1 + \frac{\lambda a_{x,0}}{b_{x,0}} + \frac{\lambda a_{y,0}}{b_{y,0}} + \frac{\lambda a_{z,0}}{b_{z,0}} \right) \end{aligned} \tag{4.57}$$

4.3.3 Outer Edges

For an outer edge, two boundary conditions applies as depicted in Figure 4.6. For the outermost right x,y-edge, we substitute the edge ghost points $p_{i+1,j+1,k-1}^n$, $p_{i+1,m+1,j}^n$ and $p_{i+1,j+1,k+1}^n$ coloured with grey surroundings using the local coherence equation (4.54). The ghost points depicted with white-coloured circles are eliminated using the interpolated formulas given by Eq. (4.38), (4.44) and (4.45) with indexes corresponding to the dimension considered. This leads to

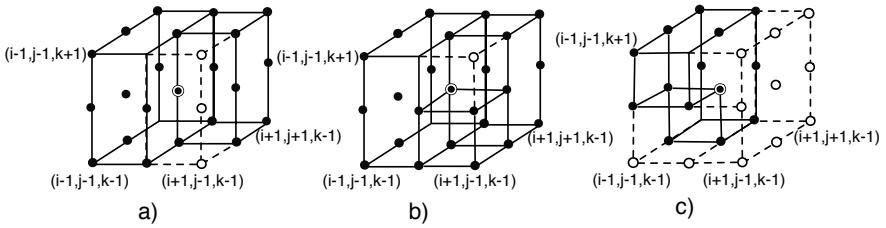


Figure 4.7: Inner points. a) Right x, z front edge with 3 edge ghost points, b) frontmost right upper corner with 1 corner ghost point, c) Lower y, z frontmost edge-corner with 11 ghost points. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles. The point that is being updated is indicated with a black-coloured circle with an surrounding circle.

the update formula for the outermost right edge:

$$\begin{aligned} p^{n+1}_{i,j,k} &= \Big[d_1(2p^n_{i-1,j,k} + 2p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1}) \\ &\quad + d_2(4p^n_{i-1,j-1,k} + 2p^n_{i-1,j,k-1} + 2p^n_{i-1,j,k+1} + 2p^n_{i,j-1,k-1} + 2p^n_{i,j-1,k+1}) \\ &\quad + 4d_3(p^n_{i-1,j-1,k-1} + p^n_{i-1,j-1,k+1}) \\ &\quad + d_4 p^n_{i,j,k} + \lambda^2 \left(\frac{g^n_x}{b_{x,0}} + \frac{g^n_y}{b_{y,0}} \right) \\ &\quad + \left(\frac{\lambda a_{x,0}}{b_{x,0}} + \frac{\lambda a_{x,0}}{b_{x,0}} - 1 \right) p^{n-1}_{i,j,k} \Big] \Big/ \left(1 + \frac{\lambda a_{x,0}}{b_{x,0}} + \frac{\lambda a_{y,0}}{b_{y,0}} \right) \end{aligned} \tag{4.58}$$

4.3.4 Inner Corners and Edges

Inner edges and inner corners are depicted in Figure 4.7a and 4.7b, respectively. No impedance boundaries apply, since we do not consider boundary impedances in oblique directions, and neighbour points exists in the 6 axial direction. However, in contrary to the SLF scheme, we have one corner ghost point for an inner corner, which can be eliminated using Eq. (4.53) for three meeting boundaries. For an inner edge, three ghost points have to be eliminated using one of the equations (4.54)-(4.56) for two meeting boundaries. Considering a frontmost right-upper inner corner (Fig. 4.7b), the ghost point $p_{i+1,j-1,k+1}$ is eliminated using

$$\begin{aligned} p^n_{i+1,j-1,k+1} &= p^n_{i+1,j+1,k+1} + p^n_{i+1,j-1,k-1} + p^n_{i-1,j+1,k-1} + p^n_{i-1,j-1,k+1} \\ &\quad - p^n_{i+1,j+1,k-1} - p^n_{i-1,j+1,k+1} - p^n_{i-1,j-1,k-1} \end{aligned}$$

yielding the update equation

$$\begin{aligned} p^{n+1}_{i,j,k} &= d_1(p^n_{i+1,j,k} + p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1}) \\ &\quad +d_2(p^n_{i+1,j,k+1} + p^n_{i+1,j,k-1} + p^n_{i+1,j+1,k} + p^n_{i+1,j-1,k} \\ &\quad +p^n_{i,j+1,k+1} + p^n_{i,j+1,k-1} + p^n_{i,j-1,k+1} + p^n_{i,j-1,k-1} \\ &\quad +p^n_{i-1,j,k+1} + p^n_{i-1,j-1,k} + p^n_{i-1,j,k+1} + p^n_{i-1,j-1,k}) \\ &\quad +2d_3(p^n_{i+1,j+1,k+1} + p^n_{i+1,j-1,k-1} + p^n_{i-1,j-1,k+1} + p^n_{i-1,j+1,k-1}) \\ &\quad +d_4 p^n_{i,j,k} - p^{n-1}_{i,j,k} \end{aligned} \tag{4.59}$$

For a right x, z frontmost edge (Fig. 4.7a), the ghost points $p_{i+1,j-1,k-1}$, $p_{i+1,j-1,k}$ and $p_{i+1,j-1,k+1}$ are eliminated using Eq. (4.54) for two meeting boundaries in the y-direction

$$p^n_{i+1,j-1,k} = p^n_{i+1,j+1,k} + p^n_{i-1,j-1,k} - p^n_{i-1,j+1,k}$$

Substituting the three ghost points using the above equation for the indexes $k-1$, k and $k+1$ gives us the update equation

$$\begin{aligned} p^{n+1}_{i,j,k} &= d_1(p^n_{i+1,j,k} + p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1}) \\ &\quad +d_2(p^n_{i+1,j,k+1} + p^n_{i+1,j,k-1} + 2p^n_{i+1,j+1,k} + p^n_{i,j+1,k+1} + p^n_{i,j.1,k+1} \\ &\quad +p^n_{i,j+1,k-1} + p^n_{i,j-1,k-1} + p^n_{i-1,j,k+1} + p^n_{i-1,j,k-1} + 2p^n_{i-1,j-1,k}) \\ &\quad +2d_3(p^n_{i+1,j+1,k+1} + p^n_{i+1,j+1,k-1} + p^n_{i-1,j-1,k+1} + p^n_{i-1,j-1,k-1}) \\ &\quad +d_4 p^n_{i,j,k} - p^{n-1}_{i,j,k} \end{aligned} \tag{4.60}$$

4.3.5 Inner Edge-Corner

An inner edge-corner is depicted in Figure 4.7c, for which one impedance boundary occur. In Appendix 9.2, the update formula has been derived from the basic inner point update equation (3.8). Another way around is to use the update formula for a plane as a starting point – doing that, only two ghost points have to be eliminated for which one of Eq. (4.54)-(4.56) is used for two meeting boundaries. For a right y, z frontmost inner edge-corner, we can use the update equation for a right plane given in Eq. (4.47), and then eliminating the ghost points $p_{i,j-1,k-1}$ and $p_{i-1,j-1,k-1}$ using Eq. (4.56) for the x-direction. This

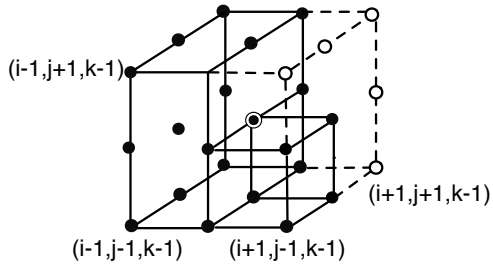


Figure 4.8: Pyramid boundary. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles. The point that is being updated is indicated with a black-coloured circle with an surrounding circle.

gives us the update formula

$$\begin{aligned} p^{n+1}_{i,j,k} &= \Big[d_1(2p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1}) \\ &\quad +2d_2(p^n_{i-1,j+1,k} + p^n_{i-1,j-1,k} + p^n_{i-1,j,k+1} + p^n_{i-1,j,k-1} \\ &\quad +p^n_{i,j+1,k-1} + p^n_{i,j-1,k+1}) \\ &\quad +4d_3(p^n_{i-1,j-1,k+1} + p^n_{i-1,j+1,k-1}) \\ &\quad +d_4 p^n_{i,j,k} + \frac{\lambda^2}{b_0} g^n + \left(\frac{\lambda a_0}{b_0} - 1\right) p^{n-1}_{i,j,k} \Big] \Big/ \left(1 + \frac{\lambda a_0}{b_0}\right) \end{aligned} \tag{4.61}$$

4.3.6 Pyramid Boundary

Finally, a pyramid boundary is depicted in Figure (4.8). This point type has not been implemented, but is given for completeness:

$$\begin{aligned} p^{n+1}_{i,j,k} &= d_1(p^n_{i+1,j,k} + p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1}) \\ &\quad +d_2(p^n_{i,j+1,k+1} + p^n_{i,j+1,k-1} + p^n_{i,j-1,k+1} + p^n_{i,j-1,k-1} \\ &\quad +2p^n_{i-1,j+1,k} + 2p^n_{i+1,j-1,k} + 2p^n_{i-1,j,k+1} + 2p^n_{i+1,j,k-1}) \\ &\quad +d_3(2p^n_{i-1,j-1,k+1} + 2p^n_{i-1,j+1,k-1} + 2p^n_{i+1,j-1,k-1} \\ &\quad +p^n_{i-1,j-1,k-1} + p^n_{i-1,j+1,k+1}) \\ &\quad +d_4 p^n_{i,j,k} - p^{n-1}_{i,j,k} \end{aligned} \tag{4.62}$$

4.3.7 Final Remarks

Having derived 8 families of update formulas for all types of boundary points, we have 8 outer corners, 12 outer edges, 6 plane boundaries, 8 inner corners, 12 inner edges and 24 inner edge-corners and 24 pyramid boundaries, giving a total of 96 different update formulas including inner points. The DIF model (i.e. the model concerning boundary points having one or more impedance boundaries) requires updating

1. The pressure $p_{i,j,k}^{n+1}$ using one of the 40 impedance boundary update formulas,
2. The input signal x^n
3. The output signal y^n
4. The impedance filter g^{n+1}

By looking at the formulas for these quantities corresponding to Eq. (4.20), (4.21), (4.19) and a given boundary update formula (e.g. Eq. (4.61)), we notice that g^n is used for updating $p_{i,j,k}^{n+1}$, $p_{i,j,k}^{n+1}$ is used for updating x^n and y^n, and x^n and y^n are used for updating g^{n+1}. Therefore, the update should be done in the order of presentation above.

4.4 Scene Geometry

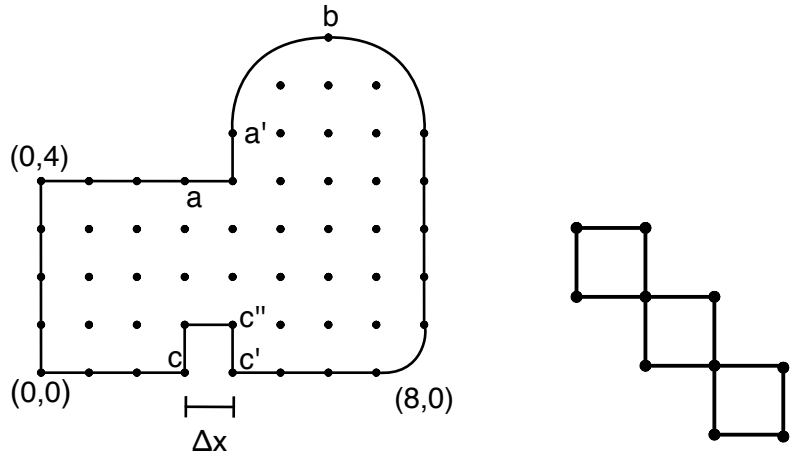
The goal of this project is to simulate the sound field in physical models occurring in the real-word. Therefore we need the ability to model arbitrary geometries and for that we have created a tool for importing CAD (Computer-Aided Design) scenes. The process includes the task of discretising the CAD model in a suitable format for the FDTD solver. The scene data needed depends on the scheme used, where the SLF scheme only needs the axial neighbour points, whereas for the general compact explicit schemes needs a combination of axial, diagonal and side-diagonal points. Functionality from the RAVEN[3] framework developed at ITA primarily for use with geometrical methods has been used for importing and processing CAD scenes. The procedure taken for discretising a

[3]RAVEN (Room Acoustics for Virtual ENvironments) is a hybrid room acoustics simulation software developed at ITA, combining deterministic image source method with a stochastic ray tracing algorithm in order to compute impulse responses in real-time which reach state-of-the-art room acoustics simulation standards.

scene works as follows: First, a start point inside the scene is chosen and all 26 neighbour coordinates are computed from the spatial resolution. If a coordinate is inside the geometry, a point is created and neighbour information is updated, otherwise the point corresponding to the coordinate is denoted as a ghost point. The algorithm is outlined below:

Initialisation:
$W = W_1 = \emptyset$
Choose a starting point $x_0 = (x_0, y_0, z_0)$ inside the scene and add a GridPoint with starting point x_0 and all neighbours set to 0 to the set W.
Loop:
while W $\neq \emptyset$ **do**
    for all GridPoint $p \in W$ **do**
        By using a step size of Δx, add to the set U the coordinates of all p's 26 neighbours in the axial, diagonal and side-diagonal directions.
        for all neighbour points $u \in U$ **do**
            if u is *inside* the geometry **then**
                Add a GridPoint p_{new} with coordinate u to W_1 and update the neighbour indexes pointing at the parent neighbour p. Set all other neighbour indexes to 0, and update p with the id to the new neighbour p_{new}.
            else
                Update the attribute material for the parent point p corresponding to the material encountered.
            end if
        end for
        Set $U = \emptyset$
    end for
    Set $W = W_1$.
end while

The method works, but some restrictions on the scene must be made, as we will explain in the following. A discretisation of a 2-D scene using the above algorithm is depicted in Figure 4.9a. The original scene is indicated by the solid line and includes sharp and soft edges with a mixture of concave and convex shapes. The lower left corner point has coordinate $(0, 0)$ and the grid spacing is Δx. The geometry is meshed as a so-called staircase, and the grid mesh procedure does not know anything about the type of points it encounters, it only checks whether a staircase path exists to a neighbour for a given resolution step. The discretisation looks fine at first sight, but problematic points have been introduced. First, let us consider the point b. The point only has a single neighbour in the axial directions located at coordinate (6,6) corresponding to a string (with no thickness). The corresponding scenario in 3-D would be a



a) b)

Figure 4.9: a) A meshed scene. The solid line is depicting the scene geometry and the dots are the grid points after meshing. b) Example of a discretised geometry non allowed in the FDTD framework.

string or a plane with no thickness, and since wave propagations in spatial 3-D environments should be modelled, these point types are not allowed. We can imagine many situations, where similar problems can occur and another example can be seen in Figure 4.9b. In 3-D, many more of these boundary types are possible. In fact, 2^{26} point combinations can be present after having meshed the scene, out of which only 95 combinations corresponding to the boundary types explained in the previous sections are handled by the FDTD framework. If all CAD scene geometries should be handled, we would need to adjust the scene after meshing to only include points corresponding to the 95 point types, but it is not a trivial task. Instead, we will restrict the CAD scenes such that only allowed point types are present in the scene after discretisation, also making the point classification an easier task (see Section 6.4).

An issue concerning ambiguity can be seen when considering the point c. The geometry has a concave notch in the geometry of width Δx being ignored by the meshing. These kinds of problems occurs when the geometry is more detailed than the spatial resolution allows, like in this case, where the neighbour points c' and c" will be considered as neighbours to the point c. A solution to the problem of "jumping over" geometric elements of sizes equal or less than the spatial resolution can be solved by shooting a ray in the direction of the neighbour elements and see if a boundary is hit. The drawback of shooting a ray is that

small obstacles, that should be ignored, may be hit, and hence be modelled with size Δx in the given direction. Since obstacles with sizes much smaller than the wave length (and hence also the spatial sampling rate Δx) can be neglected, this might lead to unprecise simulations.

Finally, a remark is made on the neighbour relation at oblique boundaries. Because the FDTD framework does not incorporate impedance boundaries across oblique boundary, the point a' should be considered as a ghost point from the point of view of a. This is handled by a post-processing step used for classifying the points which is explained in Section 6.4, and no precautions need to be taken.

Chapter 5

GPU implementation

This section will consider the GPU implementation of the 3D compact explicit schemes with boundaries modelled as DIFs. First, CUDA (Compute Unified Device Architecture) is introduced and the differences between Graphic Processing Units and Central Processing Units is outlined. Having settled the theory, three different GPU FDTD implementations are examined.

5.1 Introduction to CUDA

As the name suggests, Graphics Processing Units (GPUs) was originally designed for graphics processing. But especially with the introduction of CUDA in 2007 (NVidia, 2011b), the application of GPU programming has evolved to other areas as well, including image and video processing, computational biology and chemistry, fluid dynamics simulation, CT image reconstruction, seismic analysis, ray-tracing, and much more. CUDA is NVidia's parallel computing architecture and it enables dramatic increase in computing performance by harnessing the power of the GPU.

The main difference between CPU and GPU is, that a CPU is expected to execute a single task a fast as possible, whereas a GPU must be capable of

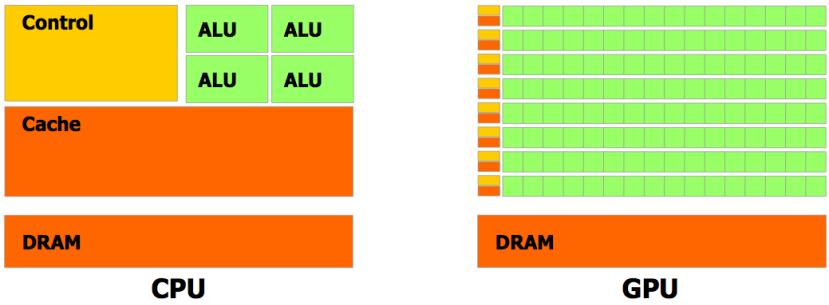


Figure 5.1: Architecture of a a) CPU, and b) GPU. (NVidia, 2011b)

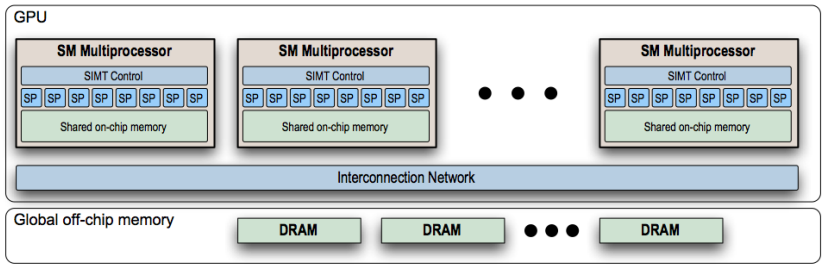


Figure 5.2: Typical GPU consisting of several streaming multiprocessors (SM) each having access to (fast) on-chip and (slow) global memory, the latter shared between all SM. Source: (Savioja, 2010)

executing a maximum amount of tasks as fast a possible. Therefore, the CPU uses more data transistors on tasks related to flow control (branching) and data caching, where the GPU is mainly designed such that more data transistors are devoted to data processing. The difference is depicted in Figure 5.1, where many more arithmetic logic units (ALU) are used in the GPU, but comes with the cost of fewer and smaller cache and control units per ALU. The general architecture of the GPU can be seen in Figure 5.2. A number of streaming multiprocessors are connected through an interconnection network, each having a number of parallel streaming processors (SP) sharing a on-chip memory where data exchange can be done fast. Data exchange between different SMs are done using the global of-chip memory (consisting of DRAM), which are slow compared to the on-chip memory. A SIMT (single instruction multiple thread) interface takes care of running the threads in parallel. The computational model is based on threads that can be run in parallel on the GPU. The programmer writes a kernel that describes the behaviour of the threads and enough threads are launched. The threads are grouped together, where it should be enforced, that each thread in the group has the same execution pattern for optimal performance. If branching

occurs for a given thread, all other threads in a group will have to wait for that thread to finish. Context switches – unlike normal CPU's – can be done without overhead, and therefore more threads that available processors should be feed to the GPU, ensuring that no processors are idle. The use of cache can provide fast access to data, which has been introduced with the Fermi hardware, however the code should still be written to take advantage of the cache utilisation by fetching data with addresses not too far from each other.

The limitation of GPUs is, that it is difficult to fully utilise the capability in practice, since keeping all threads busy may be limited by the resources available and the latency when fetching data from global memory. Also the memory link in between the global memory and the multiprocessors can quickly become a bottleneck, even though recent GPUs having bandwidths of 80-180 GB/s. Consider for example a scene consisting of one million nodes are used with an update range of 44.1 kHz. Then a throughput of 3 layers/update $\times$ 44100 Hz $\times$ 4 bytes/nodes $\times$ 10^6 nodes/layer $\approx$ 500 GB/s is required, which is impossible with recent GPUs.

5.2 The CUDA Programming Model

In CUDA, the computing system consists of a *host* and a *device*. The host is a traditional Central Processing Unit (CPU), where parts of the problem to solve exploits much logic control and less or no parallelism. The device is the domain of the GPU, consisting of massively parallel processors equipped with a large number of arithmetic execution units. The host code is implemented in C/C++, whereas the device code is written using ANSI C code extended with keywords for labelling data-parallel functions, called *kernels*, and their associated data structures. The terms *grid*, *blocks* and *threads* are the basic quantities used when executing a kernel. When launching a CUDA kernel, a grid of threads is executed by the kernel and since all the launched threads are executed using the same kernel, the threads need to rely on unique coordinate to distinguish the individual threads. For this purpose the CUDA runtime system uses a two-level hierarchy consisting of unique coordinates called *blockId* and *threadId*. These build-in variables are initialised at run-time and can be accessed within the kernel. A grid can be one or two-dimensional containing a number of blocks, where each block contains a one- two- or three-dimensional array of threads. The number of threads must be the same for all the blocks in the grid. In Figure 5.3 a 3×2 grid is shown giving a total of 6 blocks each having 12 threads ordered in a 4×3 dimensional grid. The dimensions of the block can be arbitrary chosen as long as the number of thread slots do not exceed the maximum number of allowed threads specified by the hardware.

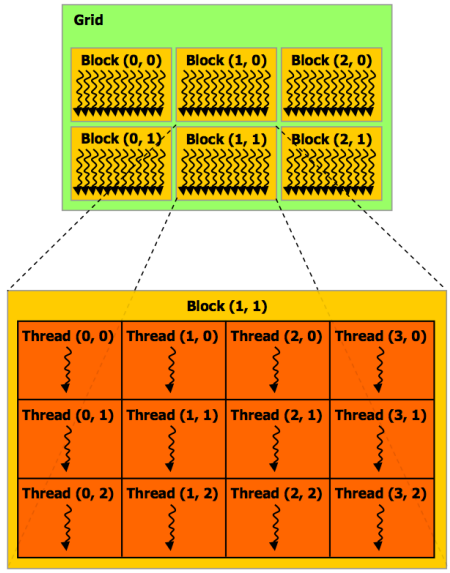


Figure 5.3: Grid of thread block. Source: (NVidia, 2011b).

The threads launched for each block are grouped into *warps*. A warp consists of 32 threads indexed from n to $n+31$. If *branch divergence* should be avoided, all threads in a warp have to execute the same execution pattern. If one thread executes another execution pattern than the rest of the threads (e.g. due a conditional branch), all other threads in the warp will stall and wait for the thread to finish before continuing.

The GPU has several types of memory available for the programmer and an overview is given below:

Global memory is accessible by all threads in the kernel and can be written and read by the host by calling API function. Global memory is implemented as DRAM and is typically the slowest of the memory types. For GPU's of compute capability ≥ 2.0, a cache has been introduced, making access to commonly used data faster.

Shared memory is allocated to thread blocks and can only be written and read by the device. Shared memory resides physically on the GPU as opposed to the global memory residing in off-chip DRAM and provides efficient means for threads to cooperate within a block by sharing the result of their work.

Constant memory is accessible by all threads in the kernel and can be written and read by the host. The constant memory is stored on chip and is suitable for storing constants.

Texture memory is accessible by all threads in the kernel and is designed for memory access patterns that exhibit a great deal of (2D) spatial locality. Texture memory is global memory, but texture are accessed through a dedicated on chip read-only cache.

Registers are allocated to individual threads and each thread can only access its own registers. The register file is the largest and fastest on-chip memory and is used to hold frequently accessed variables local to each thread.

Choosing one of the faster memory over global memory can in some applications speed up the performance, and we will discuss optimisations in Section 5.3.

The main memory resource on the GPU is the global memory which the programmer needs to allocate in a similar manner as on the CPU using

```
cudaMalloc(void** devPtr, size_t size)
```

where devPtr will point to the allocated memory of size size after invocation. Since the device memory is separated from the host memory, the data has to be copied from the host to the device, and for this task, CUDA provides the function

```
cudaMemcpy(void* dst, const void* src, size_t count,
           enum cudaMemcpyKind)
```

The function copies count bytes from the source location src to the destination location dst. The source can for example be the data on the host and the destination being the location on the device returned by cudaMalloc.

5.3 CUDA FDTD Implementation

Having founded the ideas behind CUDA programming, we are ready to go through the implementation of the FDTD solver using the GPU. The first important thing to notice about the explicit schemes is, that the nodal points needed for computing the next state only depend on previous time steps and therefore no dependencies between nodal points are present. In the following pseudo code, the inner loop can be performed in any order, meaning that each point in the geometry can be computed independently.

```
1: for n=1 to N  do
2:     for i=0 to D do
3:         p_i^n = p_{i-1}^{n-1} + p_{i+1}^{n-1} - p_i^{n-2}
4:     end for
5: end for
```

This makes the scheme highly suitable for parallel programming. By using a single thread for each node, the computations can be done in parallel. In the simple case given above, the boundary cases are not considered. The update formulas for computing the pressure values at boundaries and updating the DIFs are different for each type of boundary (inner/outer corner, inner/outer edges, etc), but also distinction between the position of the boundary type (left, right, upper, lower, etc) must be made. All these boundary cases must be determined in the code by using control statements, and will lead to *branch divergence* if care is not taken.

5.3.1 Kernel calls

We have made three different implementations for determining which approach yield the most performance:

The first implementation consists of one main kernel responsible for computing all grid points including updating the DIFs. For minimising branch divergence, the grid points are ordered with respect to grid point type, which means that threads in a warp will follow the same patterns in most cases. Examples for cubic rooms can be seen in Table 7.6 from Chapter 7, where the number of point types are depicted. Except from the corner points, we can assume little branch divergence, since a warp consists of 32 threads and most of these will exhibit the same execution pattern for all threads.

The second implementation also consists of one main kernel responsible for computing all grid points, but updating the DIFs is done in a separate kernel. This should result in a lower register usage for the main kernel, allowing for more parallel threads. For an outer corner, this would imply updating the part of Eq. (4.57) with no impedance calculations involved in the main kernel, and in an external kernel, the value computed in the main kernel is updated with the remain part of the update equation. Hence, the main kernel will calculate

$$\begin{aligned} p_{i,j,k}^{n+1} &= 2d_1(p_{i-1,j,k}^n + p_{i,j-1,k}^n + p_{i,j1,k-1}^n) \\ &\quad +4d_2(p_{i-1,j-1,k}^n + p_{i-1,j,k-1}^n + p_{i,j-1,k-1}^n) \\ &\quad +8d_3(p_{i-1,j-1,k-1}^n) + d_4 p_{i,j,k}^n \end{aligned} \tag{5.1}$$

and the DIF kernel will calculate

$$p_{i,j,k}^{n+1} = \left[p_{i,j,k}^{n+1} + \lambda^2 \left(\frac{g_x^n}{b_{x,0}} + \frac{g_y^n}{b_{y,0}} + \frac{g_z^n}{b_{z,0}} \right) \right. \tag{5.2}$$
$$\left. + \left(\frac{\lambda a_{x,0}}{b_{x,0}} + \frac{\lambda a_{y,0}}{b_{y,0}} + \frac{\lambda a_{z,0}}{b_{z,0}} - 1 \right) p_{i,j,k}^{n-1} \right] / \left(1 + \frac{\lambda a_{x,0}}{b_{x,0}} + \frac{\lambda a_{y,0}}{b_{y,0}} + \frac{\lambda a_{z,0}}{b_{z,0}} \right)$$

and update the impedance filter g^{n+1}. Since the two kernels are dependent on each other, the kernels must be executed sequentially.

For the third implementation, we will sort the grid points into seven classes corresponding to each of the family types. We will then launch seven individual kernels – one for each of the family point types. Since the kernels are independent of each other, the kernels can be processed in parallel. Kernel calls are asynchronous which means that control is immediately returned to the host after a kernel launch, but kernels are not computed in parallel unless explicitly stated. For GPU's of compute capability ≥ 2, kernels can be launched in parallel by using a stream id as a third parameter to the kernel. Inside each class we will again sort the points with regard to their types for minimising branch divergence.

Below, the three implementations are outlined:

```
// version 1
int main()
{
  ...
  while ( n < numIterations ) {
    h_srcInfo = getSourceInfo(inputSignalSrc);
    h_rceCoords = getReceiverCoord(inputSignalRce);

    cudaMemcpyAsync(d_srcInfo, h_srcInfo, sizeSrc,
                    cudaMemcpyHostToDevice);
    cudaMemcpyAsync(d_rceCoords, h_rceCoords, sizeRce,
                    cudaMemcpyHostToDevice);

    for (nGPU = n; nGPU < n + GPUchunk; nGPU++) {
      // main kernel
      solve<<<dimGrid1, dimBlock1>>>(allGridPoints, p0, p1)

      // kernel updating sources and receivers
      updateSrcRce<<<dimGrid2, dimBlock2>>>
                          (sourceInfo, receiverCoord, p0)

      // update pointers for next time step
      float* tmp_pointer = p1;
      p1 = p0;
      p0 = tmp_pointer;
    }
    n = nGPU;

    cudaMemcpyAsync(h_srcInfo, d_srcInfo, sizeSrc,
                    cudaMemcpyDeviceToHost);
    cudaMemcpyAsync(h_rceCoords, d_rceCoords, sizeRce,
                    cudaMemcpyDeviceToHost);
    ...
  }
}

// version 2
int main()
{
  ...
  while ( n < numIterations ) {
    ... // same as version 1
    for (nGPU = n; nGPU < n + GPUchunk; nGPU++) {
      // main kernel
      solve<<<dimGrid1, dimBlock1>>>(allGridPoints, p0, p1, p2)

      // DIF kernel
```

```
        updateDIFs<<<dimGrid2, dimBlock2>>>(boundGridPoints, p0, p2)

        // kernel updating sources and receivers
        updateSrcRce<<<dimGrid3, dimBlock3>>>
                                (sourceInfo, receiverCoord, p0)

        // update pointers for next time step
        float* tmp_pointer = p2;
        p2 = p1;
        p1 = p0;
        p0 = tmp_pointer;
      }
      n = nGPU;
      ... // same as version 1
    }
}

// version 3
int main()
{
  ...
  while ( n < numIterations ) {
    ... // same as version 1
    for (nGPU = n; nGPU < n + GPUchunk; nGPU++) {
      // kernels for each family type
      solve_inner<<<dimGrid1, dimBlock1, 0>>>(innerGridPoints, p0, p1)
      solve_oCorner<<<dimGrid2, dimBlock2, 1>>>(oCornerGridPoints, p0, p1)
      ...
      // kernel updating sources and receivers
      updateSrcRce<<<dimGrid3, dimBlock3>>>
                                (sourceInfo, receiverCoord, p0)
      // update pointers for next time step
      float* tmp_pointer = p1;
      p1 = p0;
      p0 = tmp_pointer;
    }
    n = nGPU;
    ... // same as version 1
  }
}
```

Common for all versions is the way source and receiver information are retrieved and copied to and from the GPU and the construction of having an inner and outer loop, explicitly shown in version 1. Since we are concerned about building a real-time system, we need to be able to change source and receiver positions during the simulation and therefore, we can not simply fix the scene parameters

in the beginning of the simulation and copy the data computed by the GPU when all time steps have been computed. Copying data to and from the GPU for every time step might be time consuming depending on the amount of data being transferred (in case of many moving sources and receivers). Therefore, computing several time-steps on the GPU without host interacting has been implemented (customised by the variable nGPU in the source code). The functions getSourceInfo and getReceiverCoord updates the location of the source and receiver and fetches the source pressures corresponding to the number of GPU iterations. The function cudaMemcpyAsync is a variant of cudaMemcpy which perform memory asynchronous memory transactions, returning the control to the host immediately. For the versions where the DIFs are updated internally, we can use just two time variables $p^{n+1}_{i,j,k}$ and $p^{n}_{i,j,k}$ for each grid point at time $n+1$, since $p^{n-1}_{i,j,k}$ will not be used by other points for the given time step and can therefore be overwritten with the new pressure value. Two pressure arrays corresponding to p^{n+1} and p^{n} are used, and when the next time step is computed, we simply swap the pointers pointing at these arrays. For version 2, we need three pressure values for each grid point, since updating the DIFs require both p^{n+1} and p^{n-1} and because these are updated externally, we can not overwrite p^{n-1} in the main kernel. For version 3, we have seven kernels updating each of the family grid points. Each has assigned an individual stream index (0-6) indicating that the kernels are independent. Notice that no explicit synchronisation is necessary for neither of the versions, since the kernels will not run in parallel unless explicitly stated and updating the pointers will not change the execution in the kernels, since the pointer address is copied by value by the kernels.

Further, variations of the three versions have been made:

- Moving the source and receiver update into the main kernel.
- Moving filter coefficients and scheme constants (such as λ) in constant memory.
- Making individual versions for frequency independent update schemes, such that the impedance is directly saved in a grid point variable member, and not by referencing an array structure.
- Individual versions of each of the seven schemes (SLF, IWB, etc) have been made, since not all schemes need to fetch all 26 neighbours. This results in fewer memory fetches from memory.

5.3.2 Memory allocation

As already mentioned, CUDA provides functions for memory allocation and memory copy. When allocating structures consisting of dynamic allocated data members, the complete structure has to be created on the host with the member pointers pointing to their corresponding memory addresses in global memory on the device. For the GPU solver, all grid points are initially copied from host to device. Copying the memory for each grid point individually turned out to be painfully slow when considering thousands of grid points, due to the performance of cudaMalloc and cudaMemcpy. The solution was unusable in practice, and therefore another approach was taken where all memory on the GPU are allocated and copied in one big chunk for all grid points. For this task, a simple – but powerful – class MemoryAssembly has been constructed with supervision of Frank Wefers. The idea is to let the class keeping track of all of the host allocated memory with references to the data, together with the number of bytes allocated for each. When all data have been allocated on the host, the corresponding memory size is allocated on the device in one big chunk. All the data allocated on the host are copied to the device using the class and all pointers referencing the original data are swapped corresponding to the new memory addressed on the device (See Appendix 9.5 for more details).

5.3.3 Memory Coalescing – Reordering the Grid

One of the most important aspect is how the CUDA kernel is accessing the data in global memory. The global memory is relatively slow compared to the other types of memory available, and can cause the performance to drop if many memory fetches are done. We have used a linear array for keeping the pressure data of the 3-D grid, enforcing consecutive memory allocation. The pressures are indexed corresponding to the id of the associated grid point. The FERMI architecture provides a Level 1 (L1) cache for each multiprocessor and a Level 2 (L2) cache shared by all multiprocessors, where the L1 cache can be configured to 48 KB, and the L2 cache is 768 KB. Because reads are merged into several chunks of consecutive memory in order to efficiently use the memory bus, it is important for the memory to be locally ordered, otherwise the surrounding data will not used and both latency and memory bandwidth are wasted. A local memory arrangement is depicted in Figure 5.4, where the boxes are indicating global memory indexed by the grid point index. Thread t0 is updating grid point 0 and assuming that the 26 neighbours are located at position 1 to 26, we will have a perfectly optimal memory locality. By reordering the grid points in such a way that locality is exploited, the cache hit is optimised due to less unnecessary data fetched from global memory into the cache. In three dimensions it is not

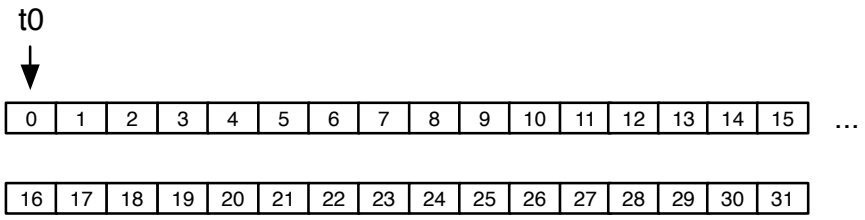


Figure 5.4: Optimal memory locality. Thread t0 is updating grid point 0. If we assume that the 26 neighbours are located at position 1 to 26, we have perfect optimal memory locality, which will ensure many cache hits.

possible to order the points, such that all neighbour points are located perfectly local for all points, but we can order the point in such a way, that some points are local to each other. For reordering arbitrary geometries, it is beneficial with a mapping from coordinate to grid point index/id. For this purpose, a lookup table is implemented as a linear array storing the ids of the grid points located in a separate array. The indexes for the linear lookup table array is ordered in row-major order with indexes corresponding to the coordinates of the cube enclosing the scene (i.e. grid points) as shown in Figure 5.5 in 2-D. For example, to find the id of the grid point with coordinate (3,0) in the 2-D grid in Figure 5.5, we get

$$\text{index} = i + X \cdot j = 3 + 9 \cdot 0 = 3$$

whereas the index for the grid point with coordinate (3,1) is

$$\text{index} = 3 + 9 \cdot 1 = 12$$

When ghost points are invoked, index 0 will be returned by the lookup table, indicating that a ghost point is reached. By accessing the grid points by coordinate makes it possible to reorder the grid points to enforce spatial locality.

Two reordering schemes have been implemented, namely a slices reordering and a cube reordering. Reordering the grid points in slices results in locality in two out of three dimensions. A graphical representation is depicted in Figure 5.6. The translation from normalised coordinates into row-major order in 3-D is given by

$$\text{index} = i + X \cdot (j + Y \cdot k) \tag{5.3}$$

where X and Y are the dimensionality in the x and y-direction, and i, j and k are the coordinate index in the x, y and z-direction.

The second reordering scheme is an attempt to ensure locality in all three dimensions by ordering the points in smaller cubes. This would ensure that points

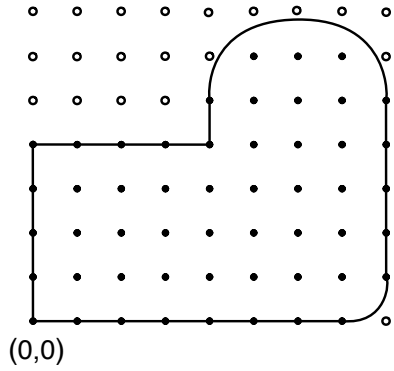


Figure 5.5: Scene enclosed by an outer rectangle in 2-D.

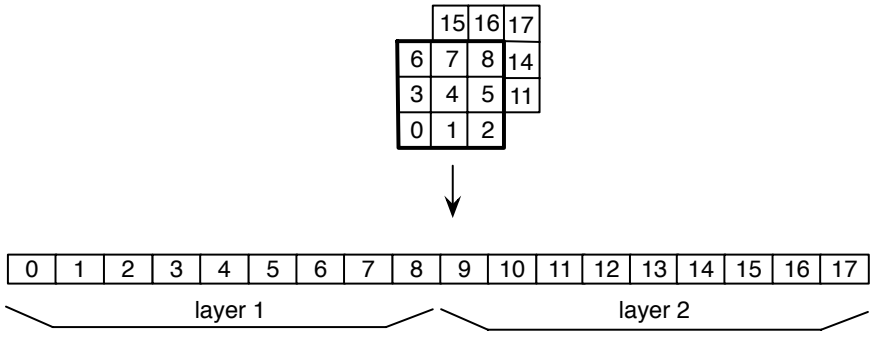


Figure 5.6: 3-D data arrangement.

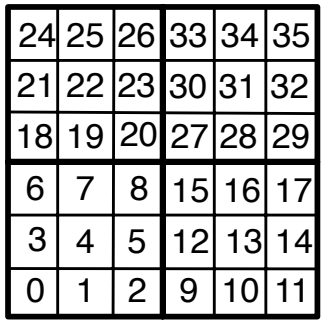


Figure 5.7: 2-D data arrangement in rectangles.

inside a cube would enforce locality, but comes with the drawback that neighbour points across a boundary will be located further away from each other. The cubic reordering scheme for the 2-D case is depicted in Figure 5.7 for a rectangle divided into four partitions. Inside each partition, the row-major order in Eq. (5.3) is used to order the grid points.

Chapter 6

System Overview

In this section, the overall system will be explained and a few details about some of the core functions will be given. In Appendix 9.4, an UML diagram showing the class interactions can be found.

6.1 Overview

In Figure 6.1, the modules of the overall system are depicted. The system consists of three core modules: GridCreator, Solver and Viewer. These three modules are individual components and can be executed separately. As the names indicates, GridCreator is responsible for importing and discretising a RAVEN scene in a format that can be used by Solver. The Solver module is where the FDTD method has been implemented and solves the wave equation for the discretised geometry with DIF filters corresponding to the materials applied on the boundaries. The module Viewer imports a scene meshed by GridCreator and visualise the meshed geometry in 3-D. The GridCreator creates a file containing information about the meshed scene. Each file entry represents a grid point with its id, boundary type, material type, coordinate and id of the 26 neighbours. The viewer has been implemented using VTK[1] and makes it possible to inspect the

[1]The Visualization Toolkit (`http://www.vtk.org/`)

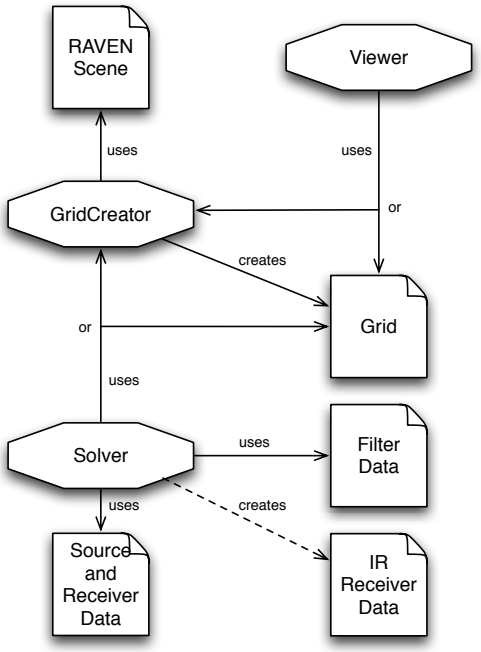


Figure 6.1: Modules of the system.

scene in 3-D. An example of visualising the Eurogress concert hall is shown in Figure 7.20.

6.2 Modelling grid and grid points

The grid keeps the overall grid information (scene dimensions, grid resolution, etc.), and incorporates functionality for reordering the grid points, adding and removing points, among others. The grid points are kept in a linear array, and can be accessed by either id or coordinate. Each grid point keeps various data, such as the ids to the 26 neighbours, structures for the values g, x and y used for modelling the impedance boundary points (from Chapter 4) and the boundary point type.

6.3 Solving

The solving part is done using the method Solve. The scheme parameters d_1, d_2, d_3, d_4 and λ are initialised with values corresponding to the scheme type, and sources and receiver are created with individual positions. Internally, a map<**int**, CSoundSource> data structure is used to keep track of the sources using an unique integer id as key and a class instance of CSoundSource including information about source positions and type. Accordingly, a map<**int**, CReceiverDesc> data structure is used to keep track of the receivers using a unique integer id as key and a class instance of CReceiverDesc keeping information about the position of each receiver.

6.4 Classifying the points

Classifying the grid points is a tedious task, since in principle 2^{26} neighbour point combinations are possible after the scene meshing process has been done. As we saw in Chapter 4, the boundary model uses only 94 boundary types in total, but to be able to correct a meshed scene incorporating point types not allowed in the framework, all 2^{26} neighbour combinations need to be recognised. This is not a trivial task and has not been investigated further, instead we assume that the CAD scenes are constructed such that points supported by the framework will only be discovered by the grid creation process. Assuming such a scene, the classification task becomes much more manageable, although still quite cumbersome.

For classifying a specific point, we must detect which neighbour points are present and also which points are not present in directions characterising the given point. It becomes more clear with an example: Let us classify an outmost right outer edge depicted in Fig. 6.2. By assuming that only the 71^2 different points supported by the framework are detected by the meshing algorithm, we do not need to check all 26 neighbour points, but in this case only the following neighbours coordinates have to be investigated:

$$(i+1, j, k)$$
$$(i, j, k+1)$$
$$(i, j-1, k)$$
$$(i, j+1, k)$$

[2]Pyramid boundaries have not been implemented, yielding 70 boundaries (+1 inner point type).

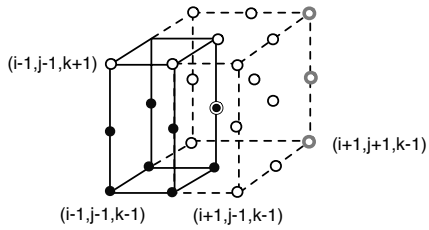


Figure 6.2: Outmost right outer edge. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles, with edge ghost points depicted with grey surroundings. The point that is being updated is indicated with a black-coloured circle with an surrounding circle.

If neighbours are present at coordinate $(i+1, j, k)$ and $(i, j, k+1)$, and no neighbours are present at coordinate $(i, j-1, k)$ and $(i, j+1, k)$, we classify the point as an outmost right outer z-edge. Following a similar approach for all 71 point types, we can classify all point types.

Chapter 7

Experiments and Results

The aim for this section is to test the FDTD method with respect to physical correctness and speed. We will first investigate the physical correctness of the FDTD method described in Chapter 4 by comparing the FDTD results with the approved FEM results. Next, the performance of FDTD solver implemented using CUDA on the GPU is measured in different geometries under various constraints with the goal of determine whether real-time simulations can be done for frequencies below the Schroeder frequency.

Finally, we will investigate an issue with numerical instability arising when simulating sound fields in complex geometries.

7.1 Physical Correctness of the Simulated Sound Field

For validating the correctness of the FDTD simulation, we will compare the FDTD method with the approved FEM method. The software tool Virtual Lab[1] has been used to compute the room transfer function using the FEM method.

[1] `http://www.lmsintl.com/virtuallab`

For each experiment, one source and two distinct receivers have been simulated, using a band-limited broadband impulse spanning the frequencies of interest. The transfer function contains information about the geometry of the room, the material property of the walls where the sound waves have been reflected, the medium in which the sound waves are travelling in and the position and direction of the sound source(s). In contrary to FEM methods, the FDTD method is a time-domain method and therefore the transfer function is not given directly, but instead the (time-domain) impulse response can be computed containing the same information as the transfer function. To obtain the impulse response using the FDTD method, the excitation signal is injected at a given position in the grid corresponding to the real-world position that is to be simulated. The injected pressure will propagate through the grid updated by the discretised wave equation and will be reflected and absorbed by the boundaries modelled by the digital impedance filters (DIFs). By simply recording the pressure at a given position, the impulse response is obtained directly. For comparing the results with the FEM solution, the impulse response is de-convolved with the injected impulse and Fourier transformed into the frequency domain. The approach is very intuitive, since the way the simulation is performed resembles the real-world very closely.

The experiments in this section will be done in a rectangular room of size 7 m $\times$ 5 m $\times$ 3 m with source (src) and receiver (rce) positions as given in Figure 7.1. Up to 100 Hz is simulated using the Kaiser-windowed sinc function from Figure 3.2 as impulse to the system injected at the source position, and the signal is then recorded at the receiver positions. The focus will primarily be on the SLF, CCP and IWB schemes. The reason for choosing the latter two is their efficiency as we will see in Section 7.2, and since the SLF scheme is widely known in the literature, this scheme is include for comparison. Furthermore, it is claimed that the IWB scheme in general leads to better simulation results when the boundaries are parallel to the world coordinate system (Kowalczyk and van Walstijn, 2011). The simulation results for all schemes can be found in Appendix 9.7 for frequency independent-, Helmholtz resonator- and porous absorber walls. The experiments were conducted with a total of four receivers, but only two receivers are included in the results due to similar results w.r.t. the errors introduced. An overview of the experiments is given below:

	Coordinates (cm)		
	x	y	z
Scene dim.	7.00	3.00	5.00
Source pos.	5.00	2.50	3.50
Receiver pos. 1	3.77	0.90	1.95
Receiver pos. 2	5.20	2.20	1.10

Table 7.1: Dimension and positions for cubic room simulations.

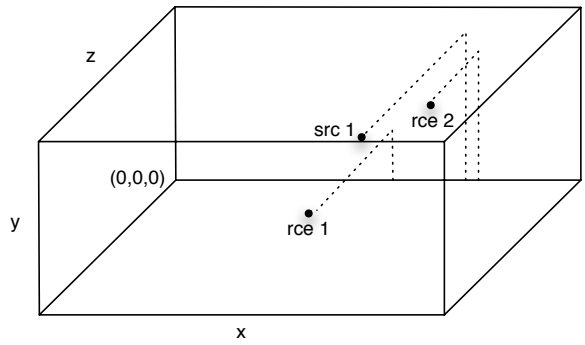


Figure 7.1: Source and receiver positions for the experiments in cubic rooms.

- 7 m×5 m×3 m cubic room simulations using all 7 schemes with frequency dependent- and independent boundaries.
- 7 m × 5 m × 3 m cubic room rotated 45 degrees around the y-axis using the SLF scheme with frequency independent boundaries.
- Impact on the results when changing the threshold of dispersion errors.
- Comparison of the result when using high and low wall absorption.
- Errors introduced due to meshing.

7.1.1 Experiments

Cubic room with frequency independent boundaries.

In this section we will investigate the physical correctness for the 7 m×5 m×3 m cubic room with frequency independent boundaries with special focus on the SLF, IWB and CCP schemes. The specific impedances (ξ) used are given in Table 7.2. We will compare the simulations performed by our FDTD method

Boundary	ξ
Walls	70
Ceiling	70
Floor	12

Table 7.2: Frequency independent impedances used for cubic room simulations.

and the FEM method with the exact same scene dimensions, source- and receiver position and material properties, and for not introducing additional errors in the comparison, we will choose the scene dimensions and source- and receiver positions such that they fit the spatial resolution for each of the schemes. In Table 7.1 the coordinates used as basis for all simulations are shown. In Table 7.3 the discretised coordinates are depicted for standard leapfrog scheme. The

Standard Leapfrog Scheme

	Discretised coord. (m)			Deviation from orig. (m)		
	x	y	z	x	y	z
Scene dim.	6.93	3.08	4.88	0.07	0.08	0.12
Source pos.	4.88	2.57	3.59	0.12	0.07	0.09
Receiver pos. 1	3.85	1.03	2.05	0.08	0.13	0.10
Receiver pos. 2	5.13	2.31	1.03	0.07	0.11	0.07

Table 7.3: Discretised scene, source and receiver coordinates discretised to fit the SLF scheme.

discretisation is straightforward and can be calculated by rounding to the nearest integer as

$$N_x = \left\lceil \frac{7}{\Delta x} - 0.5 \right\rceil \cdot \Delta x = 6.93 \tag{7.1}$$

$$N_y = \left\lceil \frac{3}{\Delta x} - 0.5 \right\rceil \cdot \Delta x = 3.08 \tag{7.2}$$

$$N_z = \left\lceil \frac{5}{\Delta x} - 0.5 \right\rceil \cdot \Delta x = 4.88 \tag{7.3}$$

where N_i is the scene dimension in the i'th dimension. The same procedure can be used for the source and receiver positions.

In Figure 7.2, the transfer function for the SLF, CCP and IWB schemes are depicted together with the error plot showing the differences between the two simulations. For the SLF scheme, a mean error of 0.3 dB is obtained for both receivers. The CCP and IWB schemes exhibit a very good fit below $\approx$ 70 Hz, however, above 80 Hz, errors up to 4 dB are introduced with mean errors of 0.3-0.6 dB for the CCP and 0.6 dB for the IWB scheme. For all schemes, the errors become more severe from around 90 Hz, due to fact that dispersion errors increases with frequency (cf. Figure 3.4). The error could also be caused by the impulse not spanning all frequencies, but experiments have shown that choosing an impulse with a cut-off frequency above 100 Hz did yield better results at higher frequencies. The differentiated Gaussian pulse from Figure 3.2 has also been used for comparison, with no observable differences in the transfer function.

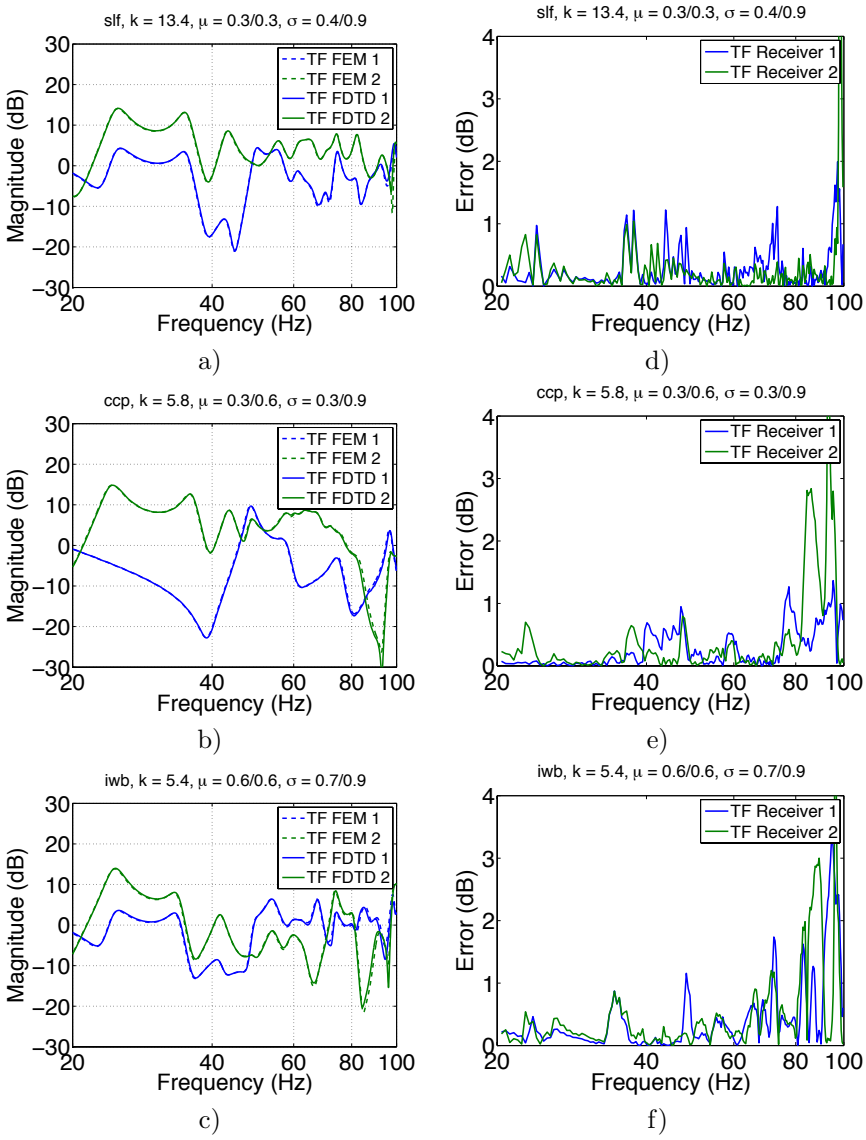


Figure 7.2: Transfer function (left) and error plot (right) for the FDTD simulation compared to FEM. Absorption coefficients from Table 7.2 are used in the Living Room scene.

We will now repeat the above experiments using twice as many sampling points per wavelength, which corresponds to a maximum of around 0.5 % of dispersion

errors being introduced. Choosing the resolution in that way, the scene size and source and receiver positions will still be a multiple of the grid resolution, making the comparisons easier. In Figure 7.3 d)-f), the results are shown. As expected, a better fit is achieved compared with the results in a)-c) for 2 % dispersion error, especially for the higher frequency. The mean errors are now roughly the same for all schemes with a mean value of 0.3 dB for the SLF and IWB scheme and 0.2 dB for the CCP. The std. deviations are also roughly the same being 0.4 dB for the SLF and CCP scheme and 0.3 dB for the IWB scheme.

We will now investigate the correctness when the spatial resolution is chosen such that 10 % of dispersion errors are allowed. In Figure 7.4, the result for the SLF and IWB schemes are depicted. The errors introduced above ≈ 60 Hz are now quite severe leading to mean value errors around 1 dB and even a mean value error of 2.1 dB is observed for receiver 2 for the IWB scheme, which may be unacceptable when physical correctness is the goal.

Finally, in Figure 7.5 the results are compared for high and low wall absorption. In a) a high absorptions of $\xi = 12$ is used for all boundaries, and in b) a low absorption of $\xi = 70$ is used. From these results it is clear that the amount of wall absorption has a quite big impact on the simulation results. For low absorption, the mean error is around 0.9 dB whereas for high absorptions, the mean error is 0.3. Because more overlapping modes will be present for higher absorption, less sharp peaks are present for high absorption compared to low absorption, leading to less errors in the simulation.

Cubic room with frequency dependent boundaries

We will now investigate the results obtained for frequency dependent wall absorption. As explained in Chapter 4, frequency dependent impedance boundaries are modelled using digital impedance filters (DIFs) formulated in Eq. (4.5). Instead of fitting the impedance data directly in terms of IIR filter coefficients, we will fit the reflectance data and afterwards convert the filter coefficients to impedance filter coefficients. Analog to the impedance filter in Eq. 4.5, we can formulate the reflectance filter by

$$R(z) \quad = \quad \frac{Z}{P} = \frac{b_0 + B(z)}{a_0 + A(z)} \tag{7.4}$$

Having obtained the coefficients a_i and b_i from the reflectance data (will be explained in moment) and ensuring that that the reflectance filter represents a passive boundary $|R_0(z)| \leq 1$, the corresponding coefficients for the impedance

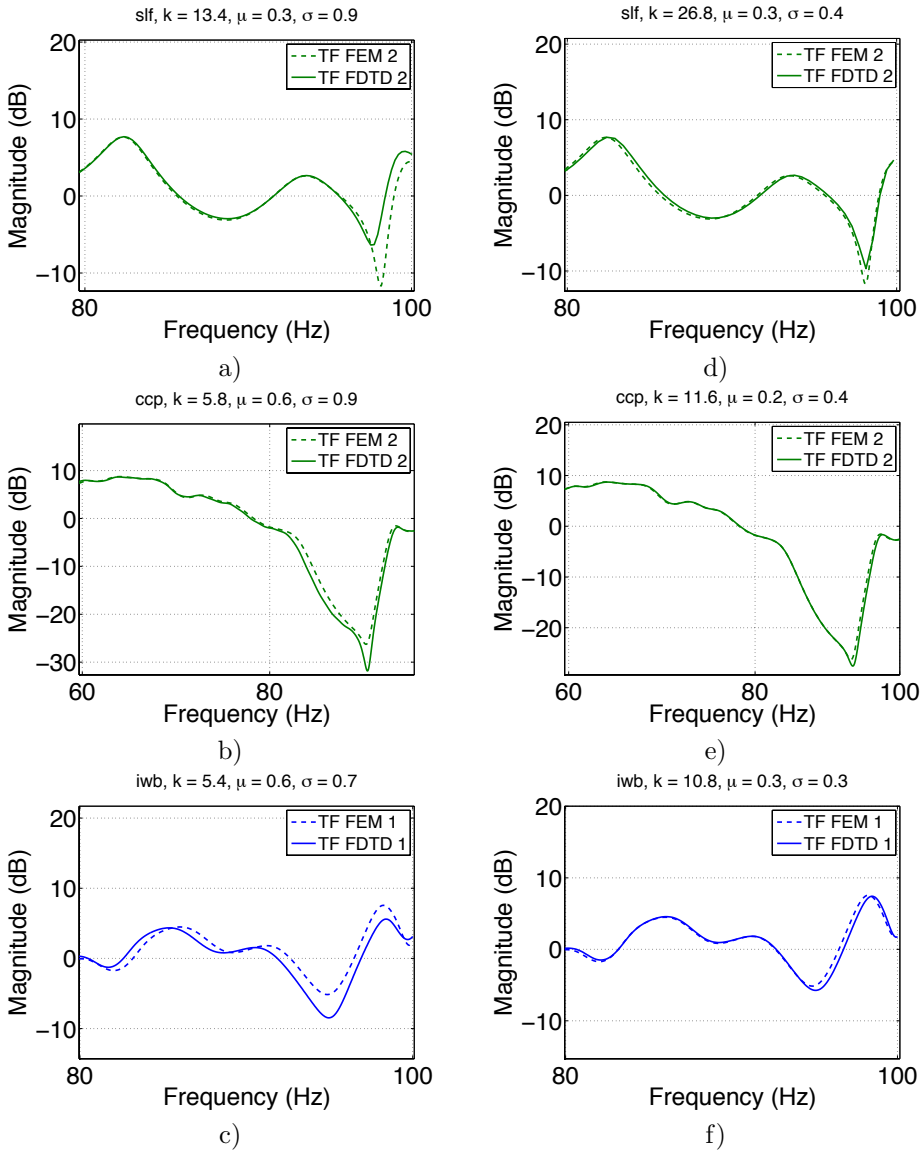


Figure 7.3: Impact of the simulation correctness when using more samples per wavelength. a)-c) are replicated from Figure 7.2 for comparison. d)-e) are the transfer functions for the SLF, CCP and IWB, respectively, corresponding to a maximum of 0.5 % of dispersion errors.

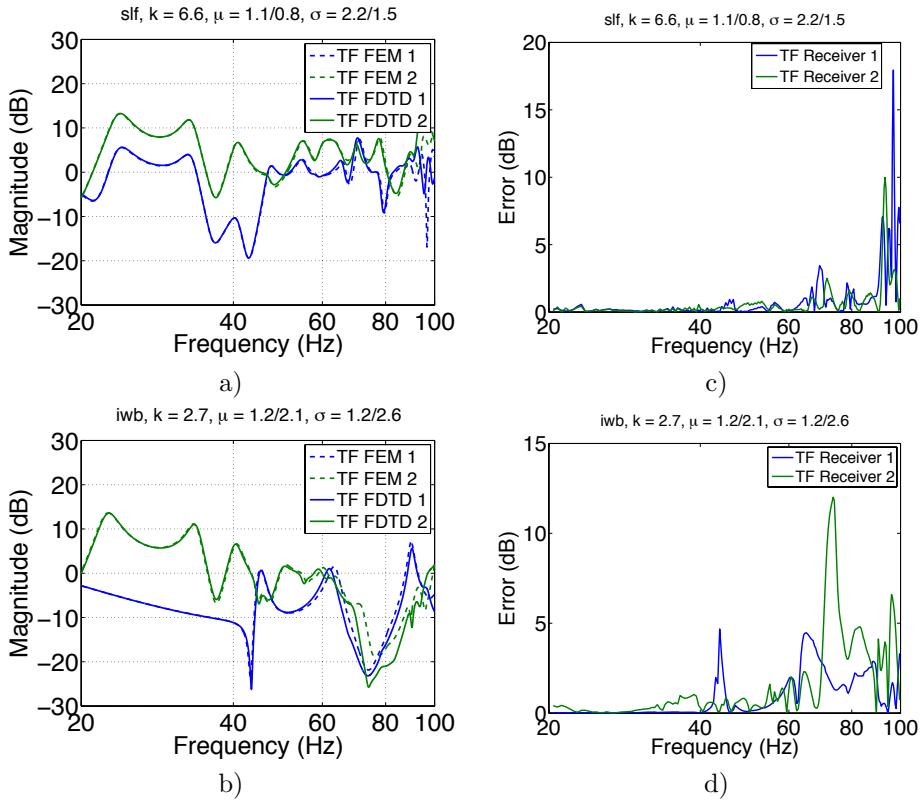


Figure 7.4: Simulation allowing for a maximum of 10 % of dispersion errors. a) Transfer functions using SLF, b) Transfer functions using IWB, c) Error plot for the SLF simulation, d) Error plot for the IWB simulation.

filter can be computed using Eq. (2.9) by

$$\xi_w(z) = \frac{1+\frac{z}{p}}{1-\frac{z}{p}} = \frac{P+Z}{P-Z} = \frac{b'_0 + b'_1 z + \ldots + b'_N z}{a'_0 + a'_0 + \ldots + a'_N}$$

where $b'_i = a_i + b_i$ and $a'_i = a_i - b_i$. We have used the Matlab function invfreqz for calculating the discrete-time transfer function corresponding to the complex-valued reflectance data, resulting in the a_i and b_i coefficients.

A Helmholtz resonator and a porous absorber in front of an air gap will be used in the following experiments. The data used contains complex valued reflectance coefficients in the range from 0 Hz up to the frequency of interest (i.e. 100 Hz). When computing the reflectance filter from the reflectance data, the z-domain

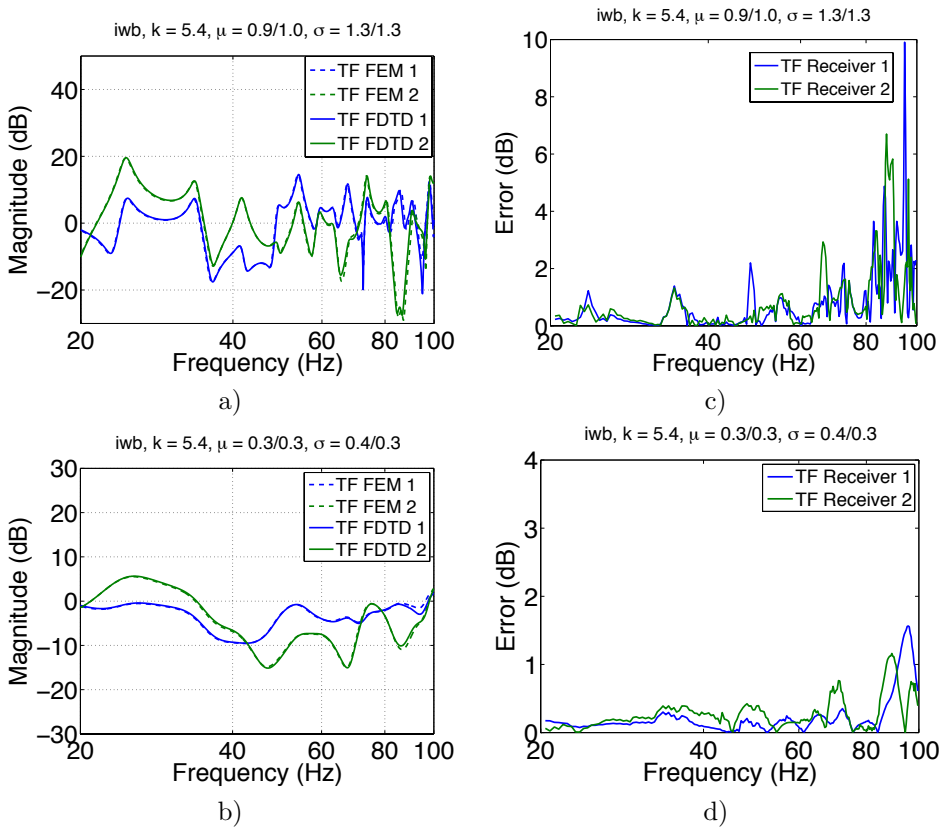


Figure 7.5: Comparison for high an and low wall absorption using the IWB scheme with 2 % dispersion errors. The impedances used are a) $\xi = 70$ and b) $\xi = 12$. The corresponding error plots are given in c) and d).

computation will stretch the filter from 0 up to the Nyquist range. Therefore, the data must be extrapolated individually for each scheme, such that the region of interest (ROI) from 20 to 100 Hz is properly scaled. Converting complex-valued frequency data into stable IIR filters coefficient can be difficult, since the reflectance data may not yield a stable filter. Therefore the reflectance data was ensured to be minimum-phase[2] before fitting the filter coefficients. The reflectance and impedance boundary data were approximated with a 5th order IIR filter, and the result is depicted in Figure 7.6 in the ROI. In Figure 7.7, the pole-zero plot of a) the Helmholtz resonator and b) the porous absorber are depicted for one of the schemes. All poles are inside the unit circle, but

[2]A minimum-phase filter is minimum-phase if the system and it's inverse are casual and stable. The Matlab function ita_minimum_phase from the ITA toolbox has been used.

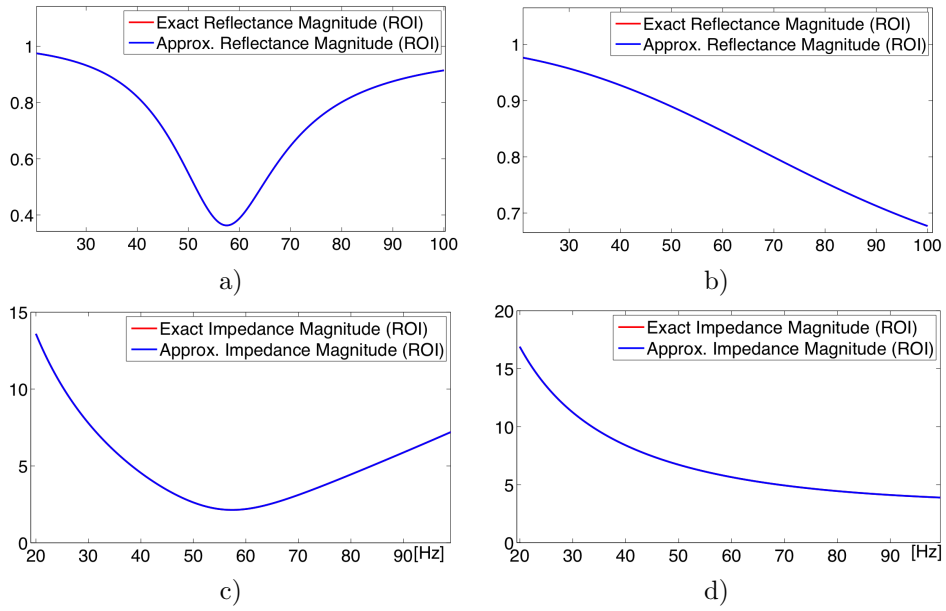


Figure 7.6: Fitting impedance IIR filters of 5th order. a) Reflectance for the Helmholtz resonator, b) Reflectance for a porous absorber, c) Characteristic impedance for the Helmholtz resonator, d) Characteristic impedance for a porous absorber.

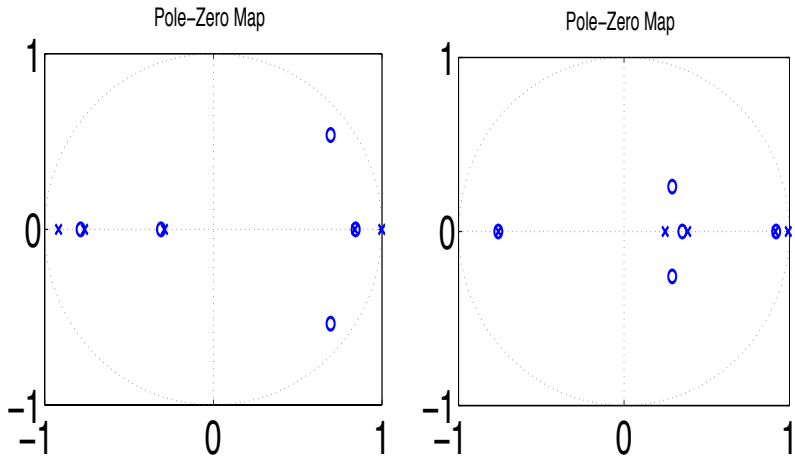


Figure 7.7: Pole-Zero plot for the impedance DIF filter, where the x-axis denoted the real part and the y-axis denotes the imaginary part. a) the Helmholtz resonator, and b) the porous absorber.

we see that one pole is very near the boundary. It has been observed that the system tends to be unstable when poles are near the boundary, caused by numerical round-off errors. In such cases, the location of the pole needs to be moved further away from the boundary. The location of the poles can be found by computing the roots of the polynomial in the denominator of Eq. (7.5). If the pole p is near the unit circle, we subtract a value such that $|p| < 0.99$, and recreate the polynomial form of the poles, resulting in a stable filter.

The simulation results using the Helmholtz resonator wall for the SLF, CCP and IWB schemes are depicted in Figure 7.8. A good correspondence between the FEM and FDTD solution is obtained, with a mean error around 0.2 dB for all schemes. For receiver 2, the mean error remains the same for the SLF, whereas a mean error of 0.5 dB for the CCP and IWB scheme is obtained. We see again, that the biggest errors occur in the high frequency range, except for the SLF.

The last experiments for frequency dependent absorption concerns the simulations with porous absorber walls for the SLF, CCP and IWB schemes depicted in Figure 7.9. For this experiment, the CCP performs best for receiver 1 with a mean error of 0.2 dB and a std. dev. of 0.2 dB. The reason that the SLF is performing slightly worse relative to the two other schemes is probably because of the sharp peaks introduced in the transfer function only for the SLF simulation. For receiver 2, the SLF is again performing best with a mean error of

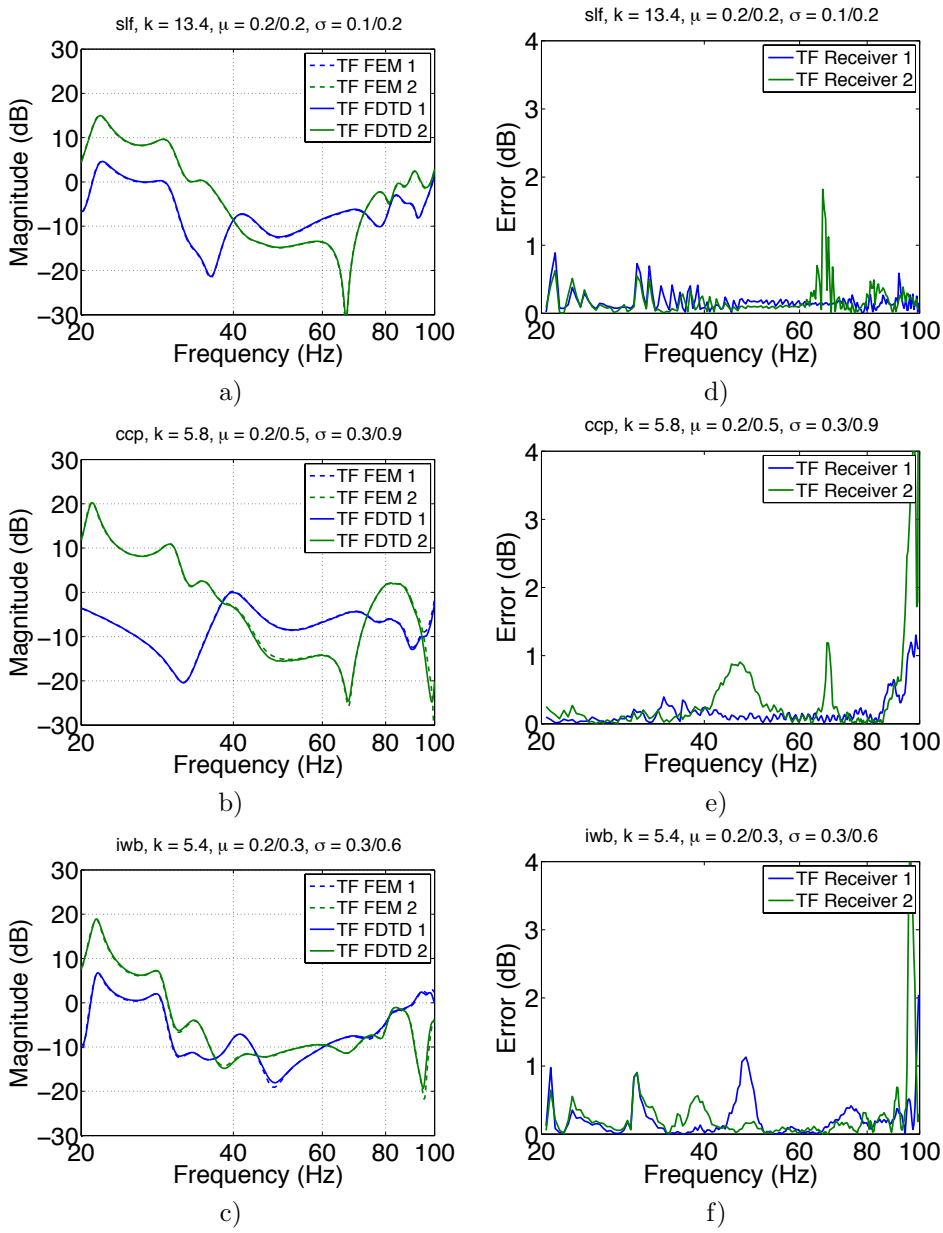


Figure 7.8: Transfer function (left) and error plot (right) for the FDTD simulation compared to FEM. The Helmholtz resonator from Figure 7.6 is used at the boundaries in the Living Room scene.

0.1 dB and a std. dev. of 0.2, which must be considered as a very good fit. In contrary to previous experiments, none of the schemes introduces more errors in the higher frequencies. If we look at the transfer functions, we see that the curves for all schemes are very flat for these high frequencies, which explains the good fit for these frequencies.

Cubic Room rotated by 45 degrees

This experiment will investigate the method's ability to simulate the sound field when having a great amount of oblique boundaries in the scene. We will again use the $7 \times 3 \times 5$ cubic room with the source and receiver positions given in Table 7.1, but instead of meshing the room oriented with the walls parallel to the world coordinate system, the room is rotated 45° around the y-axis. From a physical point of view, the simulated sound field should remain the same, assuming that the source and receivers are located at the same relative position in the room. This experiment will therefore clarify how the method copes with oblique boundaries modelled by inner and outer edges and corners. Figure 7.10 shows the meshing difficulties arising for the rotated cube. Three different discretisations are shown: a) a cubic room parallel to the coordinate system, b) a cubic room rotated 45° around the y-axis, choosing one of the corners as starting point for the meshing and c) a cubic room rotated 45° around the y-axis, choosing a starting point $\Delta x/2$ away from one of the corners for the meshing. In b) we notice the same situation as for the point b in Section 4.4, namely that the corner points have no volume. For minimising the discretisation error, we will construct the geometry such that the spatial resolution fits the staircase discretisation as optimal as possible by choosing a starting point with a distance of $\Delta x/2$ in each directions from one of the corners, resulting in the maximum number of points inside the geometry. We will only consider the SLF scheme for this experiments because of the problems explained in Section 7.3. The details about how to discretise the scene and compute the new positions for source and receivers can be found in Appendix 9.3.

The results for 2% of dispersion errors can be seen in Figure 7.11a. A slight offset in the whole frequency range is observed, which does not occur for the corresponding experiment for the room with edges parallel to the world coordinate system (Figure 7.2). This is reflected by mean errors of 0.4 and 0.6 dB for the two receivers for the rotated cube, compared to 0.3 dB for the non-rotated cube. This is caused by the staircase approximation for the boundaries, for which outer edges are used at the impedance boundaries for the rotated cube, whereas plane impedance boundaries are used in the non-rotated cube. The outer edge boundaries compensate, so to say, for all the inner edges with no impedance boundaries. Even though the results are slightly worse than for the

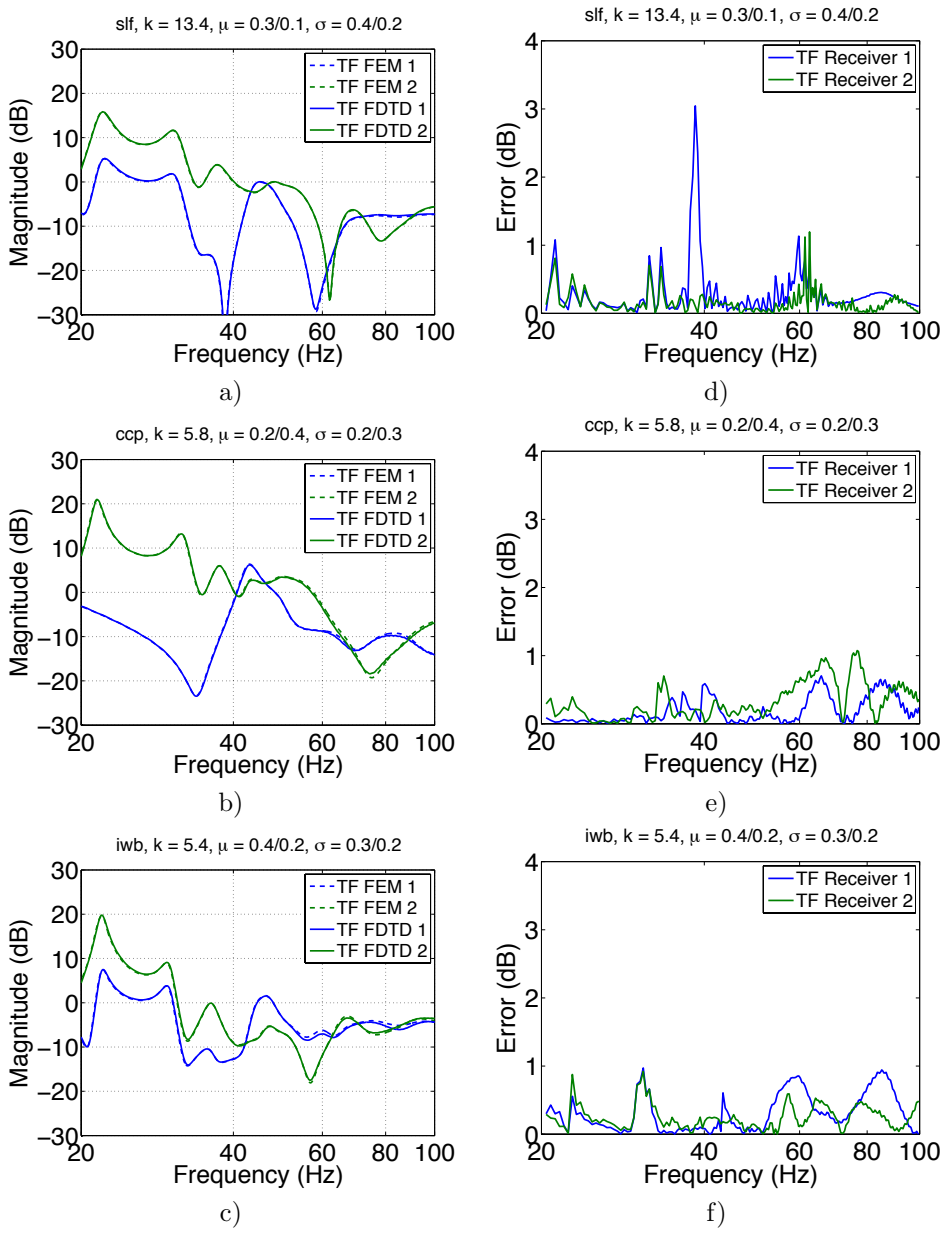


Figure 7.9: Transfer function (left) and error plot (right) for the FDTD simulation compared to FEM. The porous absorber from Figure 7.6 is used at the boundaries in the Living Room scene.

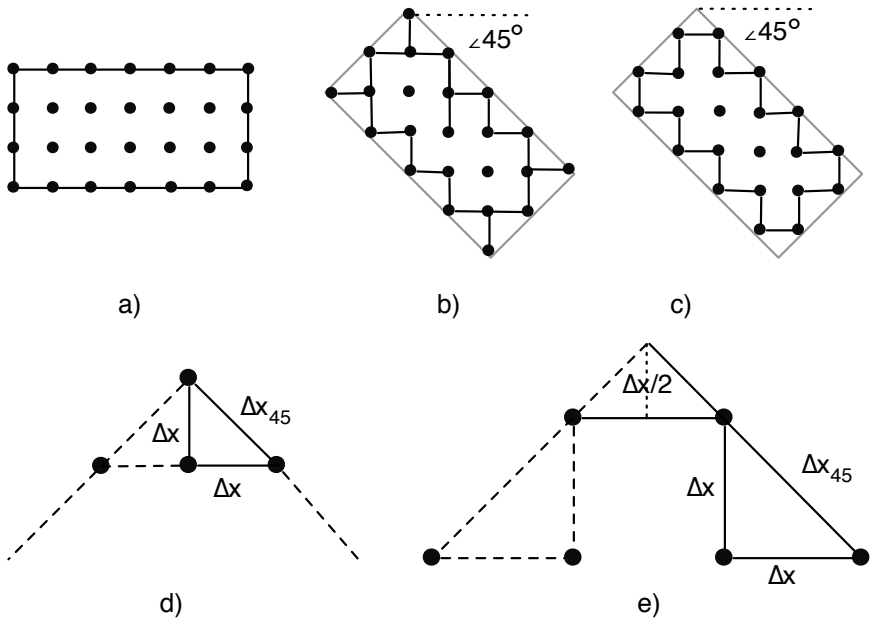


Figure 7.10: Meshing a cubic room. a) Room constructed parallel to the coordinate system, b) Room rotated 45° and meshed by using one of the corners as start point, and c) Room rotated 45° and meshed using a start point with an offset of $\Delta x/2$ in each of the dimensions relative to one of the corners.

non-rotated cube, the results are still reasonable. In b) twice as many sampling points per wavelength are used with the goal of refining the staircase approximations at the boundaries in the hope of getting a more precise discretisation of the scene. Surprisingly, the curve offset remains the same.

Errors introduced due to meshing

We have already observed that the start point choice when meshing the scene can lead to different meshes of the scene. When no restrictions are imposed on the scene size, shape or source and receiver positions, we can construct our scene such that all dimensions are multiples of the spatial resolution. However, in arbitrary geometries, it might be problematic to change the geometries such that they fit the spatial resolution, since it would lead to a model that does not reflect the real-world environment that we want to simulate. We will therefore

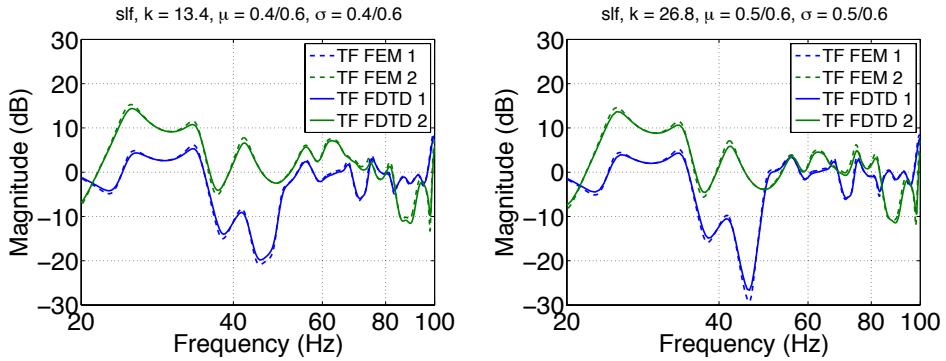
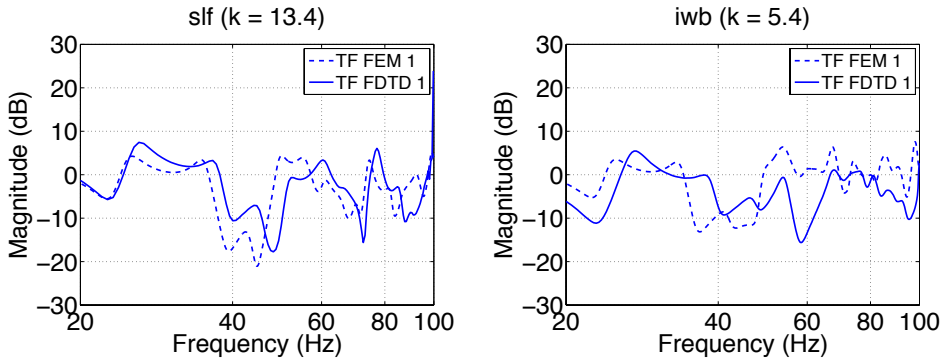


Figure 7.11: The Living Room rotated 45° using the SLF scheme. a) $k = 13.4$, b) $k = 26.8$.

	Start point (cm)			Scene dimension (cm)		
Scheme	x	y	z	x	y	z
SLF	0	0	0	6.93	3.08	4.88
IWB	0	0	0	7.01	1.19	5.10
SLF	0.13	0.13	0.13	6.67	2.82	4.62
IWB	0.32	0.32	0.32	6.37	2.55	4.46

Table 7.4: Impact on the discretisation error in a cube, when choosing a start point not being a multiple of the spatial resolution.

investigate the impact on the simulation results when choosing a start-point $(\Delta x/2, \Delta x/2, \Delta x/2)$ for a scene constructed with dimensions being multiples of Δx. We will perform the experiment for the IWB and SLF schemes, because of their different resolutions. The results are shown in Figure 7.12 for a) the SLF scheme and b) the IWB scheme. For both schemes, the precision has been degraded significantly compared to the results in Figure 7.2. The IWB performs the worst due to the coarser resolution, therefore missing a grid point resulting in a room shrinked by 0.6 meters in each dimension. For the SLF scheme, the room is only shrinked by 0.26 meters in each dimension, depicted in Table 7.4. This experiment clearly shows, that the spatial resolution not only has an impact on the dispersion error, but in practical situations also highly influences the errors introduces in the meshing process, resulting in a shift in the resonance frequencies.



a) b)

Figure 7.12: Impact on the discretisation error in a cube, when choosing a start point not being a multiple of the spatial resolution using frequency independent impedances given in Table 7.2 a) SLF, and b) IWB.

A Comparison Between the Results Obtained by all Schemes

We will briefly comment on the overall results obtained for all the schemes. In general, all schemes are performing well for both frequency dependent and -independent frequency boundaries. The SLF scheme is performing best with mean errors between 0.2 dB and 0.3 dB, whereas the IDWM scheme is performing worst with mean errors between 0.4 and 0.7 dB. For all schemes, most errors are introduces in the upper frequency range, except for the IDWM scheme, where most errors are introduced in the frequency range between 20 and 40 Hz. The reason for this behaviour remains unknown.

7.1.2 Conclusion

For frequency independent wall absorption with a maximum of 2 % dispersion errors, mean errors between 0.3 and 0.6 dB and std, deviations between 0.4 and 0.9 were obtained, with the SLF method performing the best. For frequency dependent wall absorption, mean errors between 0.2 and 0.5 dB and std. deviations between 0.1 and 0.9 were obtained, again with the SLF method performing the best. In most of the simulations, quite severe errors up to around 4 dB were introduced in the frequency band between 90 and 100 Hz which may be audible. Doubling the number of samples per wavelength for frequency independent boundaries resulted in mean errors between 0.3 and 0.4 dB and std. deviations between 0.3 and 0.4, with no notable differences between the schemes. Allowing

up to 10 % dispersion errors degrades the performance significantly, resulting in mean errors between 0.8 dB and 2.1 dB. An error of up to 0.5 dB may be acceptable, which is respected by allowing less than 2 % of dispersion errors. To determine the exact threshold of dispersion errors, a listener tests would need to be done.

In general, all schemes are performing well, with the SLF scheme performing best (less than 0.3 dB mean errors) and the IDWM performing worst (less than 0.7 dB mean errors).

It has been observed that the absorption level has an impact on the precision with mean errors of 0.3 dB for high absorption ($\xi = 12$), whereas mean errors around 1 dB for low absorption ($\xi = 70$) were obtained. This is caused by fewer overlapping modes for low absorption, leading to sharper frequency peaks.

Choosing scene dimensions not being a multiple of the spatial resolution has a dramatic impact on the simulation results. When concerned about the lower frequencies sound field – in this case up to 100 Hz – the spatial resolution is quite big resulting in discretisation errors up to more than half a meter for the SLF and around 0.3 meters for the IWB and CPP schemes for 2 % dispersion errors, yielding unacceptable results.

Finally, the method's ability to model the sound field at oblique boundaries was investigated by rotating a cubic room from the previous experiments 45° around the y-axis. The results still showed good correspondence between the FEM and FDTD simulations, though with mean errors between 0.4 dB and 0.6 dB. Even though the results are slightly worse than the experiments for the non-rotated cube with mean errors of 0.3 dB, it is encouraging that the impedance filters implemented for pronounced use of staircase approximations still yield reasonable results. It can also be assumed that most real-world geometries will have many walls parallel to the world coordinate system, leading to small areas where staircase approximations are applied.

7.2 GPU Solver

We will compare the different GPU implementations from Section 5 in terms of performance and point out the most performant implementation. The following implementations will be compared:

- Grid points computed in a single kernel/multiple kernels.

- Source and receiver computed in main kernel/separate kernel.
- DIFs computed in main kernel/separate kernel.
- Parameters in constant memory/global memory.

Having decided on which implementation performs the best, we will next consider the following:

- Scheme performance comparison.
- Boundaries and its impact on performance.
- Maximum frequencies for real-time simulation capabilities.

Finally, we will investigate the bottlenecks of the final implementation using the NVidia Profiler.

7.2.1 Test setup

The aim for the performance test is to investigate whether the simulation of the sound field in a given geometry under specific constraints (boundary type, errors allowed) can be computed in real-time. We will use a real-time factor measured by dividing the duration of the simulated sound-field with the actual running time spend on solving by

$$P_{\text{real-time}} = \frac{N \times \Delta t}{t_{\text{sim}}} \tag{7.5}$$

where $P_{\text{real-time}}$ is the real-time factor, N is the number of time steps, Δt is the temporal resolution and t_{sim} is the measured running time of the simulation. We will use $N = 2000$ for all simulations divided into chunks of 200 GPU time-steps[3].

We will test the implementation in six different geometries given in Table 7.5. The Cube 1, Cube 2 and Cube 3 scenes are used in the initial experiments to investigate which of the implementations from Section 5 performs the best, and also for comparing the individual schemes. The Car Compartment and the Living Room scene are approximations to real-world scenarios, for which

[3]Buffers of 6-30 time-steps must be used for a latency below 10 ms, but should not have any significant impact on the results, since the time spend on copying the data for the source and receiver to and from the GPU should be negligible.

we will investigate if the real-time constraint is met for frequencies below the Schroeder frequency. Finally, we will do a performance test in the Eurogress concert hall, but since the Schroeder frequency is on the limit of the human hearing threshold, we will instead find the upper frequency for which real-time simulations can be done. We will use the following denotations for the three

Space	Length (m)	Width (m)	Height (m)	Volume (m^3)	Schroeder freq. (Hz)
Car Compartment	1.5	1.15	1.5	2.81	450
Eurogress				14,000	21
Living Room	7	5	3	105	120
Cube 1	50	50	50	125,000	-
Cube 2	5	158	158	124,820	-
Cube 3	25	25	25	15,625	-

Table 7.5: The dimensions for four geometries used in this study.

GPU implementations:

A Update and DIF formula for all points computed in seven kernels, one for each family type.

B Update formula for all points computed in one kernel, DIFs updated in external kernel.

C Update and DIF formula for all points computed in one kernel.

D CPU version.

All experiments have been performed using a NVidia GTX 580 graphics card on a dual Core 2 2.16 Ghz processor running Windows 7. Microsoft Visual Studio 2010 was used with CUDA 4.1 generating 32-bit code.

7.2.2 Initial experiments

The first experiment will clarify how the three implementations A, B, C and D performs relatively. Constant memory have been used for the scheme constants d_1, d_2, d_3, d_4, and λ, but did not show a notable speedup. Experiments using asynchronous memory copy from host to device and asynchronous kernel launches for the seven kernel family grid point did not show any significant speed-up either. All three versions have been customised for handling frequency

Point family	7.4 % boundary nodes			26.2 % boundary nodes		
	N	N_{type}	Relative	N	N_{type}	Relative
Inner	456,533	456,533	0.926	327,089	327,089	0.7380
Outer corner	8	1	0.0000	8	1	0.0000
Outer edge	924	77	0.0019	1996	≈ 166	0.0040
Plane	35,574	5,929	0.0722	127,440	$\approx 21,240$	0.2620
Total	493,039	-	1	494,016	-	1

Table 7.6: Distribution of points in Cube 1 (7.4% boundary nodes) and Cube 2 (26.2% boundary nodes).

independent absorbing boundaries, implying that the filter impedance is directly written to a grid point variable instead of pointing to filter coefficients in constant memory.

Branch divergence. In Table 7.6 the distribution of the points for Cube 1 and Cube 2 are shown. For Cube 1, 92% of the point are inner points having no DIF update and since all points in this class uses the same update scheme, no branch divergence occur. For outer corners and outer edges, branch divergences occur, but since these points only contributes with 0.2% of the total amount of points, this is negligible. The plane boundaries contributes with $35,574$ point, corresponding to 7.2% of the grid points. Most of the warps will exhibit no branch divergence, only every $\lfloor 5929/32 \rfloor = 185$'s warp might have a single branch divergence. There seems to be no way to eliminate these branch divergences, and having sorted the grid points after type, this is the best we can do.

Implementation comparison. In Figure 7.13, the performance for the three implementations A, B and C are compared using Cube 1 and 2 with 7.4 % and 26.4 % of boundary nodes, respectively. Both slices reordering and cube reordering were used, but no differences were observed. In the experiments corresponding to A, B and C, the slice reordering was used, whereas for experiment C* the grid points have been randomly reordered in memory. In C*, we notice a remarkable performance drop by a factor of 3.6 confirming that it is crucial to enforce coalesced memory. We see no significant difference between the different implementations, and surprisingly no performance differences using Cube 1 and Cube 2 were observed using frequency independent boundaries. This might indicate, that the bottleneck of the system is not the computational load for each thread, but instead the time spend on memory transactions. A speed-up factor of 21 is achieved for the GPU implementation compared to the CPU

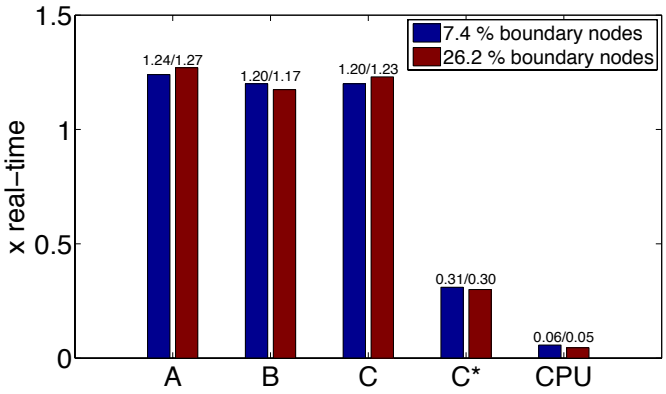


Figure 7.13: Performance comparison between the three GPU versions A, B and C is depicted along with the CPU version. Up to 100 Hz was simulated using the IWB scheme with $k = 5.4$ (2 %) in the Cube 1 and Cube 2.

implementation.

Filter order influence on performance. The performance influenced by using DIFs of order 0 to 10 is depicted in Figure 7.14 using Cube 1 and 2. The performance for implementation A and C are depicted respectively in a) and b). First, we notice that using filters of order 0 gives roughly the same results for A and C, and that the performance is not degraded compared to the customised implementation in Fig. 7.13 with filter coefficients in constant memory. The performance drops linearly to around 1 x real-time for 7.4 % boundaries and to around 0.8 x real-time for 26.2 % boundaries for both implementations using filters of order 10, corresponding to a performance factor drop of 1.2 and 1.5, respectively.

Since the previous experiments **showed no significant difference** between implementation A, B and C, implementation C will be used for the rest of the experiments.

Scheme comparison. We will now compare the performance of all the seven schemes in the Cube 3 geometry. Because the schemes uses different number of neighbour points (e.g. the IWB uses 26 neighbours, whereas SLF only uses 6), we have made individual versions for each scheme, such that only the necessary neighbour points are fetched from global memory. In Figure 7.15, the IWB, CCP

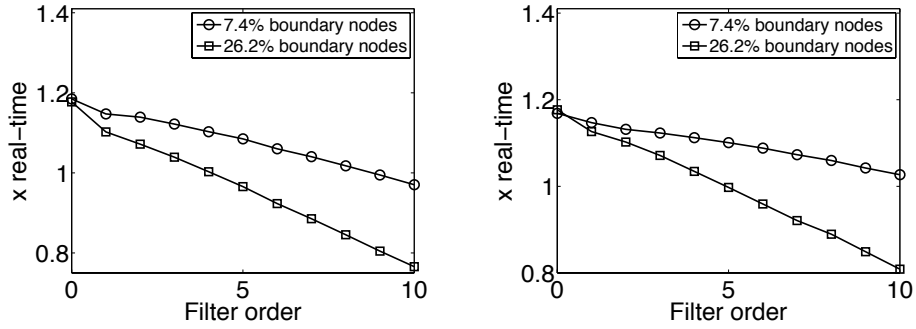


Figure 7.14: Graph showing the computation time using filters of order 0-10. a) One kernel for each family point type, b) One main kernel for all point types.

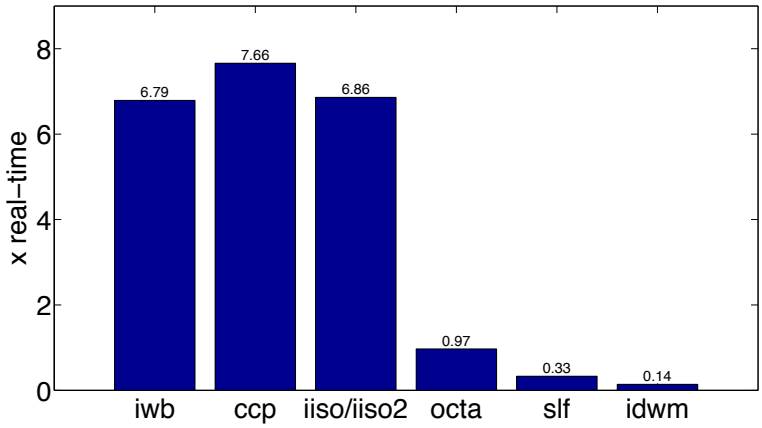


Figure 7.15: Performance comparisons for customised versions of implementation C for each of the seven schemes using Cube 3. A slightly smaller scene has been used for the IDWM scheme due to memory usage.

Scheme	Number of points
IWB	60,841
CCP	74,089
IISO/IISO2	74,089
OCTA	480,636
SLF	922,082
IDWM	907,726 (1,168,651)

Table 7.7: Number of scene point for each of the schemes used for comparison in Figure 7.15

and IISO schemes are performing more that 7 times faster than the OCTA, SLF and IDWM. This has two reason. First, the number of points shown in Table 7.7 greatly differs between the two groups due to the spatial resolution. This is the main reason for the performance differences between the schemes and apparently the fewer memory fetches and computations for the second group do not compensate for the greater amount of grid points. However, we see that the CCP and IISO schemes perform as good or better than the IWB despite the need of 21% more grid points. The reason is exactly, that fewer memory fetches are needed, namely 8 and 12 neighbour pressure value fetches for each point compared to 26 neighbour pressure values for the IWB scheme. The second reason for the performance difference is because of the Courant condition devising the sampling frequency:

$$f_{\mathrm{s}} = \frac{k}{\lambda} \cdot f_{\max} \tag{7.6}$$

The Courant number for the CCP and IWB schemes is $\lambda = 1$ and for the IISO schemes $\lambda = \sqrt{3/4}$. For the second group we have $\lambda = \sqrt{1/3}$ for the SLF and IDWM. The result is higher sampling rates for the schemes with lower Courant number than for the schemes with higher Courant number, leading to additional workload to be done in the same period of processing time.

Impact on performance when using more sources and receivers. Another important aspect to consider is the performance impact when more sources and receivers are used. In Figure 7.16a, simulations consisting of 1 transparent source and 4 receivers has been used, whereas in Figure 7.16b, simulations consisting of 100 transparent source and 100 receivers are used. We see a performance drop from 1.11 to 1.04 x real-time, which can be considered negligible.

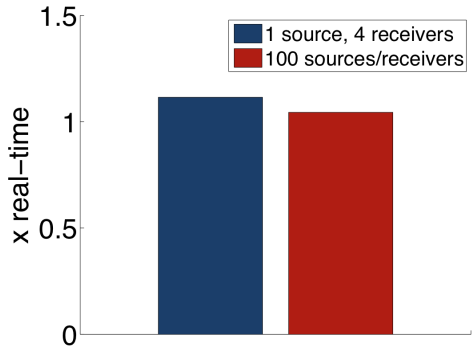


Figure 7.16: Performance comparison when using 1 transparent source and 4 receivers, and 100 transparent sources and 100 receivers.

	Level 1 cache hit (%)	Level 2 cache hit (%)
Cube 1 (coalesced)	97.6	47.8
Cube 1 (non-coalesced)	81.49	77.80

Table 7.8: Cache utilisation for coalesced and non-coalesced pressure value memory.

NSight Profiler. We have collected a few results about the GPU using NVidia's NSight Profiler. In Table 7.8 the cache hit for the Level 1 and Level 2 cache are depicted. For coalesced memory, a very good cache hit of the fast Level 1 cache of 97.6% is achieved. For the non-coalesced memory in experiment C*, only 81.49 % of the memory fetches has hit the Level 1 cache, resulting in more Level 2 cache hits. A cache hit of 97.6 % for the coalesced data may also explain why reordering in smaller cubes does not affect the performance, since more cache hits may be difficult to obtain. The L2 cache hits is given as the percentage of the data fetches not hitting in the L1 cache. These results indicate, that the processing time is not spend on memory fetches. In Figure 7.9, information about register usage, occupancy, serialisation, and branch efficiency are depicted.

As expected, we have a high branch efficiency due the ordering of points and relatively many inner points are present.

The usage of registers (and also shared memory) affects the occupancy, since only a limited number of registers can be used at a time. The multiprocessor occupancy is the ratio of active warps to the maximum number of warps supported on a multiprocessor of the GPU (NVidia, 2011a). In Figure 7.17, the

	Measure
Registers used per thread	44
Registers used per block	1408
Occupancy	16.51
GPU serialisation	0.88
Branch efficiency	1

Table 7.9: Performance measures obtained using the NVidia Nsight profiler.

occupancy can be seen as a function of threads per block with the number of registers used per block is fixed at 44 registers. In practice, it turned out that the maximum performance was reached by using 32 threads per blocks, with no performance gain acheived by using more threads per block. By using 44 registers ad 32 threads per block, the low occupancy is caused by a hardware limit of a maximum of eight concurrent blocks and not (directly) because of the number of registers used. For implementation A and B, less registers per thread are used, but also for these implementations, the performance dropped when using more than 32 threads per block. The reason that not performance gain is reached is probably because no improvement in accupancy is reached, due to the argument just given above.

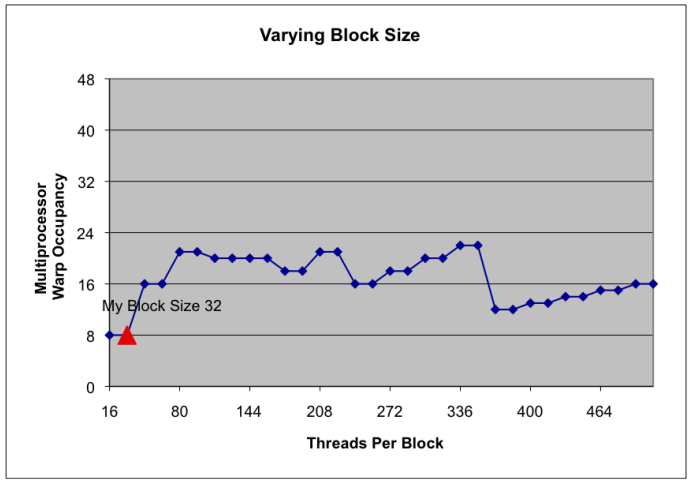


Figure 7.17: Graph showing the GPU occupancy by changing the number of threads per block while fixing the number of registers per block to 44 registers. The graph can be obtained by using the NVidia Profiler.

Though, a problem using many registers is, that data fetches from the same register can only be done sequentially. Therefore, if many threads are fetching data

from the same register – known as bank conflicts – less parallelisation can be achieved. This might be the reason for the high GPU serialisation. We believe, that a big speedup should be possible by lowering the serialisation. Since many neighbour points are shared by many points (actually 27 points for an inner point), we may lower the serialisation by letting a single thread compute more than one grid point and making use of shared memory for these common neighbour points. We may not achieve better performance w.r.t. memory latency, since we already have a very high L1 cache hit, but lowering the serialisation may increase the performance.

7.2.3 Car compartment and Living Room

Since the goal of this thesis is to investigate whether it is possible to model the sound field below the Schroeder frequency, we will now turn our focus to geometries for which the Schroeder frequency is above the limit of the human hearing. The Schroeder frequency of a car compartment is difficult to determine because of the combination of very absorbing materials (like seats), and very reflecting materials (like the windshield), leading to standing waves only for the reflecting surfaces. The Mercedes S-Class has been used for sound field simulations at ITA, where the Schroeder frequency has been determined to be in the range between 300 and 500 Hz. An exact car compartment model for the before-mentioned S-class was available, but unfortunately the meshing issues mentioned in Section 4.4 for the point b made it impossible to simulate in this exact geometry. Instead, a cubic geometry with the same volume has been used, which should not yield performance results too far from the real model, since – as shown in previous experiments – the area of the boundary surface does not have any significant impact on the performance when considering frequency independent boundaries. The geometry used in the following experiments is the Car Compartment given in Table 7.5 using a Schroeder frequency of 450 Hz. In Figure 7.18 and 7.19, the GPU and CPU versions of the SLF, CCP and IWB schemes are compared for both 0.5% and 2% dispersion errors. In Figure 7.18, frequency independent filters are used, whereas 4th order filters are used in Figure 7.18. The first, and most important thing to notice, is that the GPU implementation of the CCP and IWB schemes were able to simulate the lower sound field below the Schroeder frequency in real-time for 4th order DIFs with less than 0.5% dispersion errors. The SLF method can also perform in real-time, but only for a maximum of 2% of dispersion errors using 0th order DIFs. An interesting observation is that the CPU implementation can also be used for real-time simulations below the Schroeder frequency, if the CCP or IWB schemes are used for a maximum of 2% dispersion errors. However, if a maximum of 0.5% of dispersion errors are needed, only the GPU implementations of the CCP and IWB scheme are capable of performing in real-time.

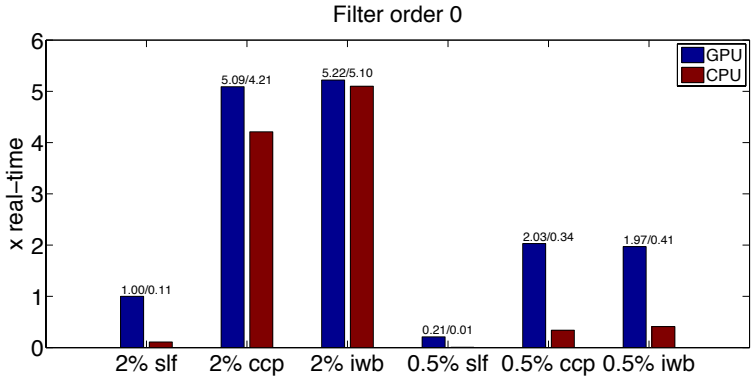


Figure 7.18: Graph showing the computation time using frequency independent filters in the Car Compartment.

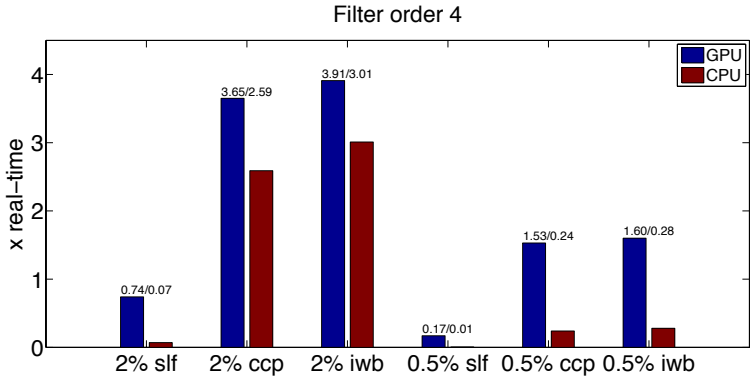


Figure 7.19: Graph showing the computation time using 4th order impedance filters for the Car Compartment.

In Table 7.10, the upper frequency limit for real-time performance for both GPU and CPU implementations of the the three schemes is depicted using frequency independent filters and maximum 2 % of dispersion errors. The highest frequency is obtained using the GPU CCP implementation, yielding 1,300 Hz with 2% of dispersion errors. The IWB follows closely with $1,250$ Hz, both capable of simulating 3 times as many frequencies as the SLF method.

The results for the Living Room scene is depicted in Table 7.12: the SLF scheme simulated up to 221 Hz, the IWB simulated up to 549 Hz and the CCP simulated up to 575 Hz, all well above the Schroeder frequency of 120 Hz. Both the

Scheme	Δx (cm)	N	f_s (Hz)	f_l (Hz)	rel.
SLF GPU	0.057	14,581	6,030	450	1
SLF CPU	0.117	1,690	2,948	220	1
CCP GPU	0.046	27,225	7,540	1,300	2.88
CCP CPU	0.100	2,700	3,451	595	1.32
IWB GPU	0.051	20,700	6,750	1,250	2.78
IWB CPU	0.103	2,700	3,348	620	1.38

Table 7.10: Maximum frequency range restricting real-time simulation performance using the IWB, CCP and SLF schemes in the Car Compartment. Both the CPU and GPU implementation have been tested allowing a maximum of 2% dispersion errors.

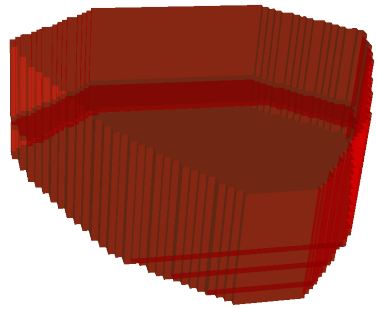


Figure 7.20: The Eurogress concert hall in Aachen visualised with the viewer.

CCP and IWB scheme were able of performing in around 20 x real-time when simulating up to 120 Hz.

7.2.4 Eurogress

For this experiment, we will find the maximum frequency limit for real-time performance in the Eurogress concert hall in Aachen. The meshed geometry can be seen in Figure 7.20, created with our viewer. We will restrict the experiments to include the GPU version of the IWB, CCP and SLF scheme, and the results can be seen in Figure 7.11. The highest frequency is again obtained using the CCP scheme, making it possible to simulate up to 185 Hz using frequency independent filters. Using a 4th order filter, the highest frequency drops to 175

Scheme	Δx (cm)	N	f_s (Hz)	f_l (Hz)	rel.
SLF filter order 0	0.357	304,050	964.8	72	1
SLF filter order 4	0.372	260,384	924.6	69	1
CCP filter order 0	0.320	411,718	1073	185	2.57
CCP filter order 4	0.339	349,926	1015	175	2.54
IWB filter order 0	0.368	277,335	934.2	173	2.40
IWB filter order 4	0.374	256,538	918	170	2.46

Table 7.11: Simulation performance of the IWB, CCP and SLF schemes in the Eurogress concert hall in Aachen.

Hz. Compared to the SLF scheme, a factor of around 2.5 more frequencies can be simulated using the CCP and IWB schemes.

7.2.5 Conclusion

Three variants of the GPU Solver have been compared showing no significant performance differences. In a cubic room consisting of around 493,000 grid points, no performance difference was observed when using 7.4% and 24.2% boundary nodes for frequency independent DIFs. However, a performance degradation factor of 1.2 and 1.5 were observed for 7.4% and 24.2% boundaries using DIFs of order 10.

A comparison of the seven schemes showed a significant performance gain when using the CCP, IWB and IISO schemes instead of the OCTA, SLF and IDWM schemes. For the Cube 3 test scene, the four first-mentioned schemes achieved real-time factors between 6.70 and 7.66, with the CCP scheme as the most efficient, whereas the latter three schemes achieved real-time factors between 0.14 and 0.97, with the IDWM being the least efficient.

It was shown, that the number of source and receiver has a negligible impact on the performance.

Real-time performance was possible for the SLF, IWB and CCP GPU implementations at or well above the Schroeder frequency for the Car compartment, Living Room and the Eurogress scene summarised in Table 7.12 allowing for a maximum of 2% of dispersion error with frequency-indepdent boundaries. Allowing a maximum of 0.5% of dispersion errors in the Car Compartment, only the CCP and IWB GPU implementations were able to perform in real-time using DIFs of order 4. If a maximum of 2% of dispersion errors is required,

Scheme	Car 2.81 m^3	Living Room 105 m^3	Eurogress 14,000 m^3
SLF	450 Hz	221 Hz	72 Hz
CCP	1,300 Hz	575 Hz	185 Hz
IWB	1,250 Hz	549 Hz	173 Hz

Table 7.12: Maximum simulation frequency restricting real-time simulations for the SLF, CCP and IWB scheme in the Car compartment, Living Room and the Eurogress scene allowing for a maximum of 2% of dispersion error with frequency-indepdent boundaries. The Schröeder frequency for the scenes are: Car Compartment 450 Hz, Living Room 120 Hz, Eurogress 21 Hz.

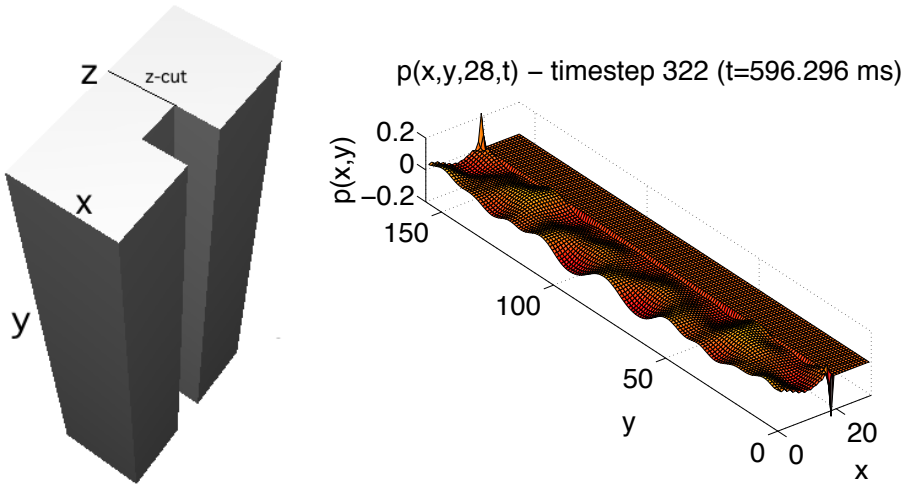
the CPU version was also able to perform in real-time. A maximum frequency limit of $f_{\max} = 1,300$ Hz was obtained using the CCP implementation for 2% of dispersion errors. Real-time performance was also possible for the Living Room scene of size $7 \times 5 \times 3$ for both 0.5 and 2% of dispersion errors.

In the Eurogress concert hall with a volume of $14,000$ m^3, the CPP implementation performed best with $f_{\max} = 185$ Hz for 0th order DIFs and $f_{\max} = 175$ Hz for 4th order DIFs, achieving a relative performance gain of around 2.5 compared to the SLF scheme.

7.3 Problems with the FDTD method for Complex Geometries

One of the main goals for this project is to investigate the physical correctness of the sound field modelled by the FDTD method in complex geometries. As we saw in Chapter 4, 8 families of boundary types with a total of 95 different update schemes are necessary for handling non-cubic scenes for the general family of 3-D non-staggered compact explicit schemes.

A problem with instability was observed when the update schemes for handling non-cubic geometry shapes were invoked, and many weeks have gone into this problem. For visualising the problem, a scene is constructed such that the wave propagation can be examined in 2-D. The outer cube dimensions are 20 m $\times$ 100 m $\times$ 40 m and is depicted in Fig. 7.21a. Two inner edges were created each including two inner edges-corners. In Fig. 7.21b, the pressure has been visualised in the x-y dimension along the z-dimension corresponding to the black line depicted in Fig. 7.21a. By choosing this z-cut, the wave propagation can be investigated along the edge including the inner edge-corners. From the figure,



a) b)

Figure 7.21: a) 20 m × 100 m × 40 m cube including two inner edges and two inner edge corners, b) Simulation along the z-axis corresponding to the black line from Fig. a).

we observe that no instabilities occurs along the edges, only at the two endpoints corresponding to edge-corners instabilities occurs. This gives us the reason to believe, that the problems in non-cubic environments are caused by problems in or near the inner edge-corners.

In the first stage, a detailed debugging session was performed with the goal of examining the logic of the program. The following was investigated:

Grid. It was confirmed that 1) all neighbours are neighbours mutually, and 2) the relative coordinate of a neighbour point is correct.

Grid point classification. After the grid is created, each point is classified into one of 71 grid point types. The correctness was tested in a small scene with four inner edges and no inner corners. By inspecting the geometry, the number of all individual points where counted by hand and compared with the number of times the update scheme for each point was invoked.

Update formula. Since the task of implementing 71 update formulas consisting of a big number of indexes is very error prone, the correctness of the update scheme was checked by choosing a simple scene, where only two

(out of 24) update formulas for inner edges corners were used and only one (out of 12) update formula for inner edges was used. By carefully comparing the implemented update scheme to the mathematic formulation, the implementation of the update scheme is considered correct.

From the logic tests, the author feels confident about the correctness of the implementation. The mathematical formulation of the update formula for the inner edge-corner might then be incorrect. In the work by Kowalczyk (Kowalczyk (2008a), Kowalczyk and van Walstijn (2011)), only outer corner, outer edges and outer planes are derived. The remaining update schemes for inner edges, inner corners and inner edge-corners were derived by the author of the thesis. The derivation of the problematic inner edge-corner has been derived in two different ways, 1) eliminating the two ghost points from the update formula for a plane, and 2) deriving the formula from the update formula directly for an inner point as done in Appendix 9.2. As expected, both approaches gave the same result. The procedure followed the same approach as done in (Kowalczyk, 2008a), so unless special cases occur for inner edge-corner, the derivation should be correct.

A mail correspondence with Kowalczyk (Appendix 9.6) supports the derived formula for inner edge and inner edge-corners. Another formula was proposed, which - for a lower-right inner-edge corner - is given as:

$$
\begin{aligned}
p_{i,j,k}^{n+1} \quad = \quad & \Big[d_1 (2p_{i-1,j,k}^n + p_{i,j+1,k}^n + p_{i,j-1,k}^n + p_{i,j,k+1}^n + p_{i,j,k-1}^n) \\
& + d_2 (2p_{i-1,j+1,k}^n + 2p_{i-1,j-1,k}^n + 2p_{i-1,j,k+1}^n + 2p_{i-1,j,k-1}^n \\
& + 2p_{i,j-1,k-1}^n + p_{i,j+1,k-1}^n + p_{i,j-1,k+1}^n) \\
& + d_3 (4p_{i-1,j-1,k-1}^n + 2p_{i-1,j-1,k+1}^n + 2p_{i-1,j+1,k-1}^n) \\
& + d_4 p_{i,j,k}^n + \frac{\lambda^2}{b_0} g^n + \left(\frac{\lambda a_0}{b_0} - 1 \right) p_{i,j,k}^{n-1} \Big] \Big/ \left(1 + \frac{\lambda a_0}{b_0} \right)
\end{aligned}
\tag{7.7}
$$

At the time of writing, implementation of the proposed formula was not possible.

Chapter 8

Conclusion and Further Work

A system capable of simulating the lower frequency sound field in real-time has been implemented using seven 3-D non-staggered compact explicit FDTD schemes incorporating frequency dependent boundaries realised as digital impedance filters consistent with locally reacting surfaces. 71 update formulas have been implemented for handling non-cubic geometries, out of which the update formulas for inner edges, inner corners and inner edge-corners have been derived by the author of the thesis.

Several experiments concerning the physical correctness of the implemented methods were done in cubic environments, where the transfer functions obtained at two different receiver positions computed by the FDTD method were compared with the transfer functions computed at equivalent source and receiver positions using the FEM method. The results for the two methods showed good correspondence for both frequency dependent and independent boundaries with the SLF scheme performing best (less than 0.3 dB mean errors) and the IDWM performing worst (less than 0.7 dB mean errors) for a maximum of 2% of dispersion errors. For a maximum of 0.5% of dispersion errors, mean errors between 0.2 dB and 0.3 dB were obtained for the SLF, CCP and IWB schemes. Allowing up to 10% of dispersion errors resulted in mean errors between 0.8 dB and 2.1 dB, which is considered as audible. It was also shown, that errors introduced due to discretisation errors of the scene can lead to severe errors in the modelled sound field.

The correctness of the modelled sound field when having a big amount of oblique boundaries were tested by rotating a cubic geometry 45° around the y-axis. The experiment showed good results, although more errors were introduced compared to the non-rotated cube.

It was shown that real-time simulation below the Schroeder frequency is possible using the CCP and IWB schemes for both frequency dependent and -independent boundaries. The results were obtained considering the performance in several cavities corresponding to a car compartment, a living room scene and three bigger geometries. For a 50 m × 50 m × 50 m room, up to 100 Hz can be simulated in real-time for frequency independent boundaries. For the Car Compartment, up to 1,300 Hz can be modelled, for the Living Room up to 575 Hz can med modelled and for the Eurogress up to 185 Hz can be modelled using frequency independent boundaries. The CCP scheme performed best in all cases.

It was shown that the number of sources and receivers did not influence the performance significantly.

A problem with instabilities for inner edge-corners was observed for all schemes, except for the SLF scheme, limiting simulations to cubic environments only. The problem has been investigated intensively, and no mismatch between the theory (Kowalczyk et al., 2011) and the implemented method is observed. Detailed debugging has been done, and no implementation errors were found. The SLF method used for the cube rotated 45° gave correct and stable results, supporting the claim that no problems with the implemented logic are present.

Further Work

The most important work to do for the future, is to investigate the instability problem in more depth by reconsidering whether the method used for eliminating the additional ghost points for inner edge-corners results in a stable formulation from both a theoretical and a practical point of view.

If arbitrary CAD scenes should be used with the system, a much more robust post-processing step needs to be implemented. Since 2^{26} point combinations are impossible to classify by brute-force, more sophisticated methods should be considered by for example exploiting rotations and mirroring of geometries.

The discretisation errors introduced due to meshing CAD scenes were shown to have a big impact on the modelled sound field. In the DWG literature,

fractional delay filters are being used for resolving these discretisation errors. To the authors knowledge, no such filters have been used in the FDTD literature, and may be applied for solving the discretisation problems.

Another topic is to evaluate the perceptual importance of the dispersion error by defining the maximum value of the dispersion error which can be considered perceptually insignificant.

Recently, (Raghuvanshi et al., 2009) have introduced the adaptive rectangular decomposition (ARD) approach to numerically solve the acoustic wave equation. The main benefit of ARD is, that much less dispersion errors are introduced. The result showed that a competitive accuracy with the FDTD reference method using 10 samples per wavelength was obtained, while consuming 12x less memory and computations. Perfectly Matched Layers (PML) (Katsibas and Antonopoulos (2002), Rickard et al. (2003)) is used for boundary modelling, and is an efficient method when perfect absorption is needed. Using PML for modelling boundaries occurring in the real-world may not yield physical correct results, therefore it would be interesting to implement DIFs into the ARD method.

Chapter 9

Appendix

9.1 Derivation of Transparent Sources for the General Family of Compact Explicit Schemes

We are interested in deriving the general formulation for the transparent source corresponding to Eq. (3.28) for the family of compact schemes. One can convince itself that for schemes where $d_4 = 0$, the same transparent source formulation (3.28) can be used, since the only change is the values of I, not the pattern. For $d_4 \neq 0$, the source node p_{src} at the previous time step is now used when updating the grid, which may change the formulation. By ignoring the exact polynomial form of the grid impulse at the moment, we can still take a 1-D approach on the problem (even though the schemes does not make sense in 1-D). We will therefore construct an update scheme in 1-D corresponding to the general compact explicit scheme

$$p^n_{\text{src}} = \lambda_1(p^{n-1}_{i_{\text{src}}+1} + p^{n-1}_{i_{\text{src}}-1}) + \lambda_2 p^{n-1}_{i_{\text{src}}} - p^{n-2}_{i_{\text{src}}} \tag{9.1}$$

The update scheme has only been constructed with the aim of examining the behaviour in the vicinity of the source position and is not useful for simulations. As mentioned before, the choice of d_1, d_2 and d_3 does not influence our transparent source formulation and therefore these are merged into λ_1. We follow the same approach as before, but now using the update scheme (9.1), yielding

the following grid impulse for an arbitrary grid driving function f

$$\begin{aligned} p^0_{\text{src}} &= 0 && (9.2)\\ p^1_{\text{src}} &= \lambda_2 f^0 && (9.3)\\ p^2_{\text{src}} &= (2\lambda_1^2 + \lambda_2^2 - 1)f^0 + \lambda_2 f^1 && (9.4)\\ p^2_{\text{src}} &= (2\lambda_1 + \lambda_2 - \lambda_2^2)f^0 + (2\lambda_1^2 + \lambda_2^2 - 1)f^1 + \lambda_2 f^1 && (9.5)\\ & && (9.6) \end{aligned}$$

The grid impulse response using the Kronecker delta function as driving function gives us

$$\begin{aligned} I^0 &= 0,\\ I^1 &= \lambda_2,\\ I^2 &= 2\lambda_1^2 + \lambda_2^2 - 1\\ I^3 &= 2\lambda_1 + \lambda_2 - \lambda_2^2 \end{aligned}$$

leading to a formulation of the pressure values at the source node expressed in terms of I by

$$\begin{aligned} p^0 &= f^0 && (9.7)\\ p^1 &= f^1 + I^1 p^0 && (9.8)\\ p^2 &= f^2 + I^1 p^1 + I^2 p^0 && (9.9)\\ p^3 &= f^3 + I^1 p^2 + I^2 p^1 + I^3 p^0 && (9.10)\\ & && (9.11) \end{aligned}$$

Interestingly, the transparent source for the general compact explicit schemes takes the same form as Eq. (3.28), only the grid impulse is changed to not include any zero values.

9.2 Detailed Derivation of Inner Edge-Corners

In this section we will derive the update formula for the left inner edge-corner depicted in Figure 9.1. As depicted in Figure 9.1, we have 11 ghost points to eliminate - 1 corner point, 2 edge points and 8 plane boundary points. We will start by eliminating the ghost points corresponding to the impedance boundaries in the axial, side-diagonal and diagonal directions as explained in Section 4.3 given by Eq. (4.38)-(4.40). Formulating the interpolated ghost points in terms of the points lying on the original rectangular grid yields Eq. (4.44) and (4.45).

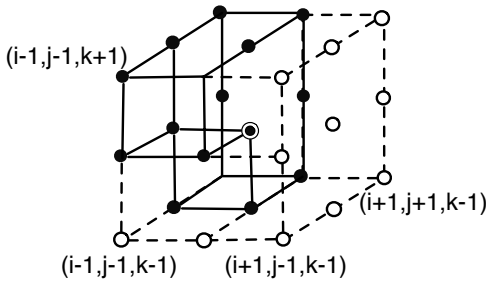


Figure 9.1: Lower y, z frontmost edge-corner with 11 ghost points. Room interior nodes are indicated with black-coloured circles, ghost nodes indicated by white-coloured circles. The point that is being updated is indicated with a black-coloured circle with an surrounding circle.

Isolating e.g $p^n_{i+1,j+1,k}$ in Eq. (4.39) gives us

$$\begin{aligned} p^n_{i+1,j+1,k} &= p^n_{i-1,j+1,k} + p^n_{i-1,j-1,k} + p^n_{i-1,j,k+1} + p^n_{i-1,j,k-1} - p^n_{i+1,j-1,k} \\ &\quad - p^n_{i+1,j,k+1} - p^n_{i+1,j,k-1} + 4\frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + 4\frac{g^n_{sd}}{b_0} \end{aligned} \tag{9.12}$$

and isolating e.g. $p^n_{i+1,j+1,k+1}$ in Eq. (4.40) gives us

$$\begin{aligned} p^n_{i+1,j+1,k+1} &= p^n_{i-1,j+1,k+1} + p^n_{i-1,j-1,k+1} + p^n_{i-1,j+1,k-1} + p^n_{i-1,j-1,k-1} - p^n_{i+1,j-1,k+1} \\ &\quad - p^n_{i+1,j+1,k-1} - p^n_{i+1,j-1,k-1} + 4\frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + 4\frac{g^n_d}{b_0} \end{aligned} \tag{9.13}$$

Inserting the above two equations together and the formulation for the axial ghost points into the compact scheme in Eq. (3.8) yields

$$\begin{aligned} p^{n+1}_{i,j,k} &= d_1(2p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1} \\ &\quad + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_a}{b_0}) \\ &\quad + d_2(2p^n_{i-1,j+1,k} + 2p^n_{i-1,j-1,k} + 2p^n_{i-1,j,k+1} + 2p^n_{i-1,j,k-1} \\ &\quad + p^n_{i,j+1,k+1} + p^n_{i,j+1,k-1} + p^n_{i,j-1,k+1} + p^n_{i,j-1,k-1} \\ &\quad + 4\frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + 4\frac{g^n_{sd}}{b_0}) \\ &\quad + d_3(2p^n_{i-1,j+1,k+1} + 2p^n_{i-1,j-1,k+1} + 2p^n_{i-1,j+1,k-1} + 2p^n_{i-1,j-1,k-1} \\ &\quad + 4\frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + 4\frac{g^n_d}{b_0}) \\ &\quad + d_4 p^n_{i,j,k} - p^{n-1}_{i,j,k} \end{aligned} \tag{9.14}$$

The above formulation corresponds to the formulation for a right plane boundary in Eq. (4.46). For the formulation of an inner edge-corner, the points $p_{i,j-1,k-1}$ and $p_{i,j-1,k-1}$ need to be eliminated, for which we can use the local coherence condition for a y,z boundary from Eq. (4.52)

$$\frac{\partial p^2}{\partial y \partial z} = 0 \tag{9.15}$$

disretised as

$$\begin{aligned} p^n_{i,j-1,k-1} &= p^n_{i,j+1,k-1} + p^n_{i,j-1,k+1} - p^n_{i,j+1,k+1} && (9.16)\\ p^n_{i-1,j-1,k-1} &= p^n_{i-1,j+1,k-1} + p^n_{i-1,j-1,k+1} - p^n_{i-1,j+1,k+1} && (9.17) \end{aligned}$$

Substituting these ghost points into Eq. (9.14) by the above equations yields

$$\begin{aligned} p^{n+1}_{i,j,k} = \; & d_1(2p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1} \\ & + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_a}{b_0}) \\ & + 2d_2(p^n_{i-1,j+1,k} + p^n_{i-1,j-1,k} + p^n_{i-1,j,k+1} + p^n_{i-1,j,k-1} \\ & + p^n_{i,j+1,k-1} + p^n_{i,j-1,k+1} \\ & + 2\frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + 2\frac{g^n_{sd}}{b_0}) \\ & + 4d_3(p^n_{i-1,j-1,k+1} + p^n_{i-1,j+1,k-1} \\ & + \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k}) + \frac{g^n_d}{b_0}) \\ & + d_4 p^n_{i,j,k} - p^{n-1}_{i,j,k} \end{aligned} \tag{9.18}$$

We have now eliminated all ghost points, and hence a formulation only consisting of points inside the geometry is obtained. Now, we only need to reorder the terms and merging the filters into on one impedance filter g^n as explained in Section 4, Eq. (4.48) ff.

Moving the terms containing the pressure difference term $(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k})$ to the

left side of the equality sign and reordering the terms yields

$$\begin{aligned}
& p^{n+1}_{i,j,k} - \frac{a_0}{\lambda b_0}(p^{n-1}_{i,j,k} - p^{n+1}_{i,j,k})(-d_1 - 4d_2 - 4d_3) \\
= \;& d_1(2p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1} + \frac{g^n_a}{b_0}) \\
& + d_2\Big(2p^n_{i-1,j+1,k} + 2p^n_{i-1,j-1,k} + 2p^n_{i-1,j,k+1} + 2p^n_{i-1,j,k-1} \\
& + 2p^n_{i,j+1,k-1} + 2p^n_{i,j-1,k+1} + 4\frac{g^n_{sd}}{b_0}\Big) \\
& + d_3(4p^n_{i-1,j-1,k+1} + 4p^n_{i-1,j+1,k-1} + 4\frac{g^n_d}{b_0}) \\
& + d_4 p^n_{i,j,k} - p^{n-1}_{i,j,k}
\end{aligned}$$

We have that $(-d_1 - 4d_2 - 4d_3) = -\lambda^2$, and moving $p^{n+1}_{i,j,k}$ outside the bracket, we get after some simple algebraic manipulations the final update scheme for a right outmost edge-corner in the y,z-direction

$$\begin{aligned}
p^{n+1}_{i,j,k} = \;& \Big[d_1(2p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1} \\
& + 2d_2(p^n_{i-1,j+1,k} + p^n_{i-1,j-1,k} + p^n_{i-1,j,k+1} + p^n_{i-1,j,k-1} \\
& + p^n_{i,j+1,k-1} + p^n_{i,j-1,k+1}) \\
& + 4d_3(p^n_{i-1,j-1,k+1} + p^n_{i-1,j+1,k-1}) \\
& + d_4 p^n_{i,j,k} + \frac{\lambda^2}{b_0} g^n + (\frac{\lambda a_0}{b_0} - 1)p^{n-1}_{i,j,k}\Big] \Big/ \left(1 + \frac{\lambda a_0}{b_0}\right)
\end{aligned} \tag{9.19}$$

where the impedance filters g_a, g_{sd} and g_d have been moved into one filter using the superposition principle.

9.3 Computing Receiver and Source Position for the 45° Rotated Cube

We will need more care to construct the geometry for the rotated cubic scene, since we are concerned with oblique edges. For the experiment, we will rotate the cube 45° only in the x and z direction, and therefore the y dimension is given as in 7.3. We will need the oblique boundaries to fit the spatial resolution, but since these edges are approximated by staircase edges, the length of the x and z dimensions should fit a multiple of Δx_{45} given by

$$\Delta x_{45} = \sqrt{0.257^2 + 0.257^2} \tag{9.20}$$

as depicted in Figure 7.10 c). The length of the y dimension is given as in (7.2) and the x and z boundaries is computed in a similar way as

$$N_x = N_z = \left\lceil \frac{7}{\Delta x_{45}} - 0.5 \right\rceil \cdot \Delta x_{45} = 6.90 \tag{9.21}$$

Assuming that the uppermost corner is located at $(0, 0, 0)$, we choose the start position $\Delta x/2$ in the x and z dimensions as

$$s_0 = \begin{pmatrix} 0.1857 \\ 0 \\ 0.1857 \end{pmatrix} \tag{9.22}$$

depicted in 7.10 d). By using this procedure for meshing the smallest possible discretisation/meshing error for a cube rotated 45° is obtained.

Translating the source and receiver positions can be done by rotating a point around the y-axis expressed by the rotation matrix R

$$R = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{9.23}$$

where θ is the rotation angle in radians ($45° = \pi/4$). Applying 9.23 for all source and receiver positions from Table (7.1) we have now the positions rotated 45° around the y-axis with oregon in the corner point $c = (0, 0, 0)$. Since the resulting geometry after meshing will be translated such that all coordinates only takes positive values, we also have to translate the rotated coordinates. By looking at Figure 9.2, the corner point c has to be translated $\overrightarrow{p}$ in the x and z directions.

The first step is to compute the length B by

$$C = N_y - \Delta x_{45} \tag{9.24}$$

$$B = \sin\left(\frac{\pi}{4}\right) \cdot C \tag{9.25}$$

$$a = \begin{pmatrix} B \\ 0 \\ 0 \end{pmatrix} \tag{9.26}$$

Because of the start position used for meshing the scene, the corner point p is located at

$$p = \begin{pmatrix} B + \Delta x/2 \\ 0 \\ -\Delta x/2 \end{pmatrix} = \begin{pmatrix} 2.05 \\ 0 \\ -0.1284 \end{pmatrix} \tag{9.27}$$

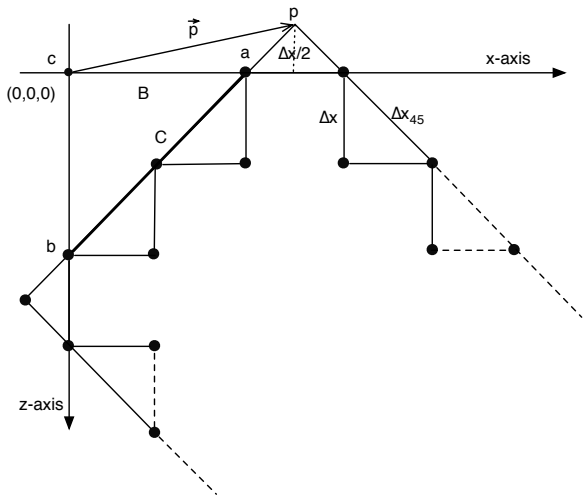


Figure 9.2: Translating source and receiver points for the cube rotated by 45°.

	Coordinates (cm)		
	x	y	z
Scene size	6.90	3.08	5.08
Source pos.	5.00	2.57	3.54
Receiver pos. 1	3.72	1.03	1.90
Receiver pos. 2	5.18	2.31	1.18

Table 9.1: Source and receiver coordinates for the cube rotated 45° and discretised to fit the SLF scheme.

Applying the translation vector $\overrightarrow{p}$ to each of the coordinates and discretising the coordinates according to the spatial resolution gives us the positions in Table 9.1. The position in the corresponding non-rotated cube used for the FEM simulation is obtained by doing the inverse translation and rotation.

9.4 Class Interactions

The four main classes Grid, GridPoint, CSolverBase and CHostSolver are shown in Figure 9.3. CHostSolver is incorporating methods for solving the wave equation using the FDTD method, and is a subclass of CSolverBase responsible for initialising the system. The Grid and GridPoint models a grid and its grid points.

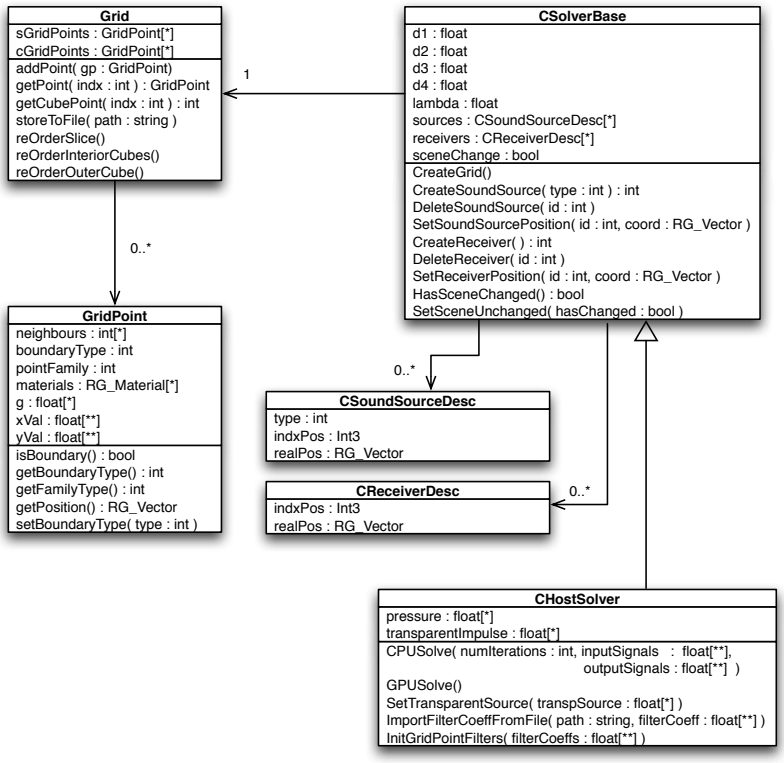


Figure 9.3: UML diagram showing the interaction between the main classes of the system.

9.5 Memory Allocation on the GPU

Copying the memory for each grid point individually turned out to be painfully slow when considering thousand of grid points, due to the performance of cudaMalloc and cudaMemcpy. The solution was unusable in practice, and therefore another approach was taken where all memory on the GPU is allocated and copied in one big chunk for all grid points. For this task, a simple – but powerful – class MemoryAssembly has been constructed with supervision of Frank Wefers. The idea is to let this class keeping track of all of the host allocated memory and

all references to this data, together with the number of bytes allocated for each. A pointer to the allocated memory is returned by the class, and the class member function bind is used when the memory is references by other structures. In this way, the class always knows who are referencing the data by copying the pointer variable by reference.

```
void allocateGridPoint( GridPoint_cu& tmp_gp,
                        MemoryAssembly& gMemAssembly,
                        int dim)
{
  // allocate g
  float* g;
  g = (float*) g.malloc( dim*sizeof(float) );

  // init with 0's
  for (int d = 0; d < dim; ++d)
    gp[d] = 0;

  tmp_gp.g = g;
  gMemAssembly.bind( (void *&) tmp_gp.g, g);
  ...
}
```

When all grid points have been allocated and bind by MemoryAssembly to the temporary grid point structure tmp_gp as before (but this time only host memory is referenced), the caller function allocates memory on the device for the given sub-structure in one big chunk as

```
cudaMalloc((void**) &d_all_g, totalNumBytesAlloc_g);
```

The goal is now to automatically swap all the pointers in the temporary structure, such that they point on the corresponding memory on the device. This is done by calling the class function assemble which will swap all the pointers in the temporary sub-structure to the newly allocated device memory given as argument to assemble. The memory allocated on the host does not contain any data yet, therefore all the allocated memory is copied in correct order to the destination on the host pointed to by a second argument to assemble. This data is then copied as one big chunk to the allocated memory on the device:

```
float *pDest_g = (float*) malloc(memSize_g);
gMemAssembly.assemble(pDest_g, d_all_g);
cudaMemcpy( d_all_g, pDest_g, memSize_g, cudaMemcpyHostToDevice);
```

The MemoryAssembly class has been keeping track of all memory, and therefore all pointers in the temporary structure points at the correct device memory

locations. Finally, the array tmp_gridPoints of temporary GridPoint_cu is copied to the device and free'd afterwards.

9.6 Mail Correspondence with Konrad Kowalczyk

Dear Nikolas,

Please find enclosed the pdf with update equations for all 7 boundary types. If you still have a problem with this one 3/8 boundary type, you can try an alternative equation which is given at the end of that file. Another reason for some instabilities might be that you used this 3/8 equation also for boundary points denoted as 5/8 in the enclosed pdf, which you did not mention in the thesis.

In general, you can always test which equations give rise to instabilities by setting all other boundary points in your simulation as 0, i.e. not updating them at all, which yields a perfect -1 reflection, and let your simulation run for a very long time.

Anyway, I think you really grasped the topic, thought about these simulations really thoroughly, and finally tried (mostly correctly) deriving many equations yourself, which I hope your supervisors will take into account when evaluating your M.Sc. thesis.

Best regards,
Konrad

On Tue, 02/07/2012 12:55 AM, Nikolas Borrel-Jensen ¡mail@nikolasborrel.com¿ wrote:
cc Frank Wefers, RWTH Aachen University
cc Jakob Grue Simonsen, University of Copenhagen

Dear Konrad Kowalczyk

To your knowledge, I might attach this conversation in the appendix of my thesis, if you don't have any complains about that.

Again, thanks a lot for your willingness to help, it is very helpful! I hope you have the time for guiding me to a solution for my instability problems for arbitrary geometries.

I have derived the update schemes for inner corners, inner edges and inner edge corners. As far as I can see from my investigation of the problem, the instability problems arises only at the inner edge-corners. I have been debugging my implementation very carefully, and I feel quite sure about the correctness of the logic in my implementation. Therefore, the only problem that I can think of is, that my derivation of inner edge-corners somehow is wrong. For inner edge-corners, the easiest way to derive these is to use a plane boundary as a starting point, and eliminating the two ghost points using the formula for two meeting boundaries.

I have attached an extract from my thesis, I hope you can find some time to confirm or (hopefully) disconfirm the correctness of inner edge-corners given in Eq. (5.62).

By the way, I think that there is a mistake in the formula you send me for inner edges. In the d_3-terms, one more ghost-point elimination has to be done, as far as I can see?

Again, thanks for you detailed answers, I hope you can help me resolving this last problem preventing me from simulating in other geometries than the shoebox room...

Kind regards,
Nikolas Borrel-Jensen

Den 05/02/2012 kl. 19.18 skrev Konrad Kowalczyk:

Hi,

1) As for the DIF implementational issues, this is true that there are a few more boundary updates possible for interpolated schemes. The most important thing to remember when deriving them is that the same set of equations as describe in the paper can be used as a basis and next they can be used to derive the actual updates. Staircase implementation actually reduces the number of possible boundary update types. I prepared a short pdf for you that hopefully explains how this is done.

2) Please find enclosed the figures for Matlab, which should make it easier for you to analyze dispersion.

All the best for your MSc project.

Regards,
Konrad

9.7 Comparison Between the FDTD and the FEM method: TF plots

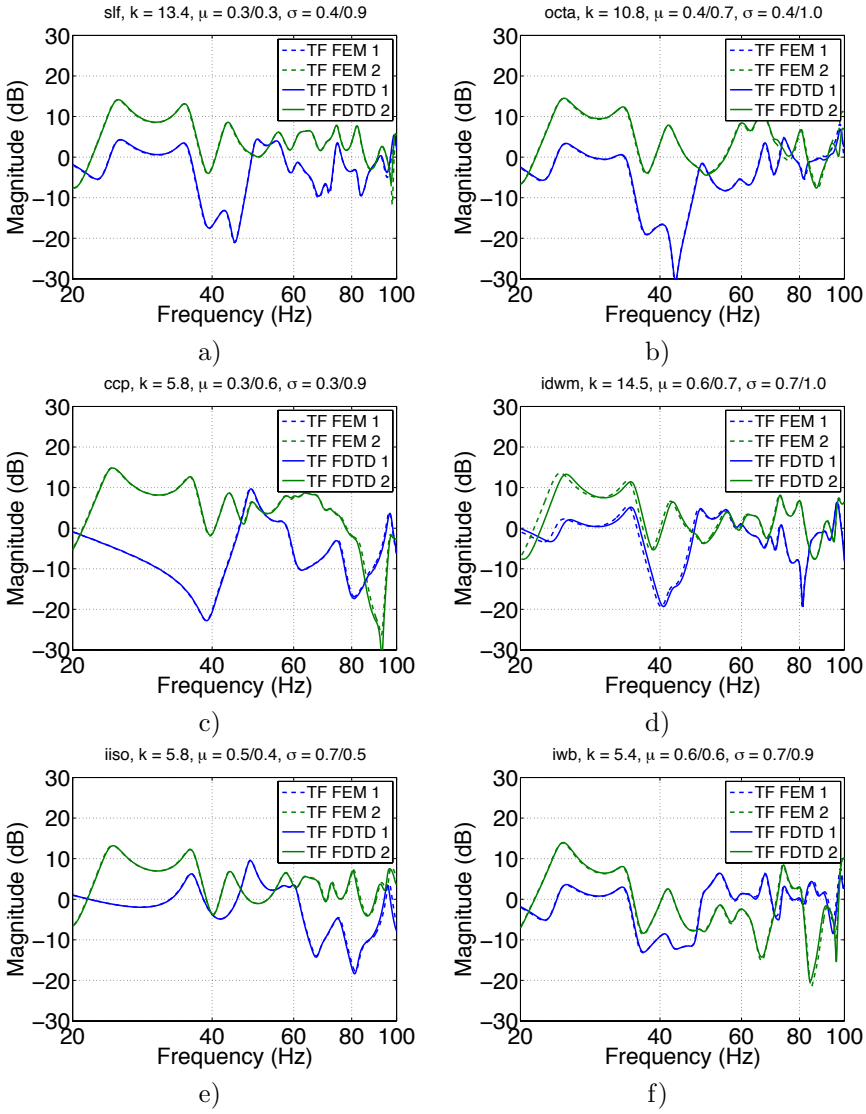


Figure 9.4: The FDTD method compared with the FEM method. The setup is depicted in Fig. 7.1 for frequency indep. absorption given in Table 7.2. One source and two receivers are used.

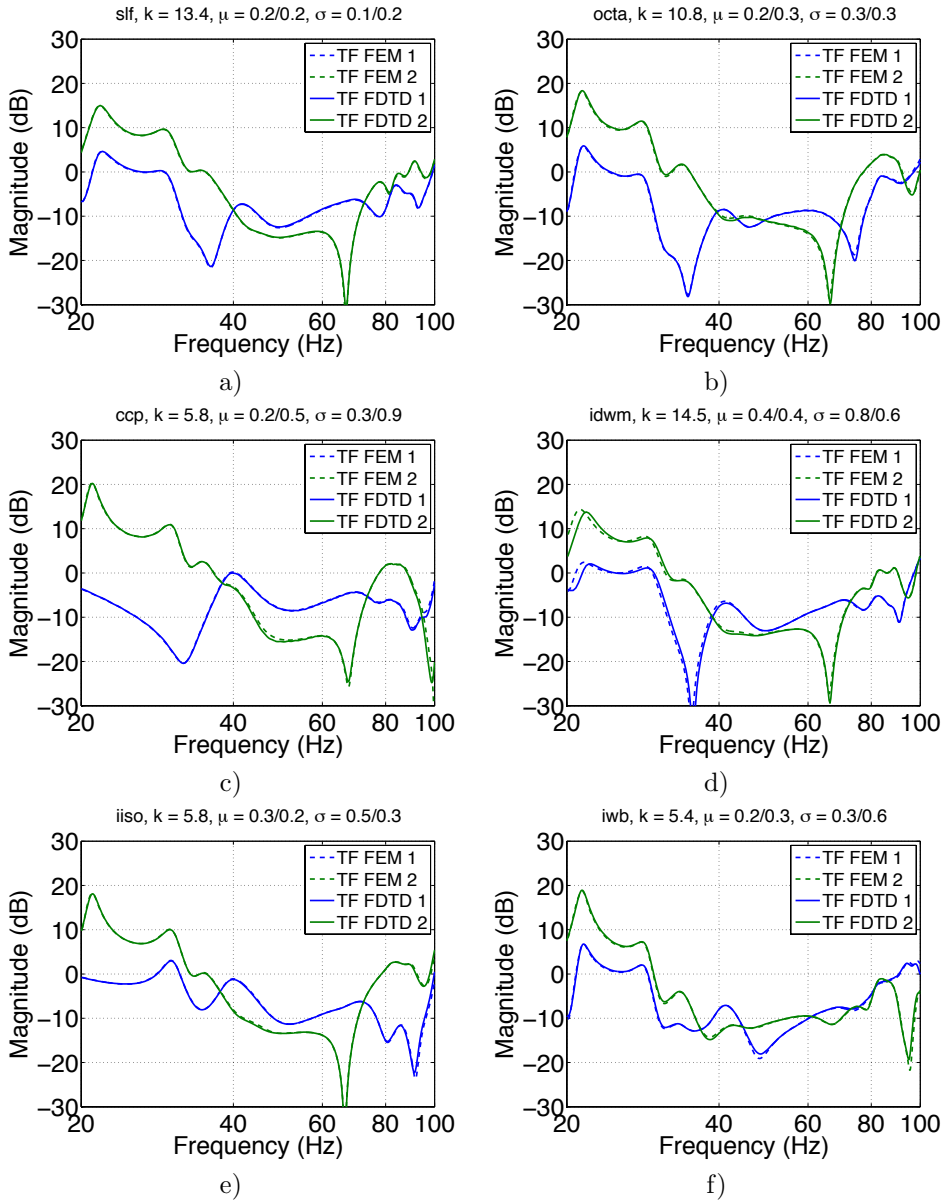


Figure 9.5: The FDTD method compared to the FEM method. The setup is depicted in Fig. 7.1 using the Helmholtz resonator from Figure 7.6 for all boundaries. One source and two receivers are used. a) SLF, b) OCTA, c) CCP, d) IDWM, e) IISO, f) IWB.

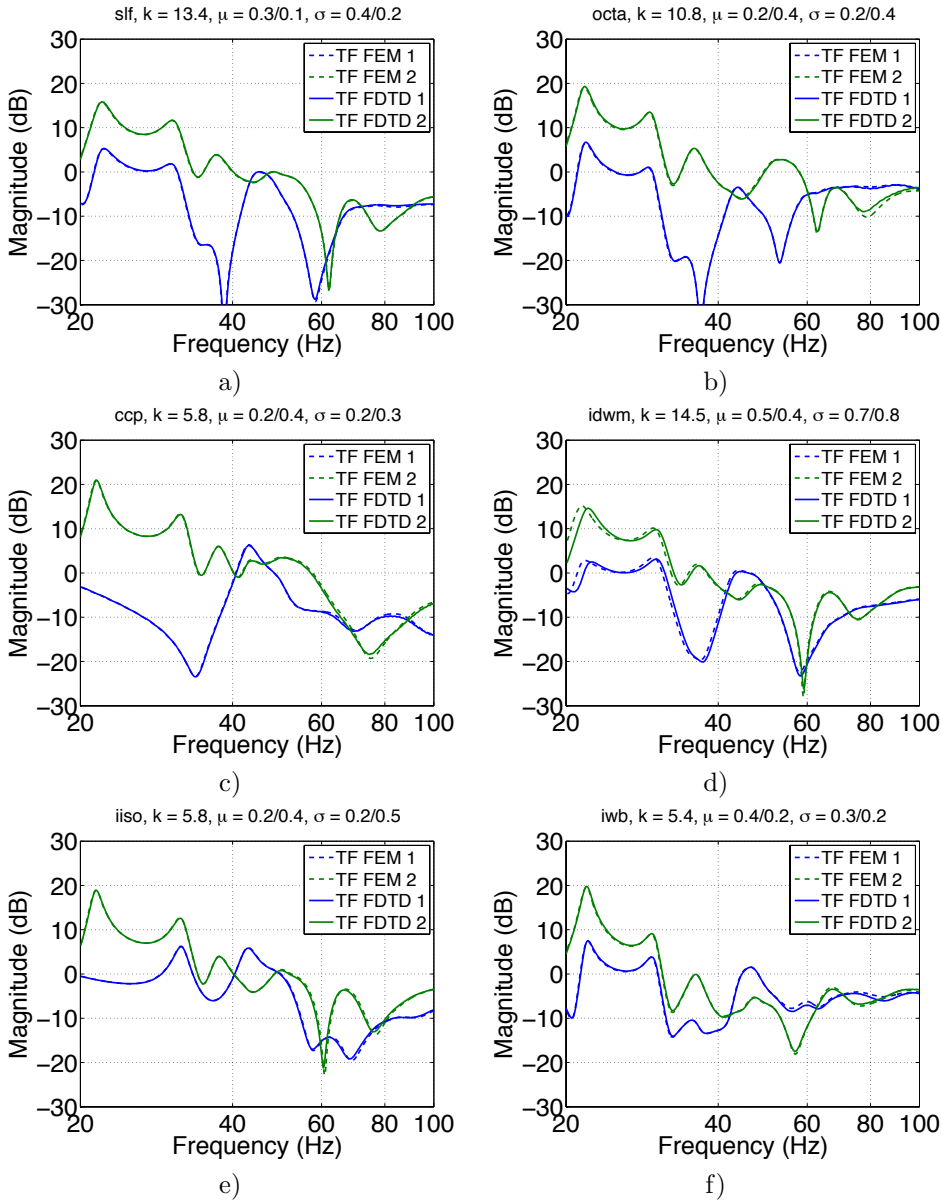


Figure 9.6: The FDTD method compared to the FEM method in the Living Room scene using the porous absorber from Figure 7.6 for all boundaries. One source and two receivers are used. a) SLF, b) OCTA, c) CCP, d) IDWM, e) IISO, f) IWB.

Bibliography

Botteldooren, D. (1995). Finite-difference time-domain simulation of low-frequency room acoustic problems. *The Journal of the Acoustical Society of America 98*(6), 3302–3308.

Duyne, S. A. V. and J. O. Smith III (1993). Physical modeling with the 2-d digital waveguide mesh.

Gedney, S. (2010, december). *Introduction to the Finite-Difference Time-Domain (FDTD) Method for Electromagnetics amazon*. Morgan Claypool Publishers.

Hunter, P. (2001). Fem/bem notes. Technical report.

Huopaniemi, J., L. Savioja, and M. Karjalainen (1997). Modeling of reflections and air absorption in acoustical spaces a digital filter design approach. *Applications of Signal Processing to Audio and Acoustics, 1997. 1997 IEEE ASSP Workshop on*, 4 pp.+.

Karjalainen, M. and C. Erkut (2004, January). Digital waveguides versus finite difference structures: equivalence and mixed modeling. *EURASIP J. Appl. Signal Process. 2004*(1), 978–989.

Katsibas, T. K. and C. S. Antonopoulos (2002). An efficient pml absorbing medium in fdtd simulations of acoustic scattering in lossy media. *Proceedings of the IEEE Ultrasonics Symposium 1*, 551–554.

Kelloniemi, A. (2006, September). Frequency-dependent boundary condition for the 3-d digital waveguide mesh. In *In Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06)*, pp. 161–164.

Kowalczyk, K. (2008a). Boundary and medium modelling using compact finite difference schemes in simulations of room acoustics for audio and architectural design applications.

Kowalczyk, K. and M. van Walstijn (2010a, march). A comparison of nonstaggered compact fdtd schemes for the 3d wave equation. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 197–200.

Kowalczyk, K. and M. van Walstijn (2010b, jan.). Wideband and isotropic room acoustics simulation using 2-d interpolated fdtd schemes. *Audio, Speech, and Language Processing, IEEE Transactions on 18*(1), 78 –89.

Kowalczyk, K. and M. van Walstijn (2011). Room Acoustics Simulation Using 3-D Compact Explicit FDTD Schemes. *IEEE Transactions on Audio, Speech, and Language Processing 19*(3), 34–46.

Kowalczyk, K., M. van Walstijn, and D. Murphy (2011). A Phase Grating Approach to Modeling Surface Diffusion in FDTD Room Acoustics Simulations. *IEEE Transactions on Audio, Speech, and Language Processing 19*(3), 528–537.

Kowalczyk, M. (2008b). Modeling Frequency-Dependent Boundaries as Digital Impedance Filters in FDTD and K-DWM Room Acoustics Simulations. *J. Audio Eng. Soc 56*(7/8), 569–583.

Kuttruff, H. (2000, October). *Room Acoustics* (4 ed.). Taylor & Francis.

Murphy, D., A. Kelloniemi, J. Mullen, and S. Shelley (2007, march). Acoustic modeling using the digital waveguide mesh. *Signal Processing Magazine, IEEE 24*(2), 55 –66.

Murphy, D. T. and M. Beeson (2007). The KW-Boundary Hybrid Digital Waveguide Mesh for Room Acoustics Applications. *Audio, Speech, and Language Processing, IEEE Transactions on [see also Speech and Audio Processing, IEEE Transactions on] 15*(2), 552–564.

NVidia (2011a). Nvidia cuda c best practice guide, version 4.0, http://developer.nvidia.com/category/zone/cuda-zone.

NVidia (2011b). Nvidia cuda programming guide, version 4.0, http://developer.nvidia.com/category/zone/cuda-zone.

Raghuvanshi, N., R. Narain, and M. C. Lin (2009, September). Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Transactions on Visualization and Computer Graphics 15*, 789–801.

Rickard, Y. S., N. K. Georgieva, and W.-P. Huang (2003). Application and optimization of PML ABC for the 3-D wave equation in the time domain. *Antennas and Propagation, IEEE Transactions on 51*(2), 286–295.

Savioja, L. (2010, September). Real-Time 3D Finite-Difference Time-Domain Simulation of Mid-Frequency Room Acoustics. In *13th International Conference on Digital Audio Effects (DAFx-10)*.

Schneider, J. B., C. L. Wagner, S. L. Broschat, S. E. Al, J. Acoust, and S. Am (1998). Implementation of transparent sources embedded in acoustic finite-difference time-domain grids.

Smith, J. (2012). Physical audio signal processing for virtual musical instruments and digital audio effects, https://ccrma.stanford.edu/ jos/pasp/.

van Walstijn, M. and K. Kowalczyk (2008). On the numerical solution of the 2d wave equation with compact fdtd schemes. In *11th Int. Conference on Digital Audio Effects (DAFx-08), Espoo, Finland*, pp. 205–212.

Vorländer, M. (2007, November). *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality (RWTHedition)* (1 ed.). Springer.

Yee, K. (1966, May). Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media. *IEEE Transactions on Antennas and Propagation 14*(3), 302–307.