Computer Physics Communications 203 (2016) 298–308

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

openPSTD: The open source pseudospectral time-domain method for acoustic propagation☆

Maarten Hornikx[a,*], Thomas Krijnen[b], Louis van Harten[c]

[a] Building Physics and Services, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[b] Urban Science and Systems, Eindhoven University of Technology, The Netherlands
[c] Electrical Engineering, Eindhoven University of Technology, The Netherlands

ARTICLE INFO

Article history:
Received 23 July 2015
Received in revised form
8 February 2016
Accepted 25 February 2016
Available online 7 March 2016

Keywords:
Python
Blender
Linearized Euler equations
Fourier pseudospectral method
GPU

ABSTRACT

An open source implementation of the Fourier pseudospectral time-domain (PSTD) method for computing the propagation of sound is presented, which is geared towards applications in the built environment. Being a wave-based method, PSTD captures phenomena like diffraction, but maintains efficiency in processing time and memory usage as it allows to spatially sample close to the Nyquist criterion, thus keeping both the required spatial and temporal resolution coarse. In the implementation it has been opted to model the physical geometry as a composition of rectangular two-dimensional subdomains, hence initially restricting the implementation to orthogonal and two-dimensional situations. The strategy of using subdomains divides the problem domain into local subsets, which enables the simulation software to be built according to Object-Oriented Programming best practices and allows room for further computational parallelization. The software is built using the open source components, Blender, Numpy and Python, and has been published under an open source license itself as well. For accelerating the software, an option has been included to accelerate the calculations by a partial implementation of the code on the Graphical Processing Unit (GPU), which increases the throughput by up to fifteen times. The details of the implementation are reported, as well as the accuracy of the code.

Program summary

Program title: openPSTD v1.1 (v1.0 is the version without the GPU acceleration)
Catalogue identifier: AFAA_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AFAA_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: GNU General Public License, version 3
No. of lines in distributed program, including test data, etc.: 45339
No. of bytes in distributed program, including test data, etc.: 4139815
Distribution format: tar.gz
Programming language: Python (as comes with Blender 2.72).
Computer: Variable.
Operating system: Windows, Linux, Mac OS X.
RAM: From 250 MB for a typical geometry up to 1 GB for large geometries (with about 4M grid points)
Classification: 4.3, 12.
External routines: Blender 2.72, NumPy, SciPy, PyFFT, PyOpenCL, PyCUDA
Nature of problem: Sound propagation
Solution method: Fourier pseudospectral time-domain method

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: m.c.j.hornikx@tue.nl (M. Hornikx).

http://dx.doi.org/10.1016/j.cpc.2016.02.029

Restrictions: Structured grid, two dimensions, real-valued boundary conditions only

Unusual features: Implementation of code using Blender/Python including GPU acceleration; subdomain modelling within Fourier pseudospectral method

Running time: Depending on the dimension of the problem, calculation times take minutes up to hours

1. Introduction

In the built environment, prediction of the acoustics of indoor and outdoor scenarios is important regarding e.g. the evaluation of noise levels due to road traffic. Computing acoustics of indoor and outdoor spaces is traditionally carried out by simplified calculation methods relying on geometrical acoustic principles at a relatively low computational cost. A requirement for these methods to be valid is that spaces and surfaces should be large compared to the wavelength of interest. Acoustic descriptors of spaces could then reliably be predicted by these methods. An overview of these methods can be found in the literature [1]. The use of such methods does however exclude various acoustical aspects such as outdoor meteorological influences, modal effects in small spaces, diffraction effects at corners and around screens and a realistic auralization. Full wave-based models are needed if accurately modelling of these acoustical aspects is desirable, and these methods have received attention in recent years, see e.g. references in [2]. The limitations of full wave-based methods mostly relate to their computational complexity and the quality of the input data of the geometry and boundaries [2,3]. With the advances in computer power, as well as the developments of partly implementing wave-based acoustics methods on graphic processing units, see e.g. [4,5], full wave-based methods are getting more applicable for realistic geometries and higher frequencies. Besides, research to develop efficient wave-based methods will also boost its applicability, and recent efficient methods are efficient finite-difference time-domain (FDTD) schemes [6], the fast multipole boundary element method (FMBEM) [7], and the Fourier pseudospectral time-domain method (PSTD) [8]. For acoustic propagation problems involving an inhomogeneous propagation domain (e.g. atmospheric acoustics) as described by the linearized Euler equations, numerical solutions that discretize the whole domain are needed. From the available methods, the PSTD method is most attractive as it only requires two spatial points per acoustic wavelength for accurate results. Developments in the Fourier PSTD method towards sound propagation in the built environment include modelling of real-valued boundary media and a moving atmosphere [8], a local grid refinement [9] and the PSTD method in orthogonal curvilinear coordinates for moderately curved surfaces [10]. The Fourier PSTD method has been used for a variety of acoustic applications, such as a sports hall [3], exhaust pipe noise [11], green roofs and facades in an urban setting [12] and ground roughness [13].

This work presents an open-source implementation of the PSTD method accessible under the GPL license [14], intended to be used by all interested researchers for applications in the built environment, i.e. both for indoor and outdoor problems. The implementation is the first step towards a fully fledged open-source software for computing acoustic propagation with this wave-based solution method. The current implementation method allows for modelling a configuration in two dimensions composed out of rectangular sub-domains. The boundaries of the domain can either be assigned an arbitrary real valued acoustic surface impedance or contain conditions that completely absorb the acoustic waves. Software tools for other, less efficient, wave-based solution methods (FDTD) have recently been presented [15,16], but open source software for acoustic propagation based on the PSTD method has not appeared thus far. The unique features of openPSTD are that, for the first time, the highly efficient Fourier PSTD method is implemented in an open source software interface to solve non trivial problems. Also, the implementation using the Blender/Python programs is new as well as the way the geometrical subdomains are implemented in Fourier PSTD. Finally, the code is accelerated by a partial implementation of the code on the Graphical Processing Unit (GPU).

The paper is organized as follows. In Section 2, the partial differential equations governing sound propagation solved in openPSTD are explained. The technique to numerically solve these equations is then described. Section 3 details the computational architecture of openPSTD, its user interface, details of the computational implementation in Python and user options. The appropriateness of the software is demonstrated in Section 4. Finally, Section 5 concludes the paper and gives an outlook on ongoing developments in the numerical solution technique and future developments of the openPSTD software.

2. Physical and numerical method

2.1. Linearized Euler equations

Sound propagation in the built environment, involving configurations with arbitrary boundary conditions and effects of inhomogeneous media as caused by meteorological effects, can be described by the linearized Euler equations. These equations are derived from the Navier–Stokes equations, in which effects of molecular viscosity are neglected and all physical variables are decomposed into their background values, including convective and thermal effects, denoted by subscript 0, and acoustic fluctuations:

$$\begin{aligned} \rho_{\text{tot}} &= \rho_0 + \rho, \\ \mathbf{u}_{\text{tot}} &= \mathbf{u}_0 + \mathbf{u}, \\ p_{\text{tot}} &= p_0 + p, \end{aligned} \tag{1}$$

with ρ the density, $\mathbf{u} = [u_x, u_y, u_z]^T$ the velocity vector and p the pressure. For most applications in the built environment, amplitudes of the acoustic variables are small. The error, introduced by linearizing with respect to the acoustic variables, is therefore expected to be low. Furthermore, for the intended applications of sound propagation in the built environment, the propagation medium is considered to be incompressible and isentropic, and terms proportional to ∇p_0 can be ignored [17]. In this way, the linearized Euler equations (LEE) in non-conservative form are obtained:

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= -(\mathbf{u}_0 \cdot \nabla)\mathbf{u} - (\mathbf{u} \cdot \nabla)\mathbf{u}_0 - \frac{1}{\rho_0}\nabla p, \\ \frac{\partial p}{\partial t} &= -\mathbf{u}_0 \cdot \nabla p - \rho_0 c^2 \nabla \cdot \mathbf{u}, \end{aligned} \tag{2}$$

with c the adiabatic speed of sound. Computations are initiated by a non-zero pressure condition:

$$p(\mathbf{x}, t=0) = e^{-\beta|\mathbf{x}-\mathbf{x}_s|^2}, \qquad (3)$$
$$\mathbf{u}(\mathbf{x}, t=0) = 0,$$

with $\beta = 3e^{-6}c^2/\Delta^2$, $\mathbf{x} = [x, z]^T$ the coordinate vector, $\mathbf{x}_s = [x_s, z_s]^T$ the coordinate vector of the source and Δ the uniform grid spacing. This pressure condition ensures acoustic energy for all resolvable frequencies (determined by the chosen spatial discretization), and a decrease of the power with high frequencies such that the effect of aliasing is small. The factor β scales this initial pressure distribution to the highest resolved frequency.

Acoustic boundary conditions of configurations of interest can be set by defining the normalized surface impedance Z_s:

$$Z_s = \frac{p}{\rho_0 c u_p}, \qquad (4)$$

where u_p denotes the velocity component of the propagation direction of the sound wave incident to the surface. The acoustic absorption coefficient can be computed as:

$$\alpha(\theta) = 1 - |R(\theta)|^2,$$

with

$$R(\theta) = \frac{Z_s \cos\theta - \cos\theta'}{Z_s \cos\theta + \cos\theta'}, \qquad (5)$$

with θ the angle with the normal to the surface of the sound wave incident to the surface and θ' the angle with the normal to the surface of the sound wave propagating into the boundary medium. Many boundaries in outdoor acoustics can be treated by locally reacting porous media. For such locally reacting boundaries, $\theta' = 0$.

2.2. Fourier pseudospectral time domain method

Wave propagation in the openPSTD software is implemented according to the extended Fourier PSTD method as presented in [8]. With this numerical method, the linearized Euler equations (LEE) of Eq. (2) are solved. A geometry of interest as drawn in the openPSTD software (see Section 3) is discretized by a Cartesian mesh. The solution of Eq. (2) is discrete in time as well. At every discrete time step, the spatial derivatives of Eq. (2) are computed using the Fourier pseudospectral method and the equations are marched in time by a low-storage optimized 6-stage Runge–Kutta method. The analysis below is presented for a two-dimensional configuration and follows [8]. The Fourier pseudospectral method relies on an eigenfunction expansion which is based on the LEE for a non-moving medium.

$$\frac{\partial \mathbf{q}}{\partial t} = \left[jR_m^{-1}L\right]\mathbf{q}, \qquad (6)$$

with $\mathbf{q} = [u_x, u_z, p]^T$, and

$$L = \begin{bmatrix} 0 & 0 & j\partial_x \\ 0 & 0 & j\partial_z \\ j\partial_x & j\partial_z & 0 \end{bmatrix} \qquad R_m = \begin{bmatrix} \rho_m & 0 & 0 \\ 0 & \rho_m & 0 \\ 0 & 0 & \dfrac{1}{\rho_m c_m^2} \end{bmatrix}, \qquad (7)$$

where index m denotes the medium index. The following eigenvalue problem is now solved:

$$[L - \epsilon R_m]\,\psi = 0. \qquad (8)$$

Here ϵ are the eigenvalues and ψ the orthogonal eigenfunctions. The four eigenvalues, found from $\det(L - \epsilon R_m) = 0$, are $\epsilon = 0,\ 0,\ -k_m c_m,\ k_m c_m$, with k the wave number. These eigenvalues correspond to two time independent components, which are uninteresting, and two propagating waves in time with $k_m \in [0, \infty)$, which constitute the solutions of interest for, respectively, the $e^{-j\omega t}$ and $e^{j\omega t}$ convention. The eigenfunctions can be seen as a set of plane wave solutions of the problem of interest with different angles of incidence. To facilitate the use of fast Fourier transforms (FFTs), the eigenfunctions are written as a function of the wave number vector $\mathbf{k} = [k_x, k_z]$. Using the eigenvalue equation, we may write:

$$\mathbf{Q}_\pm(\mathbf{k}, t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathbf{q}'(\mathbf{x}, t)\overline{\psi_\pm(\mathbf{k}, \mathbf{x})}^T R_m \,\mathrm{d}\mathbf{x}, \qquad (9)$$

where the overbar denotes the complex conjugate and with $\mathbf{Q}_\pm(\mathbf{k}, t)$ the wave number variables. The action of the spatial derivative operator $L\mathbf{q}'(\mathbf{x}, t)$ may be computed using Eqs. (8) and (9):

$$L\mathbf{q}'(\mathbf{x}, t) = \sum_\pm \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \epsilon R_m \mathbf{Q}_\pm(\mathbf{k}, t)\psi_\pm(\mathbf{k}, \mathbf{x})\,\mathrm{d}\mathbf{k}. \qquad (10)$$

The spatial derivative operator can thus be calculated by applying a transform to $\mathbf{q}'(\mathbf{x}, t)$ through the eigenfunctions over the spatial variables in Eq. (9), multiplying the obtained wave number variables by ϵR_m and inverse transforming over the wave number through the eigenfunctions. Most of the transforms in Eqs. (9) and (10) can be done by FFTs as $\psi_\pm$ mainly consists of Fourier kernels. When considering media with different densities only, the spatial derivatives can be calculated using one-dimensional eigenfunctions and one-dimensional transforms in the separate directions. This approach was shown to be successful [8]. Thus, the spatial derivatives arising in Eq. (2) are computed with PSTD in the wavenumber domain. This implies that a spatial Fourier transform is applied to the acoustic variables, and after an operation in the wavenumber domain, an inverse Fourier transform returns the derivatives. This way of computing spatial derivatives leads to an accurate solution close to 2 spatial points per wavelength. The computation of spatial derivative operators of the acoustic variables normal to a boundary (here, the x-direction is taken) is shown below. The domain ranges from $-M_1\Delta x$ to $M_2\Delta x$, with the interface between the two media located at $x = 0$.

$$\frac{\partial p(n\Delta x^+)}{\partial x} = \begin{cases} \mathcal{F}_x^{-1}\left(jk_x e^{-jk_x\frac{\Delta x}{2}}\,\mathcal{F}_x\begin{bmatrix}p_1\\ p_2\end{bmatrix}\right), & -M_1 \le n \le -1, \\ & 0 \le n \le M_2 - 1, \end{cases}$$
$$\frac{\partial u_x(n\Delta x)}{\partial x} = \begin{cases} \mathcal{F}_x^{-1}\left(jk_x e^{jk_x\frac{\Delta x}{2}}\,\mathcal{F}_x\begin{bmatrix}u_{x,1}\\ u_{x,2}\end{bmatrix}\right), & -M_1 \le n \le -1, \\ & 0 \le n \le M_2 - 1. \end{cases} \qquad (11)$$

$\mathcal{F}_x$ is the forward and $\mathcal{F}_x^{-1}$ the inverse Fourier transform over and to the x-variable, and with p_1, p_2, $u_{x,1}$ and $u_{x,2}$ defined in Eqs. (12) and (13). For the derivatives of the pressure and normal velocity component in medium 1, i.e. $x \le 0$, the following is used

$$p_1(m\Delta x) = \begin{cases} p(m\Delta x), & -M_1 \le m \le -1, \\ R_{1,1}p(-m\Delta x) + T_{2,1}p(m\Delta x), & 0 \le m \le M_2 - 1, \end{cases}$$
$$u_{x,1}(m\Delta x^+) = \begin{cases} u_x(m\Delta x^+), & -M_1 \le m \le -1, \\ -R_{1,1}u_x(-m\Delta x^+) + T_{1,2}u_x(m\Delta x^+), & 0 \le m \le M_2 - 1. \end{cases} \qquad (12)$$

Note that for p_1 and $u_{x,1}$, derivatives are computed for $-M_1\Delta x \le x < M_2\Delta x$, but are only physically valid and used for $x \le 0$. For the

derivatives in medium 2, i.e. $x \geq 0$, the following is used

$$p_2(m\Delta x) = \begin{cases} R_{2,2}p(-m\Delta x) + T_{1,2}p(m\Delta x), & -M_1 \leq m \leq -1, \\ p(m\Delta x), & 0 \leq m \leq M_2 - 1, \end{cases}$$
$$u_{x,2}(m\Delta x^+) = \begin{cases} -R_{2,2}u_x(-m\Delta x^+) + T_{2,1}u_x(m\Delta x^+), & -M_1 \leq m \leq -1, \\ u_x(m\Delta x^+), & 0 \leq m \leq M_2 - 1, \end{cases} \quad (13)$$

with $R_{i,i}$ and $T_{i,j}$ physical reflection and transmission coefficients from medium i to medium i and j respectively and Δz^+ denotes $\Delta z + \frac{\Delta z}{2}$. The pressure and velocity nodes are staggered in space. To evaluate derivatives at positions staggered by $\mp\frac{\Delta x}{2}$, the spatial derivatives have in Eq. (11) been multiplied in the wave number domain by $e^{\pm jk_x\frac{\Delta x}{2}}$. When both media have the same properties, $R = 0$ and $T = 1$ and the equations reduce to:

$$\left.\frac{\partial p}{\partial x}\right|_{l\Delta x^+} = \mathcal{F}_x^{-1}\left(jk_x e^{-jk_x\frac{\Delta x}{2}}\,\mathcal{F}_x[p]\right), \quad -M_1 \leq l \leq M_2 - 1,$$
$$p = p(m\Delta x) \quad -M_1 \leq m \leq M_2 - 1,$$
$$\left.\frac{\partial u_x}{\partial x}\right|_{l\Delta x} = \mathcal{F}_x^{-1}\left(jk_x e^{jk_x\frac{\Delta x}{2}}\,\mathcal{F}_x[u_x]\right), \quad -M_1 \leq l \leq M_2 - 1,$$
$$u_x = u_x(m\Delta x^+) \quad -M_1 \leq m \leq M_2 - 1. \quad (14)$$

In the Fourier PSTD method, boundaries can be modelled with a quasi locally reacting approach, which is accomplished by not computing spatial derivatives of the acoustic variables in the boundary media parallel to the surface. This method was shown to be reliable [12,18]. In the current implementation, the frequency dependency of the boundary conditions can be taken into account by separate PSTD computations per octave band. For outdoor acoustics, it is possible to include meteorological effects as for example a mean wind speed field $\mathbf{u}_0$ in Eq. (2) [8]. In openPSTD v1.1, meteorological effects are not yet incorporated. An often adopted method to obtain a reflection free boundary at open domain ends is the perfectly matched layer, PML. A PML has been implemented as an absorption layer at the edge of the computational domain acting on the normal components of the velocity and pressure, which returns a layer without reflections for all incoming propagation angles. The adopted approach splits the pressure into components p_x and p_z, similar to the velocity components [8]. Eq. (2) for the split pressure equations are solved in time by a six-stage Runge–Kutta method which can be written as:

$$\mathbf{q}'(\mathbf{x}, t_0) = \mathbf{q}'(\mathbf{x}, t),$$
$$\mathbf{q}'(\mathbf{x}, t_i) \approx \mathbf{q}'(\mathbf{x}, t_0) - \gamma_i\Delta t(W\mathbf{q}')(\mathbf{x}, t_{i-1}), \quad \text{for } i = 1, \ldots, 6,$$
$$\mathbf{q}'(\mathbf{x}, t + \Delta t) \approx \mathbf{q}'(\mathbf{x}, t_6), \quad (15)$$

with $\mathbf{q}' = [u_x, u_z, p_x, p_z]^T$ and W the right hand side operator of Eq. (6), i the integer stage number within a Runge–Kutta time step, γ_i a real number between 0 and 1 and Δt a discrete time step. In the numerical solution, this method only requires storage of the variables at two time steps. The effect of the PML is now imposed by an additional operator after each time step:

$$\mathbf{q}'(\mathbf{x}, t + \Delta t) = \mathbf{q}'(\mathbf{x}, t + \Delta t)e^{-\sigma'\Delta t}, \quad (16)$$

with $\sigma' = [\sigma_x, \sigma_z, \sigma_x, \sigma_z]^T$, with:

$$\sigma_x = \beta\left(\frac{x - x_{\text{PML}}}{D_x}\right)^4, \qquad \sigma_z = \beta\left(\frac{z - z_{\text{PML}}}{D_z}\right)^4. \quad (17)$$

The coefficient β can be chosen in the software, D_x is the thickness of the PML layer in x-direction. The PSTD method requires the solution to be periodic on the spatial domain. For this reason, a PML is applied to all boundaries, both at open domain ends as well as in boundary media. The influence of the size of the PML on the accuracy of the results can be found in Section 4.1.

3. openPSTD framework

3.1. Software architecture overview

The openPSTD framework is built entirely around open source components and can be decomposed into two principal parts. First, there is the user interface, which is built as a plug-in for the open source 3D modelling application Blender [19]. The second part is the simulation kernel that takes as input a description of the boundary geometry and source and receiver positions, as it is compiled by the user interface. The kernel in turn outputs the resulting pressure values for the computational grid as well as at specified receiver positions, which are read back into the user interface to be visualized there. These two parts are built as modular components so that potentially, the user interface could use another simulation back-end. Even more importantly, the simulation kernel can be used independently from the interface, for example to defer calculations to a more powerful calculation cluster. On the other hand, because both components rely on the Python programming language, an instalment of the software can easily be assembled that appears as an integrated solution to the user. The implementation of the simulation back-end relies on the Numpy and Scipy stack [20], which is a framework for numerical and scientific computing for the Python programming language [21]. The core parts of the implementation depend on linear algebra functionality and fast Fourier transform (FFT) functions, which are supplied by Python [20].

After exporting the computed results at a receiver position in Blender as a .bin file, the results of a computation with openPSTD can be post-processed in any scientific programming environment as preferred by the end-user, as the results are written to the file system as a binary stream of standard IEEE 754 floating point numbers. In addition to manually reading the binary output files, the Blender interface to openPSTD reads the output interactively and updates the user interface accordingly. For the receiver positions this means that, for the currently selected receiver in the interface, the range of output frames that have been processed appears as a highlighted area along with the pressure level curve of the impulse response. Likewise, pressure levels for grid nodes are read and mapped to a RGB-colour on a logarithmic scale, which are presented in the 3D area of the Blender interface where they are overlaid on top of the boundary geometry. An illustration of the Blender interface with an active simulation running is presented in Fig. 6.

Blender uses Python as its scripting language. Therefore, the Blender executable itself is a Python interpreter that comes bundled with Numpy. When the user commences a new simulation, this results in the Blender interface starting a new Blender process in background mode that will run the simulation code, which is in a different module from the interface code. The user interface is notified when the simulation background process finishes and is also able to kill it when instructed by the user.

3.2. Composition of a spatial domain by subdomains

In openPSTD, the total spatial domain is composed out of subdomains. The advantage of this decomposition is a higher geometrical flexibility, a lower computational complexity, it promotes object orienting programming and a parallel computational architecture. Fig. 1(a) schematically shows a geometry decomposed into four

subdomains. Typically, the user models the air domains and supplies coefficients that model the boundary conditions. The boundary media are inherently modelled by openPSTD by additional subdomains, and include an absorbing layer (PML). These PML domains are not shown in Fig. 1(a). Boundaries of subdomains are populated by acoustic velocity components, which are spatially staggered from the pressure components as explained in Section 2. The open triangles collocate, i.e. the boundaries of adjacent subdomains are at the same position and have the same value in two subdomains. The computation of the horizontal spatial derivatives as detailed in Section 2, at the pressure positions in subdomain 3, is schematically depicted in Fig. 1(b). Three subdomains are needed to compute these derivatives. To obtain spatial periodicity on the three subdomains, the variable values are multiplied by a window function prior to computing the derivatives. This super-Gaussian window $w(l)$ is defined as in [9]:

$$w(l) = \begin{cases} e^{-\alpha \ln(10)\left(\frac{l-N_w}{N_w}\right)^6} & \text{for } 0 \le l < N_w - 1, \\ 1 & \text{for } N_w \le l \le N_s + N_w, \\ e^{-\alpha \ln(10)\left(\frac{l-(N_s+N_w)}{N_w}\right)^6} & \text{for } N_s + N_w + 1 \le l \le N_s + 2N_w, \end{cases} \tag{18}$$

where N_w is the length of the single-sided exponential part of the window and N_s the length of the subdomain of interest. The error introduced by the window function is related to N_w and the value of α, see [9]. In openPSTD v1.1, the value of N_w can be specified and the related error for the frequency corresponding to 2.5 discrete points per wavelength is displayed in the user interface. Section 4 demonstrates the accuracy of the window function in openPSTD. The window function imposes the minimum dimension of a subdomain to be at least N_w cells in all directions.

Composing the geometry out of rectangular subdomains obviously poses restrictions on the type of geometry that can be modelled. However, the interface, at which the boundaries of two adjacent subdomains coincide, does not necessarily have to match the full length of both subdomain boundaries, see Fig. 2(a). Hence, subdomains can have multiple neighbours at a domain boundary and hence local refinement does not necessarily propagate throughout the entire topology, as depicted in Fig. 2(b). This is beneficial because in its current state the openPSTD implementation does not cope well with narrow subdomains, i.e. domains smaller than N_w. In the current implementation subdomain-neighbour information is deduced solely from geometrical collinearity of domain edges. For that reason, an infinitely thin screen between two domains can for example not be represented, as the geometrical coincidence of the domain boundaries would imply that the domains are connected.

3.3. Python implementation

openPSTD is implemented according to Object-Oriented Programming (OOP) principles. In particular, the principle of encapsulation, the coupling of data and functions that operate on the data by defining classes of objects, has proven to be suitable for the way the environment is modelled in openPSTD, i.e. a geometry decomposed in a set of subdomains. Additionally, the concept of inheritance, in which sibling classes inherit behaviour from a common ancestor class, is a useful device to model the different types of domains, being either PML layers or air domains. With every subdomain modelled as a class instance, each domain does its own bookkeeping to maintain information about neighbouring domains, from which they need information for computing derivatives, see Fig. 1(a). The decoupling in subdomains simplifies the implementation logic, because all possible situations are modelled using the same generic strategy and also reduces the effort required to implement parallel processing in order to shorten calculation time.

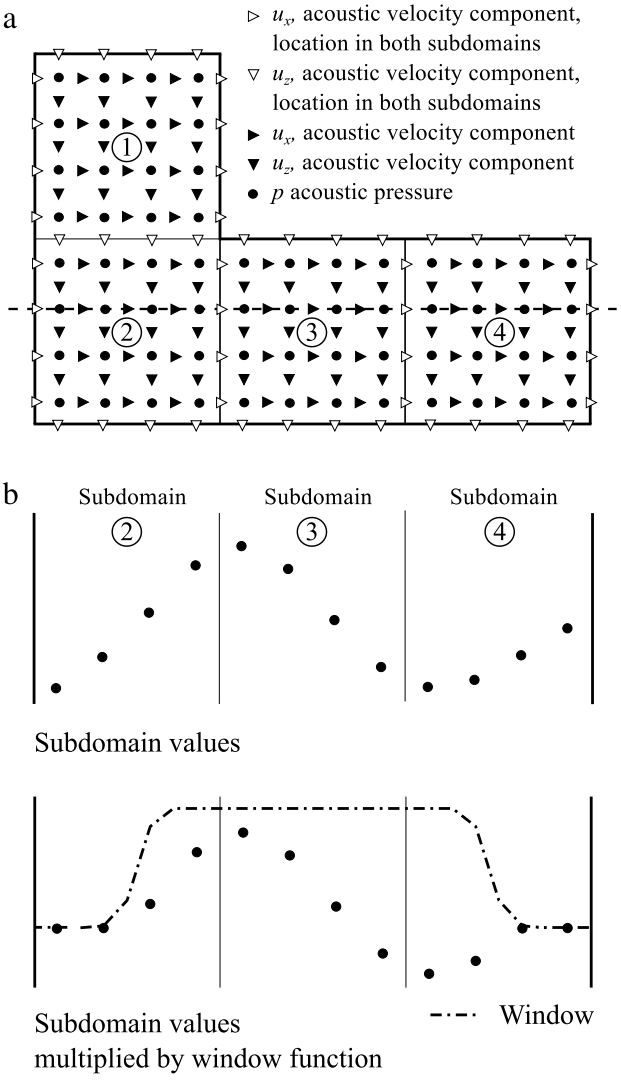

Fig. 1. (a) Schematic representation of a 2D geometry decomposed in 4 subdomains, excluding PML subdomains, (b) schematic representation of windowing the solution at the pressure positions along the dashed line of (a) for computing spatial derivatives in subdomain 3.

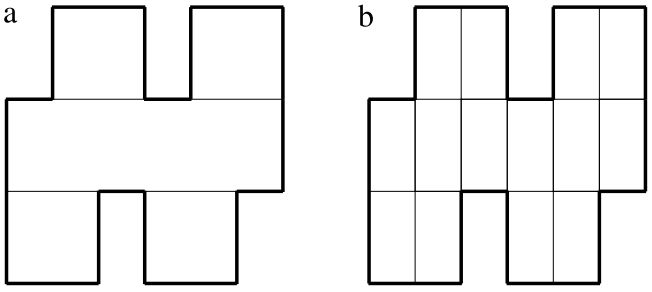

Fig. 2. Schematic representation of a situation in which adjacent subdomain boundary pairs (a) do not overlap along their full length or (b) agree completely.

The main part of the computational effort consists of the FFTs, see Eqs. (12)–(14). The algorithm comprises of an FFT and its inverse for all axes in the domain for both the pressure nodes as well as the velocity nodes. In openPSTD v1.1, the domain is always two-dimensional, meaning a total of two sets of FFT and IFFT operation pairs on each subdomain. As described above, subdomains do not necessarily share an entire boundary together. More precisely, to calculate the derivatives along a particular axis, say the vertical axis

of the central subdomain in Fig. 2(a), every subdomain in the set of upper neighbouring subdomains is paired with every neighbouring bottom subdomain to see if a column of grid cells runs over all three domains. Every grid cell in an air subdomain has exactly one horizontal line and one vertical line running over the grid cell connecting two subdomains from opposing sets of neighbour subdomains. Hence, no grid cell in an air subdomain is excluded from being updated. Boundary subdomains, see Fig. 3(a), differ in the sense that they do not necessarily have neighbours in every direction. In these cases the algorithm will still concatenate three arrays in order to compute the derivatives for the computational cells of that domain, but will populate an array with zeros for missing neighbours. Also, contrary to air domains, which will never intersect, boundary subdomains will overlap with other subdomains in the case of a concave corner, as is illustrated in Fig. 3(b). However, these boundary subdomains are physically not directly connected. Subdomains located in a concave part of the boundary might not have neighbouring subdomains in the direction parallel to the boundary surface. For the calculation of the derivative parallel to the boundary for such subdomains, sides of the boundary subdomain perpendicular to the boundary are assumed to be acoustically rigid. In case of a convex corner as shown in Fig. 3(b), a corner boundary subdomain is created that is connected to both other boundary subdomains, and has the impedance value of the boundary subdomain with the highest impedance.

The pseudocode of openPSTD as implemented in Python is made schematic in Fig. 4. It contains several parts: interpreting the Blender input and initializing the calculation, calculating the sound propagation by looping of time steps and writing the data to receiver positions.

3.4. GPU acceleration

Simulating large scenes in openPSTD can take up to several hours on a modern CPU. The most computationally intensive operation is computing the spatial derivatives: this accounts for between 84% and 96% of CPU time, depending on the input scene and parameters. openPSTD includes an option to compute these spatial derivatives on a Graphical Processing Unit (GPU). While the program uses 64-bit floats by default, the GPU code includes an option to use 32-bit floats, as not all modern desktop GPUs support 64-bit float operations. Using 32-bit floats significantly improves efficiency, at the expense of a very slight loss of accuracy, see Section 4.2. The code has been implemented in both CUDA and OpenCL and it therefore supports a wide range of GPUs. To perform the needed FFT operations for evaluating the spatial derivatives, the PyFFT library [22] was used. All the other operations in the spatial derivative function were implemented in handwritten CUDA/OpenCL kernels. The relative computational time performance is plotted in Fig. 5 and the accuracy in Section 4.1. The GPU used to gather these results was an NVIDIA GeForce GTX 760 and the CPU used was an Intel i7-920 clocked at 2.67 GHz. Fig. 5 shows the relative computational performance of the different GPU implementations (CUDA and OpenCL, 32 and 64 bit) with respect to the original CPU implementation. From this figure it is clear that for bigger input domains, the GPU implementations yield a bigger performance gain. On the used hardware, this performance gain capped out at a maximum speed-up factor of approximately 9× for the 64-bit implementation and a speed-up factor of approximately 15× for the 32-bit implementation. Especially when openPSTD is extended into a three-dimensional scenes, these GPU performance improvements are expected to be essential in order to be fast enough to ensure usability in practice. Also, the currently used FFT algorithm could be optimized by using (combinations of) higher order FFT radix algorithms.

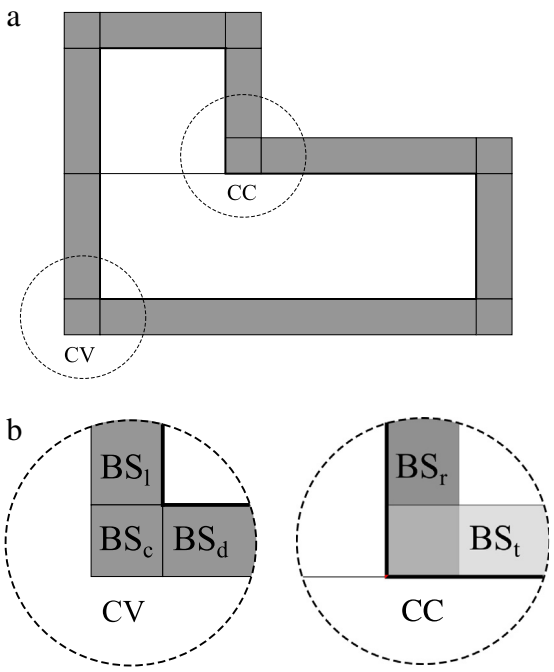

Fig. 3. (a) Depiction of a geometry with boundary subdomains (BS). (b) Situations of boundary subdomains around a convex (CV) and concave (CC) corner of the configuration (a). Subscripts l, r, t, d, c represent the left, right, top, down and corner BS for a certain subdomain.

3.5. User interface

The use of Blender as user interface of openPSTD is very well-suited because of the same open source license as openPSTD. A depiction of the visual user interface that has been developed in the modelling package Blender is shown in Fig. 6. It provides a modelling interface that is useful to model an arbitrary configuration relying on rectangular subdomains for simulation in openPSTD. The configuration appears in part A of Fig. 6. Geometrical details can be adjusted in part B, as well as locations of the source and receiver(s) positions. Apart from the geometrical input, the simulation kernel takes several input parameters, which are shown as properties in the openPSTD blender add-on, see part C in Fig. 6. The parameters are categorized in general and advanced settings, see Sections 3.5.1 and 3.5.2. The simulation kernel is invoked by starting a separate process in which the simulation code runs. As long as the simulation kernel is running, the user interface will keep listening for new simulation frames appended to the result and visualize these graphically. These pressure values are mapped to a colour (RGB-triplet) using a logarithmic scale and drawn in area A of the Blender interface. In area D of the interface finally, the impulse response for the pre-defined receiver positions, if any, is visualized. This same area is used to control which frame of the simulation is currently displayed in viewport A. Computed impulse responses at predefined receiver positions can be exported as binary files, which can then be imported and post-processed in a scientific computing environment. Simulation output is written to disk so that it persists and the user is able to inspect the results in a later session as well. Documentation on the user interface and a tutorial on a computation can be found in the openPSTD wiki [23].

3.5.1. General settings

General settings contain settings that need to be verified before a simulation is started.

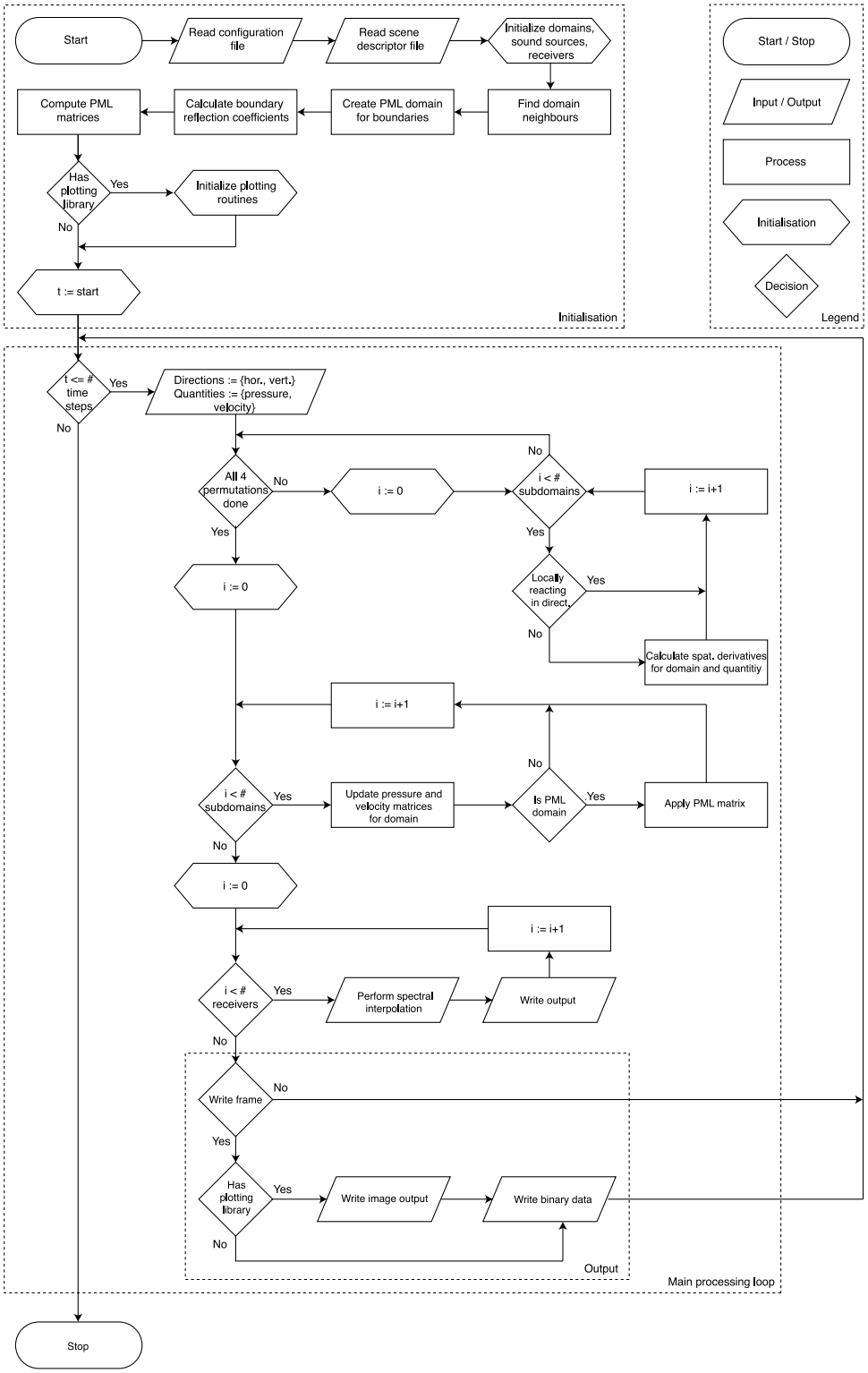

Fig. 4. Pseudocode of openPSTD v1.1.

- **Grid spacing** specifies the uniform distance between grid points in metres and, hence, is inversely related to the highest frequency that can be resolved by the method;
- **Window size** equals to N_w and corresponds to the dimension of a window, see Eq. (18);
- **Render time**. The simulation length of a calculation in openPSTD v1.1 is denoted by Render time in s. The calculation is terminated after this time;
- **Absorption**. An acoustic absorption coefficient $\alpha(\theta = 0)$ can be assigned to every boundary edge by setting this value between 0 and 1. If the **local** box is ticked, the boundary is in approximation treated as a locally reacting boundary.

3.5.2. *Advanced settings*

Upon ticking the advanced settings box, more options appear in the openPSTD v1.1 add-on.

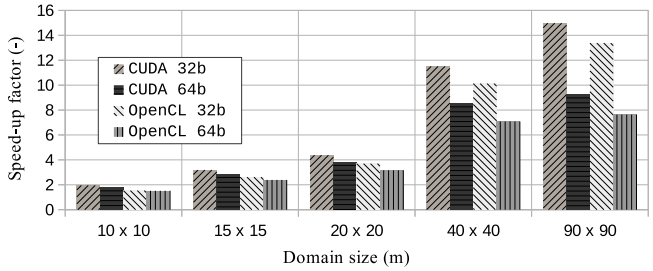

Fig. 5. Relative performance of openPSTD v1.1 (GPU implementation) with respect to openPSTD v1.0 (CPU implementation) for square domains at grid spacing of $\Delta x = 0.2$ m.

- **Pressure level visualization scale** in dB sets the dynamic range of the relative sound pressure level in the 2D visual animation;
- **Number of PML cells** sets the number of grid points in the boundary media that are affected by the perfectly matched layer (PML), see Eq. (17);
- **Density** and **Sound speed** represent the density in kg/m^3 and the adiabatic speed of sound in m/s of the propagation medium;
- **CFL number RK-scheme**. The stability of the calculation is controlled by the CFL number of the Runge–Kutta scheme;
- **Enable GPU acceleration**. If this box is checked, openPSTD calculations will partly be executed on the GPU (only available for computers that have a GPU that supports either CUDA or OpenCL). **Use 32 bit** can be selected to use 32-bit floats in the GPU computations;
- **Save every *n*th** sets the interval of discrete time steps used for the visualization of the 2D results;
- **Visualization subsampling** sets the interval of discrete spatial steps used for the visualization of the 2D results;
- **Bake openPSTD simulation**. The computed time dependent 2D sound field can be exported as a movie. The Bake openPSTD simulation command turns the calculations into a movie.

4. openPSTD performance

Various calculations have been carried out to demonstrate the accuracy and applicability of openPSTD v1.0 (CPU implementation). For all these calculations, the computed impulse responses at receivers in openPSTD v1.0 have been exported as .bin files and have been post-processed in Matlab. For all the cases, the default speed of sound and medium density have been used.

4.1. Accuracy

The accuracy of openPSTD v1.0 is first demonstrated for a free field situation and a situation with a reflecting surface. Fig. 7(a) shows the studied configuration, with a source position and two receiver positions. The relative amplitude error from openPSTD calculations is quantified as follows for the free field situation:

$$\epsilon(f) = 20 \log_{10} \left| \frac{|P_{n,2}(f)| - |P_{n,1}(f)|}{|P_{n,1}(f)|} \right|, \tag{19}$$

with

$$P_{n,j} = \frac{P_j(f)}{H_0^{(2)}(kr_j)},$$

$$P_j(f) = \mathcal{F}_t\left(p_j(t)\right),$$

with $p_j(t)$ the time signal recorded at receiver position j and $H_0^{(2)}(kr_j)$ the Hankel function of order zero and second kind, k the wave number and r_j the distance between source and receiver j. The Hankel function is used to normalize the source amplitude. The variables $P_{n,1}$ and $P_{n,2}$ denote the normalized pressure amplitudes at the subsequent receiver positions. For the computation of $\epsilon(f)$ in the presence of a reflecting surface in Fig. 7(b), the error is computed as:

$$\epsilon(f) = 20 \log_{10} \left| \frac{|P_{n,2,R}(f)| - |P_{n,2}(f)|}{|P_{n,2}(f)|} \right|, \tag{20}$$

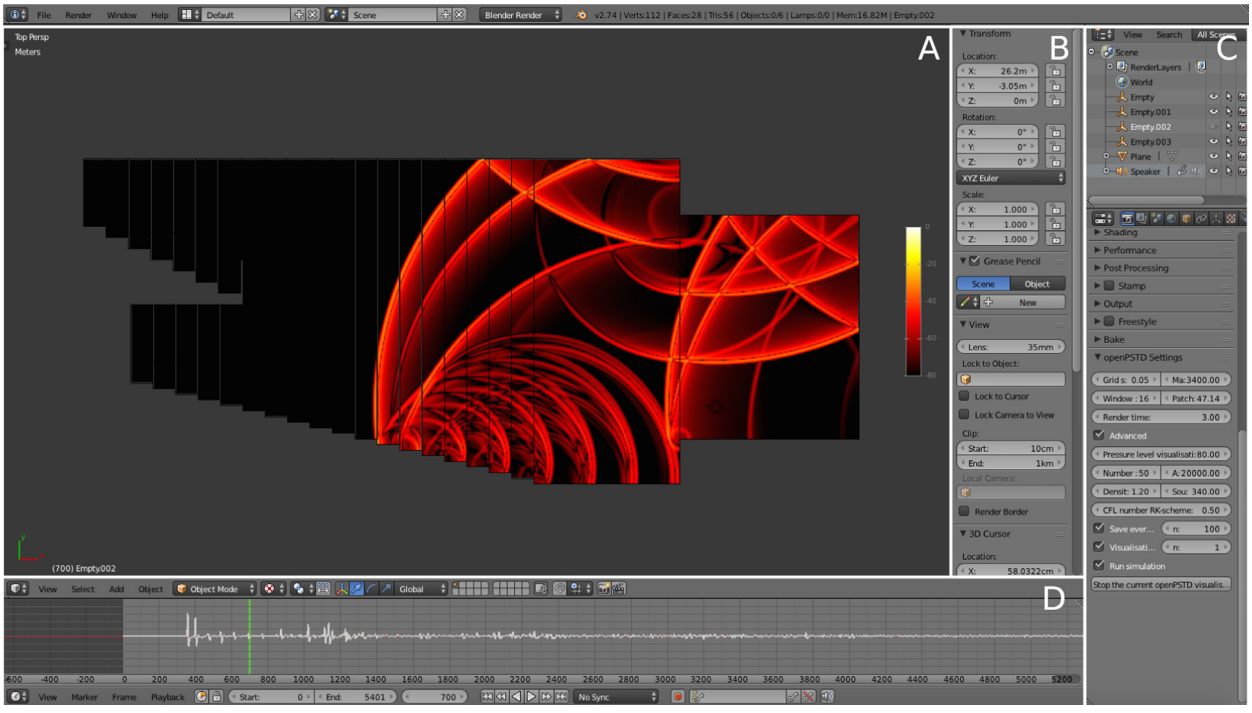

Fig. 6. Screen-shot of the openPSTD interface, with different parts indicated. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

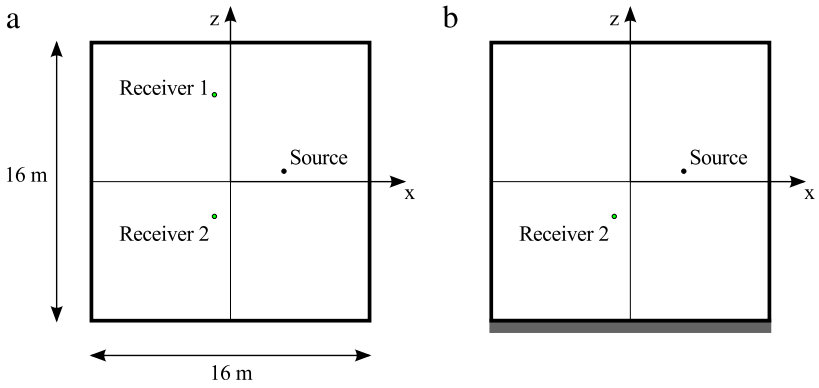

Fig. 7. Configurations for openPSTD v1.0 performance study. 16 m × 16 m domain is modelled by a single domain or divided into 4 subdomains. (a) Free field with source (3 m, 0.5 m) and receivers, R1 at (−1 m, 5 m) and R2 at (−1 m, −2 m), (b) reflective surface with source (3 m, 0.5 m) and receiver R2 at (−1 m, −2 m).

with

$$P_{n,2,R} = \frac{P_{2,R}(f)}{QH_0^{(2)}(kr_2')},$$

$$P_{2,R}(f) = \mathcal{F}_t\left(p_{2,tot}(t) - p_2(t)\right),$$

with $p_{2,tot}(t)$ the time signal recorded at receiver position 2 in the presence of the reflecting surface, $p_{2,tot}(t) - p_2(t)$ the time signal at receiver position 2 due to the reflecting surface only, r_2' the distance between source and image receiver, and Q the cylindrical wave reflection coefficient.

Various errors arise in an openPSTD calculation. The domain is bounded by PML layers, which do not perfectly absorb the incoming sound waves. This error is described in [8]. Furthermore, the window applied to compute the derivatives introduces an error, as detailed in [9]. Then, the error from the Runge–Kutta method applied to march the linearized Euler equations in time can be computed analytically according to [24]. Since the executed calculations are in 2D, the afterglow effect plays a role as well [25]. This effect implies that a signal arising from an initial distribution as used in openPSTD takes a long time before its energy has dropped below the truncation error. Finally, the aliasing error will be present close to the frequencies corresponding to 2 points per wavelength. Fig. 8 shows computed results for the free field configuration. In Fig. 8(a), the error $\epsilon(f)$ is computed using a PML with a length of 100 grid points and a window length of $N_w = 48$ for a single domain configuration. This calculation is supposed to yield a low error. The results show that for low frequencies, the error is growing. This is caused by the afterglow effect, as illustrated by the difference between calculations with different time lengths. Close to the 2 points per wavelength number, the highest errors are found. These errors are the aliasing errors. The error in the region between 2.5 and 5 points per wavelength matches well with the analytically computed Runge–Kutta error. Finally, the region between 5 and 20 points per wavelength is bounded by the PML error and window error. The latter is illustrated by lowering the number of grid points in PML and window, as shown in Fig. 8(b). The choice of 20 PML point also shows to dominate the error for the region 5 to 2 points per wavelength. Finally, the additional error of the multi-domain implementation is shown in Fig. 8(c) using a PML layer with 50 cells and a window length of 16 cells. The domain has been divided into 4 subdomains. It is obvious that the additional error is low.

For the reflecting surface of Fig. 7(b), two absorption coefficients are used, 0 and 0.8. In Fig. 9, the error ϵ according to Eq. (20) is computed using a PML layer with 50 cells and a window length of 16 cells. The Runge–Kutta errors are higher as in Fig. 8 since the travel time difference between the direct and reflected wave is larger than between the two receiver positions of Fig. 7(a).

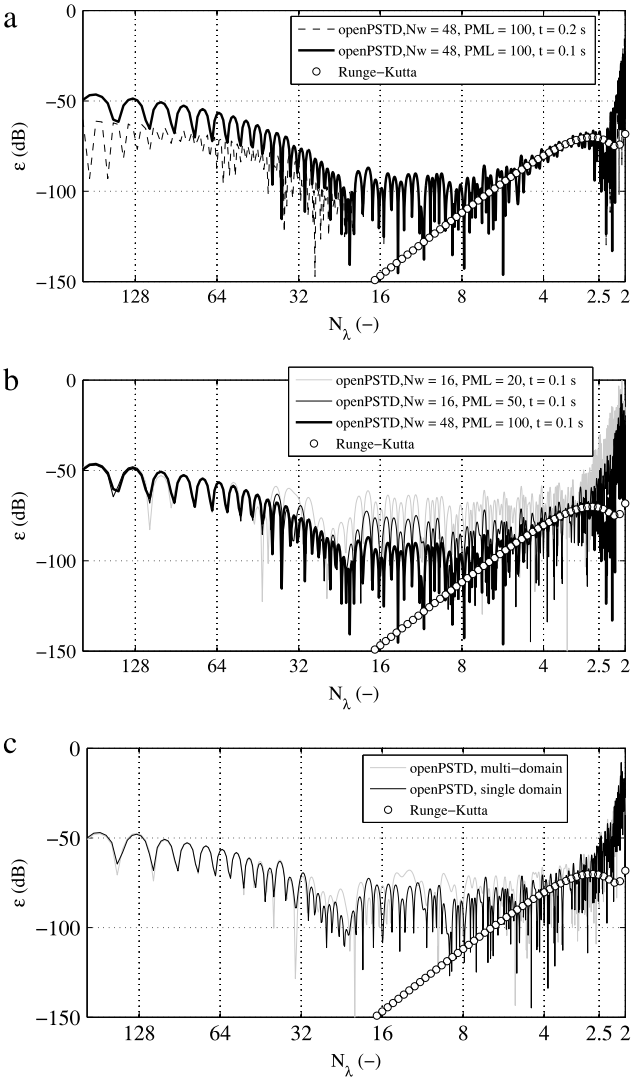

Fig. 8. Accuracy of openPSTD v1.0 for a free field situation of Fig. 7(a). Error ϵ as defined in Eq. (19) is shown as a function of the number of points per wavelength N_λ, computed as $N_\lambda = c/(f\Delta x)$ with c the adiabatic speed of sound. (a) Single domain configuration, (b) single domain configuration, various PML and window lengths, (c) multi subdomain configuration.

A similar explanation for the obtained errors can be made as in the free field configuration. The figures show that the errors are similar for both absorption coefficients, and that the multi-domain implementation does not entail additional errors. In Fig. 9(c) finally,

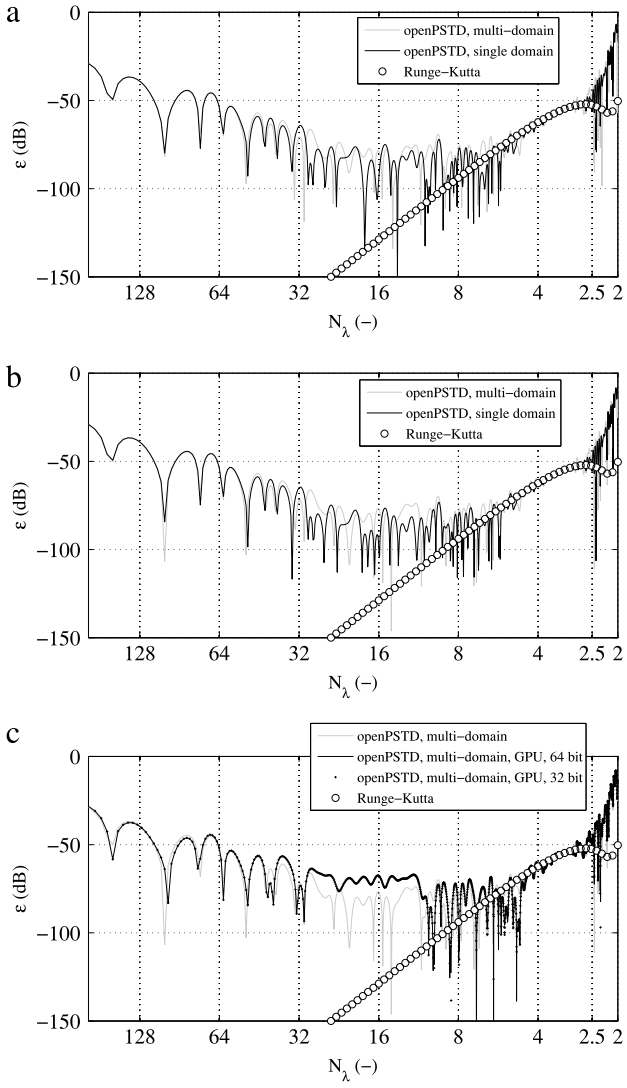

Fig. 9. Accuracy of openPSTD v1.0 for a reflecting surface situation of Fig. 7. Error ϵ as defined in Eq. (20) is shown as a function of the number of points per wavelength N_λ. (a) Absorption coefficient of 0, (b) absorption coefficient of 0.8. (c) absorption coefficient of 0.8, including openPSTD v1.1 results (GPU implementation).

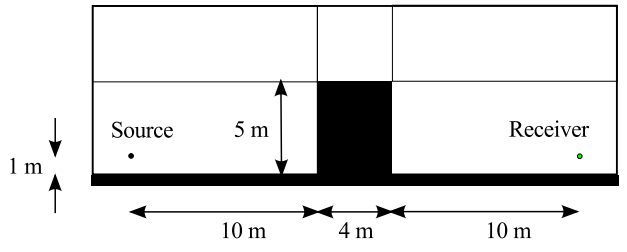

Fig. 10. Configuration of a noise barrier, using 5 subdomains in openPSTD. Ground and barrier are acoustically rigid.

results from using openPSTD v1.1 (GPU acceleration) with 32 bit and 64 bit have been included for the surface with an absorption coefficient of 0.8. The agreement between results computed using openPSTD v1.0 and openPSTD v1.1 is very good.

4.2. Noise barrier application

To further demonstrate the applicability of the openPSTD software, a calculation is carried out for sound propagation over a wide noise barrier. A barrier with a width and height of 4 m × 5 m has been modelled, see Fig. 10. A source and receiver are located at 10 m from either side of the barrier, and have a height of 1 m. All surfaces are acoustically rigid. A spatial discretization of $\Delta = 0.05$ m has been used, corresponding to a highest resolved frequency of 3400 Hz. The results are compared with a calculation using the boundary element method (BEM) [26]. The excess attenuation EA, i.e. the sound pressure level relative to the level in the configuration without the barrier, is plotted in Fig. 11(a). The agreement between the openPSTD results (using a PML layer with 20 cells and a window length of 16 cells) and BEM is good and starts to deviate towards the frequency corresponding to 2 spatial points per wavelength. This error can be understood from the error analysis in Section 4.1 and in addition is influenced by the fact that the corners of the barrier are not explicitly modelled in a solution method as PSTD. This error has been identified earlier, e.g. [8]. Also for the barrier application, results from openPSTD v1.0 (CPU implementation) and openPSTD v1.1 (GPU implementation) have been computed and show a good agreement, see Fig. 11(b).

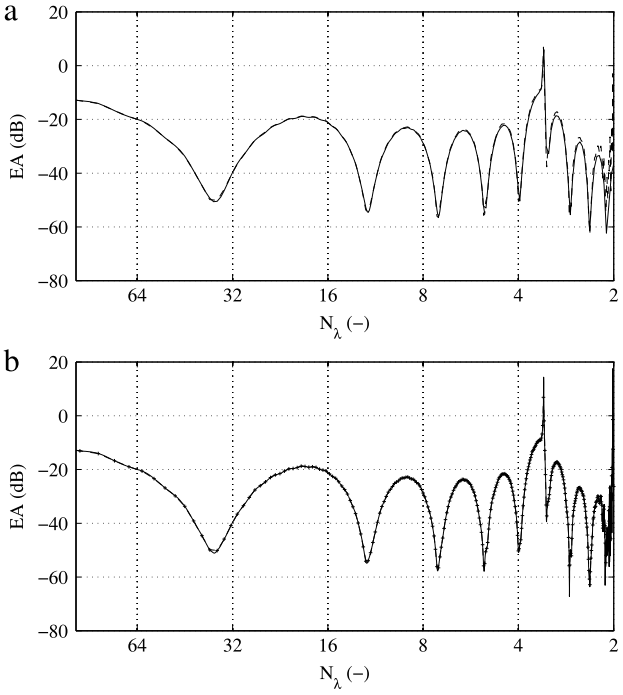

Fig. 11. Excess attenuation of noise barrier. (a) openPSTD v1.0 (dashed) and Boundary Element Method (solid), (b) openPSTD v1.0 (dashed) and openPSTD v1.1 (GPU implementation) 64 bit (solid) and openPSTD v1.1 (GPU implementation) 32 bit (+) results.

5. Conclusions and ongoing work

The open source framework openPSTD is presented. It comprises the implementation of the Fourier PSTD method for solving the linearized Euler equations to render sound propagation in the built environment. openPSTD is composed of a graphic user interface in Blender and a computational kernel in Python, with optional computational acceleration by a partial GPU implementation. The graphic interface supports the comprehension of sound propagation in the time domain and post-processing of signals recorded at predefined receiver positions can be executed in any scientific programming software. This first version of openPSTD is very well suited for academic purposes of computing sound propagation in two dimensional configurations as noise barriers, concert hall cross-sections and small scale inner city layouts. Research is still ongoing on strategies to model non-orthogonal geometries to allow arbitrary geometries as input. Furthermore, impedance

boundary conditions and source directivity will be added in subsequent instalments, based on current ongoing research [27,28], as well as local grid refinement and a moving medium.

Currently, the GPU in the computations of this paper was only actively computing for approximately 18% of the program's runtime. One reason is that the program is single-threaded and thus the GPU is inactive whenever anything other than spatial derivatives are being computed. Another reason is that a significant amount of time is spent transferring the needed data to the GPU and transferring the results back to the main memory. Further versions of the program are expected to be CPU multi-threaded and will overlap computation and transferring data. In the ideal case, this is expected to speed up the program by an additional five times. Finally, as the code has several loops and calculations can be operated per subdomain, an even more cost efficient implementation can be established by distributing the computational operations over the multiple CPU processors.

Acknowledgement

This research was partly supported by Marie Curie Career Integration Grant project nr. 321932 within the 7th European Community Framework Programme.

References

[1] V. Välimäki, J.D. Parker, L. Savioja, J.O. Smith, J.S. Abel, IEEE Trans. Audio Speech Lang. Process. 20 (2012) 1421–1448.
[2] M. Vorländer, J. Acoust. Soc. Am. 133 (2013) 1203–1213.
[3] M. Hornikx, C. Hak, R. Wenmaekers, J. Build. Perform. Simul. 8 (1) (2015) 26–38.
[4] R. Mehra, N. Raghuvanshi, L. Savioja, M. Lin, D. Manocha, Appl. Acoust. 73 (2) (2012) 83–94.
[5] D. Albert, P. Eller, J. Cheng, Fast computations of outdoor sound scattering using graphical processing hardware, in: Proceedings of InterNoise, New York City, New York, USA, 2012.
[6] K. Kowalczyk, M. van Walstijn, IEEE Trans. Audio Speech Lang. Process. 19 (1) (2011) 34–46.
[7] O. Atak, K. Huijssen, M. Rychtarikova, B. Pluymers, W. Desmet, in: Proc. of ISMA, Leuven, Belgium, 2010, pp. 1969–1983.
[8] M. Hornikx, R. Waxler, J. Forssén, J. Acoust. Soc. Am. 128 (2010) 1632–1646.
[9] M. Hornikx, W. De Roeck, W. Desmet, J. Comput. Phys. 231 (14) (2012) 4759–4774.
[10] M. Hornikx, D. Dragna, J. Acoust. Soc. Am. 138 (1) (2015) 425–435.
[11] M. Hornikx, W. De Roeck, T. Toulorge, W. Desmet, Comput. Fluids 116 (2015) 176–191.
[12] T. Van Renterghem, M. Hornikx, J. Forssén, D. Botteldooren, Build. Environ. 61 (2013) 34–44.
[13] I. Bashir, T. Hill, S. Taherzadeh, K. Attenborough, M. Hornikx, Appl. Acoust. 83 (2014) 1–15.
[14] openPSTD: the open source pseudospectral time-domain method for acoustic propagation available from http://www.openpstd.org/.
[15] J. Saarelma, L. Savioja, An open source finite-difference time-domain solver for room acoustics using graphics processing units, in: Proceedings of Forum Acusticum, Krakow, Poland, 2014.
[16] J. Sheaffer, B.M. Fazenda, WaveCloud: an open source room acoustics simulator using the finite difference time domain method, in: Proceedings of Forum Acusticum, Krakow, Poland, 2014.
[17] V.E. Ostashev, D.K. Wilson, L. Liu, D.F. Aldridge, J. Acoust. Soc. Am. 117 (2005) 503–517.
[18] M. Hornikx, Y. Smyrnowa, T. Van Renterghem, C. Cheal, J. Kang, Acoustic simulation tools for urban streets, squares and road-side courtyards integrating vegetation. Deliverable 5.3 of HOSANNA, Collaborative Project under the Seventh Framework Programme, Theme 7, Sustainable Surface Transport, 2012.
[19] Blender: professional free and open-source 3D computer graphics software product available from http://www.blender.org/.
[20] T.E. Oliphant, Comput. Sci. Eng. 9 (2007) 10–20.
[21] Python: programming language available from http://www.python.org/.
[22] PyFFT v0.3.6 documentation available from https://pythonhosted.org/pyfft/.
[23] openPSTD wiki: documentation pages of open source pseudospectral time-domain method for acoustic propagation available from http://www.openpstd.org/wiki/.
[24] C. Bogey, C. Bailly, J. Comput. Phys. 194 (2004) 194–214.
[25] J. Escolano, C. Spa, A. Garriga, T. Mateos, Appl. Acoust. 74 (2013) 818–822.
[26] V. Cutanda Henriquez, P.M. Juhl, OpenBEM—An open source Boundary Element Method software in Acoustics, in: Proceedings of Internoise, Lisbon, Portugal, 2010.
[27] F. Georgiou, M. Hornikx, Incorporating directivity in the Pseudospectral time-domain method by using spherical harmonics, in: Proceedings of Internoise, Melbourne, Australia, 2014.
[28] R. Pagan Munoz, M. Hornikx, A hybrid PSTD/DG method to solve the linearized Euler equations, in: Proceedings of Euronoise, Maastricht, The Netherlands, 2015.