Aalto University
School of Electrical Engineering

Jukka Saarelma

Finite-difference time-domain solver for room acoustics using graphics processing units

Master's Thesis
Espoo, November 11, 2013

Supervisor: Professor Lauri Savioja
Instructor: Ph.D Jonathan Botts

Aalto University
School of Electrical Engineering

ABSTRACT OF
MASTER'S THESIS

Author:	Jukka Saarelma		
Title:	Finite-difference time-domain solver for room acoustics using graphics processing units		
Date:	November 11, 2013	**Pages:**	8 + 64
Department of Mediatechnology			
Professorship:	Mediatechnology	**Code:**	IL3011
Supervisor:	Professor Lauri Savioja		
Instructor:	Ph.D Jonathan Botts		

Several acoustic simulation methods have been introduced during the past decades. Wave-based simulation methods have been one of the alternatives, but their applicability for wideband acoustic simulation has been limited by the computing power of available hardware. During recent years, the processing power and programmability of graphics processing units have improved, and therefore several wave-based simulation methods have become potential alternatives. In this thesis, a finite-difference time-domain solver is implemented. The performance of the solver is accelerated with the use of graphics processing units. Different performance considerations are reviewed and the system is evaluated by comparing the simulated responses to known analytic solutions.

The resulting system is C++ software, which is interfaced with Matlab with the use of a mex-function. It is found that the forward difference boundary formulation is the most efficient for parallel implementation due to a lesser number of operations. The usage of double precision data type in the simulation decreases the performance significantly. The system is found to follow the analytical solutions with accuracy expected of the method, apart from the reflection characteristics of the forward difference boundary formulation that deviate slightly from the analytical solution.

Keywords:	acoustics, acoustic simulation, CUDA, finite-difference time-domain method, parallel computing, visualization, wave equation
Language:	English

Aalto-yliopisto
Sähkötekniikan korkeakoulu

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Jukka Saarelma		
Työn nimi:	Aaltoyhtälön numeerinen ratkaisija aika-alueen differenssimenetelmällä käyttäen graffiikkaprosessoreja		
Päiväys:	11. marraskuuta 2013	**Sivumäärä:**	8 + 64
Mediatekniikan laitos			
Professuuri:	Mediateknologia	**Koodi:**	IL3011
Valvoja:	Professori Lauri Savioja		
Ohjaaja:	TkT Jonathan Botts		

Erilaisia akustisia simulaatiomenetelmiä on kehitetty viime vuosikymmenien aikana. Yhtenä vaihtoehtona on käytetty aaltopohjaisia ratkaisioita, mutta laskennallinen tehokkuus on usein rajoittava tekijä niiden käytölle. Viimevuosina grafiikkaprosessoreiden ja ohjelmistorajapintojen kehitys on mahdollistanut erilaisten aaltopohjaisten menetelmien käytön. Tässä työssä toteutetaan aaltoyhtälön ratkaisija aika-alueen differenssimenetelmällä. Toteutuksen tehokkuutta parannetaan hyödyntämällä grafiikkaprosessoreita ja eri toteutusvaihtoehtoja verrataan. Jäjestelmällä estimoituja vasteita verrataan tunnettuihin analyyttisiin ratkaisuihin.

Toteutettu järjestelmä on C++-ohjelma jota voidaan käyttää Matlab-ympäristöstä hyödyntäen Matlab-ohjelmiston mex-rajapintaa. Päivitysyhtälö jossa reunaehdot on toteutettu etenevällä differenssillä todetaan tehokkaimmaksi vaihtoehdoki. Simulaation tehokkuus alenee huomattavasti käytettäessä kaksikertaista laskentatarkuutta. Voidaan todeta, että järjestelmän estimoimat vasteet toteuttavat odotetulla tavalla analyyttiset ratkaisut, poislukien etenevällä differenssillä toteutetun reunaehdon heijastusominaisuudet, jotka eroavat analyyttisestä mallista.

Asiasanat:	akustiikka, akustinen simulaatio, CUDA, aika-alueen differenssimenetelmä, rinnakkaislaskenta, visualisointi, aaltoyhtälö
Kieli:	Englanti

Acknowledgements

I would like to offer my gratitude first of all to Professor Lauri Savioja who offered this thesis position to me. Most probably I would not have had the guts to pursue this topic without such straightforward encouragement. I would like to thank my instructor Ph.D. Jonathan Botts for various helpful discussion on topics concerning this thesis and acoustic simulation in general, and M.Sc. Henrik Karlson for providing support and new features to the voxelization libary when needed in a truely professional manner.

In addition I would like to thank my co-worker B.Sc Perttu "Liitto" Laukkanen for making my tendencies to be unconcerned with attending conferences during the spring and summer 2013 look ridiculous and overall for keeping the morale high around the office. The conference visits during this work were extremely motivational.

Research leading to these results has received funding from the Academy of Finland project 'Efficient perceptually optimal simulation of room acoustics' (No. 265824).

Espoo, November 11, 2013

Jukka Saarelma

Contents

Symbols and abbreviations

Symbols

c	Speed of sound
f_s	Sampling frequency
k	Wavenumber
p	Pressure
R	Reflection coefficient
T	Temperature of air
t	Time
v	Particle velocity
Z	Surface impedance
α	Absorption coefficient
λ	Courant number
ω	Angular frequency
ρ_0	Air density
θ	Angle of incidence
ξ	Specific acoustic impedance

Operators

∇^2	Laplacian
$\frac{\partial f}{\partial x}$	Partial derivative of f with respect to x

Abbreviations

2-D	Two-Dimensional
3-D	Three-Dimensional
API	Application Programming Interface

ARD	Adaptive Rectangular Decomposition
ART	Acoustic Radiance Transfer
BEM	Boundary Element Method
BRDF	Bidirectional Reflectance Transfer Function
CUDA	Computer Unified Device Architecture
DWM	Digital Waveguide Mesh
FDTD	Finite-Difference Time-Domain
FEM	Finite Element Method
FLOPS	Floating-Point Operations Per Second
GDDR	Graphics Double Data Rate
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
HLSL	High-Level Shader Language
ISA	Instruction Set Architecture
IWB	Interpolated Wideband
MEX	Matlab Executable
OS	Operating System
PBO	Pixel Buffer Object
PTX	Parallel Thread Execution
SLF	Standard Leapfrog
SM	Streaming Multiprocessor
SP	Streaming Processor
SRL	Standard Rectilinear
VBO	Vertex Buffer Object

Chapter 1

Introduction

Several different simulation methods have been introduced to predict acoustical characteristics of spaces. The methods are commonly divided into geometric and wave-based methods. Geometric methods estimate the propagation of sound as rays whose propagation can be computed geometrically from the orientations of the reflecting surfaces. Such methods can estimate the propagation of sound in a wide bandwidth. However they do not take into account the wave phenomena of sound. This leads to severe inaccuracies at low frequencies. Wave-based methods are used to solve the wave equation directly. Such methods can predict the wave phenomena of sound but they are computationally expensive.

During the past decade, the physical limits of single processor cores have become evident. This fact has lead processor manufacturers to develop multi core processor architectures. Applications executing intense graphics processing, such as games, have used the parallel hardware of *graphics processing units* (GPUs) for real time rendering, and maintained a demand for parallel processor architectures with more processing power. In 2007, Nvidia published a parallel computing platform CUDA, which allowed programmers to use GPUs for general purpose computing. The use of GPUs allows significant performance improvements in data parallel applications.

The *finite-difference time-domain* (FDTD) method is a well-known method to numerically solve partial differential equations. The method has been used from the beginning of the 20th century when it was used to prove the existence of the solutions of partial differential equations. For simulation purposes, FDTD was first used in electromagnetic field problems but was eventually adapted to acoustics. The method introduces severe direction- and frequency-dependent dispersion to the simulated sound field. The least dispersion occurs at low frequencies. To decrease the dispersion error, the effective sampling frequency of the mesh must be increased by oversampling,

which increases the domain size and therefore leads to high computational requirements. The advantage of the method is that the field update equation can be formulated in a completely data parallel manner. This leads to the problem statement of this thesis.

1.1 Problem statement

The FDTD method has inherently high computational requirements due to large domain sizes. Estimating the propagation of sound in an enclosure using FDTD with serial programming is possible, but significant speedups are achievable by using GPUs and parallel programming. The advantage over other wave-based methods is that the formulation is easily parallelizeable, the update equation can be applied uniformly across the domain, and it does not need precomputation. The aim of this thesis is to develop an efficient FDTD solver for room acoustics using GPUs. As the method allows the observation of the propagation of sound in the time domain, an efficient visualization scheme taking advantage of the use of GPUs is implemented. Different options regarding the design of the implemented system are evaluated.

1.2 Outline of this thesis

This thesis is structured as follows. In Chapter 2, the physical background relevant to the implementation is reviewed and a brief introduction to acoustic simulation methods is presented. In Chapter 3, a more detailed description of the FDTD method is given. In Chapter 4, the architecture of the device family used in this work is reviewed and the programming model is introduced. In Chapter 5, a general review of several similar systems is given and the developed system is described in more detail. In Chapter 6, the capabilities of the developed system are reviewed and the performance is evaluated. In Chapter 7, concluding remarks are presented and the future direction of the system development is discussed.

Chapter 2

Background

Room acoustic simulation is based on the physical principles of wave propagation. This work consists of implementing a system which effectively estimates the progress of a sound field in the time-domain, therefore in order for the reader to follow the derivation of the implementation, the theoretical formulation of the propagation of sound is reviewed. As several different methods for the same task have been proposed over time, to understand the problem this work seeks to solve, the principles and the limitations of existing methods are discussed.

2.1 Fundamentals of Room Acoustics

2.1.1 Wave Equation

A sound wave is defined as the vibration of particles about their mean position. Fluids such as air have mass density and volume elasticity and therefore they have similar characteristics as a chain of masses and springs. Harmonic motion is generated by two forces: the restoring force and the inertia of the mass of the object. In the case of sound, the restoring force is caused by the elasticity of air which resists it being compressed and the force causing the motion is the inertia of the mass density of air. The vibration in a sound wave is not uniform within the medium. Particles move in a different phase in different points in a sound field. The variation between both pressure and velocity is a function of time and position.

The detailed properties of the acoustic wave motion depend on whether the air is in thermodynamic equilibrium or not, the ratio between the amplitude and frequency of the acoustic motion, the molecular mean-free-path and the collision frequency [24, p. 227]. The behavior of wave motion in

fluids is nonlinear because the pressure changes affect the temperature of the medium and therefore the speed of sound. Generally, the effects resulting from inhomogeneities in the medium in room acoustics are considered so small that they can be neglected [20] and therefore a simplified model can be used. Common assumptions for a simplified model are that the medium is an idealized fluid, its properties are uniform and continuous, and it is in thermodynamical equilibrium. In such a homogeneous and isotropic medium, the velocity of sound is constant with reference to space and time. The speed of sound can be described with the equation

$$c = (331.4 + 0.6\,T)\frac{m}{s}, \tag{2.1}$$

where T is the temperature of the air in degrees Celsius. The linear approximation of sound propagation can be described by the first-order equations defining the acceleration of the fluid produced by pressure gradient

$$\rho_0 \frac{\partial \boldsymbol{u}}{\partial t} = -grad\ p, \tag{2.2}$$

and the compression produced by the velocity gradient

$$\kappa \frac{\partial p}{\partial t} = -div\ \boldsymbol{v}, \tag{2.3}$$

where p is the sound pressure, $\boldsymbol{v}$ the vector particle velocity, t the time, ρ_0 the gas density of air, c the speed of sound and κ the adiabatic exponent defined by

$$\kappa = \rho_0 c^2. \tag{2.4}$$

Combining these two equations by eliminating the particle velocity $\boldsymbol{u}$, a differential equation describing the propagation of sound waves in any lossless fluid is achieved:

$$c^2\ \nabla^2 p = \frac{\partial^2 p}{\partial^2 t}, \tag{2.5}$$

where

$$c^2 = \kappa \frac{p_0}{\rho_0}. \tag{2.6}$$

The operator ∇^2 is given as

$$\nabla^2 p = \frac{\partial^2 p}{\partial^2 x^2}, \tag{2.7}$$

$$\nabla^2 p = \frac{\partial^2 p}{\partial^2 x^2} + \frac{\partial^2 p}{\partial^2 y^2}, \tag{2.8}$$

$$\nabla^2 p = \frac{\partial^2 p}{\partial^2 x^2} + \frac{\partial^2 p}{\partial^2 y^2} + \frac{\partial^2 p}{\partial^2 z^2}, \tag{2.9}$$

in 1, 2 and 3-dimensional cartesian coordinates.

2.1.2 Reflection and Scattering

Room acoustics is by definition related to structures that reflect sound energy. How sound propagates inside an enclosed space is determined by the geometry and the material properties of the boundaries, namely the surfaces, floor, ceiling, and walls. Room boundaries reflect a fraction and transmit a fraction of the sound energy that hits the surface. The combination of the fractions of sound energy is what usually is considered the acoustics of a room.

When a plane wave hits an unbounded and uniform surface, part of the sound energy is reflected in a form which is differing in phase and amplitude. The amplitude and phase change is expressed by the complex reflection factor

$$R = |R|\, e^{i\omega}, \tag{2.10}$$

that is a property of the surface. The magnitude and phase angle are frequency and incident angle dependent. The intensity of the reflected wave is smaller by a factor $|R|^2$ and the energy lost during reflection is therefore $1 - |R|^2$. This quantity is known as the *absorption coefficient* of the surface:

$$\alpha = 1 - |R|^2. \tag{2.11}$$

The reflection factor completely describes the properties of a locally reacting surface in a room acoustical point of view for all angles of incidence and for all frequencies. The reflection characteristics of a surface can be described based on the normal particle velocity, which is generated by the sound pressure on the surface. This is called the surface impedance and is given as

$$Z = \frac{p}{v_n}, \tag{2.12}$$

where v_n denotes the velocity component normal to the surface. Like the reflection factor, the surface impedance is generally complex. A frequently used variant of the formulation is called the *specific acoustic impedance*, which is defined by the ratio of the surface impedance and the characteristic impedance of air:

$$\xi = \frac{Z}{\rho_0 c}. \tag{2.13}$$

The reciprocal of the surface impedance is called the *wall admittance*, and the reciprocal of ξ is the *specific wall admittance*.

Reflection at normal and oblique incidence

The reflection coefficient of a surface can be formulated from the equations describing a progressive plane wave. In the case that the normal of the surface is parallel to the direction that the incident wave is traveling, the following formulation can be derived.

A plane wave traveling in a cartesian x-y coordinate system in the direction of the x-axis can be defined with two equations, which describe the pressure component

$$p(x,t) = p_0 e^{i(\omega t - kx)}, \tag{2.14}$$

and the velocity component

$$v(x,t) = \frac{p_0}{\rho_0 c} e^{i(\omega t - kx)}, \tag{2.15}$$

of the wave. The variable $k = \omega/c$ denotes the wavenumber. When such a wave intersects a boundary at normal incidence, part of the wave is reflected and part of it is transmitted into the boundary medium. The reflected part of the wave changes the direction. The amplitude of the reflected part reduces due to the boundary absorption, and the phase of the wave changes. Both changes are fully defined by the reflection coefficient R. As the direction of the reflected part of the wave is reversed, the sign of the velocity component is inverted. Now the equations for the reflected wave can be defined as

$$p(x,t) = R p_0 e^{i(\omega t + kx)}, \tag{2.16}$$

$$v(x,t) = -R \frac{p_0}{\rho_0 c} e^{i(\omega t + kx)}. \tag{2.17}$$

To solve the values of two components in the boundary plane, one can simply assign $x = 0$ and sum the reflected and incident parts of the wave:

$$p(0,t) = p_0(1+R)e^{i(\omega t+kx)}, \tag{2.18}$$

$$v(0,t) = R(1-R)\frac{p_0}{\rho_0 c}e^{j(\omega t+kx)}. \tag{2.19}$$

The wall impedance is obtained by dividing $p(0,t)$ by $v(0,t)$:

$$Z = \rho_0 c\frac{1+R}{1-R}, \tag{2.20}$$

and consequently the reflection coefficient can now be expressed with:

$$R = \frac{Z-\rho_0 c}{Z+\rho_0 c}. \tag{2.21}$$

In a more general case, when the angle of incidence of the wave θ varies between 0° and 90°, the plane wave formula can be expressed with the use of a coordinate rotation $x' = x\cos\theta + y\sin\theta$. The solution for the rotated plane wave is inserted into the pressure and velocity components (2.16) and (2.17). Proceeding in similar manner as in the derivation of the reflection at normal incidence as presented in [20, p. 43], a formulation for reflection at oblique incidence is attained and has the form

$$R_\theta = \frac{\xi_w\cos\theta - 1}{\xi_w\cos\theta + 1}, \tag{2.22}$$

where θ is the angle of incidence, and ξ_w is the specific acoustics impedance of the surface.

2.2 Acoustic simulation methods

Acoustic simulation of spaces has become a widely used tool in a variety of applications. Primary use of modeling algorithms has been in room acoustic prediction and auralization. All the methods are based on approximating the wave equation with given boundary conditions on the surfaces of the geometry.

Different methods of acoustic modeling have different computational requirements and characteristics. The size, shape and properties of the room

dictate which method is possible to use. Usually modeling methods are divided into two main categories: geometric and wave-based methods. In this section, an review of different methods is given.

2.2.1 Geometric Methods

The first main category is usually referred to as *geometric methods*. In geometric methods the propagation of sound is represented as rays describing the path of the traveling sound energy. The path that the sound propagates through is calculated using knowledge of the geometric structure of the enclosure.

Specular reflections from the surfaces of the room can be estimated with the use of the *image source method* [1]. When surfaces have diffuse characteristics, they can be simulated by assigning a certain degree of randomness to the ray path, or by prescribing the directional reflection characteristics of the surfaces. In these cases the response of room can be estimated with *ray tracing* or *radiance transfer*.

Image Source

The concept of the image source method is based on the principle that each reflection from the boundaries of an enclosure can be represented as a source radiating in free space. This is achieved by calculating the positions of such sources from the geometry of the room by mirroring the physical source according to the orientation of each reflecting surface. The image source method can be described as sound field decomposition where the sound field is represented as spherical waves in superposition.

The image source method results in an exact solution to the wave equation in a rectangular room with rigid boundaries [1]. In the case of arbitrary geometries, the basic form of the method estimates efficiently only the specular reflections [4]. To take into account the wave phenomena of sound, the image source method can be extended by estimating edge diffraction with separate image sources [34].

A major drawback of the image source method is computational cost. The total number of image sources which are calculated is $n_i = s(s-1)^{n-1}$ where s is the number of surfaces and n is the chosen limit of reflection order. Several other methods of searching the specular reflection paths have been proposed to reduce the computational cost such as *beam tracing* [12].

Ray Tracing

As stated previously, the drawbacks of the image source method are the exponential growth of number of image sources and that the method cannot estimate diffuse reflections. In order to model diffuse reflections with geometric methods, statistical boundary properties are assigned. In such case a Monte Carlo method must be utilized. One of the most-used of such methods in acoustics is ray tracing [19]. In ray tracing a large number of rays is emitted from a source position and the path which each ray travels between the surfaces of the simulated geometry is traced. Each boundary reflection is saved into the computer's memory. After each ray has either traveled up to a predefined reflection order or the ray has hit the receiver volume, the impulse response of the room is constructed from the generated ray path information.

The advantage of ray tracing is that the late part of the room response which is stochastic in nature can be estimated in a feasible way. The downside of the method is that possible paths contributing to the early reflections might not be traced at all because the method launches a discrete number of rays with a given angular distribution. The method is found to be able to estimate acoustic parameters with useable accuracy, and it is used in commercial software jointly with the image source method.

Radiosity Methods

Radiosity methods are closely related to ray tracing. In radiosity methods the surfaces of the model are divided into elements. Every element is considered as a potential emitter and reflector of radiosity. Between each element pair a form factor is calculated which defines the fraction of radiosity which is carried between the elements. The characteristics of the emitter are usually defined by the scattering coefficient of the surface.[26]

A general form of such methods, or to be exact, a generalization of all geometric methods, the acoustic radiance transfer function (ART) was introduced by Siltanen [43]. In ART the energy exchange of each surface is defined with *bidirectional reflectance transfer function* (BRDF). BRDF maps each incoming angle of incidence into a distinct reflection angle and amplitude. The method has high memory requirements due to the high number of energy exchanges between the elements, but it is capable of estimating the late part of the impulse response efficiently with the complex diffusing properties o.

Another method that is similar to radiosity methods is the prediction of sound propagation with the diffusion equation [33]. In this method, the

propagation of sound is presented as a statistical measure describing the probability that a particle is localized in a certain position with a certain velocity. The goal is to prescribe the progress of the sound field statistically. The method can be derived from the ART method [25] which, as was noted previously, is proposed as a general theory for geometric acoustics.

2.2.2 Wave-based Methods

The second main category of acoustic simulation methods is wave-based methods. In wave-based methods, the propagation of sound is described with different forms of the wave equation. All wave-based methods need a discrete representation of the geometry. The geometry is either discretized into volume or surface elements. Element methods are commonly used to solve the wave equation in the frequency domain. Time domain solvers are used for specific forms of the wave equation. Such methods are computationally expensive, but can be formulated in data parallel manner.

Element methods

Element methods are commonly divided into the *finite element method* (FEM) and the *boundary element method* (BEM). The functional difference between these two methods is that in FEM, the propagation is estimated with volumetric elements and in BEM the propagation is estimated with the use of surface elements.

In FEM, the geometry is represented with a volumetric grid of elements which can be either uniform or non-uniform. The sound field in each volume is represented with an analytical shape function which describes the behavior of the material. In the case of air, the calculation involves forcing the pressure to be continuous between the elements and force the sound field inside the element to fulfill the wave equation [49].

In BEM, the surfaces of the geometry are subdivided into elements and the contribution of particle velocity and the sound pressure on each surface element describes the sound field inside or outside the given geometry. The method is a numerical solver of the Kirchhoff-Helmholtz equation [49]. The main benefit in such technique is that the sound field can be described in continuous medium without discretization.

Time-domain solvers

Instead of solving the sound field with integral equations solely in the spatial domain, it is also possible to simulate the propagation of sound in the time

domain. In such methods, the wave equation is first solved in general manner with the use of a suitable discretization. One of the most used method is the FDTD method, which is the method used in this work. A review of the FDTD is presented in the next chapter.

Another formulation of discretized wave propagation used in room acoustics and sound synthesis is through traveling wave solution. Such formulation is used in digital waveguides [44], also referred to as *digital waveguide meshes* (DWMs) [50] in multiple dimensions. The DWM is a mesh of bidirectional delay units which are connected with scattering conjunctions. DWMs and finite-difference schemes have high degree of functional equivalence. The fundamental difference between the two methods is that finite-difference methods process signals and DWMs the wave decompositions of signals.

A different approach to solve the sound field in the time domain is using discrete cosine transform in the spatial domain. On such method as described by Ranghuvanshi et al.[35], the volume is divided into rectangular decompositions which allows to describe the sound field with cosine terms. This formulation leads to more efficient algorithms because the need for oversampling is significantly smaller when the sound field is prescribed in the frequency domain. The downside of the method is that the handling of the interfaces between the different rectangular domains is a fairly complex task and reduces the efficiency of the method. Additionally, with a lower spatial sampling rate the volumetric representation of the geometry is less accurate.

Chapter 3

The Finite-Difference Time-Domain method

Finite-difference methods have been popular among researchers for several decades. The method is used in several fields of science to numerically solve partial differential equations of different types. The origins of the method date back to 1928 when Courant, Friedrichs and Lewy used difference schemes to prove the existence of solutions of partial difference equations [7]. The finite-difference time-domain method is a finite-difference method where the partial difference equation of interest is discretized in the time domain. The method was introduced to engineering by Yee [55] to solve electro magnetic field problems. The scheme introduced is known as the staggered or interleaved grid where the magnetic and electric fields are solved separately in an interleaved manner. Yee's staggered grid approach was adapted for room acoustics by Botteldooren [5] where the field components used were pressure and particle velocity. A finite-difference scheme using pressure-only grids instead of interleaved grids in the form of DWG was introduced by Savioja [37]. Bilbao formulated a group of schemes using a pressure-only grids [2] which were further refined by Kowalczyk [18] to a family of compact explicit FDTD schemes with a general formulation for the boundary conditions. The pressure-only formulation has then been shown to have functional equivalence to Yee's staggered grid approach [6]. The digital waveguide approach [44], which can be categorized as a finite-difference method, has been researched extensively in the field of sound synthesis.

There are several issues regarding the computational efficiency of different schemes which include the density of the grid points, the spectral response, the possibility to decompose a given scheme into more computationally efficient form, the operation count, the maximum value of the time step and the simplicity of the implementation of the boundary conditions [3, p. 303]. The

question, which scheme is useful and which is not for the problem at hand is an important aspect in finite-difference methods.

Finite difference schemes can be categorized according to several aspects. If the scheme uses only nodes which are directly adjacent to it, it is stated as a *compact* scheme. The number and direction of neighboring nodes which are used in the update equation is referred to as *stencil*. The update equations can be divided into *explicit* and *implicit*. In explicit schemes, the new value of the current element is derived from the previous values of the elements adjacent to it, or depending on the scheme, from the previous values of several non-adjacent elements. In implicit schemes, the equation used for the update of a given element require solving a system of linear equation at the new time step [18].

In this chapter the basic principles of the FDTD method for room acoustics are introduced and more detailed description of the family of compact, explicit schemes is given. The schemes considered in this thesis are limited to compact explicit schemes due to their applicability to parallel implementation. The compact explicit schemes are solved incrementally in time and their memory requirements for a single update are relatively modest due to the use of only the directly adjacent nodes.

3.1 Compact explicit FDTD schemes for room acoustics

In this section the general form of compact explicit FDTD schemes is reviewed following the formulation of Kowalczyk [18]. This general formulation of the compact, explicit schemes captures several existing schemes introduced in other studies such as standard rectilinear [47], the 3-D interpolated waveguide mesh [38] and the octahedral scheme [3, p. 317] with the use of several free parameters in the formulation.

For room acoustic simulation the partial differential equation of interest is the linear wave equation (2.5). A standard second order accurate difference scheme for the wave equation is given as [47, p. 158]:

$$\frac{\partial^2 p}{\partial^2 t} = \frac{p^{n+1}_{i,j,k} - 2p^{n}_{i,j,k} + p^{n-1}_{i,j,k}}{\Delta t^2} + O(\Delta t^2), \tag{3.1}$$

$$\frac{\partial^2 p}{\partial^2 x} = \frac{p^{n}_{i+1,j,k} - 2p^{n}_{i,j,k} + p^{n}_{i-1,j,k}}{\Delta x^2} + O(\Delta x^2), \tag{3.2}$$

$$\frac{\partial^2 p}{\partial^2 y} = \frac{p^{n}_{i,j+1,k} - 2p^{n}_{i,j,k} + p^{n}_{i,j-1,k}}{\Delta y^2} + O(\Delta y^2), \tag{3.3}$$

$$\frac{\partial^2 p}{\partial^2 z} = \frac{p^n_{i,j,k+1} - 2p^n_{i,j,k} + p^n_{i,j,k-1}}{\Delta z^2} + O(\Delta z^2), \tag{3.4}$$

where Δt is the chosen time step and Δx denotes the spacing of the grid. The subscrips i, j and k indicate the location indices in 3-D cartesian coordinates. The family of compact explicit schemes can be described with the use of centered finite difference operators,

$$\delta_t^2 p^n_{i,j,k} \equiv p^{n+1}_{i,j,k} - 2p^n_{i,j,k} + p^{n-1}_{i,j,k}, \tag{3.5}$$

$$\delta_x^2 p^n_{i,j,k} \equiv p^n_{i+1,j,k} - 2p^n_{i,j,k} + p^n_{i-1,j,k}, \tag{3.6}$$

$$\delta_y^2 p^n_{i,j,k} \equiv p^n_{i,j+1,k} - 2p^n_{i,j,k} + p^n_{i,j-1,k}, \tag{3.7}$$

$$\delta_z^2 p^n_{i,j,k} \equiv p^n_{i,j,k+1} - 2p^n_{i,j,k} + p^n_{i,j,k-1}, \tag{3.8}$$

in 2-D,

$$\delta_t^2 p^n_{i,j,k} = \lambda^2 \left[(\delta_x^2 + \delta_y^2) + b(\delta_x^2 \delta_y^2) \right] p^n_{i,j}, \tag{3.9}$$

and in 3-D,

$$\begin{aligned} \delta_t^2 p^n_{i,j,k} = \lambda^2 \big[& (\delta_x^2 + \delta_y^2 + \delta_z^2) \\ & + a(\delta_x^2 \delta_y^2 + \delta_y^2 \delta_z^2 + \delta_x^2 \delta_z^2) \\ & + b\delta_x^2 \delta_y^2 \delta_z^2 \, \big] p^n_{i,j,k}, \end{aligned} \tag{3.10}$$

with two free parameters a and b. λ denotes the Courant number, which defines the relationship between the spacing of the grid and the time discretization defined by the sampling frequency, and is given as

$$\lambda = \frac{c\Delta t}{\Delta x}, \tag{3.11}$$

where c is the speed of sound. The value of the Courant number has a range of values for stable time-stepping for each scheme, which are derived in [18] and not included here. By substituting the center difference operators into equations (3.9) and (3.10) the update equations for the explicit schemes are achieved in the form that they can be used in computational simulation. Update equations are given in 2-D by

$$\begin{aligned} p^{n+1}_{i,j,k} = & \, d_1(p^n_{i+1,j,} + p^n_{i-1,j,} + p^n_{i,j+1} + p_{i,j-1,}) \\ & + d_2(p^n_{i+1,j+1} + p^n_{i+1,j-1} + p^n_{i-1,j+1} + p^n_{i-1,j-1}) \\ & + d_3 p^n_{i,j} - p^{n-1}_{i,j}, \end{aligned} \tag{3.12}$$

and in 3-D,

$$\begin{aligned} p_{i,j,k}^{n+1} &= d_1 \,(\, p_{i+1,j,k}^n + p_{i-1,j,k}^n + p_{i,j+1,k}^n + p_{i,j-1,k}^n + p_{i,j,k+1}^n + p_{i,j,k-1}^n) \\ &+ d_2 \,(\, p_{i+1,j+1,k}^n + p_{i+1,j-1,k}^n + p_{i+1,j,k+1}^n + p_{i+1,j,k-1}^n \\ &+ p_{i,j+1,k+1}^n + p_{i,j+1,k-1}^n + p_{i,j-1,k+1}^n + p_{i-1,j,k-1}^n \,) \\ &+ d_3 \,(\, p_{i+1,j+1,k+1}^n + p_{i+1,j-1,k+1}^n + p_{i+1,j+1,k-1}^n + p_{i+1,j-1,k-1}^n \\ &+ p_{i-1,j+1,k+1}^n + p_{i-1,j-1,k+1}^n + p_{i-1,j+11,k-1}^n + p_{i-1,j-1,k-1}^n \,) \\ &+ d_4 p_{i,j,k}^n - p_{i,j,k}^{n-1}. \end{aligned} \tag{3.13}$$

The free parameters in equations (3.12) and (3.13) determine which of the scheme the equation reduces to. In the case of the *standard rectilinear* (SRL), in 2-D the parameters d_1, d_2, d_3 take values $\frac{1}{2}$, 0 and $2(1-2\lambda)$, respectively. In 3-D, d_1, d_2, d_3, and d_4 take values $\frac{1}{3}$, 0, 0, and $2(1-3\lambda)$, respectively. The update equations with these substitutions are

$$\begin{aligned} p_{i,j}^{n+1} &= \lambda^2 (p_{i+1,j}^n + p_{i-1,j}^n + p_{i,j+1}^n + p_{i,j-1}^n) \\ &+\ 2(1-2\lambda) p_{i,j}^n - p_{i,j}^{n-1}, \end{aligned} \tag{3.14}$$

and

$$\begin{aligned} p_{i,j,k}^{n+1} &= \lambda^2 (p_{i+1,j,k}^n + p_{i-1,j,k}^n + p_{i,j+1,k}^n + p_{i,j-1,k}^n + p_{i,j,k+1}^n + p_{i,j,k-1}^n) \\ &+\ 2(1-3\lambda) p_{i,j,k}^n - p_{i,j,k}^{n-1}. \end{aligned} \tag{3.15}$$

The update equations for *interpolated wideband scheme* (IWB) and several other schemes are achieved by substitutions. The parameters for different FDTD schemes are presented in [18, p. 72] and [18, p. 90]. The fundamental difference between different schemes is the number of adjacent nodes used in the update, namely the stencil of the scheme. The stencils used in SRL and IWB update are presented in Figure 3.1.

3.2 Boundary Conditions

In simulation of room acoustics, the boundaries are typically assumed to be locally reacting. By this it is meant that no waves propagate in the boundary itself. For a complete physical model the transmission of sound along the surfaces should be taken into account, but in this work it is ignored due to computational complexity and cost. In this section, a method of describing

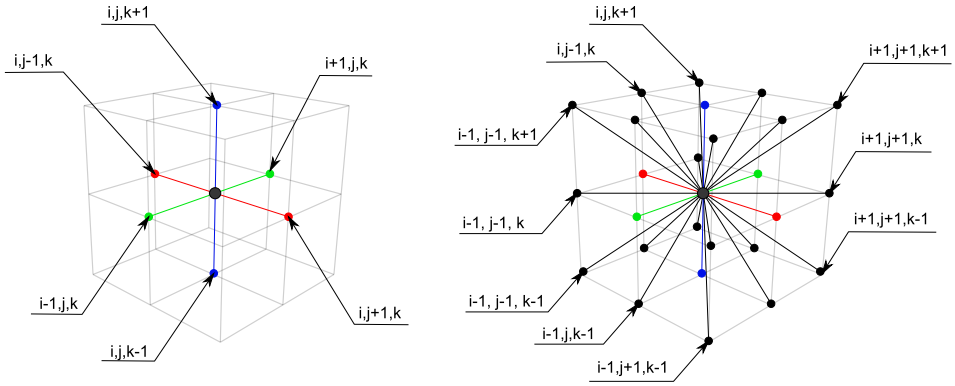

Figure 3.1: Stencils representing the nodes used in the update equation of a) SRL scheme and b) IWB scheme.

boundaries in the SRL FDTD scheme is reviewed as presented in [18]. For the sake of simplicity, the derivation of a 2-D wall and corner boundary are given. The same principle is used to formulate boundary conditions in 3-D.

In the case of 2-D SRL schemes, for a boundary node that has a wall on the right side, a single grid point is lying outside the modeled space. These points outside the geometry are referred to as *ghost points*. The ghost points of a 2-D SRL scheme are presented in Figure 3.2. The discrete boundary condition is combined with the update equation to eliminate the ghost points and solve the pressure value at the boundary node. The boundaries of rectilinear FDTD scheme can be derived by approximating the first-order derivatives of the continuous boundary conditions with centered operations. The continuous boundary conditions in 2-D are given

$$\frac{\partial p}{\partial t} = -c\,\xi\,\frac{\partial p}{\partial x}, \quad \frac{\partial p}{\partial t} = -c\,\xi\,\frac{\partial p}{\partial y}, \tag{3.16}$$

where c is the speed of sound and ξ the specific acoustic impedance. The centered difference operators for the first order derivatives of the boundary condition are given

$$\frac{\partial p}{\partial t} = \frac{p_{i,j}^{n+1} - p_{i,j}^{n-1}}{2\Delta t} + O(\Delta t^2), \tag{3.17}$$

$$\frac{\partial p}{\partial x} = \frac{p_{i+1,j}^{n} - p_{i-1,j}^{n}}{2\Delta x} + O(\Delta x^2), \tag{3.18}$$

$$\frac{\partial p}{\partial y} = \frac{p_{i,j+1}^{n} - p_{i,j-1}^{n}}{2\Delta y} + O(\Delta y^2). \tag{3.19}$$

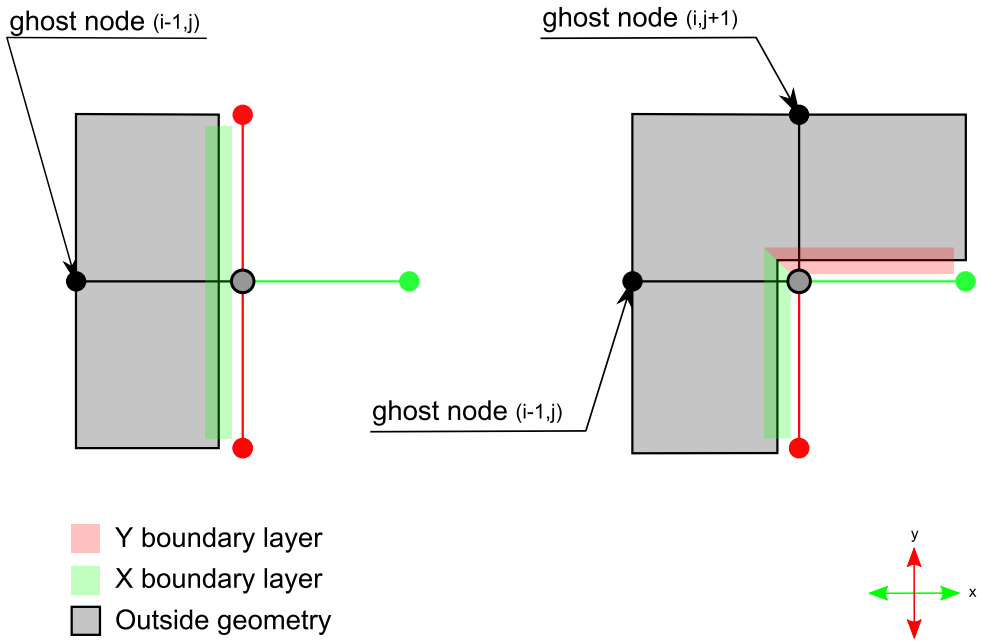

Figure 3.2: Ghost nodes in the case of 2-D boundaries. Each ghost node is substituted with a boundary condition in the update equation.

The equation for a point lying outside a wall on the side of the positive x-axis of the space can be now expressed with the difference operator and the specific acoustic impedance in the direction of x-axis:

$$p^n_{i+1,j} = p^n_{i-1,j} + \frac{1}{\lambda \xi_x}(p^{n-1}_{i,j} - p^{n+1}_{i,j}), \tag{3.20}$$

where $\lambda = \frac{c\Delta t}{\Delta x}$. Now by eliminating the ghost point from the update equation (3.14) an update equation for the boundary node is formulated as

$$\begin{aligned} p^{n+1}_{i,j} = \big[2(1-2\lambda^2) + \lambda^2(p^n_{i,j+1} + p^n_{i,j-1}) \\ +2\lambda^2 p^n_{i-1,j} + (\frac{\lambda}{\xi} - 1)p^{n-1}_{i,j}\big] / \frac{\lambda}{\xi}. \end{aligned} \tag{3.21}$$

In the case of a corner node, both boundary conditions for the x- and y-dimension must be met simultaneously, so both of the ghost points lying outside the space normal to x- and y-axes must be eliminated with the boundary conditions in their respective directions. The equation for the point outside

the space in the direction of the x-axis is given in (3.20), and in a similar manner the equation for the point lying outside the space in the direction of the y-axis is

$$p^n_{i,j+1} = p^n_{i,j-1} + \frac{1}{\lambda\xi_y}(p^{n-1}_{i,j} - p^{n+1}_{i,j}), \tag{3.22}$$

where ξ_y is the specific acoustic impedance of the boundary normal to y-axis. The update equation for the corner node is derived from the 2-D update equation (3.14) with substituted boundary conditiond, and is given by

$$\begin{aligned} p^{n+1}_{i,j} = \big[&2(1-2\lambda^2)p^n_{i,j} + 2\lambda^2(p^n_{i-1,j} + p^n_{i,j-1}) \\ &+(\frac{\lambda}{\xi_x} + \frac{\lambda}{\xi_y} - 1)p^{n-1}_{i,j}\Big] /(1 + \frac{\lambda}{\xi_x} + \frac{\lambda}{\xi_y}). \end{aligned} \tag{3.23}$$

Inner corners do not have ghost points to eliminate, so the node can be updated with the 2-D update equation (3.14). The specific acoustic impedance at the boundary can be expressed with the reflection coefficient data by rearranging the equation (2.21)

$$\xi_w = \frac{1+R}{1-R}. \tag{3.24}$$

3.2.1 Boundary definition with forward difference operator

One formulation of the update equation, which is specifically useful for parallelization, uses a different method to derive the boundary condition. The method that was introduced by Webb and Bilbao [52] presumably defines the discretized boundary condition with a forward difference instead of centered difference as in equation (3.18). The referred paper does not show the derivation of the boundary condition, but the resulting update equation is similar to update equation presented here. The forward difference operation is given by

$$\frac{\partial p}{\partial x} = \frac{p^n_{i+1} - p^n_i}{\Delta x} + O(\Delta x^2). \tag{3.25}$$

Using the centered difference formulation (3.17) of the partial derivative in time domain and the forward difference formulation in the spatial domain, the continuous boundary condition (3.16) can be expressed with

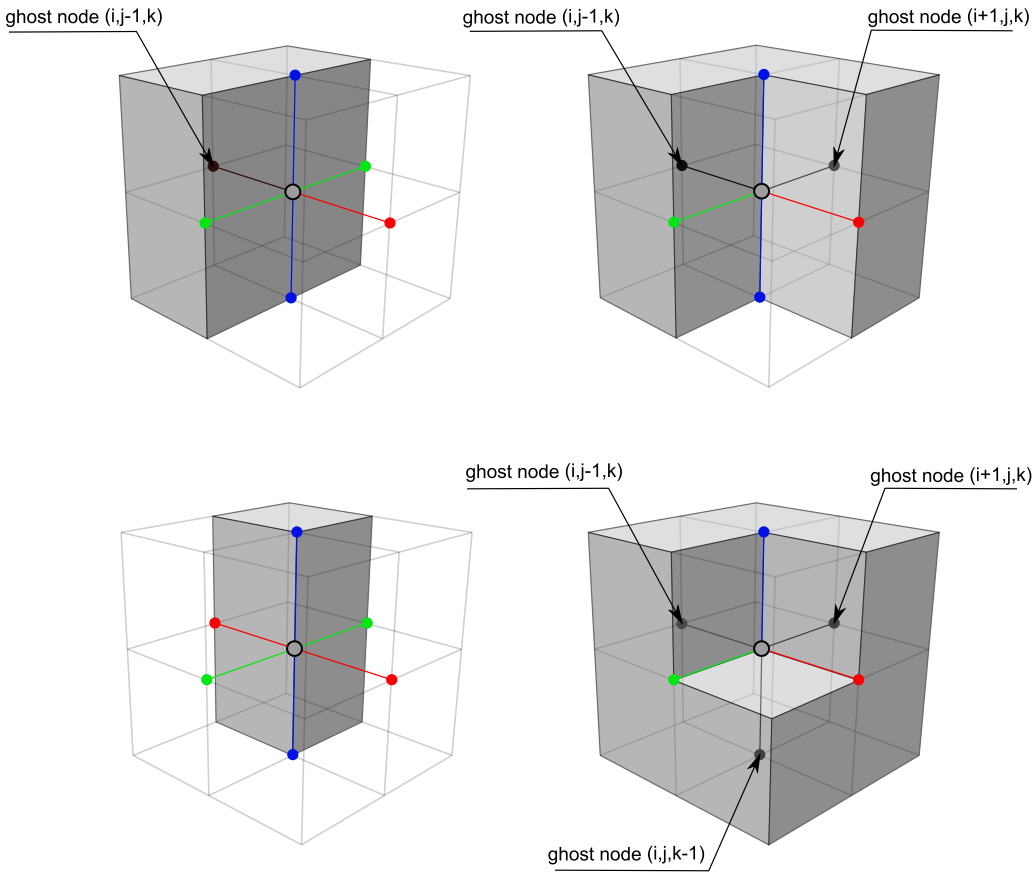

Figure 3.3: Ghost nodes in the case of 3-D boundaries. Each ghost node is substituted with a boundary condition in the update equation. It is notable that inner corners do not have boundary conditions introduced in the update equation in the SRL scheme.

$$\frac{p_{i,j}^{n+1} - p_{i,j}^{n-1}}{2\Delta t} = -c\,\xi\,\frac{p_{i+1,j}^{n} - p_{i,j}^{n}}{\Delta x}. \tag{3.26}$$

The ghost point $p_{i+1,j}^{n}$ takes the form

$$p_{i+1,j}^{n} = p_{i,j}^{n} + \frac{1}{2\lambda\xi}\,(p_{i,j}^{n-1} - p_{i,j}^{n+1}). \tag{3.27}$$

By substituting the ghost node into the 2-D SRL update equation (3.14), the

following update equation for a wall boundary in 2-D is achieved:

$$\begin{aligned} p_{i,j}^{n+1} = \frac{1}{1+\lambda\,\beta}[\,&(2-3\lambda^2)p_{i,j}^{n} + (\lambda\,\beta-1)p_{i,j}^{n-1} \\ &+ \lambda^2(p_{i-1,j,}^{n} + p_{i,j-1}^{n} + p_{i,j+1}^{n})\,], \end{aligned} \tag{3.28}$$

where $\beta = (\frac{1}{2\xi})$ and ξ is the specific acoustic impedance of the boundary. It can be noticed from the boundary condition (3.27) that the value which is introduced to the equation by the boundary condition is the current pressure node $p_{i,j}^{n}$ which leads to the fact that no additional direction dependent memory fetches must be made in the computation.

To include this position information in the equation, a variable K is introduced. Variable K defines how many ghost nodes are present in the current node. K is formulated as $dimensions \times 2 - number\ of\ ghost\ nodes$. For example, a wall node in 2-D is evaluated as: $K = 2 \times 2 - 1 = 3$, and a corner node in 3-D $K = 2 \times 3 - 3 = 3$. To ignore the ghost nodes from the update equation, the pressure values outside the geometry are assigned to zero during each update. By deriving the update equations for different node positions a general solution for node update can be presented in 2-D as

$$\begin{aligned} p_{i,j}^{n+1} = \frac{1}{1+\lambda\,\beta_{2d}}[\,&(2-K\lambda^2)p_{i,j}^{n} + (\lambda\,\beta_{2d}-1)p_{i,j}^{n-1} \\ &+ \lambda^2(p_{i+1,j}^{n} + p_{i-1,j}^{n} + p_{i,j-1}^{n} + p_{i,j+1}^{n})\,], \end{aligned} \tag{3.29}$$

where $\beta_{2d} = (\frac{4-K}{2\xi})$ and in 3-D

$$\begin{aligned} p_{i,j}^{n+1} = \frac{1}{1+\lambda\,\beta_{3d}}[\,&(2-K\lambda^2)p_{i,j,k}^{n} + (\lambda\,\beta_{3d}-1)p_{i,j,k}^{n-1} \\ &+ \lambda^2(p_{i-1,j,k}^{n} + p_{i,j-1,k}^{n} + p_{i,j+1,k}^{n} + p_{i,j,k-1}^{n} + p_{i,j,k+1}^{n})\,], \end{aligned} \tag{3.30}$$

where $\beta_{3d} = (\frac{6-K}{2\xi})$ and ξ is the specific acoustic impedance of the boundary node.

3.3 Dispersion Error

The phenomenon of the waves of different frequencies traveling with different speeds is called *dispersion* [47, p. 103]. In the air, the speed of sound is constant for all frequencies and in all propagation directions. When the acoustic system is discretized, the velocity of the traveling wave of a given frequency called *phase speed* differs from the theoretical phase velocity. It is

common for all FDTD schemes that the numerical phase velocity approaches the theoretical phase velocity at low frequencies, but the wave speed at high frequencies is lower than the theoretical one. The error is direction dependent and varies between different schemes. The dispersion errors of SRL and IWB schemes in 2-D domain are presented as a function of direction and frequency in Figure 3.4.

It can be noticed that the wave speed error in SRL scheme is the highest in the axial direction whereas in IWB scheme the error is the highest in the diagonal direction. The band limit up to which SRL scheme can estimate the propagation of sound with a dispersion error lower than 2 % is approximately $0.1f_s$ whereas with the IWB the band limit is approximately $0.22f_s$ where f_s denotes the spatial sampling frequency.

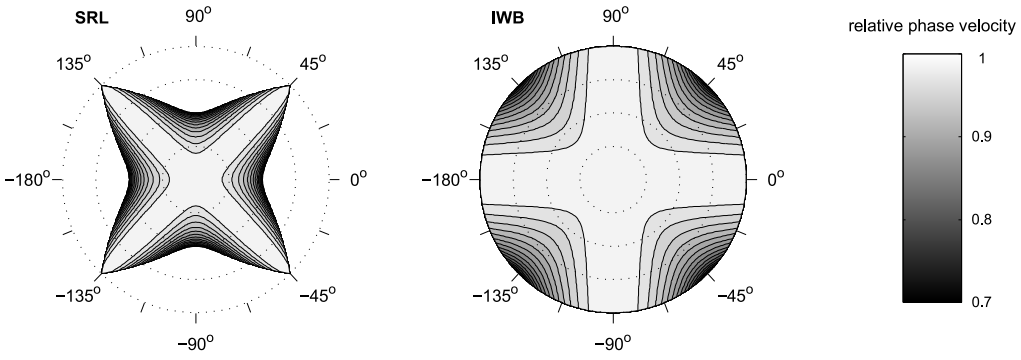

Figure 3.4: Relative phase velocity as a function of frequency from [18, p. 82]. The radius of the polar plot indicates the frequency and the angle the propagation direction. Each dotted circle indicates a frequency $f = (\frac{1}{8}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2})f_s$ from the innermost circle to the outermost respectively. The variable f_s denotes the spatial sampling frequency.

3.4 Source modeling

In acoustic simulation the source energy is usually embedded within the FDTD grid. The most common way to excite the mesh is to use one or several pointwise sources. The types of single pointwise source found in literature can be divided into three categories: *hard*, *soft* and *transparent*, where transparent can be considered a special case of a soft source.

A hard source is set up by simply assigning a value of a desired time function f^n to a specific element in the mesh. The index n denotes the current time step. The function can be written as $f^n = f(n\Delta t)$, where Δt is the length of the time step. The problem in a such source definition is that the direct assignment of the element value overrides the update equation, and such a source will therefore effectively scatter any incident field. If the location of the source and the number of time-steps is such that no incident fields will reach the source position during the simulation, a hard source can be efficiently used. A hard source is assigned to the mesh with equation:

$$p^n_{src} = f^n, \tag{3.31}$$

where f^n is the source function, and p^n_{src} the pressure node where the source is positioned. A soft source is set up by adding the value of the desired time function to the pressure value of the source position in the mesh. This method does implement a source which does not introduce scattering, but the problem which arises is that the actual excitation does not match the time function f^n [39]. A soft source is assigned to the mesh with equation:

$$p^n_{src} = p^n_{mesh} + f^n. \tag{3.32}$$

To overcome the limitation of the scattering of the hard source and the deviation from the source function of soft source a transparent source was introduced by Schneider [39]. A transparent source is defined as a source which radiates the same field as a hard source but does not act as a scatterer. The simplest way to implement a transparent source is to record the impulse response of the mesh with a single simulation using a hard source. This is achieved by using an impulse-like source function where $f^1 = 1, f^n = 0 \; when \; n \neq 1$ and recording the response at the source node position. A transparent source sample is created by convolving the source function with the impulse response of the mesh. The source update of a transparent source is formulated with the impulse response and the time function of the source:

$$p^{n+1}_{src} = p^{n+1}_{mesh} + f^{n+1} - \sum_{m=0}^{n} h^{n-m+1} f^m. \tag{3.33}$$

The first source update is done in with the soft source update and therefore the step index n is incremented in the equation.

3.5 Medium Viscosity

The propagation of sound in the air introduces frequency dependent viscous losses to the signal transmitted. The effect has been compensated in FDTD simulation with different approaches.

Webb and Bilbao suggest the usage of a form of the wave equation which takes it into account. The equation is given as

$$\frac{\partial^2 p}{\partial^2 t^2} = c^2 \nabla^2 p + c\alpha \nabla^2 \frac{\partial p}{\partial t}, \tag{3.34}$$

where α is derived from the linearized Navier Stokes equations [52]. The downside of such formulation is that in order to calculate the Laplacian of the time derivative, the knowledge of the past and current pressure value in each neighboring node must be present at the same time, which leads to a need for additional data storage.

Another approach is to run the simulation assuming lossless medium and post process the simulated response with a specific air absorption filter. The method used by Southern et al. [46] uses an overlap-add convolution to apply time varying filtering to the simulated response. This method is extremely efficient when calculating room impulse responses, but in case of continuous simulation such method is not applicable.

Chapter 4

CUDA architecture

General purpose graphics processing unit (GPGPU) programming has become popular in a variety of fields in science. *Compute unified device architecture* (CUDA) is one of the first device architectures designed for GPGPU programming. The system developed as a part of this thesis is implemented usign GPGPU programming with CUDA. Therefore an introduction to parallel programming and the device architecture is necessary for the reader in order to understand the motivation for this work and the considerations concerning the implementation. The organization of the chapter is as follows: first, the progression from sequential *central process unit* (CPU) computing to large scale GPU computing is discussed; then, the general architecture of CUDA capable computing devices is specified; and last, the programming interfaces of CUDA are described in more detail.

4.1 From CPU to GPU

For the past several decades microprocessor manufacturers have introduced performance improvements and cost reduction on single CPU microprocessors. This drive had led most developers to rely on the advances in hardware to increase the speed of applications, as Kirk states: "..The same software simply runs faster as each new generation of processors is introduced"[17, p. 1]. As the physical limits such as heat-dissipation and energy consumption have become evident to single microprocessors, practically all microprocessor manufacturers have switched to models with several processor cores on each chip. This development has led to the change in programming practices since traditional sequential programming does not introduce performance improvements. Application software that uses parallel programming is able to take advantage of each new processor generation. This new incentive for parallel

programming has been referred to as the *concurrency revolution* [48].

Microprocessor design has led to two distinguished trajectories. The *multicore* trajectory is putting effort to maximize the speed of sequential programs with several cores. The *many-core* trajectory focuses on the execution throughput of a parallel program, which is achieved with a large number of much smaller cores compared to the multicore trajectory. A group of devices which is currently following the multi-core design trajectory are GPUs. The fast progress of the computational performance of GPUs has been influenced by the demand from the video game industry [17]. GPUs are utilizing hundreds of small cores in highly parallel manner that allows them to achieve a large throughput of floating-point operation. What should be noted is that they do not perform well in tasks which can not be formulated in parallel manner. Therefore it is practical to use both CPUs and GPUs utilizing the GPU in the computationally intense parts. One of the programming platforms allowing such a joint execution of applications is CUDA, which was released in 2007 by Nvidia. With CUDA and other GPGPU programming platforms introduced later, it is possible to significantly increase the performance of applications that have distinct numerically intensive and data parallel parts, such as the explicit FDTD update.

4.2 Device

CUDA capable devices have gone through several device generations. Device generations are prescribed with a specifier called *compute capability*, which consists of major and minor revision numbers. The major revision number ranges from 1 to 3 and defines the core architecture of the device, 1 being the oldest architecture called *Tesla*, 2 the following called *Fermi* and 3 the most recent, *Kepler*. The minor version number corresponds to incremental improvements on the core architecture. Each of the device generations has similar functionalities and they can all be programmed with the same *application programming interfaces* (APIs). The differences between device generations are mainly the number of floating point operations that the device can perform per time unit, the size of the memory available on the device, memory bandwidth which the device can achieve between the registers of the processors, and the device memory and the caching mechanisms.

In this section, the general architecture of a CUDA devices is reviewed in the following order: first, the processor architecture and threading mechanism is reviewed; second, the types of memory are introduced; and third, the architecture which is used in the development of the system is reviewed in more detail.

4.2.1 Streaming multiprocessors and thread hierarchy

Typical CUDA-capable GPU is build around a scalable array of highly threaded *streaming multiprocessors* (SMs). Streaming multiprocessors contain a number of *streaming processors* (SPs) that share control logic and instruction cache. A single SM is designed to execute hundreds of threads concurrently. The architecture of a single SP is called *single instruction multiple data* (SIMD). This means that the SPs, which a single SM controls, run the same instruction, but each SP uses different data. The GPGPU device as a whole is not classified as SIMD machine because it consists of multiple SMs which can all run different instructions on SPs they control. Therefore the device itself is classified as *single instruction multiple thread* (SIMT) [10]. The number of cores in each SM varies between device generations.

A CUDA program which is launched from the host CPU is called a *kernel*. A kernel defines the code which is to be executed by all threads assigned for the task. A kernel invokes a *grid* of threads which is further divided into *blocks*. Threads of a single thread block are executed on one SM and the blocks are scheduled between the SMs. As a block terminates, a new block will be launched on the specific SM.

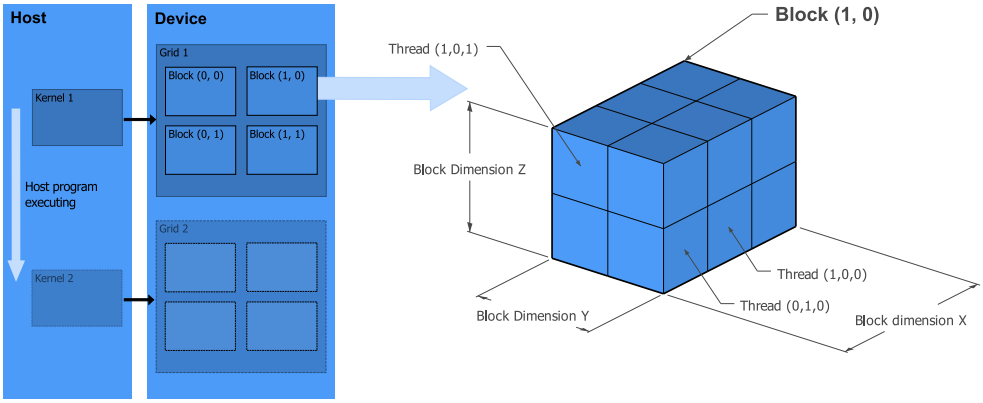

Figure 4.1: CUDA grid organization. Kernels are launched from the host code. Each kernel invokes a grid of threads which is further subdivided into blocks. Each block contains threads which are organized either into 1, 2, or 3 dimensions.

Each thread of a grid executing the same kernel function has unique thread coordinates. Threads are organized into a two-level hierarchy using

coordinates for blocks (blockIdx) and threads (threadIdx). Both blockIdx and threadIdx can be either one, two, or three dimensional. Organization of threads into multiple dimensions helps assigning threads to multidimensional data. The thread hierarchy is presented in Figure 4.1. The number of threads per block is limited because all threads of a block share the limited memory of the SM executing it.

SMs create, manage, schedule and execute threads in groups of 32 parallel threads. These groups are called *warps*. Each thread block is partitioned into warps and each warp is scheduled by a *warp scheduler*. Threads of a single warp execute one common instruction at a time. SM which has more than one warp scheduler [28] can issue and execute as many warps as there are schedulers concurrently. In the case of conditional branching within a warp, each branch is serially executed. When each branch is completed, the threads converge back to the same execution path. As a warp is the smallest partition of threads, it is beneficial to use a block size which is a multiple of warp size. If the block size is not a multiple of the warp size, the reminder will be padded with idle threads.

Table 4.1: Differences between computing capabilities of CUDA devices.

Compute Capability	1.x	2.0	2.1	3.0	3.5
Number of cores per SM	8	32	48	192	192
Maximum dimensionality of grid of thread blocks	2	3	3	3	3
Maximum number of threads per block	512	1024	1024	1024	1024
Number of warp schedulers	1	2	2	4	4
Maximum number of blocks per grid dimension	$2^{16}-1$	$2^{16}-1$	$2^{16}-1$	$2^{16}-1$	$2^{16}-1$

From the table 4.1 it can be observed that the most significant difference between device generation is the number of cores per SM. The number of cores translates to the throughput of floating point operations. The number of SMs per device varies between different devices of the same device generation.

4.2.2 Memory

CUDA devices have several different types of memories. Memory types can be roughly divided into three categories: off-chip memory which is the largest of the memory areas, on-chip memories which reside in the SMs include registers

and shared memories and cache memories which are utilized depending on the declaration of variables used in the code.

Off-chip memory, is a *graphics double data rate* (GDDR) memory chip that ranges in size from 128 MB to 6 GB depending on the device. This memory area is accessible from both the host and device code and is used to store most of the data used in the processing due to its capacity. The off-chip memory can be declared as global, or constant in the program code. Different declaration of the memory affects the caching mechanisms used for the allocated variables.

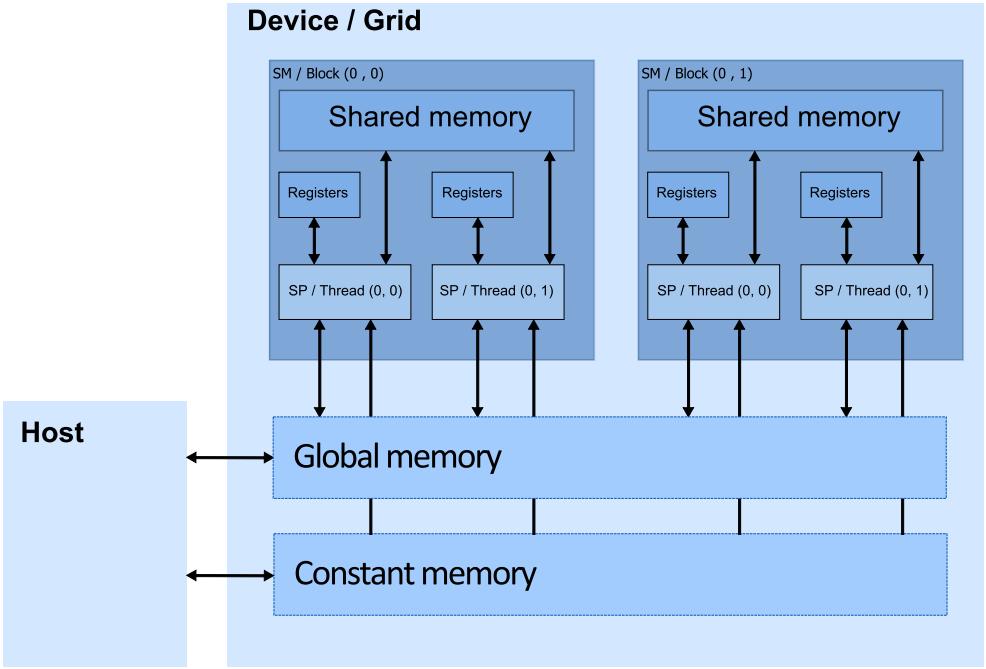

Figure 4.2: CUDA Memory organization. Global and constant memories reside in the off-chip GDDR memory. Shared memory and registers are located in the SMs. The thread unit processed by each part of the device is noted in the figure. Figure adopted from [17].

On-chip memories are memories that reside in the SMs of the device. On-chip memory includes shared memory and registers. Shared memory can be accessed by all threads of a single block. Registers of a SM are assigned for each thread separately. The bandwidths of these memories are fast compared to the off-chip memory. Volkov [51] states, that in most cases the access to per thread registers is much faster than to shared memory. The memory organization of a typical CUDA device is presented in Figure

4.2. The relationship between the device and the thread organization is presented with the notion of the thread unit which is processed by each part of the device.

Cache memories are small memory units that are used to store recently loaded and stored variables to increase the speed of consecutive memory accesses. Cache memories which appear in all device generations are texture and constant caches. The texture cache is utilized when a kernel is fetching data from a texture object that is allocated in the global memory. The memory area which is allocated for texture objects is commonly referred to as *texture memory*. The constant cache is utilized for variables which are allocated in global memory and declared as constant. Additional L1 and L2 cache memories were introduced with the Fermi architecture [28]. These two memories form a two-level cache hierarchy. L2 is a large cache which is accessible by all SMs. L1 caches are located in each SM. A specific read-only cache was introduced with the Kepler architecture [31] that uses the caching mechanism of the texture cache without the need of declaring a specific texture object.

4.2.3 Kepler Architecture

The device used in the development and benchamarking of the system was Nvidia Geforce GTX 690. The device has a Kepler architecture and its the computing capability is 3.0. The different components of the Kepler architecture are illustrated in Figure 4.3. The device contains two independent GPUs which can be used concurrently. The specification for the device is presented in table 4.2.

The Kepler architecture has several technical differences in comparison to older device generations. The number of CUDA cores per SM is 192, which translates to the number of floating point operations the device can perform per second. The number of warp schedulers is increased to four which allows the device to assign four warps for each multiprocessor to be executed concurrently [31]. As mentioned in the previous section, the Kepler architecture has a two level cache hierarchy, which consists of 1,536 KB L2 cache and configurable L1 cache which ranges from 16 to 48 KB depending on the configuration. In addition, a 48 KB read-only cache is introduced into the architecture. The variables with which the read-only cache is used is controlled in the code with a `const __restrict` keyword.

The data transfer from the host to the off-chip memory is bounded by the PCI express interface. The memory clock rate and the bus width define the memory bandwidth from the off-chip memory to the SMs. The speed of the PCI express version 3.0 [29] is 15.75 GB/s. The theoretical memory

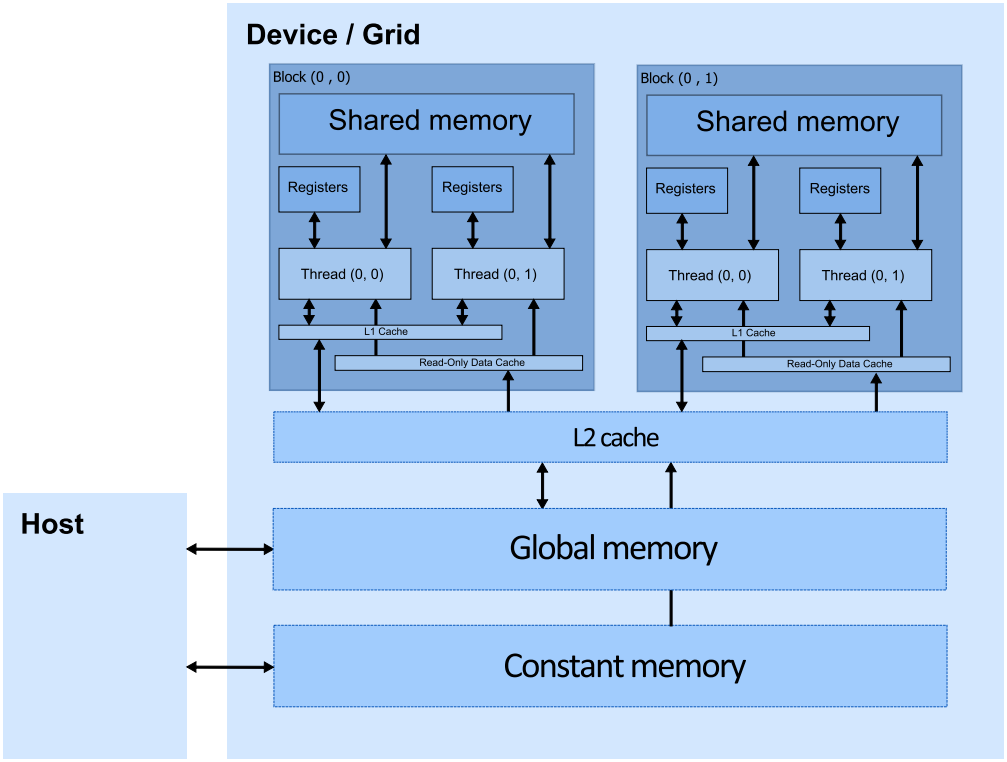

Figure 4.3: Overview of the Keppler architecture.

Table 4.2: Device spesification for the Nvidia Geforce GTX 690.

Memory GB	Core clock (MHz)	Memory clock (MHz)	Memory bandwidth (bit)	SM count	Registers / SM
2 x 2048	915	6008	256	2x8	2^{16}

bandwidth from the off-chip memory to the SMs can be determined with the following formula:

$$bandwidth(bytes/s) = memory\,clock(Hz) \times bus\ width\,(bytes) \times 2. \quad (4.1)$$

In the case of the GTX 690, by substituting the values from table 4.2 the equation (4.1) a theoretical memory bandwidth of 384 GB/s per device can be determined. The multiplication of 2 is added because of the double data rate of the memory. The number of cores of each device can be again derived from values introduced in tables 4.1 and 4.2 which leads to a total number of 1536 cores per device. The theoretical maximum of *floating point operations per second* (FLOPS) that the device can achieve can be calculated with the

formula:

$$peak(FLOPS) = core\,clock(Hz) \times number\,of\,cores \times 2\,(operations), \quad (4.2)$$

which in case of the GTX 690 leads to 2,810 $GFLOPS$ per device. It is notable that the peak FLOPS of the device is approximately 7 times larger than the memory bandwidth of the off-chip memory and 178 times larger than the memory bandwidth of the PCI express interface. Therefore in order to achieve optimal perfromance, a minimization of memory transfers between the host and the device should be conducted. The method to minimize the memory traffic between the off-chip memory and the SMs is a question which should be assessed separately for each device generation.

4.3 Programming

As it has been reviewed in the past sections, CUDA devices have distinct device architecture and threading model. The programming of CUDA capable device can be done with several different APIs. The level of control that can be achieved on the device depends much on the chosen API. In this section, the programming model of CUDA is reviewed as shown in Figure 4.4. First a brief description of the *instruction set architecture* (ISA) is given. Then the programming interfaces of different levels are reviewed.

4.3.1 PTX instruction set architecture

Instruction set architecture (ISA) is the part of hardware which is visible to the programmer or compiler writer. ISA serves as a boundary between the software and the hardware. In general, ISA defines the native data types, instructions, registers, addressing modes, memory architecture, and interruption and exception handling. In the case of CUDA there are several different devices which have different capabilities. In such cases, the compilation would have to be targeted to the machine in use in order to produce working hardware instructions. This can be avoided with the use of *parallel thread execution* virtual machine (PTX) [32]. When a CUDA program is compiled it is targeted to PTX instructions instead of hardware instructions. The PTX instructions are translated to native device code in run time. This method is referred to as *just-in-time* compilation. It increases the application load time, but allows the code to benefit from new compiler improvements, updated device drivers and it makes possible for the code to work on devices which did not exist when the application was compiled [30].

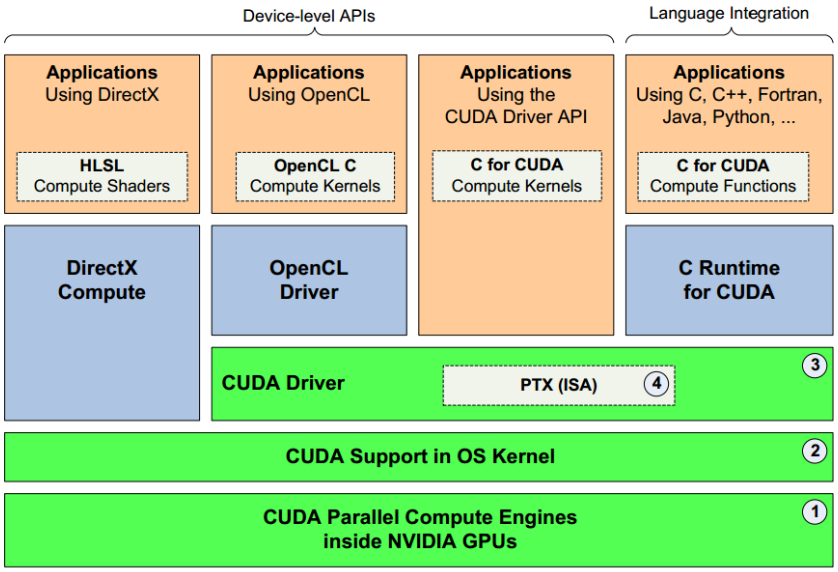

Figure 4.4: Components of a CUDA architecture: 1: the parallel computing engines inside the GPU, namely the hardware, 2. OS kernel-level support for hardware initialization and configuration, 3. User-mode driver, which provides a device-level API for developers and 4. Parallel Thread Execution (PTX) instruction set architecture (ISA) which is translated for each device during runtime [27].

4.3.2 Programming interfaces

The programming interfaces of CUDA are divided into two categories: Device-level interfaces and language integration interfaces. The fundamental difference between these two is that with the device-level interfaces the actual configuration of the GPU device is made by the developer and with the language integration interface the configuration is handled by the CUDA automatically. Which API to use depends solely on how much control the developer wishes to have over the GPU and in which language the host application is written. First an overview of Device-level interfaces including DirectX compute, OpenCL and CUDA Driver API is given. Then the language integration programming interface is described.

Device-level programming interface

Device-level programming interfaces include DirectX compute, Open CL and CUDA driver API. DirectX compute is Microsoft's API for GPGPU program-

ming for applications using DirectX. DirextX is a graphics programming API for Windows platform. In DirectX compute programs are written in *high-level shader language* (HLSL). A shader is a similar concept as the kernel is in CUDA, a function which is run with a group of threads in parallel. As can be seen from Figure 4.4, DirectX does not use the CUDA driver, so it is solely working on top of the *operating system* (OS). DirectX compute was not considered as an option for this work because it is OS specific.

OpenCL (*Open Computing Language*) is an open standard for cross-platform parallel programming of GPGPUs. Open CL makes it possible to use other GPGPUs than the devices of NVIDA. The programs are written in *OpenCL C*, which has similar syntax to C. Several performance comparison of Open CL and CUDA have been conducted [9][13] concluding that CUDA has slightly better performance for the cost that it can only be utilized with Nvidias devices.

CUDA driver API is a low-level C programming interface for CUDA capable devices. Driver API gives the programmer more control over the device on the cost that the developer must take care of more complex device functionalities such as CUDA *contexts*, which are analogous to host processes, and *modules* which are analogous to dynamically loaded libraries on the host [30]. Such control might be necessary with a multi-threaded host application using several device contexts per device. The driver API was not utilized in this work due to the off-line nature of the solver.

Language integration programming interface

Language integration programming interfaces are namely abstractions of the device-level API. A CUDA native language integration programming interface is the CUDA Runtime API that is built on top of the CUDA driver API. The runtime API automatically creates a CUDA context for each device in the system and this context is shared by all of the host threads of the application. The API provides a core set of functionalities to allocate and transfer memory between the host and the devices, select devices, and run computing kernels. The runtime API was utilized in this work due to its functional convenience. There are other language integration programming interfaces that interface different programming languages such as Fortran, Python or Java with CUDA. These APIs were not considered because there was no existing code which would have needed to be integrated into the system.

Chapter 5

Implementation

The aim of this thesis is to develop an efficient parallel implementation of a FDTD solver for room acoustics. The method has already been used in several fields of science, and it has been found applictions for room acoustics especially with the use parallel processing with GPUs. In this chapter the developed system is described. Considerations concerning the parallel implementation of FDTD have been studied in numerous publications, so first an overview of previous implementations is given. Second, the overall architecture of the developed system is presented following with the description of several more detailed implementation considerations.

5.1 Previous Work

Several different implementations of parallel room acoustic FDTD solvers have been suggested. All of the published implementations for the use of room acoustic simulation are developed using CUDA.

A real-time implementation for low frequency simulation was introduced by Savioja [36]. Savioja's solution uses Kowalczyk's general formulation of compact explicit schemes and extends the functionality to frequency dependent boundaries. Southern et al. [45] describes an implementation of a 2-D solver with special attention on the handling of the boundary nodes of the mesh by dividing the mesh into distinguished tile types.

Webb and Bilbao have suggested a compact explicit scheme which is particularly suited for parallel computation by using a forward difference boundary definition [52]. Webb extends the functionality of the previously suggested solver for multiple GPUs [53] and reports major improvements on the possible mesh sizes and solver speed. Webb also reviews different optimization paradigms for CUDA kernels with the most recent device ar-

chitectures [54] Lopez et al. overview several different methods to optimize FDTD simulation on two different device architectures [21]. Sheaffer reviews basic programming paradigms concerning the parallelization of FDTD [42] and recently released an open source FDTD solver for room acoustics [41].

Suggested methods

Two solutions to efficiently process a 3-D volume have been suggested: the tiling method and the slicing method which are both benchmarked by different authors [21][42][54]. In the tiling method, the whole mesh is represented as a large 2-D tile where the 3-D location of each node is calculated either with the use of modulo operations when using 2-D kernel grids, or from the thread indices when using a 3-D grid. The method allows to process the whole mesh in such a way that each thread operates on one element at a time. The method is presented in Figure 5.1. The downside of this method is that the modulo operation is computationally expensive and that memory reuse with the help of shared memory is not possible. A method of avoiding modulo operations has been presented by Scheaffer [42] which is reported to increase the efficiency substantially. The modulo operations are not needed in the case that the dimensions of the mesh are a multiple of the block size used in the kernels.

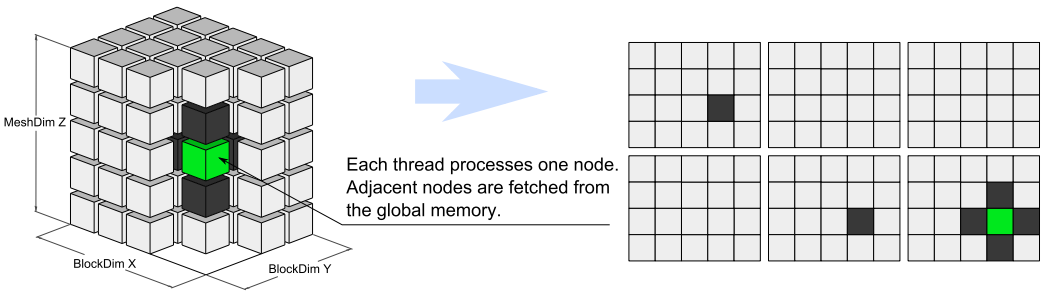

Figure 5.1: The tiled method for processing a 3-D mesh. Each node is processed by a single thread. Each neighboring node of the node being updated is fetched from the linear memory independently. No conditional statements have to be applied.

In the slicing method, each thread does not operate only on one element. The mesh is divided into slices and each thread goes through all the elements of single positions in all slices. The slicing method allows efficient memory reuse since each slice is used in three occasions during the progress of the

simulation. The downside of the method is that each thread block requires data from the memory which the thread block adjacent to it is processing. Therefore conditional statements have to be used to control the memory loads at the thread block boundary. The method is presented in Figure 5.2.

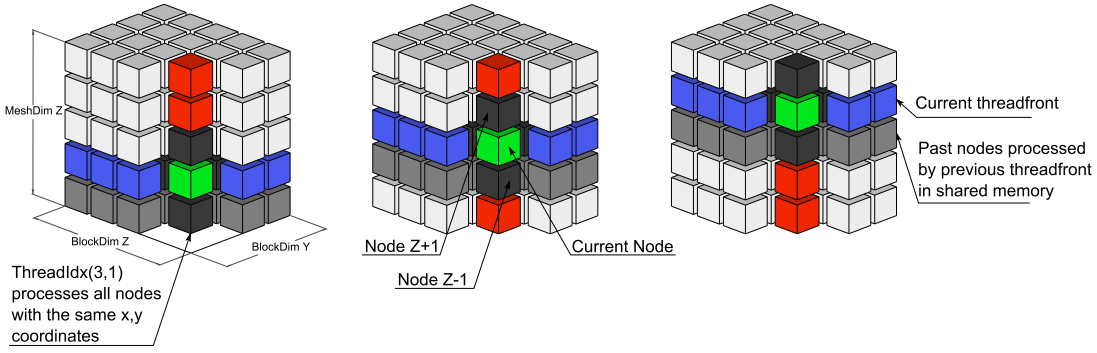

Figure 5.2: The slicing method for processing a 3-D mesh. Two shared memory areas with the size of the thread block are allocated. Each thread loads the node above the current node (z+1) to the register. After the update the z+1 slice is transferred from the registers to the z slice in the shared memory, and the z slice from the shared memory is transferred to z-1 slice in the shared memory.

Lopez et al. [21] utilizes the texture memory of CUDA. Texture memory allows the use of the texture cache that allows efficient caching of memory which is located spatially in close proximity. It is concluded that the usage of the texture memory does not increase performance over the sliced method. Webb [54] discusses different memory optimization techniques and their applicability to both Fermi and Kepler architectures. It is concluded that the two-level cache increases the performance of a kernel using solely global memory fetches to such level that it is only marginally slower than the kernel using shared memory. The slicing method used by Webb differs slightly from the one used in this thesis and it is reported to have better performance opposed to the method used in this system.

There has been much research done on the parallel execution of FDTD simulation in room acoustics. Most of the suggested programming paradigms have been independently benchmarked by several authors. It is important to note that some of the benchmarks have been done with rigid boundaries [21] [54] which decrease the amount of memory traffic during the simulation and hence the performance metrics may not be comparable in all cases.

5.2 System Architecture

The system can be divided into five distinct sections: data input, voxelization, memory partition, FDTD solver and data output. The system includes visualization and capturing functionalities which use a different execution path where several conditional statements and functions are done between every solver step. The flow diagram of the system is presented in Figure 5.3.

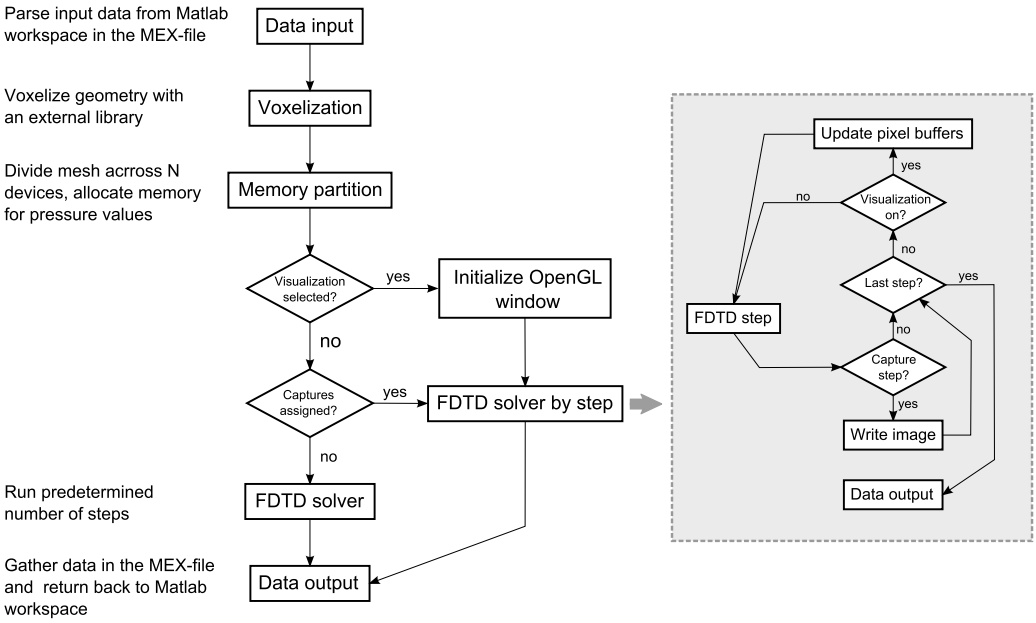

Figure 5.3: Overview of the system architecture.

Data input is handled through Matlab with the use of a MEX function. The input arguments of the MEX function include vertex coordinates, triangle indices, material parameters, visualization selection, update type selection, source and receiver definitions. Separate functions are used to load and validate the geometry in Matlab.

Voxelization of the surface model is done with an external library which is reviewed in section 5.3. The voxelizer outputs a rectangular mesh of nodes which contains the orientation information and material indices of each element.

The voxelized geometry is then inserted into a data structure which handles the allocation and partition of material and position indices, pressure meshes, source data and simulation parameters across the devices selected to use.

The actual FDTD update is done with a launch function which first assigns pressure values to each source position, then launches the FDTD update kernel for each device in use, and last, collect pressure values from the receiver positions. Output data is passed back to the Matlab workspace through the MEX-function which was used to start the simulation.

5.3 Voxelization

Voxelization of the surface geometry is done with an external GPU voxelization library implemented by Karlsson [14]. The implementation is based on a tile-based parallel solid voxelization algorithm [40]. The base of the voxelization is a rectangular grid of voxels which size is determined by the spatial sampling frequency of the simulation. The grid is divided into 4x4 voxel columns in a chosen coordinate plane. Each triangle is assigned to tiles it overlaps, yielding a work queue of tile/triangle pairs. The tiles are then processed in parallel with one warp processing each tile and its associated triangles. Considerations in the performance of the library are reviewed in more detail by Karlsson [14].

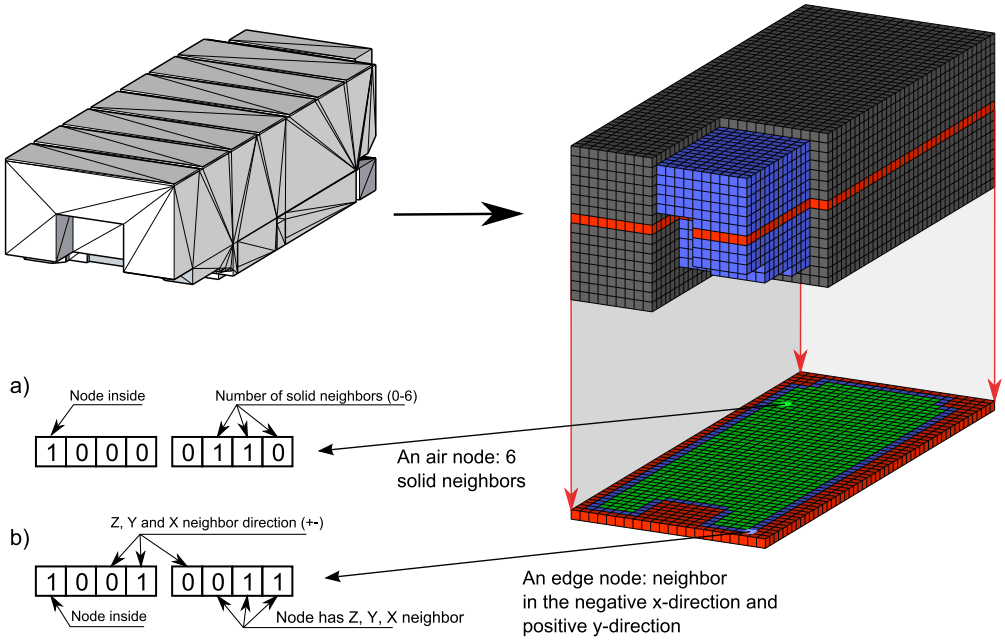

Figure 5.4: Bits of a single boundary node byte with a) forward difference boundary b) centered difference boundary.

Four meshes are needed to run a simulation: a position/orientation mesh that contains the information about whether the node is air, edge, corner or outside node, a material index mesh that contains the information on which material from a material list the node has and two pressure meshes for the current and past pressure values. The voxelization library produces the position/orientation and material index meshes. Pressure meshes are allocated during the memory partition, either with single or double precision data type. Mesh dimensions include additional padding which makes the dimensions of the mesh multiples of the block size dimensions used in the voxelizer kernels. A single layer of padding is left at the beginning of each coordinate plane as suggested by Webb [52] to avoid indexing out of bounds when the FDTD update kernel reads the neighboring values of nodes located at the boundary of the geometry.

The position mesh is further processed depending on the FDTD scheme selected. The format of the position mesh differs depending on the type of the boundary formulation. If a forward difference boundary is used, only the number of surrounding solid nodes is needed. If a centered difference boundary is used, the information on which side of the node the solid neighbors are located is also needed. The position bytes are presented in the Figure 5.4. The reason for two distinct byte types is that for the forward difference boundary a single type conversion is sufficient to decode the node type in the update equation instead of decoding the neighbor switches of the center difference byte.

5.4 FDTD Kernels

Several guidelines are given across literature and publications to optimize the performance of GPGPU kernels [29] [51]. The most important general principles that one comes across are:

- Minimize memory traffic between the host and the device
- When transferring memory from global memory to the kernels, ensure coalescent memory access
- Avoid conditional statements inside warps. Conditional statements inside a warp lead to branching and sequential execution of the code
- Minimize memory traffic between the off-chip memory of the device and the shared memory and the registers of the the SMs

Minimizing memory traffic between the host and the device can be ensured by allocating and partitioning all needed memory on the device before the simulation. The data used in the simulation such as simulation parameters, material information and pressures meshes are allocated and transferred to the device before the actual simulation steps. The maximum size of the mesh is limited by the size of the device memory.

The coalescent memory access means that all threads of a half-warp access off-chip memory at the same time. This is achieved with the selection of thread block size so that it is a multiple of the warps size, and having a uniform update kernel without conditional branching.

To avoid conditional statements inside the kernels, a uniform update equation for the whole mesh is formulated. For the SRL scheme a uniform update equation can be achieved for both forward difference and center difference boundaries. With the IWB scheme, such formulation is inefficient to implement since the boundary conditions for inner corners have numerous special cases. In the reference kernel such cases are ignored. Formulation of a position uniform update equation is achieved with the forward difference operator as presented in section 3.2.1. In a similar manner it is possible to formulate the update equation for SRL with centered difference boundaries with the use of bits indicating the dimension where the current node has a solid neighbor and the sign of the vector indicating on which side of each dimension the solid neighbor lies.

The methods for minimizing the memory traffic between the SMs and the off-chip memory are a question which is under investigation in this thesis. As discussed in section 4.2.2 and 4.2.3, CUDA devices have several different memory types which can be used to optimize the performance of the kernel. With older device generations significant speedups could be achieved with the use of shared memory and texture cache as was reviewed in section 5.1. As the read-only data cache was introduced with the Kepler architecture, it is not as evident that complex optimization techniques will work as efficiently compared with the built-in caching mechanisms.

The performance of two-level cache has been reported improving the computational efficiency significantly, so a comparison to a shared memory implementation is made. The effect of increased memory traffic is evaluated with an IWB kernel that uses all the 26 adjacent nodes in the update. The IWB kernel does not include all special cases of node positions, and hence is not fully functioning with complex geometries. It is included only for theoretical performance comparison. A total number of four different kernels were developed for evaluation:

- SRL update with centered difference boundary

- IWB update with centered difference boundary
- SRL update with forward difference boundary
- SRL update with forward difference boundary and with the use of shared memory

All the variables that are not modified during the update equation are declared with a `const __restrict` keywords to utilize the read-only cache. The kernel utilizing the shared memory of the device is implemented with the slicing method. The other kernels use direct memory fetches from the off-chip memory.

5.5 Visualization

When investigating the propagation of sound in an enclosed space, an informative tool for the end user is to be able to visualize the sound field. The FDTD method is an iterative method where the progress of the field is simulated in a time-marching loop, so the visualization of the sound field in the time-domain is inherently simple to implement.

When the FDTD simulation is implemented in platforms such as MATLAB, visualization can be done with built-in functions. In the case of a C / C++ application, such visualization tools must be implemented with separate libraries. A potential choice for rendering graphics is *Open GL* via *GLUT* [16] open source library because Open GL functionality can be coupled with CUDA using existing library functions. The GLUT library is utilized in the developed system.

The visualization scheme in the developed system is implemented with the use of CUDA-GL interoperability in the similar manner described by Demir et al. [8]. CUDA-GL interoperability allows a mapping of *pixel* and *vertex buffer objects* (PBO, VBO) initialized in the GL context to CUDA memory. This way it is possible to update PBOs and VBOs without first transferring the data to host memory. For 2-D visualization each coordinate plane of the 3-D geometry, x-y, x-z and y-z, is given a separate pixel buffers, which are updated according to the controls of the visualization and the pressure values of the mesh. In addition a functionality to capture the data from a single slice or from the whole mesh was implemented, so that the whole sound field or a single slice at predefined time steps can be analyzed after the simulation.

5.6 Matlab Integration

The integration of C and C++ functions into Matlab is made possible with the use of Matlab executables or in short, MEX-files [22]. A MEX file can be defined in several different ways including source MEX-files, binary MEX-files, MEX function libraries and MEX-build scripts. Source MEX-file is a C, C++ or Fortran source file which contains the subroutine definition. Binary MEX-files are dynamically-linked binary files, which the Matlab loads and executes. The binary MEX-file is created with MEX-build script from the source MEX-file. MEX function libraries are C/C++ and Fortran libraries which can be used inside source MEX-files. An example of such a library is is MX Matrix library, which can be used with Matlab's matrix datastructure.

The entry point to external C or C++ code is done in the source MEX-file. The source MEX-file is essentially acting as a main function of a given program with the difference that the input argument list of a regular C-program is replaced with a specific argument list which interfaces variables from the Matlab workspace to the function.

The source MEX-file can be compiled to a binary MEX-file with a command `"mex"` in Matlab. If the source MEX-file contains dependencies to external libraries or object files, the files and library locations are defined with option arguments typed after the compilation command in similar manner as with C / C++ compilers. When the compilation is done, the MEX-function can be called with the source MEX-file name. In this work, the Matlab integration is made by first compiling the C++ code into object files with Visual Studio, and then compiling a source MEX-file in Matlab containing the initialization of the solver with the arguments given from Matlab workspace.

Chapter 6

Evaluation

In this chapter, the functionality of the implemented system is evaluated. The evaluation is divided into two distinct parts. First, the responses of each scheme are analyzed in spectral and time domains. Second, the computational performance of the software is evaluated.

6.1 Analysis of simulated responses

To evaluate the results of a physical model, the most straightforward way is to compare the results of the simulation with analytical solutions. An exact analytical solution is either difficult or even impossible to derive in general form, hence simple test cases are used where the analytical solution is known and relatively easy to formulate. The test cases cover the most important parts of sound propagation. First, the spectral analysis of a room response of a rectangular room is compared with the analytic mode frequencies. Second, the correctness of the solver is validated by showing that the error in the estimated sound field reduces at an expected rate as the spatial resolution of the mesh is increased. Finally, the magnitudes of boundary reflections at different angles of incidence are compared with the analytic reflection coefficients values.

6.1.1 Spectral analysis of a simulated room impulse response

A convenient way to evaluate the simulated room impulse response is to compare the theoretical room modes with the magnitude spectrum of the simulated impulse response. It is possible to solve the theoretical room modes for a rectangular enclosure with rigid boundaries by solving the eigenfrequencies

of the wave equation. The formula for the eigenfrequencies for a rectangular room with rigid boundaries is given by [20]

$$k_{n_x,n_y,n_z} = \frac{c}{2}\left[\left(\frac{n_x}{L_x}\right)^2 + \left(\frac{n_y}{L_y}\right)^2 + \left(\frac{n_z}{L_z}\right)^2\right]^{\frac{1}{2}}, \tag{6.1}$$

where c is the speed of sound, n_x, n_y and n_z indicate the number of nodal planes in the direction of the subscript and L_x, L_y, and L_z are the dimensions of the room respectively.

A simulation for each scheme implemented was carried out with a model of cubic room with an edge dimension of 1 m. A source and a receiver were positioned at the opposite corners of the room. The reflection coefficient at the boundaries is set to 1. A hard source was used because the usage of a soft source introduced instability to the simulation due to high reflection order. The spatial sampling frequency was chosen so that the node dimensions are close to a fraction of room dimensions in order to match the mesh as close as possible to the geometry. Spatial sampling frequency is 59,583 Hz in the case of SRL schemes and 34,400 Hz with IWB scheme. Speed of sound in the simulation was set at 344 $\frac{m}{s}$. Sampling frequencies corresponds to a node size of 1 cm. The resulting impulse response was filtered with a DC-blocking IIR filter with a cut-off frequency at $0.0003 \times f_s$.

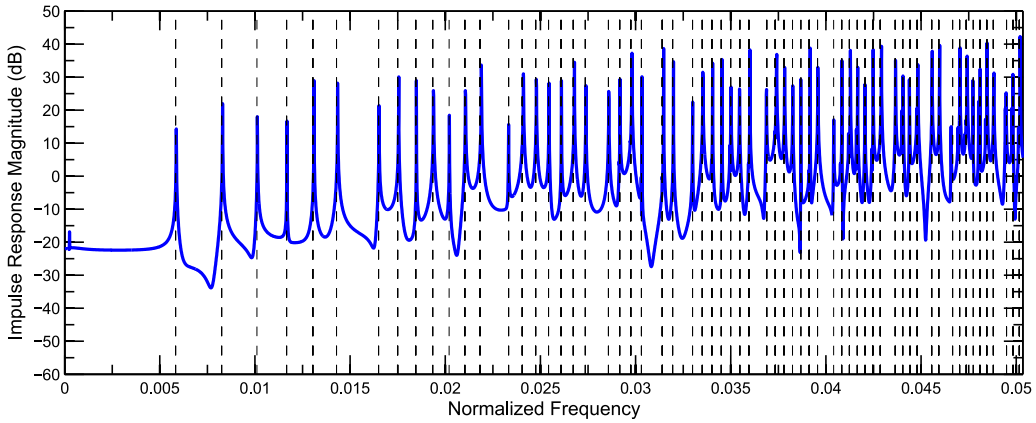

Figure 6.1: The magnitude spectrum of an impulse response simulated using the SRL update with center difference boundary formulation.

It can bee seen from Figures 6.1, 6.2 and 6.3 that the peaks in the spectrum of the simulated room response show significant correspondence to the analytic eigenfrequencies. With the forward difference boundary formulation, the peaks in the spectrum are located in slightly lower frequencies than the

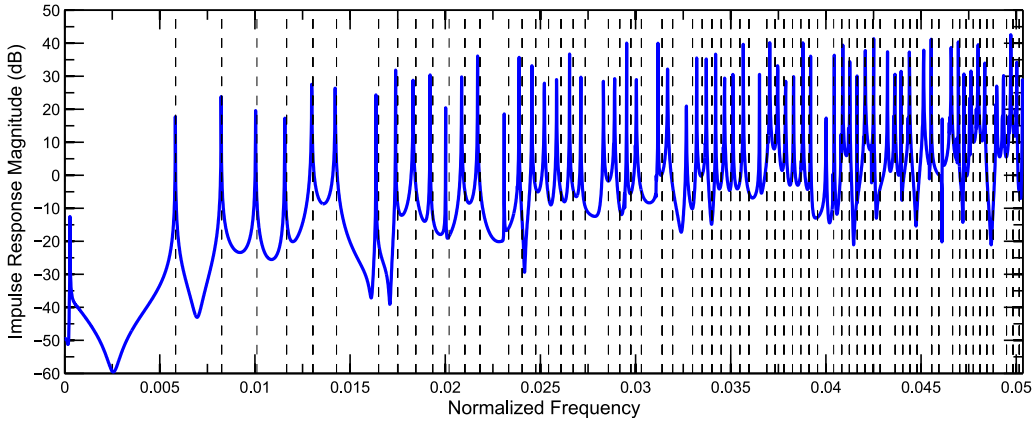

Figure 6.2: The magnitude spectrum of an impulse response simulated using the SRL update with forward difference boundary formulation.

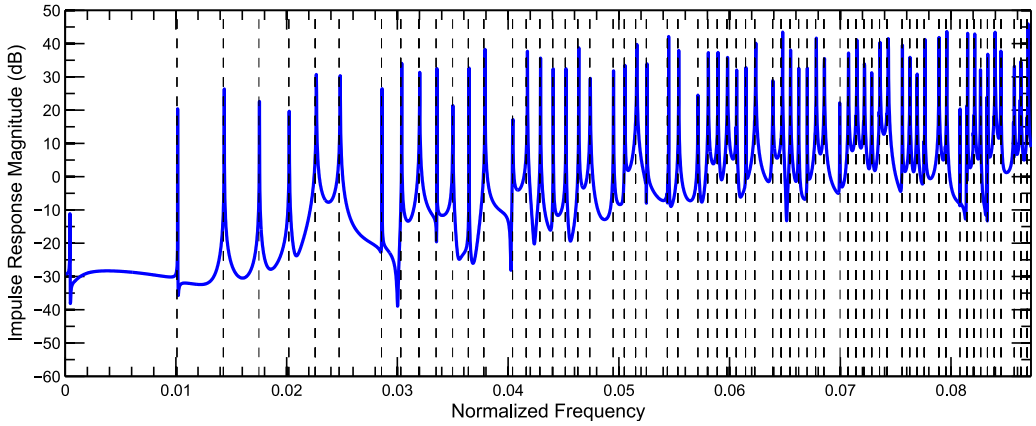

Figure 6.3: The magnitude spectrum of an impulse response simulated using the IWB update with center difference boundary formulation.

analytic eigenfrequencies. This might be due to the effect of the boundary formulation on the domain size. If the dimensions of the room are incremented with one Δx, the eigenfrequencies match up in a similar accuracy as in the case of the center difference boundary.

The peaks in both IWB and SRL are slightly tuned up. This is most probably due to the use of a hard source. The source node acts as a scatterer at the corner node and hence reduces the domain size. It can be concluded that these methods can estimate room modes of a cubic geometry with good accuracy. In the case of complex geometries, a more detailed evaluation should be carried out, for example against measurement results.

6.1.2 Free field propagation

To validate the free field propagation in the mesh, it can be shown that the result of the numerical solution approaches the analytical solution at a given rate as the resolution of the mesh is increased respectively. Namely this procedure ensures that the implemented scheme is *consistent* with the partial differential equation for any smooth function [47, p. 20].

The analytical solution for a simple pressure point source is

$$p(x,t) = \frac{1}{4\pi||x||} s(t - x/c), \tag{6.2}$$

where s is the source function. A smooth function which is easily scaled to different pulse widths which is used in the evaluations is

$$s(t) = \frac{d}{dt} \sin^{\alpha}(k_m c t), \tag{6.3}$$

where k_m is the wavenumber, c the speed of sound and α a fixed variable. A single period of the source function is used.

The procedure consists of running three different simulations in different mesh densities with a cubic geometry with an edge width of 1 m. Simulation was run in 3-D with the source located in the center of the room. Simulation time was chosen so that the wavefront does not reach any of the boundaries of the geometry and that the source excitation reaches the end of the pulse. The chosen time instance was 1.4 ms, which corresponds approximately to 48 cm of waves travel. The value of the fixed variable α was set to 6. Resolution of the mesh used are 50, 100 and 200 elements per dimension which resulted in the total mesh sizes of 125,000, 1,000,000 and 8,000,000 elements respectively. The value of Δt varies between different resolutions and is dependent on the Courant number λ of the scheme used.

The solution estimated with the FDTD solver was found to match the analytical solution with a delay of Δt assigned to the source signal of the FDTD simulation. The reason for this is arguable and needs more investigation. The delay might be due to the order in which the update is done, which is exciting the mesh with the source function at step n and then updating the mesh to step $n + 1$. The analytical solution used in the evaluation is for simple point source with no radius. Variables for different simulations are presented in table 6.1. The wavefronts in each case and the difference between the analytical and simulated wave fields are visualized in Figures A.1 and A.2.

The analytical solution is evaluated in the element positions of each mesh with the formula 6.2. A *convergence rate* is calculated from the absolute value

Table 6.1: Simulation setup for the evaluation of free field propagation.

Scheme	Number of Elements	λ	dx (m)	dt (ms)	pulse width (time steps)
SRL	125,000	$\frac{1}{\sqrt{3}}$	0.02	0.0033	35
SRL	1,000,000	$\frac{1}{\sqrt{3}}$	0.01	0.0017	69
SRL	8,000,000	$\frac{1}{\sqrt{3}}$	0.005	0.0008	139
IWB	125,000	1	0.02	0.0058	20
IWB	1,000,000	1	0.01	0.0029	40
IWB	8,000,000	1	0.005	0.0015	80

of the maximum error ϵ of the numerical solution of different mesh densities with the formula

$$r = \log(\frac{\epsilon_N}{\epsilon_{2N}})/\log(2), \tag{6.4}$$

where the subscript $2N$ indicates the doubling of the resolution. The error of the scheme is proportional to the squared error of the spatial difference in equations (3.2), (3.3), and (3.4). Therefore the expected convergence rate is $\log((2\Delta x)^2)/\Delta x^2)/\log(2) = 2$.

Table 6.2: Convergence rates of the two schemes when the resolution of the mesh is doubled.

Resolution difference	Convergence rate SRL	Convergence rate IWB
$50 \rightarrow 100$	1.9424	1.9635
$100 \rightarrow 200$	2.0261	2.0017

It can be seen from Figures A.1 and A.2 that the error decreases significantly when the resolution of the simulation is increased. As can be seen at table 6.2 the implemented solution follows the analytically expected convergence, and therefore it can be concluded that the solver can estimate free field propagation within the limits of the method.

6.1.3 Reflectance magnitude analysis

An important part of room acoustic simulation is the boundary conditions. To evaluate the reflection characteristics at the boundary of the model, it is possible to approach the problem in similar manner as when measuring boundary conditions in a real-world situation. One such method is described

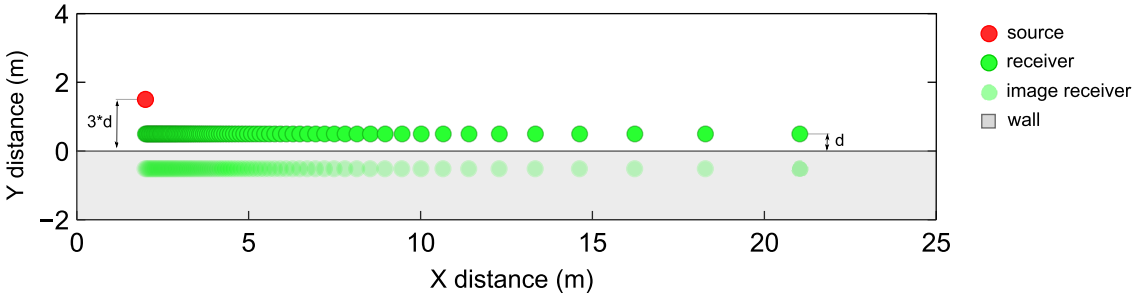

Figure 6.4: Layout of the simulation setup for measuring the reflection magnitude of a FDTD boundary condition.

by Mommertz [23]. In the method the first reflection from the surface is isolated and compared with an "ideal" reflection from a rigid boundary. An ideal reflection can be generated with a source propagating in free space at a distance which corresponds to the distance of the reflected signal, namely with the image source principle [1]. This approach has been previously used with finite-difference methods by Kelloniemi [15] and Haapaniemi [11].

The procedure consists of running three separate simulations for each reflection coefficient. The simulation is run with one receiver per angle of incidence and one source. The configuration is presented in Figure 6.4. the first simulation is run with the receivers located at close distance of the reflecting surface. The second simulation is run with the same source/receiver setup as with the first one but in a free field to attain the isolated direct sound. The third simulation is run in the free field so that the source is located in similar manner as an image source would be located behind the reflecting surface in the first simulation. In this way, the response of an ideal reflection signal is achieved.

The response from the second simulation is subtracted from the first which results in an isolated reflection from the surface. The energy of the surface reflection is then compared with the energy of the ideal reflection. A reflection coefficient estimate is achieved by taking the square root of this fraction. The results of the reflection coefficient estimates are presented in Figures 6.5, 6.6 and 6.7. The analytic reflection coefficient for each angle of incidence is calculated from equation (2.22).

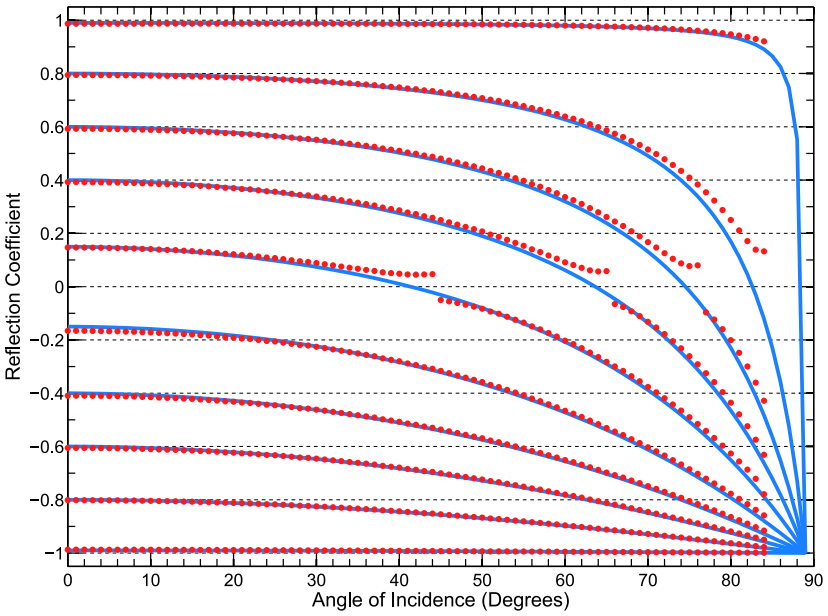

Figure 6.5: Reflection coefficient estimates for each angle of incidence calculated from the magnitude of the reflected impulse. The estimates are presented as red dots and analytic reflection coefficients as a solid blue line. The simulation is run using the SRL scheme with the centered difference boundary

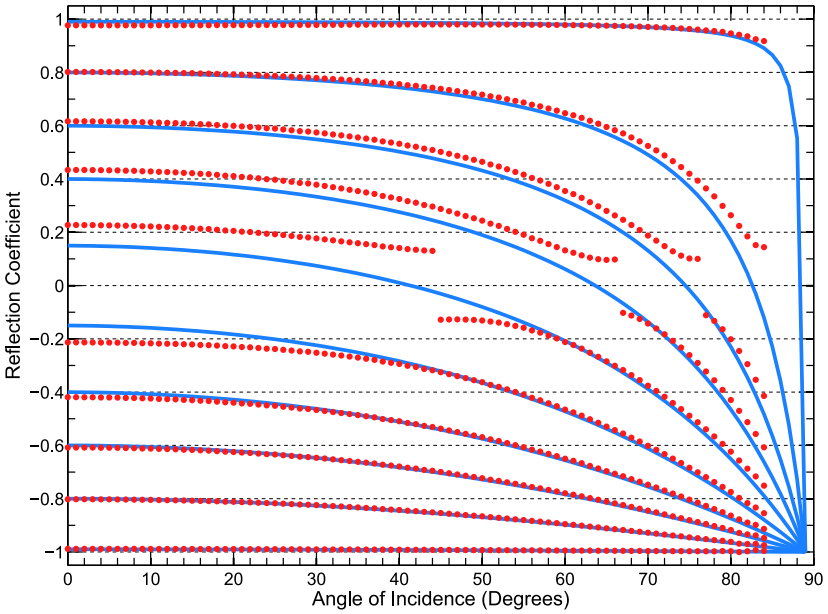

Figure 6.7: Reflection coefficient estimates for each angle of incidence calculated from the magnitude of the reflected impulse. The estimates are presented as red dots and analytic reflection coefficients as a solid blue line. The simulation is run using the SRL scheme with the forward difference boundary.

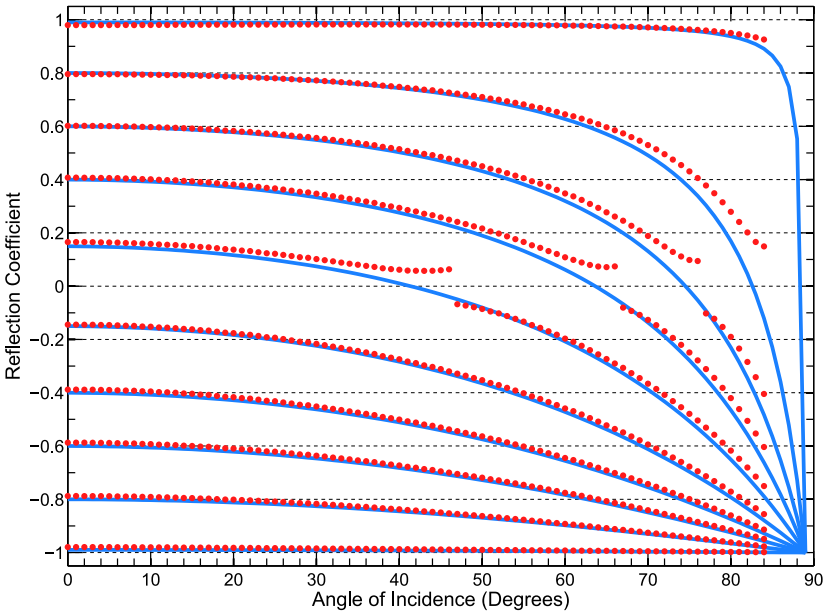

Figure 6.6: Reflection coefficient estimates for each angle of incidence calculated from the magnitude of the reflected impulse. The estimates are presented as red dots and analytic reflection coefficients as a solid blue line. The simulation is run using the IWB scheme with the centered difference boundary

The reflectance magnitude follows the analytical solution of the reflection fairly well when the center difference boundary is utilized. In the case of the forward difference boundary, the magnitude response values do not follow the analytical solution in similar accuracy. It can be observed from the Figure 6.7 that the reflection characteristics of the forward difference boundary introduces more reflected energy at the low values of the reflection coefficient. In each of the cases there is a severe step around the reflection coefficient values around zero. This is due to the fact that a simple admittance boundary is in fact frequency dependent as presented by Kowalczyk [18] and therefore to achieve a perfectly absorbing boundary, a simple admittance formulation is not sufficient. It can be concluded that the implemented boundary conditions follow the analytical characteristics of a boundary reflection with accuracy that can be expected from the method and therefore the implementation is valid.

6.2 Computational Performance

In this section, the performance of the different schemes is evaluated. Four different kernels are evaluated which are, as reviewed in section 5.4, SRL update with a centered difference boundary and read-only cache (SRL centered), SRL update with a forward difference boundary and read-only cache (SRL forward), SRL update with a forward difference boundary with shared memory (SRL shared) and IWB with a centered difference boundary and read-only cache (IWB). The implementation of the IWB kernel can not handle geometries with inner corners and is included here as a reference. All of the kernels are implemented with frequency independent boundary conditions with the value of the admitance fetched for each update. Evaluation is performed with four different mesh densities of a rectangular enclosure. Results of the performance test are presented in Figure 6.8.

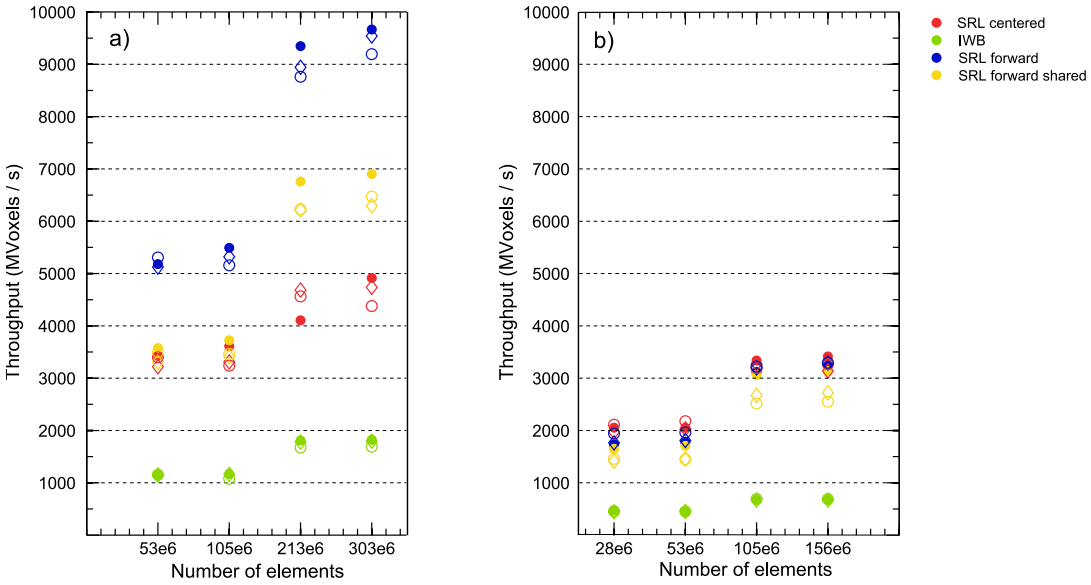

Figure 6.8: Throughput of different schemes in a) single precision and b) double precision. Different symbols stand for different block sizes. • refers to a block size of [32, 4, 1], ◇ to [32, 8,1] and ○ to [64, 8, 1]. Different colors refer to different schemes as presented in the legend. The number of devices used is two with the two larger mesh sizes.

Notable for the results is that the performance difference between the forward difference boundary and the center difference boundary is significant even with the same memory load. SRL centered uses more operations when decoding the boundary update. This indicates that the kernels are not mem-

ory bound but operation bound. The reference implementation of IWB kernel has significantly lower throughput which is due to the increased number of nodes which have to be fetched from the global memory for each update, and the higher number of arithmetic operations needed for the update.

The throughput with double precision is significantly lower in comparison with the throughput with single precision. This is due to lower throughput of arithmetic operations. The number of add, multiply and multiply-add operations in one clock cycle with single precision with the compute capability of 3.0 is 192 whereas with double precision it is only 8. The increased memory traffic due to the doubled size of each pressure value increases the load on the memory bus. It can be observed that the kernels tend to become less sensitive to the operation count when using double precision, which can be seen from the effect of similar throughputs of the SRL centered and SRL forward.

Table 6.3: Results of reported implemetations of FDTD solvers using CUDA. The notion (double) indicates if the benchmark is carried out in double precision.

	Stencil size	GPU	Computing capability	Memory	Number of cores	Reported performance Mvoxels/s
Savioja SRL [36]	6	Quatro FX 5800	1.3	4 GB	240	1310
Savioja IWB [36]	26	Quatro FX 5800	1.3	4 GB	240	753
Webb [52]	6	Tesla C1060	1.3	4 GB	240	818
Webb [52]	6	Tesla C2050	2.0	3 GB	448	2044
Webb [54] (double)	6	Tesla K20	3.5	5 GB	2490	4698
Lopez et al. [21]	6	GTX 260	1.3	896 MB	192	2808
Lopez et al. [21]	6	GTX 480	2.0	1.5 GB	448	7279
Sheaffer[42]	6	Tesla T10	1.3	4 GB	240	1064
The developed system, single device (float)	6	GTX 690	3.0	2 GB	1536	5500
The developed system, two devices (float)	6	GTX 690	3.0	4 GB	3072	9700
The developed system, single device (double)	6	GTX 690	3.0	2 GB	1536	2200
The developed system, two devices (double)	6	GTX 690	3.0	4 GB	3072	3400

The implementation of the sliced method using shared memory does not introduce similar performance gains as presented by Lopez et al. [21]. This might be due to the reported sensitivity of the sliced method to different boundary formulations [42].

In comparison to previously suggested methods, the developed system performs well. The reported performance of each implementation is trans-

formed into the units / second format, which in this case is millions of voxels the implementation is capable of processing in second. The throughput in single precision with a single device is the second highest. The throughput of the kernel developed by Lopez et al. [21] is the highest, which can be dues to the use of fixed admittance value that reduces the memory traffic. The throughput with double precision is not as large as in [54], which would indicate that several optimizations could be carried out to increase the performance. It can be concluded that the performance of the developed system is comparable to the most recent publications.

Chapter 7

Conclusions

The aim of this thesis was to develop an efficient FDTD solver for the use of room acoustics simulation. The main design principles of the system were to make the concurrent use of multiple GPU devices possible, interface the system through Matlab to allow an efficient usage of the solver through scripting and maintain an approachable code base for later development and benchmarking of kernels.

The implemented system was evaluated against the analytical solutions of different aspects of room acoustics. The system was proven to estimate modal frequencies in a simple box geometry, estimate the reflection characteristics of a simple admittance boundary, and simulate the free field propagation of sound with accuracy expected of the method. The reflection characteristics of different boundary formulation deviate slightly. The computationally most efficient boundary formulation does introduce higher reflection energy at low reflection coefficient values.

The computational performance of the system is similar in comparison with other published systems. The further optimization of the kernels could introduce better performance. The performance in double precision is severely lower than with single precision which was expected. The results indicate that the double precision kernels become memory bound opposed to single precision where the operation count affects the performance.

7.1 Future Work

As new GPU device generations are introduced, many of the problems concerning the computational cost of FDTD in room acoustics can be overcome with increased memory bandwidth and processing power. What can not yet be said is whether the responses achieved with purely oversampled mesh are

perceptually sufficient when used in collaboration with rendering techniques.

Several parts of the current system should be further developed in the light of current research. The implementation of frequency-dependent boundary conditions could be carried out and investigated further. A stable version of the IWB scheme with attention given to the code optimization could result in an efficient alternative for purely oversampled SRL scheme. Comparison against spectral methods such as ARD should be made.

Validation of simulation results against measurements should be carried out both numerically and perceptually to gain more insight on what the current version of the system can explain. The integration of different simulation algorithms, such as beam tracing, and tools for measuring room impulse responses in the system, would result in a powerful toolbox for gathering data and evaluating the aspects of different methods and guide the way to more accurate and useful acoustic simulation.

Bibliography

[1] Allen, J. B., and Berkley, D. A. Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America 65* (1979), 943.

[2] Bilbao, S. *Wave and scattering methods for the numerical integration of partial differential equations*. PhD thesis, Stanford University, 2001.

[3] Bilbao, S. *Wave and scattering methods for numerical simulation*. Wiley, 2004.

[4] Borish, J. Extension of the image model to arbitrary polyhedra. *The Journal of the Acoustical Society of America 75* (1984), 1827.

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

[6] Botts, J., and Savioja, L. Integrating finite difference schemes for scalar and vector wave equations. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing* (2013).

[7] Courant, R., Friedrichs, K., and Lewy, H. On the partial difference equations of mathematical physics. *IBM Journal of Research and Development 11*, 2 (1967), 215–234.

[8] Demir, V., and Elsherbeni, A. CUDA-OpenGL interoperability to visualize electromagnetic fields calculated by FDTD. *Journal of the Applied Computational Electromagnetics Society 27*, 2 (2012).

[9] Fang, J., Varbanescu, A. L., and Sips, H. A comprehensive performance comparison of CUDA and OpenCL. In *Proceedings of the International Conference on Parallel Processing* (2011), IEEE, pp. 216–225.

[10] Farber, R. *CUDA Application Design and Developement.* Elsevier Inc., 2011.

[11] Haapaniemi, A. Simulation of acoustic wall reflections using the finite-difference time-domain method. Master's thesis, Aalto University, Finland, 2012.

[12] Heckbert, P. S., and Hanrahan, P. Beam tracing polygonal objects. *ACM SIGGRAPH Computer Graphics 18*, 3 (1984), 119–127.

[13] Karimi, K., Dickson, N. G., and Hamze, F. A performance comparison of CUDA and OpenCL. *arXiv preprint arXiv:1005.2581* (2010).

[14] Karlsson, H. Solid voxelization algorithm for room acoustics. Master's thesis, Aalto University, Finland, 2013.

[15] Kelloniemi, A. *Room Acoustic Modeling With The Digital Waveguide Mesh - Boundary Structures and Approximation Methods.* PhD thesis, Aalto University, Finland, 2008.

[16] Kilgard, M. J. The OpenGL utility toolkit (GLUT) programming interface, 1996.

[17] Kirk, D. B., and Hwu, W. *Programming Massively Parallel Processors.* Morgan Kaufmann Publishers, 2010.

[18] Kowalczyk, K. *Boundary and medium modelling using compact finite difference schemes in simulation of room acoustics for audio and architectural design applications.* PhD thesis, School of Electorincs, Electrical Engineering and Computer Science, Queen's University Belfast, 2008.

[19] Krokstad, A., Strom, S., and Sørsdal, S. Calculating the acoustical room response by the use of a ray tracing technique. *Journal of Sound and Vibration 8*, 1 (1968), 118–125.

[20] Kuttruff, H. *Room acoustics*, 5 ed. Taylor & Francis, 2009.

[21] López, J. J., Carnicero, D., Ferrando, N., and Escolano, J. Parallelization of the finite-difference time-domain method for room acoustics modelling based on CUDA. *Mathematical and Computer Modelling 57*, 7-8 (2011), 1822–1831.

[22] Matlab. *Introducing MEX-files*, 2013. Online, accessed May, 2013, `http://www.mathworks.se/help/matlab/matlab_external/introducing-mex-files.html`.

[23] Mommertz, E. Angle-dependent in-situ measurements of reflection coefficients using a subtraction technique. *Applied Acoustics 46*, 3 (1995), 251–263.

[24] Morse, P., and Ingard, K. *Theoretical acoustics*. Princeton University Press, 1987.

[25] Navarro, J. M. *Discrete-time modelling of diffusion processes for room acoustics simulation and analysis*. PhD thesis, Technical University of Valencia, 2012.

[26] Nosal, E.-M., Hodgson, M., and Ashdown, I. Improved algorithms and methods for room sound-field prediction by acoustical radiosity in arbitrary polyhedral rooms. *The Journal of the Acoustical Society of America 116* (2004), 970.

[27] NVIDIA corporation. *NVIDIA CUDA Architecture, Introduction & Overview*, 2009. Online, accessed February, 2013, `http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf`.

[28] NVIDIA corporation. *NVIDIA's Next Generation CUDA Computer Architecture: Fermi*, 2009. Online, accessed Match, 2013, `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`.

[29] NVIDIA corporation. *CUDA C Best practices guide*, 2012. Online, accessed August, 2013, `http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html`.

[30] NVIDIA corporation. *CUDA C Programming guide*, 2012. Online, accessed August 2013, `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.

[31] NVIDIA corporation. *NVIDIA's Next Generation CUDA Computer Architecture: Kepler GK110*, 2012. Online, accessed August, 2013, `http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf`.

[32] NVIDIA corporation. *Parallel Thread Execution ISA Version 3.1*, 2012. Online, accessed March, 2013, `http://docs.nvidia.com/cuda/parallel-thread-execution/index.html`.

[33] Picaut, J., Simon, L., and Polack, J. A mathematical model of diffuse sound field based on a diffusion equation. *Acta Acustica united with Acustica 83*, 4 (1997), 614–621.

[34] Pulkki, V., Lokki, T., and Savioja, L. Implementation and visualization of edge diffraction with image-source method. In *Proceedings of the Audio Engineering Society Convention 112* (2002).

[35] Raghuvanshi, N., Narain, R., and Lin, M. Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Transactions on Visualization and Computer Graphics 15*, 5 (2009), 789–801.

[36] Savioja, L. Real-time 3D finite-difference time-domain simulation of low-and mid-frequency room acoustics. In *Proceedings of International Conference on Digital Audio Effects* (2010), vol. 1, p. 75.

[37] Savioja, L., Backman, J., Järvinen, A., and Takala, T. Waveguide mesh method for low-frequency simulation of room acoustics. In *Proceedings of the International Computer Music Conference* (1994), pp. 463–466.

[38] Savioja, L., and Valimaki, V. Interpolated rectangular 3-D digital waveguide mesh algorithms with frequency warping. *IEEE Transactions on Speech and Audio Processing 11*, 6 (2003), 783–790.

[39] Schneider, J., Wagner, C., and Broschat, S. Implementation of transparent sources embedded in acoustic finite-difference time-domain grids. *The Journal of the Acoustical Society of America 103* (1998), 136.

[40] Schwarz, M., and Seidel, H.-P. Fast parallel surface and solid voxelization on GPUs. *ACM Transactions on Graphics 29*, 6 (2010), 179.

[41] Sheaffer, J. Wavecloud, accelerated acoustics FDTD, march 2013. Online, accessed May, 2013 `http://wavecloud.jonsh.net/`.

[42] Sheaffer, J., and Fazenda, B. FDTD/K-DWM simulation of 3D room acoustics on general purpose graphics hardware using compute unified device architecture (CUDA). *Proceedings of the Institute of Acoustics 32*, 5 (2010).

[43] Siltanen, S., Lokki, T., Kiminki, S., and Savioja, L. The room acoustic rendering equation. *The Journal of the Acoustical Society of America 122* (2007), 1624.

[44] Smith III, J. Physical modeling using digital waveguides. *Computer Music Journal* (1992), 74–91.

[45] Southern, A., Murphy, D., Campos, G., and Dias, P. Finite difference room acoustic modelling on a general purpose graphics processing unit. In *Proceedings of the Audio Engineering Society Convention 128* (2010).

[46] Southern, A., Siltanen, S., Murphy, D., and Savioja, L. Room impulse response synthesis and validation using a hybrid acoustic model. *IEEE Transactions on Audio, Speech, and Language Processing 21*, 9 (2013).

[47] Strikwerda, J. *Finite difference schemes and partial differential equations*. Wadsworth & Brooks, 1989.

[48] Sutter, H., and Larus, J. Software and the concurrency revolution. *Queue 3*, 7 (2005), 54–62.

[49] Svensson, P., and Kristiansen, U. R. Computational modelling and simulation of acoutic spaces. In *Proceedings of the Audio Engineering Society Conference: Virtual, Synthetic, and Entertainment Audio* (2002).

[50] Van Duyne, S., and Smith, J. Physical modeling with the 2-D digital waveguide mesh. In *Proceedings of the International Computer Music Conference* (1993), pp. 40–40.

[51] Volkov, V. Better performance at lower occupancy. In *Proceedings of the GPU Technology Conference, GTC* (2010), vol. 10.

[52] Webb, C., and Bilbao, S. Computing room acoustics with CUDA-3D FDTD schemes with boundary losses and viscosity. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing* (2011), pp. 317–320.

[53] Webb, C., and Gray, A. Large-scale virtual acoustics simulation at audio rates using three dimensional finite difference time domain and multiple graphics processing units. In *Proceedings of Meetings on Acoustics* (2013), vol. 19, p. 070092.

[54] Webb, C. J. Computing virtual acoustics using the 3D finite difference time domain method and Kepler architecture GPUs. In *Proceedings of the Stockholm Musical Acoustics Conference* (2013).

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

Appendix A

Analytic and simulated free field propagation

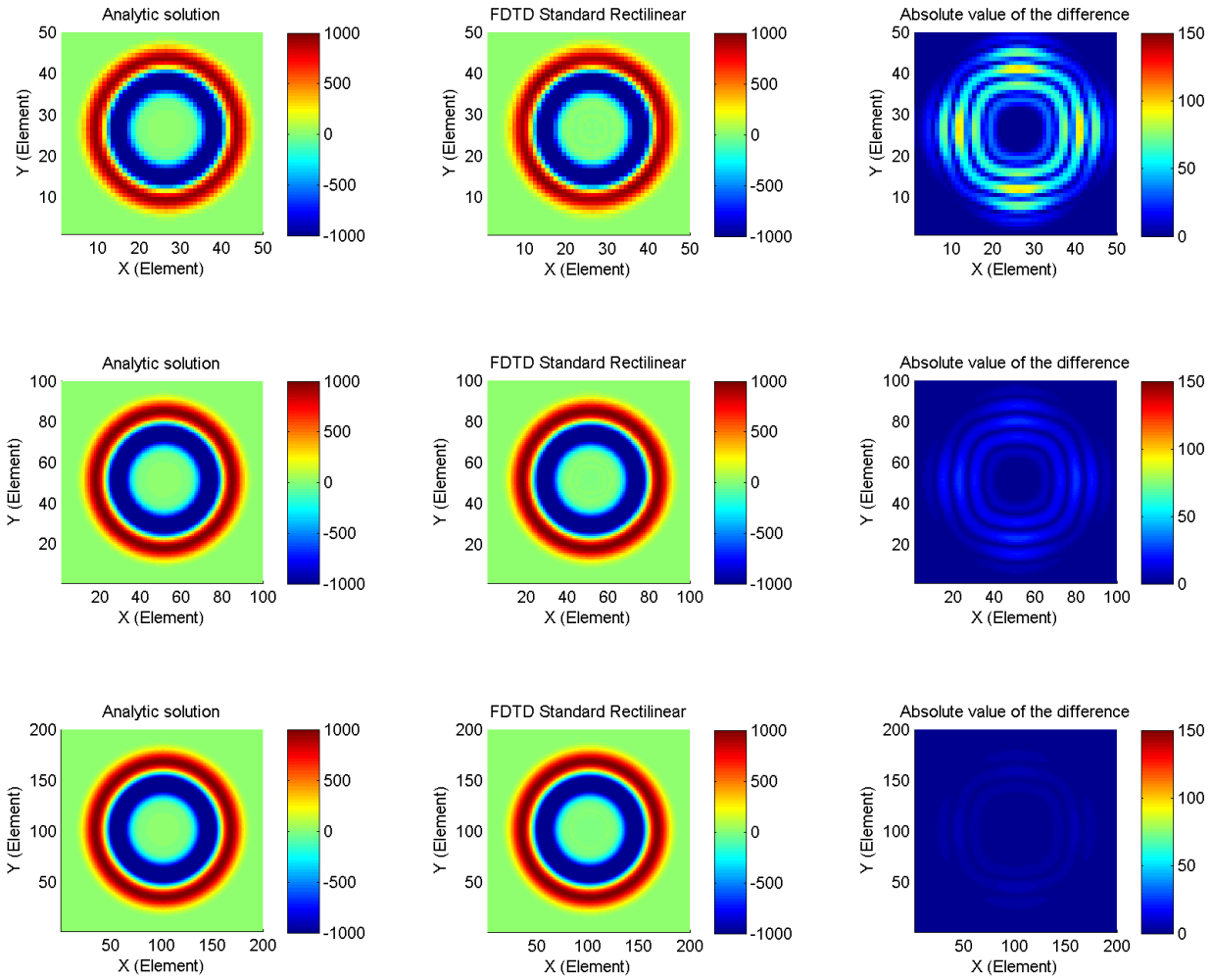

Figure A.1: Analytic and simulated results using the SRL scheme inside a cubic space with 1 m edge in different resolutions.

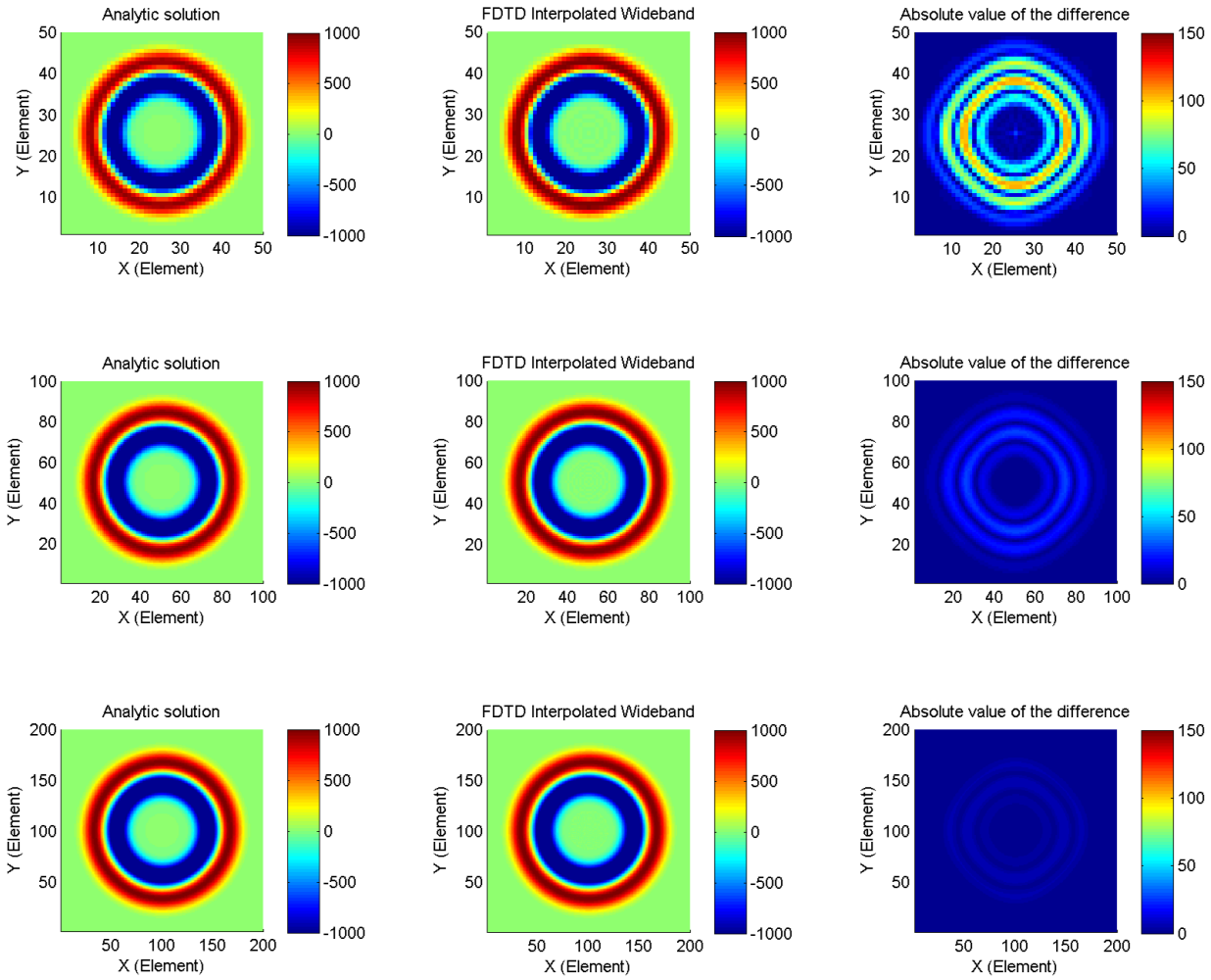

Figure A.2: Analytic and simulated results using the IWB scheme inside a cubic space with 1 m edge in different resolutions.