CHALMERS
UNIVERSITY OF TECHNOLOGY

Method Development of Automotive Ultrasound Simulations

Master's thesis in Complex Adaptive Systems

JOHAN NORDEVALL

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

Master's thesis 2015:58

Method Development of Automotive Ultrasound Simulations

JOHAN NORDEVALL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2015

Method Development of Automotive Ultrasound Simulations
JOHAN NORDEVALL

© JOHAN NORDEVALL, 2015.

Examiner: Lennart Löfdahl, Chalmers
Supervisor: Christophe Duwig, Volvo Cars

Master's Thesis 2015:58
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Automated parking illustration. Image courtesy of Volvo Cars.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2015

Method Development of Automotive Ultrasound Simulations
JOHAN NORDEVALL
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

Within the automotive industry, autonomous driving is a promising avenue with potentially major advantages in terms of safety, fuel economy, and convenience. In order to secure the engineering of autonomous vehicles, active safety systems will need to be combined to provide a 360 degree view around the vehicle at all times. Risks arise when the different sensors are contaminated or yield conflicting information, especially when determining a proper response in a critical situation.

One area of special interest in active safety systems is geared towards parking applications, such as parking assistance or self-parking systems. These systems are primarily based on ultrasonic sensors that are used in a sonar-like fashion to detect potential obstacles in the proximity of the vehicle. In order to gain a better understanding of the strengths and limitations of ultrasonic sensors in applications like these, computer modeling can be used.

In this report a proof-of-concept framework for ultrasonic simulations has been designed and implemented. High performance computational aspects have been considered and domain decomposition techniques have been investigated for large-scale simulations on computer clusters. The framework has a high level interface for building virtual parking simulation scenarios, where the following aspects can be controlled and/or specified: the three-dimensional simulation domain, various environment and control parameters, placement of cars and other potential obstacles, the exportation of data.

Keywords: automotive simulation, ultrasonic parking sensors, finite difference method

Acknowledgements

This master thesis was carried out at Volvo Cars. I would like to thank everyone involved in this project, particularly Christophe Duwig and Lennart Löfdahl.

Johan Nordevall, Gothenburg, June 2015

Contents

1

Introduction

Within the automotive industry, autonomous driving is a promising avenue with potentially major advantages in terms of safety, fuel economy, and convenience. In order to secure the engineering of autonomous vehicles, active safety systems will need to be combined to provide a 360 degree view around the vehicle at all times. An overview of current automotive sensor systems are shown in figure 1.1. The sensor systems are mostly complementary but there is some overlap, hence risks arise when the different sensors are contaminated or yield conflicting information, especially when determining a proper response in a critical situation.

It is therefore vital to control the risks and build virtual strategies for testing a wide range of situations. For example in self-parking scenarios, one may optimize the number and location of ultrasonic sources around the vehicle for providing a complete and clear image of the dynamic surrounding. To do so, the simulation of ultrasound propagation between vehicles and in complex geometry is of great importance.

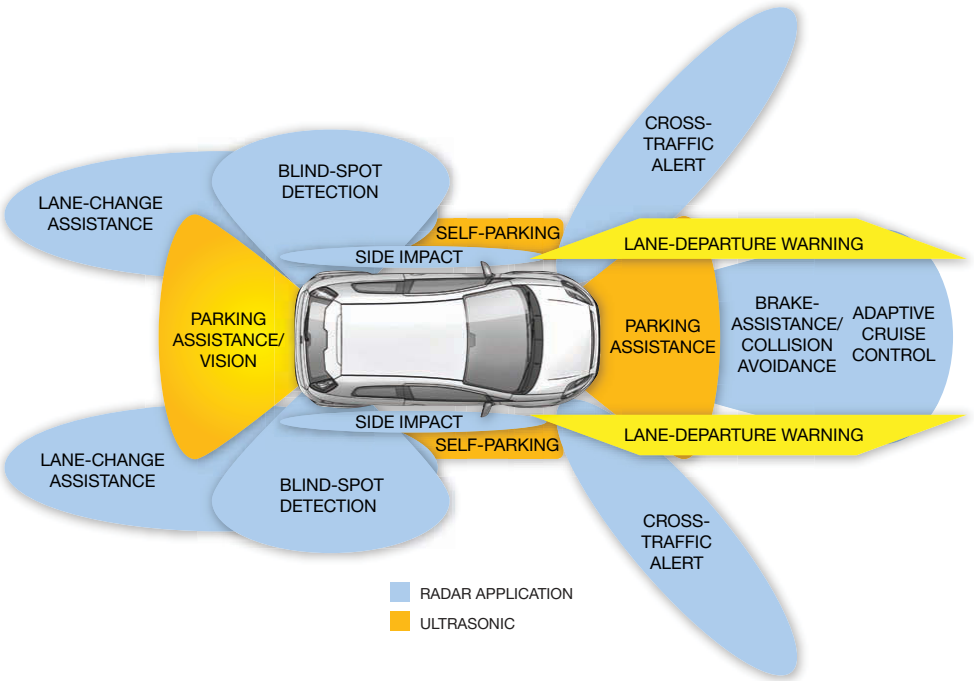


Figure 1.1: A selection of modern automotive sensor systems. [1]

1.1 Background

Volvo Cars currently uses commercial off-the-shelf (COTS) ultrasonic sensors in their parking assistance systems and thus no in-house data analysis is performed on the data from these sensors. The sensor system simply acts as a so-called black box that (after empirical calibration) outputs the perceived distance to an object – which may or may not actually be the actual distance. To gain a better understanding of the limitations of obstacle detection via ultrasonic sensors, a simulation framework is desired in order to study the behavior in complex parking scenarios.

Modeling the behavior of ultrasonic detection and/or sensor systems requires modeling of the propagation of the ultrasonic waves in the near field of the vehicle. Ultrasonic waves is an umbrella term for sound waves operating at a higher frequency than that the human ear can detect, typically ca 20 kHz. Sound waves in turn are a special case of waves that may be modeled by the mathematical wave equation, which shall be used as the driving equation in the following work.

The wave equation and its related formulation of wave propagation, is a well-researched area and articles based on implementing the wave equation in different research fields are numerous. These research areas include seismic modeling – from oil and gas exploration to earthquake simulations, acoustics – from concert halls to real time auralization[1] in computer games, and modeling of electromagnetic fields. [2, 3, 4, 5, 6, 7]

In addition, this thesis will combine and lightly cover other areas of intense research, many of them being separate research fields in their own right: sensor modeling, acoustic wave modeling, high performance computing and parallelization techniques.

[1] Perceived sound accurately matches what one would expect from the virtual/simulated environment, in terms of sound direction, echo, and reverberation.

2

Aim

The aim of this thesis is to develop a proof-of-concept framework supporting virtual testing and high fidelity simulations of ultrasonic sensors in automotive applications. For example it may be used in simulating parking sensors to make sure that obstacles behind a car are properly detected, and if not, to find out what configuration of ultrasonic sensors that are needed to accurately capture the surrounding area.

In summary, when using the framework one should be able to specify:

- Parameters such as: domain dimensions, ultrasonic sensor frequency, speed of sound, etc.
- The location of cars and other models (e.g. pedestrians and other obstacles) in the domain
- What data to export from the simulation

Before designing the framework different simulation methods and techniques will need to be investigated and evaluated, and as a part of this a sub target in this thesis will be to develop a two dimensional (2D) ultrasound simulation script. This will speed up the development cycle when comparing simulation methods and simplify initial visualization and verification efforts. Once an appropriate method have been found the script can be extended to 3D simulations and a framework can be built around it.

2.1 Limitations

To keep within the time frame set out for this work the following assumptions and scope limitations apply:

- The wave speed in the simulation domain is constant – no air temperature gradients
- The ultrasonic source is point sized and its wave propagates with uniform intensity – even perpendicularly to its mounting plane
- No attenuation – the wave energy is not partially absorbed by the medium its traveling in

- All cars and models are static – the sound travels orders of magnitude faster than any object in parking situations
- No absorption at domain boundaries/obstacles
- Only specular reflections are considered
- No data analysis, only export data
- Virtual testing only – no comparison to real world data

Addressing these limitations is no doubt interesting and a logical extension in future work, however, they are beyond the scope of this thesis.

3

Theory

This chapter will cover most of the groundwork required for a basic understanding of the different parts involved in implementing a simulation framework for ultrasonic sensors.

3.1 Ultrasonic sensors

An ultrasonic sensor, or rather ultrasonic transducer, is a type of device that converts high frequency sound into electric signals, or vice versa. It can be used as an active sonar, meaning that it emits a pulse and listens for an echo. By measuring how long it took for the sensor to detect echo, the distance to the object causing the echo can be calculated. See figure 3.1.

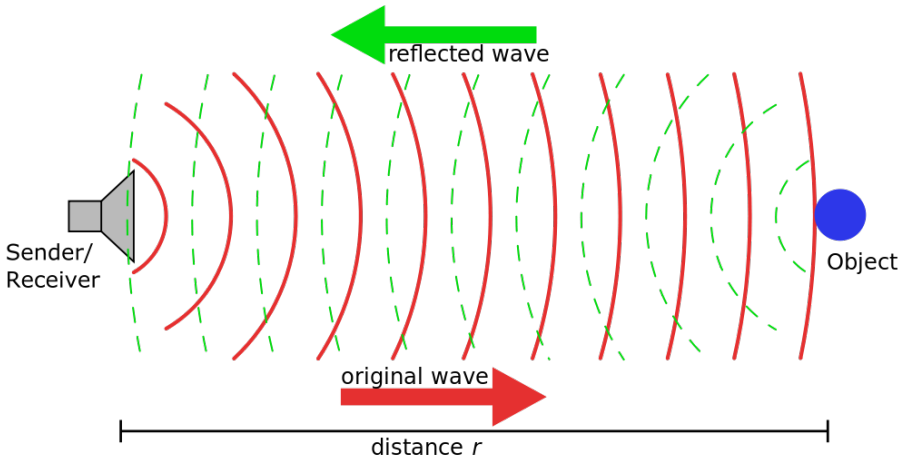


Figure 3.1: Sonar principle. The distance r may be calculated as $r = c\Delta t/2$, where c is the speed of sound and Δt is the time difference from that the wave was emitted till that the echo was received. [8]

It should be noted that the speed of sound is temperature dependent, an approximate relation can be derived from the ideal gas law[1]: $c \approx 331.4\sqrt{1 + T/273.15}\,\mathrm{m/s}$, where T is the temperature in degrees Celsius[9]. The resulting air speed in the temperature

[1]For an isentropic wave at constant heat capacity ratio and zero percent air humidity.

span -30 to 40 °C ranges from ca 312 to 355 m/s, which will affect the accuracy of ultrasonic parking sensors if not taken into account.

Commercial ultrasonic sensors for automotive parking applications typically operate in the frequency band 40 to 60 kHz and are specified to accurately detect obstacles from 30 up to 450 cm away from the sensor. Integrated temperature sensors may be used to compensate for the temperature dependency in the speed of sound and increase the accuracy. [10, 11, 12]

It should also be noted that the so-called beam pattern of ultrasonic sensors is non-uniform, typically meaning that the signal emitted from the sensor will be strongest straight ahead of the sensor. Should an object be at an angle to the sensor it may fail to be detected or only be detected below the maximum range of the sensor. See figure 3.2 for an example of how this affects the maximum sensing range.

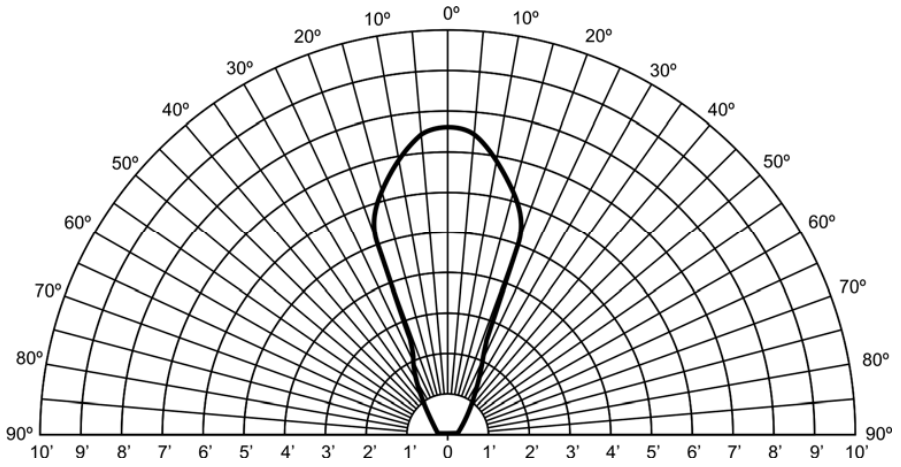


Figure 3.2: Example result of a non-uniform beam pattern in an ultrasonic sensor. The figure shows at which distance (in feet, 1 ft ≈ 0.3 m) a 9 cm cylinder was detected. [13]

3.2 The wave equation

The wave equation is a linear partial differential equation that describes how waves propagate in a medium. It can apply for sound waves as well as light and water waves. For a time dependent pressure field $u = u(\mathbf{x}, t)$ and a source $f = f(\mathbf{x}, t)$ in the spatial domain $\mathbf{x}$ and time domain t, the wave equation is given by:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \nabla^2 u = f, \tag{3.1}$$

where ∇^2 is the Laplace operator and c the propagation speed of the wave in the domain.

3.3 Numerical methods

3.3.1 Finite difference method

In order to solve equation (3.1) it will be discretized in space and time using the finite difference method (FDM). In the one-dimensional case the discretized solution to equation (3.1) may be represented as u_i^n at time step n and point i in the domain. Here it is assumed that the temporal domain spans $n = 0, \ldots, T$ equal parts of length Δt while the spatial domain is divided into $i = 0, \ldots, N$ equal parts of length h such that $N = L/h$, where L is the length of the domain. The length h is often referred to as the cell size.

The second derivative in time at a specific time n and position i may then be approximated as

$$\frac{\partial^2 u_i^n}{\partial t^2} \approx \frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} \tag{3.2}$$

using the central difference method. Similarly, the Laplacian operator in one dimension, $\nabla^2 = \frac{\partial^2}{\partial x^2}$, will be approximated as

$$\frac{\partial^2 u_i^n}{\partial x^2} \approx \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}. \tag{3.3}$$

Using these approximations equation (3.1) may be rewritten in its explicit time stepping form:

$$u_i^{n+1} = 2u_i^n - u_i^{n-1} + \gamma^2 \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n \right) + f_i^n, \tag{3.4}$$

where $\gamma = \frac{c\Delta t}{h}$ is the so called Courant number. Assuming initial conditions $u(x, 0) = g(x)$ and $u_t(x, 0) = h(x)$, the first iteration steps can be derived to:

$$u_i^0 = g_i, \quad u_i^1 = g_i + \Delta t h_i + \frac{\gamma^2}{2} \left(g_{i+1} - 2g_i + g_{i-1} \right) + \frac{f_i^1}{2} \tag{3.5}$$

3.3.1.1 Courant number

The Courant number, also known as the Courant–Friedrichs–Lewy condition, is a dimensionless quantity that specify how fast a solution will propagate in one time step in terms of fractions of the cell size. Thus, for general convergence and stability in diffusion problems it is required that the Courant number is less than one.

3.3.1.2 Stencils

When referring to the discrete Laplacian it is often useful to represent it in terms of its stencil, which easily generalizes when using higher order approximations and/or higher dimensions. The above Laplacian approximation would be represented as $\frac{1}{h^2}(1, -2, 1)$, where the coefficients (weights) correspond to the factors seen in the

numerator in equation (3.3). The same coefficients also appear in equation (3.4), the time stepping rule. Higher order approximations improve the accuracy but leads to larger stencils which are costlier to compute.

Furthermore, the above analysis is easily generalized to higher dimensions, and the general two dimensional, second order stencil is given by

$$\frac{1}{h^2}\begin{bmatrix} c_1 & c_2 & c_1 \\ c_2 & c_3 & c_2 \\ c_1 & c_2 & c_1 \end{bmatrix} \tag{3.6}$$

where $c_2 = 1 - 2c_1$ and $c_3 = 4c_1 - 4$. The two-dimensional stencils evaluated in this report are listed in table 3.1. Similarly, the general 3D stencil is given by

$$\frac{1}{h^2}\begin{bmatrix} c_1 & c_2 & c_1 \\ c_2 & c_3 & c_2 \\ c_1 & c_2 & c_1 \end{bmatrix}\begin{bmatrix} c_2 & c_3 & c_2 \\ c_3 & c_4 & c_3 \\ c_2 & c_3 & c_2 \end{bmatrix}\begin{bmatrix} c_1 & c_2 & c_1 \\ c_2 & c_3 & c_2 \\ c_1 & c_2 & c_1 \end{bmatrix} \tag{3.7}$$

where $c_3 = 1 - 4c_1 - 4c_2$ and $c_4 = 16c_1 + 12c_2 - 6$. [14]

Table 3.1: 2D stencil schema names and their abbreviations

Stencil scheme	c_1-value
5-point standard (std)	0
9-point isotropic (iso)	1/6
9-point mixed (mix)	1/4
9-point interpolated wideband (iwb)	1/3
9-point extrapolated (ext)	-0.084

3.3.2 Spectral methods

A different approach in solving the wave equation is to approximate the Laplacian using spectral methods. By transferring the problem to the frequency domain the differentiation can be replaced with a simple multiplication. Denoting the Discrete Fourier Transform (DFT) by $\mathcal{F}$, the Laplacian may be approximated with

$$\nabla^2 u \approx \mathcal{F}^{-1}\left(-k^2\mathcal{F}(u)\right),\ k = -\sum_{d=1}^{D}\left(\frac{2\pi}{L_d}i_d\right)^2 \tag{3.8}$$

where k is a scaling factor incurred by the DFT related to the dimension D of the problem[15].

In order for this approximation to hold however, it is generally assumed that the wave field should be periodic. If the field is non-periodic and/or discontinuous using this method will introduce errors causing wrap-around effects where waves exiting from one end of the domain enter from the opposite end. On the other hand, one key advantage in using this method over the finite difference is that a coarser mesh

can be used when sampling the wave field, the spectral method allows for a sampling size h approaching the Nyquist limit[16] (). An additional benefit to this method is that there exists highly optimized algorithms for calculating the DFT, namely the Fast Fourier Transform (FFT)[17].

3.3.3 Finite element methods

In addition to the finite difference and spectral methods for solving high frequency wave propagation problems, there are also the finite element method (FEM)[18] and boundary element method (BEM)[19]. One of the main strengths of these methods is the possibility to dynamically adapt the mesh over the simulation domain/object and for instance use smaller and/or differently shaped cells where greater precision is needed.

BEM is a variant of FEM which uses a boundary integral formulation that assumes a homogeneous medium and represents the domain field values through values only on the boundary. Thus, BEM is mainly used in exterior and FEM in interior scattering problems. However, these two methods are mainly used for solving the steady-state wave equation, and not the full time-domain wave equation. [16]

3.4 Dispersion

In numerical modeling and simulation it is important to consider and be aware of the effect of dispersion. Numerical dispersion occurs in all modeling methods and is primarily caused by the modeled fields being approximated in some way, and not being an exact representation of the actual fields. Generally, dispersion errors tend to smooth strong or sharp variances in the signal.

A common realization of this is a frequency-dependent phase velocity, specifically lower frequencies travel slower than faster ones, and thus the modeled signal is flattened or smeared out. A non isotropic mesh, i.e. the number of cells per wavelength is not equal in every direction, may also cause dispersion - in different amounts depending on the direction of the wave. See figure 3.3 for an example on how dispersion may effect the solution in a simulation.

In time dependent methods (such as solving the wave equation) the time stepping scheme is also a source of dispersion. In this form however, it is primarily causing a direction independent phase error in the propagation of the field.

The amount of dispersion is often tightly coupled with the computational complexity of the discretization method. Achieving low numerical dispersion requires high computational density and vice versa, particularly when using the FDM. [20]

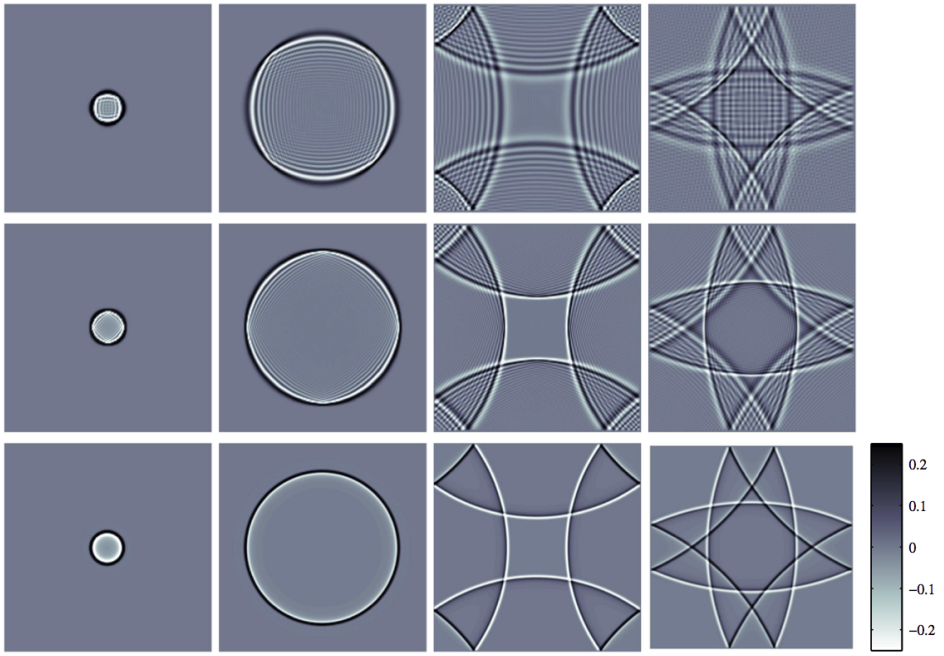


Figure 3.3: Example effect of dispersion in a simulation. The figure shows, from left to right, a snapshot from a 2D simulation at time steps 0.03, 1.50, 3.5, and 5.0 ms respectively. The different discretization stencils, 5-point standard (top) and 9-point interpolated wideband (middle) are shown together with the exact solution (bottom). [21]

3.5 Computational considerations

There are several factors that needs to be taken into account when implementing a high-performance simulation framework. In order to gain maximum performance special care must be taken in regards to the program structure, what data types are used and how they map to the underlying architecture and hardware that perform the calculations.

Nevertheless, FDM modeling is inherently computationally heavy, especially at an automotive scale at ultrasonic frequencies. For instance, keeping track of all the field values in a $1\,\mathrm{m}^3$ cube discretized with $1\,\mathrm{mm}^3$ cells requires 7.45 GB of memory, while computing the discrete 3D Laplacian using a 27-point stencil on the entire field requires upwards 30 billion floating point operations[22].

3.5.1 CPU vs. GPU

Broadly speaking, central processing units (CPUs) are very well suited for general purpose, sequential computation while graphics processing unit (GPUs) have been optimized to run a simple operation on data in parallel. While CPUs might have a few separate cores that can handle a few software threads[2] at a time, GPUs

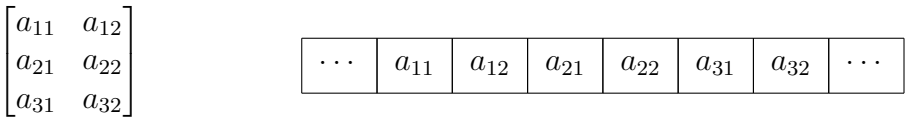
[2]A thread is in essence a sequence of instructions that the processor carries out.

today may have several hundreds of cores that can process thousands of threads simultaneously.

GPU-based computing have increased in popularity the last few years and graphic cards have proven to be well suited for numerical calculations. High end GPUs can achieve up to a several thousand billion floating point operations per second (FLOPS) compared to high end CPUs at a few hundred billion FLOPS[23]. However, code written to run on CPUs can generally not be run on GPUs without large modifications.

3.5.2 Memory layout

Computers generally store data in a linear fashion in its physical memory. This has consequences when a program has to store non-linear data such as 2D- or 3D-matrices; usually what happens is that the matrix is stored row by row (or column by column). See figure 3.4 for an example.



(a) 2D matrix **(b)** Memory layout

Figure 3.4: How a 2D matrix is stored in memory using row-major ordering.

Without going into much detail on how CPUs operate on data (L1-L3 caching, cache misses etc.) one can in general say that accessing and operating on data (i.e. numbers) is faster the closer together the data is stored in memory.

This affects the performance when calculating the discrete Laplacian for large, multidimensional matrices. In the 2D case using the standard 5-point stencil to compute the differential value at a single point, three of those points are stored sequentially but two of them are stored at a positive/negative offset equal to the width (or height) of the matrix.

3.5.3 Parallelization

The structure of the FDM formulation of the wave equation problem is easily parallelized since in each time step the new field value of a certain cell only depends on its closest neighboring cells. This part, in particular the discrete Laplacian, is among the most compute intensive components of the FDM-based algorithm and by distributing the computation the simulation can be sped up greatly. However, achieving good parallelization is non-trivial and customizing the parameters to the hardware is vital for high performance. [22]

3.5.3.1 MPI

Message Passing Interface (MPI) is a specification of a communication standard for distributed computing and several implementations of it exist. The main goal of MPI is to "establish a portable, efficient, and flexible standard for message passing that will be widely used for writing message passing programs"[24]. In more practical terms this translates to a toolbox that provides abstractions which enables programmers to better utilize distributed computing resources, be it several cores on a single CPU, a high number of CPUs spread out on separate machines in a data center, or even a combination of the two scenarios.

To take advantage of the inherent parallel structure of the FDM formulation the domain can be divided into sub-domains, each of whose solution can be calculated independently. MPI can be used to facilitate the communication between neighboring cells on the interface between two such sub-domains. Figure 3.5 illustrates how 3D-data typically is divided into sub-domains. Note that special care is usually taken so that the memory layout in each of the sub-domain's largest dimension is contiguous.

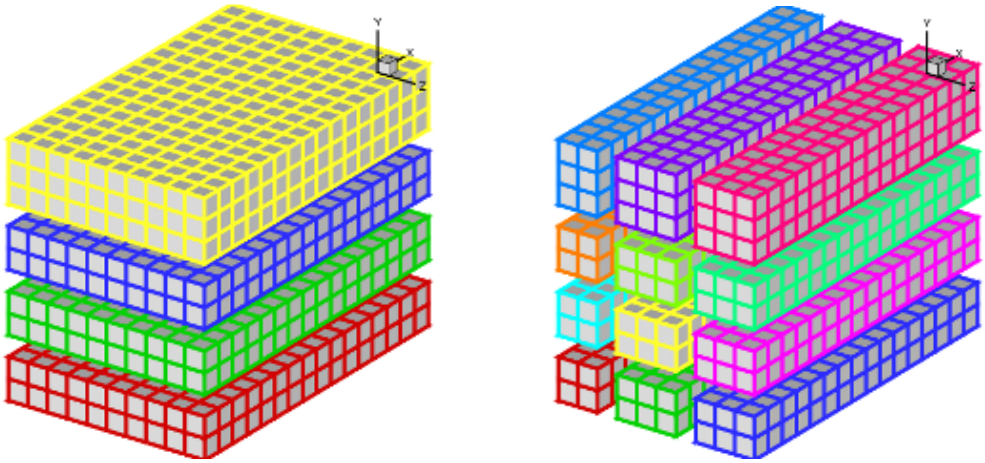


Figure 3.5: Typical 1D- and 2D-decomposition of 3D data. Each (same colored) continuous piece of data is processed separately. To the left is a so called 'book' or 'slab' decomposition along the Y-axis, while to the right a 'pencil' or 'drawer' decomposition along the YZ-plane is shown. [25]

3.5.3.2 OpenMP

A different approach in writing parallel code is to use the Open Multi-Processing (OpenMP)[26] compiler extension. The basic idea is that by using special comments around blocks of code the compiler can instruct that portion of the code to be run simultaneously on multiple threads (possibly on different processors). This is especially useful in loop constructs where each iteration can be performed independently. By using OpenMP the user don't have to deal with low-level details such as: iteration space partitioning, thread creation, scheduling, data sharing, and synchronization[27].

Its greatest benefit is that it allows for adding parallelism to existing source code without having to do major rewrites, which means that it is not as difficult as MPI to debug. However, the resulting code is not as portable as it requires a shared memory computer, which may be a problem for very large FDM simulations, and ultimately performance is generally sacrificed by yielding control of the system's memory management to the compiler.

3.6 Programming language considerations

When writing and designing algorithms for numerical simulations special care has to be taken in order for the resulting code to be performant. A major factor in this is the language of choice. Up until the most recent years a very large part of high-performance numerical code have been written almost exclusively in low-level programming languages as Fortran or C. Recently however, Python – a high-level language – have gained traction in scientific computing due to its ease of use and its wide range of available tools and packages.

In addition, to fully utilize the compute capabilities of clusters or processors with multiple cores the language is first of all required to support it. The most widely used parallelization techniques for high-performance computing, MPI and OpenMP, are foremost available in C and Fortran. When it comes to GPU calculations, writing code that runs on GPUs requires even more consideration due to its parallel architecture and is to a greater extent tied to the exact specifications of the graphic card in use.

In this work Python was used for quick experimenting and prototyping while C or Fortran – either included and hidden away in Python packages such as Numpy or Scipy, or reimplemented on an as-needed basis – handled underlying numerical calculations.

4

Method

Initially, a script for two-dimensional simulation was developed in Python to evaluate different implementation methods and algorithm design choices. The discrete Laplacian compute step was quickly determined to consume the majority of the run time and various implementations of it was tested for performance. By offloading all numerical, non-setup code to a Fortran program the computation time was further reduced.

In the projects next phase the numerical computation was extended from 2D to 3D and kept in a separate Fortran module. Since the 2D implementation was limited to rectangular obstacles in its simulation domain methods for importing 3D CAD models were studied next. To enable large scale simulations on a computer cluster a 2D domain decomposition scheme and efficient non-blocking communication structure via MPI was investigated and implemented.

Lastly, an API for specifying and saving different simulation scenarios was implemented together with a main Fortran program for simulating the scenario files. Basic visualization scripts were also developed to aid in presenting the generated data.

4.1 Framework overview

The framework consists of two major parts, a scenario generation script and a standalone simulation program. The scenario generation script is used to define the simulation domain, which then can be passed to the simulation program.

4.1.1 Scenario generation

The domain parameters are saved to a file generated by a separate Python script. It defines a so-called scenario which can then be simulated by the main program. The scenario requires the following parameters to be specified:

- The domain width, depth, and height
- The frequency of the ultrasonic sensor(s)

- The Courant number
- The number of cells per wavelength
- One or more CAD models plus their insertion coordinates
- One or more coordinates where ultrasonic sources shall be placed
- Zero or more coordinates where the field value should be saved (every iteration)
- How often the air pressure field data should be saved, if at all

4.1.2 Simulation program

The simulation program is written in Fortran and utilizes domain decomposition via MPI for increased performance. The scenario file is specified as a command line argument. Pseudocode for the program is shown in listing 4.1.

Listing 4.1: Structure of the simulation program. The MPI steps were omitted in a first implementation.

```
Load domain parameters from file
Initialize MPI domain decomposition
Initialize field state u_0, u_1

for n from 2 to the number of time steps
    Calculate the new field state u_n from u_n-1 and u_n-2
    Add field correction at obstacle boundary cells in u_n
    Communicate between MPI instances
    Save specified field data
end for
```

4.2 Discretization strategy

4.2.1 Spatial

The simulation domain is limited to a rectangular box with dimensions L_x, L_y, and L_z. This is internally represented as an $N_x \times N_y \times N_z$ matrix, with $N_d = \lceil L_d/h \rceil$ and where h is generally referred to as the cell size. Furthermore, the cell size is in turn defined as a parameter in terms of the number of cells per wavelength (CPW), which in similar simulations/models range from 4-20, very much depending on other parameters and the underlying discretization method[28, 29, 30]. The corresponding matrix index (i, j, k) of a spatial coordinate at (x, y, z) is given by

$$(i, j, k) = (\lfloor x/h \rfloor, \lfloor y/h \rfloor, \lfloor z/h \rfloor). \tag{4.1}$$

It is assumed that the first element in each dimension has index zero.

One of the implications of this is that a range of coordinates maps to the same index, i.e. cell, in the data matrix. This can be illustrated using a 1D example (the 3D case is entirely analogous), let $L_x = 1\,\text{m}$ and $h = 0.25\,\text{m}$, yielding $N_x = 4$. This means that along the x-axis ranging from 0 to L_x meters where one in reality could expect a continuous band of numerical values of some sort (air pressure in our case), one only finds 4 discrete data points in the model. This corresponds to the 4 averaged values along the uniform segments $0 \leq x < h$, $h \leq x < 2h$, $2h \leq x < 3h$, and $3h \leq x < L_x$. Put another way, the field values in any discretized cell is collapsed into a single point value in the model.

4.2.2 Temporal

The time is discretized in equal steps of Δt. It is not set explicitly but instead determined by the Courant number γ, cell size h, and speed of sound c according to $\Delta t = \gamma h/c$.

4.3 Implementing reflections

When the simulated wave reaches the domain boundaries or an obstacle it should be reflected. It is assumed that this reflection occurs without absorption, this is normally referred to as zero Neumann boundary conditions. To enforce this, the field value in cells on the boundary (of the domain or obstacles) will be updated using a modified stencil. Recall that the stencil is used for calculating the discrete Laplacian, which is used in the update rule. The modified stencil is simply mirrored around its center for those weights that correspond to cells outside the domain or cells on an obstacle.

An example should clear this up: In the 1D case, remembering that the original stencil was $\frac{1}{h^2}(1, -2, 1)$, if an obstacle should be present to the left of the current cell, the modified stencil correctly adjusting for the reflection when summed and added to the already calculated field value will be $\frac{1}{h^2}(-1, 0, 1)$. The result of using the modified stencil can been seen by combining the two: $\frac{1}{h^2}(0, -2, 2)$, the influence of the field value to the left is zero and instead the influence of the field value to the right is doubled. This corresponds to Neumann boundary conditions where no absorption is present and the energy is conserved.

Note that calculating the new field state is actually done in a two-step process. First, the field state is updated using the standard stencil over the entire domain as if no obstacles were present. Secondly, the new field value at each cell neighboring a boundary is adjusted by adding a correction weight calculated from the modified stencil discussed above.

4.4 Source signal

The ultrasonic signal is modeled as a sine wave. A Gaussian envelope is applied to mimic a wave packet with a short amplitude build-up, as may be emitted from a ultrasonic source. The signal depends on the frequency f, a time delay d to ensure zero initial state, and σ, the standard deviation or Gaussian RMS width:

$$s(t) = \exp\left(-\frac{(t-d)^2}{2\sigma^2}\right)\cos\left(2\pi f(t-d)\right) \tag{4.2}$$

Both the time delay and standard deviation are typically set in relation to the frequency, for example to $d = 5/f$ and $\sigma = 1.5/f$.

An alternative to the modulated sine wave is the Ricker wavelet. It is the second derivative of a Gaussian and has the property that it consists of a wide spectrum of frequencies in contrast to a harmonic wave[31]. The Ricker function is given by

$$s(t) = \left(1 - r(t)^2\right)\exp\left(-r(t)^2\right), \quad r(t) = \pi f(t-d) \tag{4.3}$$

where f is the peak frequency and $d = 3/f$ is a time delay. It is mainly used in geophysics where it allows for wide band analysis in a single simulation[32].

4.5 Importing 3D models

4.5.1 The stl data format

In order to achieve realistic and/or meaningful simulations it is necessary to be able to include 3D models (of for example cars, pedestrians, or other obstacles found in parking scenarios) in the simulation domain. One such data format for storing 3D models is the stl file format.

The stl (derived from 'stereolithography') file format is a data format widely used in modeling environments. It describes the surface geometry of three-dimensional objects by mapping it to a series of connected triangles. As the triangles become smaller the surface is more accurately captured, however the file size of the model grows as well. The triangles are fully defined by the coordinates of their three vertices and surface normal[33]. Listing 4.2 shows the raw data structure of a stl file in its ASCII form, while figure 4.1 shows an actual stl model viewed in a CAD program.

Listing 4.2: Basic structure of the ASCII stl format.

```
solid <name>
...
facet normal ni nj nk
    outer loop
        vertex v1x v1y v1z
        vertex v2x v2y v2z
        vertex v3x v3y v3z
    endloop
endfacet
...
endsolid <name>
```

Figure 4.1: Example stl model. The car is built up with almost 4 million triangle surfaces. Model courtesy of Volvo Cars.

4.5.2 Data pre-processing

To incorporate stl models in the simulation program their triangle/vertex representation of surfaces must be transformed to the chosen 3D matrix structure. This is accomplished by mapping each vertices' coordinate to its corresponding matrix index with equation (4.1). It should be noted that in order to produce a high quality discretized model the original stl model should be of high enough resolution. A high enough resolution in this context means that no triangle side should be larger than the chosen cell size, otherwise the discretized model will have gaps in its surface.

5

Result

In all simulations, unless other specified, the speed of sound was set to 340 m/s, the frequency to 55 kHz, and the Courant number to 0.5. The number of cells per wavelength is abbreviated CPW.

5.1 Stencil performance

Early in the project different Python packages were evaluated for their stencil computation performance, as shown in table 5.1. The differences in running time are perhaps not as large as one would expect, but could be attributed to the fact that all packages more or less perform the calculations using compiled C (or Fortran) code internally. With these results and future parallelization (MPI) in mind, Fortran became the language of choice for numerical calculations.

Table 5.1: Stencil performance evaluation. The table lists the running times for computing the discrete 2D Laplacian using the 5-point standard stencil. The matrices were populated with random (0,1)-uniform data.

Matrix size	$10^3 \times 10^3$		$10^4 \times 10^4$	
Method	Time [s]	Scaled	Time [s]	Scaled
Fortran (via Cython)	0.005	1.00	0.486	1.00
C (via Cython)	0.005	1.00	0.529	1.09
Scipy weave	0.006	1.20	0.603	1.24
Fortran (via f2py)	0.006	1.20	0.677	1.39
Numba jit	0.009	1.80	0.944	1.94
CUDA[1]	0.019	3.80	1.610	3.31
Numpy	0.025	5.00	2.019	4.15

[1]Includes overhead incurred in transferring the data to and from the graphic card.

5.2 Accuracy & dispersion

The 2D stencils listed in table 3.1 were evaluated and their accuracy and dispersion compared. To study the relationship between dispersion and CPW a wave packet was simulated in two differently sized domains with no obstacles, 30 cm by 30 cm (Case 1) and 95 cm by 95 cm (Case 2), at 8, 16, 32, 64, and 128 CPW. The simulated wave packet at 128 CPW was taken as the reference, representing the correct result. Figure 5.1 shows the normalized root mean square (RMS) error between the reference wave and the remaining waves at lower CPW. As seen in the figure, the analysis is repeated for grid angles at 0, 30, and 45 degrees. The grid angle is defined counter-clockwise from the x-axis in the 2D plane.

The difference in error depending on the grid angle highlights the dispersion and wave propagation angle relationship, and it is here the stencils reveal their strengths and weaknesses. The main concern in choosing a suitable stencil for general simulations is that dispersion is minimized, especially propagation angle dependent dispersion. As seen in figure 5.1 the stencils perform rather similarly with only small differences in dispersion error among them at higher CPW.

The effect of dispersion on the wave packets can be seen more clearly in figure 5.2, where the pulse shape is shown at the two measured distances in Case 1 and 2. Another noteworthy property of the simulations is, as expected, that the computational effort grows very quickly when increasing the domain dimensions and the CPW. Table 5.2 lists the running times for these simulations. The computational effort is tightly tied to the dimensionality of the simulation domain and 3D simulations will have an even worse performance scaling.

Table 5.2: Listed is the computation time for the simulations in figure 5.1. Note that doubling the CPW requires roughly 8 times the computational effort.

CPW	Case 1 [s]	Case 2 [s]
8	1.57	132
16	13.1	1070
32	111	7870
64	881	60500
128	7220	512000

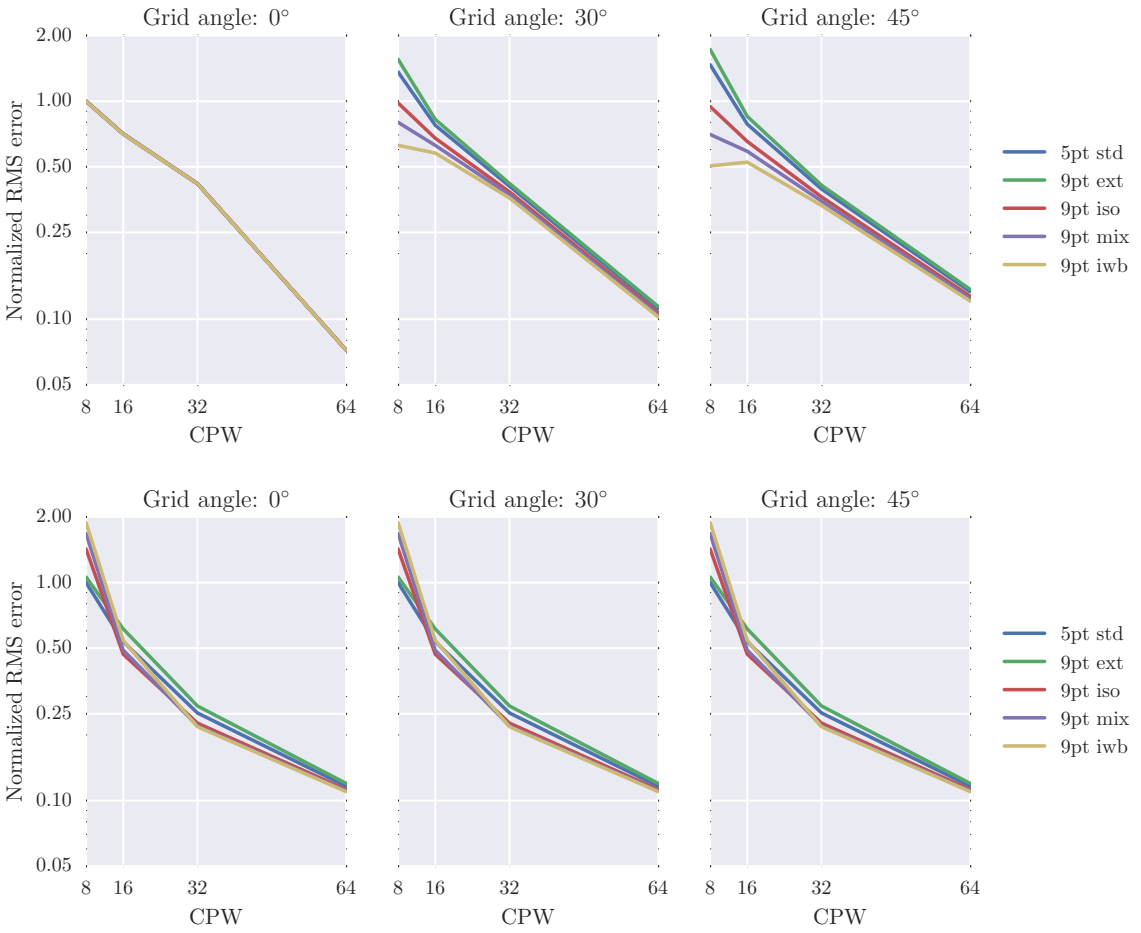


Figure 5.1: The figure shows the relationship between RMS and CPW at different grid angles for wave packets that have been simulated using five different stencils. The top row shows data generated in Case 1 while the bottom row shows the same from Case 2 (see additional description in previous section). Note that the y-axis is scaled logarithmically and that the data overlap in the top left figure.

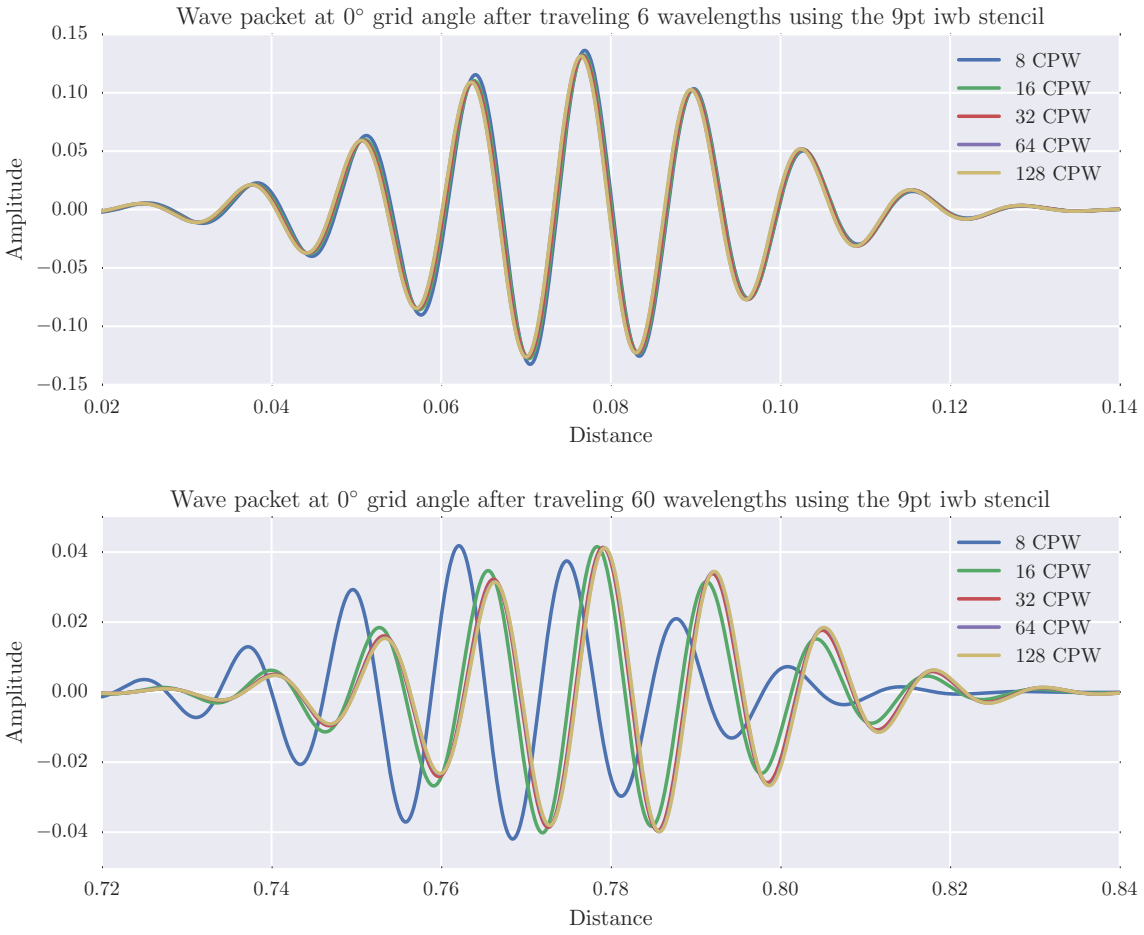


Figure 5.2: The figures illustrate the shape of the modulated sine wave at different CPW and time steps. The top figure is from the Case 1 simulation while the bottom from Case 2. The effect of dispersion is seen clearly in the second case at longer distances, evidently caused by the lower CPW. The distance is given in meters and the maximum amplitude was normalized to unity when emitted at the source.

5.3 Spectral methods

Using the DFT proved very promising in simple domains without obstacles, where it outperformed the FDM by achieving lower dispersion at lower CPW and at the same time having a lower computational cost. However, as obstacles were added to the domain the side effects of no longer having a symmetrical/periodic wave became apparent. High frequency noise was introduced and greatly amplified when using the Ricker source signal and using phase-neutral low-pass filters to combat the noise was investigated but quickly dismissed due to the added complexity and substantial extra computational cost. In figure 5.3 snapshots from simulations based on the FDM and DFT methods are compared; similar negative effects are seen when using the modulated sine signal in the DFT case, where the reflected wave is both erroneously duplicated and mirrored along the obstacle.

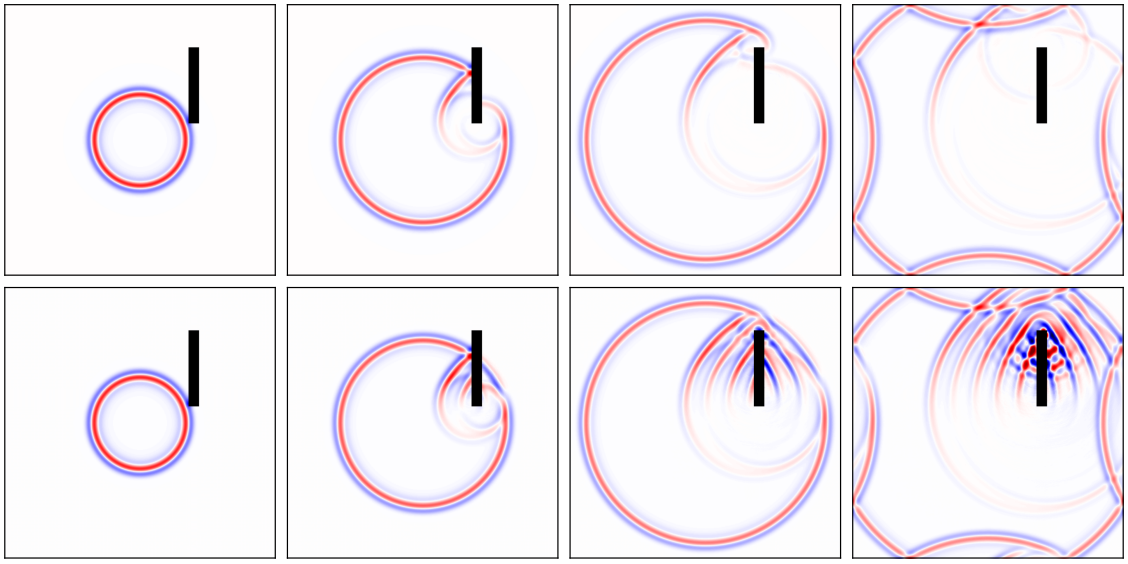


Figure 5.3: Comparison between simulations when using the finite difference method (top) and the discrete Fourier transform (bottom) for approximating the discrete Laplacian. From left to right snapshots of the solution at time steps 10, 15, 20, and 25 ms are seen. The simulation domain was 25 cm by 25 cm, Courant number 0.45, and CPW 16.

5.4 3D simulations

The 3D simulations are based on the finite difference method and the discrete Laplacian was approximated using the 27 point isotropic stencil given in equation (3.7), letting $c_1 = 1/30$ and $c_2 = 1/10$. This was done to minimize the direction dependent dispersion as discussed in section 5.2.

Example results from a 3D simulation using the developed simulation framework are shown in figure 5.4. Note that these types of figures, though perhaps difficult to

decipher, should solely be used for illustrative purposes. Due to time constraints the simulation had to be done at a sub-ultrasonic frequency, 17 kHz. The CPW was set to 8 and the simulation domain was 4.0 m by 3.0 m by 2.5 m (wide/deep/high).

Under normal (intended) framework usage the main interest is the field pressure under the length of the simulation, specifically at the driving sensor, and possibly also at any neighboring sensors. The estimated distance to the obstacle may then be calculated, and obstacle location potentially triangulated.

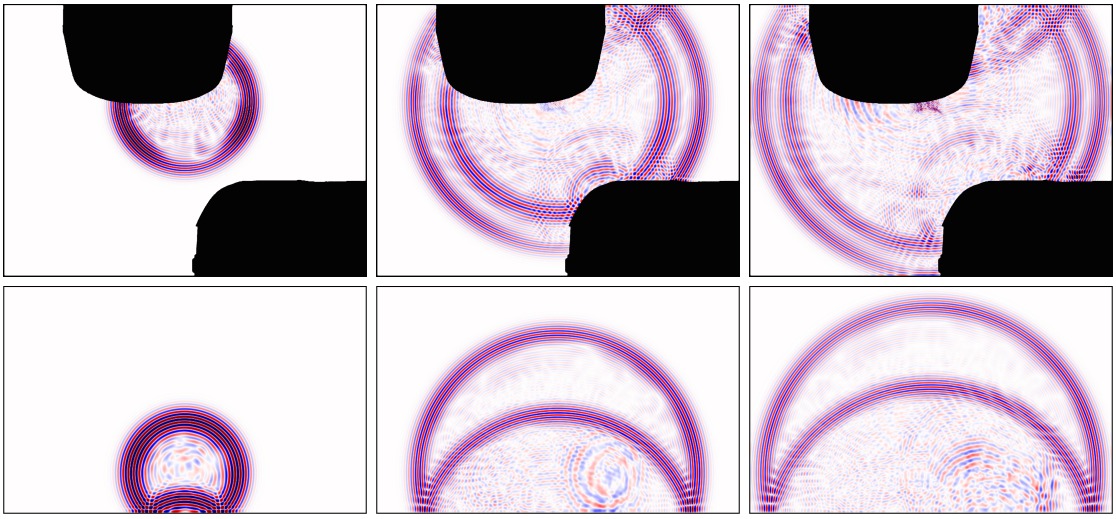


Figure 5.4: The figure shows 2D slices of the air pressure field in a 3D simulation at three succeeding time steps. Shown top is a cut in the XY-plane 15 cm above the ultrasonic source, at bottom a cut in the XZ-plane midway between the two cars.

5.5 Framework design

An easy to use interface has been implemented and a straightforward usage pattern have been devised:

1. Generate a scenario file
2. Run the simulation program
3. Analyze saved data

A Python script using the constructed API is shown in listing 5.1. The API have been designed to hide away complexity tied to the actual simulation code and its parameters to provide an easy to use, high-level interface.

Listing 5.1: Scenario generation

```
#!/usr/bin/env python

from ultrasound import Domain

"""This script shows how to create scenario files with the Domain class.

In summary:
  * A rectangular domain and simulation parameters are defined
  * Two cars are added, one rotated 90 degrees around its z-axis
  * Two ultrasonic sources are placed near the first car
  * The air pressure matrix should be saved every 50:th iteration
  * The scenario file is saved to 'example_scenario.txt'

Below is a 2D schematic showing the domain layout. Note
that models may be placed partially outside the domain.

       Car 1 origin, domain
       | coordinate (2,-3,0)
       |
       +-> +---------+
           |         |                    Coordinate
           |         |                     system
           |         |                    +--> X
+ Domain   |  Car 1  |                    |
| origin   |         |                    v
v (0,0,0)  |         |
           |         |                    Y
+----------------------------------+
|          |         |             |
|          +--x---x--+             |
|             ^   ^                |
|     Ultrasonic sensors           |
|                                  |
|                                  |
|                   +--------------|-----+
|                   |              |     |
|                   |    Car 2     |     |
+----------------------------------+     |
                    |                    |
                +-> +--------------------+
                |
                Car 2 origin, domain
                 coordinate (3,3,0)
"""

domain = Domain(dimensions=(5.0, 3.0, 2.0), source_frequency=55000,
                cells_per_wavelength=12, courant_number=0.5)
domain.add_model(file_path='car_1.stl', coordinate=(2, -3, 0))
domain.add_model(file_path='car_2.stl', coordinate=(3, 3, 0),
                 rotation_angle=90, rotation_axis=(0, 0, 1))
domain.add_sources(coordinates=[(2.25, 1, 0.5), (2.75, 1, 0.5)])
domain.export_field(every_nth_iteration=50)
domain.save_scenario(file_path='example_scenario.txt')
```

6

Discussion & Conclusion

In this thesis methods of simulating the wave equation at ultrasonic frequencies have been studied and evaluated, and a proof-of-concept framework supporting accurate large scale 3D simulations have been developed. The high level framework enables engineers to study the dynamics of using ultrasonic sensors for obstacle detection and distance estimation, and with a greater understanding of its limitations and strengths, more reliable and safer sensor systems can be built.

One of many possible ways this framework can be used is to create a test suite of a wide range of simulation scenarios. This test suite may then be used, for instance, in the design process of optimizing the location of the ultrasonic sensors. However, the developed framework is by no means fully polished nor perfect, but provides a solid foundation that can be extended and built upon.

As for a last note, running high precision ultrasonic simulations require substantial computational effort, mainly due to the combination of modeling small, millimeter scale elements in large, meter sized domains. Regular parking scenarios typically demand more computing resources than available in typical desktop computers – the limiting factor is memory requirements – so that these types of simulations must currently be run on clusters of 10+ machines. It is though very likely that with the current rate of hardware advancements that it will be possible to run such simulations on high end desktop computers in the next few years.

6.1 Future work

A possible avenue for future work involves improving the modeling of the ultrasonic sensors. In their current form they behave as point sources that is rather unphysical. Real ultrasonic sources are directional – the emitted waves have peak intensity when aligned with the sensor. The actual shape of the wave packet may also not be consistent with what is modeled; however this should only have a very minor effect when the data is analyzed. Another concern, which perhaps is more related to the data analysis than the modeling, is that ultrasonic sensors have limited sensitivity. Thus it is needed to decide at what fraction of the initial amplitude a reflected wave should no longer count as detected.

One potentially major difference between the simulation model and real world sce-

narios is the lack of modeling temperature variations. The majority of internal combustion cars expel hot exhaust gases at the rear of the car, often in close vicinity any ultrasonic parking sensors. As the air temperature affects the wave propagation speed it may cause the estimated obstacle distance to become inexact. This is particularly important in parking scenarios where the speed of the car is low and the surrounding space often is limited, both of which hinders dispersion of the heat.

Absorption will need to be considered in future models to accurately simulate reflections from cloth, rubber and other soft or irregular surfaces. Typical examples of these include pedestrians, animals, and vegetation. Likewise attenuation, which is not taken into account in the current model, may cause the model to report inaccurate and possibly somewhat optimistic obstacle detection ranges.

Bibliography

[1] Margery Conner. "Automobile sensors may usher in self-driving cars". In: *EDN* 56.10 (2011), p. 42.

[2] Kenneth Harold Waters. "Reflection Seismology: A tool for energy resource exploration". In: (1987).

[3] J Ripperger, H Igel, and J Wasserman. "Seismic wave simulation in the presence of real volcano topography". In: *Journal of volcanology and geothermal research* 128.1 (2003), pp. 31–44.

[4] John T Etgen and Michael J O'Brien. "Computational methods for large-scale 3D acoustic finite-difference modeling: A tutorial". In: *Geophysics* 72.5 (2007), SM223–SM230.

[5] Lauri Savioja. "Real-time 3D finite-difference time-domain simulation of low- and mid-frequency room acoustics". In: *13th Int. Conf on Digital Audio Effects*. Vol. 1. 2010, p. 75.

[6] Nikunj Raghuvanshi et al. "Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes". In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 4. ACM. 2010, p. 68.

[7] Dennis M Sullivan. *Electromagnetic simulation using the FDTD method*. John Wiley & Sons, 2013.

[8] Wikimedia Commons. *Sonar Principle — Wikipedia, The Free Encyclopedia*. 2008. URL: http://commons.wikimedia.org/wiki/File:Sonar_Principle_EN.svg (visited on 05/18/2015).

[9] Department of Physics C.R. Nave and Georgia State University Astronomy. *Sound Speed in Gases*. 2014. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/sound/souspe3.html (visited on 05/18/2015).

[10] SICK. *Distance Sensors: Short range (displacement)/mid range/long range distance sensors, linear measurement sensors, ultrasonic sensors, double sheet detector, optical data transmission, position finders*. 2012. URL: https://www.mysick.com/saqqara/pdf.aspx?id=im0045108 (visited on 05/17/2015).

[11] Texas Instruments. *Automotive ultrasonic sensor interface for park assist or blind spot detection systems*. TIDA-00151. 2014. URL: http://www.ti.com/lit/ug/tidu327/tidu327.pdf (visited on 05/18/2015).

[12] Bosch. *Automotive Electronics: Parking made easy – Parking assistance systems from Bosch*. 2014. URL: http://www.bosch-mobility-solutions.us/media/db_application/downloads/pdf/comfort_1/en_5/einparkenleichtgemachtparkassistenzsystemevonbosch.pdf (visited on 05/18/2015).

[13] Parallax. *PING: Ultrasonic Distance Sensor*. #28015. 2014. URL: https://www.parallax.com/sites/default/files/downloads/28015-PING-Documentation-v1.6.pdf (visited on 05/18/2015).

[14] Michael Patra and Mikko Karttunen. "Stencils with isotropic discretization error for differential operators". In: *Numerical Methods for Partial Differential Equations* 22.4 (2006), pp. 936–953.

[15] Steven G Johnson. *Notes on fft-based differentiation*. 2011.

[16] Ravish Mehra et al. "An efficient GPU-based time domain solver for the acoustic wave equation". In: *Applied Acoustics* 73.2 (2012), pp. 83–94.

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

[18] Lonny L Thompson. "A review of finite-element methods for time-harmonic acoustics". In: *J. Acoust. Soc. Am* 119.3 (2006), pp. 1315–1330.

[19] Robert D Ciskowski and Carlos Alberto Brebbia. *Boundary element methods in acoustics*. Springer, 1991.

[20] Shah Muhammad Raiyan Kabir et al. "Elimination of numerical dispersion from electromagnetic time domain analysis by using resource efficient finite element technique". In: *Progress In Electromagnetics Research* 137 (2013), pp. 487–512.

[21] Brian Hamilton and Stefan Bilbao. "Fourth-order and optimised finite difference schemes for the 2-D wave equation". In: *Proc. 16th Conference on Digital Audio Effects (DAFx-13)*. 2013.

[22] Shah M Faizur Rahman, Qing Yi, and Apan Qasem. "Understanding stencil code performance on multicore architectures". In: *Proceedings of the 8th ACM International Conference on Computing Frontiers*. ACM. 2011, p. 30.

[23] Takayuki Aoki. "Application Performances on Many-core Processors: Xeon Phi versus Kepler GPU". 2013.

[24] Lawrence Livermore National Laboratory Blaise Barney. *The Message Passing Interface*. 2014. URL: htts://computing.llnl.gov/tutorials/mpi/ (visited on 05/15/2015).

[25] N. Li and S. Laizet. "2DECOMP&FFT – A highly scalable 2D decomposition library and FFT interface". In: Cray User Group 2010 conference. Edinburgh, 2010.

[26] OpenMP Architecture Review Board. *OpenMP Description*. 2015. URL: http://openmp.org/ (visited on 05/01/2015).

[27] Intel Corporation. *Developer Zone: Parallelization Using OpenMP*. 2014. URL: https://software.intel.com/en-us/articles/parallelization-using-openmp (visited on 05/20/2015).

[28] U.S. Inan and R.A. Marshall. *Numerical Electromagnetics: The FDTD Method*. ISBN: 9781139497985.

[29] K.S. Kunz and R.J. Luebbers. *The Finite Difference Time Domain Method for Electromagnetics*. ISBN: 9780849386572.

[30] L. Tarricone and A. Esposito. *Grid Computing for Electromagnetics*. Artech House electromagnetic analysis series. ISBN: 9781580537780.

[31] Jade Rachele S. Garcia. “3D finite-difference time-domain modeling of acoustic wave propagation based on domain decompostion”. MA thesis. UMR Geosciences Azur.

[32] J Redondo et al. “Time domain simulation of sound diffusers using finite-difference schemes”. In: *Acta acustica united with acustica* 93.4 (2007), pp. 611–622.

[33] Wikipedia. *STL (file format) — Wikipedia, The Free Encyclopedia*. 2015. URL: `http://en.wikipedia.org/w/index.php?title=STL_(file_format)&oldid=657070763` (visited on 05/12/2015).