

SimSonic Suite
User's guide for SimSonic2D

Emmanuel Bossy

November 20, 2012



Contents

1	Introduction	3
1.1	The SimSonic Suite	3
1.2	About this document	3
1.3	Quick overview of using SimSonic2D	3
2	Physical Model	4
2.1	Model Equations	4
2.2	FDTD discretization	4
2.2.1	Temporal mesh	6
2.2.2	Spatial mesh	6
2.2.3	Stability condition	6
2.2.4	Choice of grid steps	7
2.3	Wave generation	8
2.4	Boundary conditions	9
3	Description of SimSonic2D	10
3.1	Overview of input and output files	10
3.2	Grids in SimSonic2D	11
3.3	The Geometry.map2D input file	12
3.4	The Parameters.ini2D input file	13
3.4.1	Principle	13
3.4.2	General parameters	13
3.4.3	Boundary conditions	14
3.4.4	Sources	15
3.4.5	Receivers	17
3.4.6	Snapshots	19
3.4.7	Definition of material properties	20
3.5	Signals files	20
3.6	Snapshots files	21
3.7	Operating Systems and memory requirements	21
3.7.1	Operating systems	21
3.7.2	Memory requirements	21
3.8	How to run a simulation	21
4	Tutorial	23
	References	24
A	Physical Units in SimSonic2D	25
B	SimSonic2D Matlab Toolbox	26
C	File Formats	27

1 Introduction

1.1 The SimSonic Suite

SimSonic is freely available 3rd party software suite for the simulation of ultrasound propagation, based on finite-difference time-domain (FDTD) computations of the elastodynamic equations. It is intended as a tool for researchers and teachers communities. The SimSonic suite consists of several compiled programs and C source codes, free for use and download from www.simsonic.fr, under the [GNU GPL](#) license. In exchange for free access to the SimSonic suite, the users are asked to make proper reference (in their research publications or any other types of oral or written communication) to www.simsonic.fr and to [Bossy et al., JASA, 115, 2314-2324, 2004](#) (in which SimSonic has been used for the first time).

The development of SimSonic was initiated in 2003 by Emmanuel Bossy during his PhD work at the [Laboratoire d'Imagerie Paramétrique](#) (CNRS-University Paris 6) in Paris, France [1]. Since then, SimSonic has been maintained by Emmanuel Bossy, now at [Institut Langevin](#), CNRS-ESPCI ParisTech, Paris, France, and has regularly been enriched with new options and versions. SimSonic is currently being used by several research laboratories ([references](#)). The various versions of SimSonic correspond to different characteristics in terms of spatial dimensions and symmetries, but are otherwise based on the same physical model. In short, SimSonic currently models linear propagation in both fluid and solid media, which can be anisotropic and heterogeneous. Versions with absorption exist, but are still under development and beta-testing, and are therefore not described in this document.

1.2 About this document

This document is the user's guide for the SimSonic2D program, the 2-D version (on a cartesian mesh) of the SimSonic software suite. It describes the physical model and computation methods on which SimSonic2D is based, and explains how to use SimSonic2D. It also contains a tutorial section, where various examples of simulation are described in detail, from the generation of the input files to the visualization of the results. This version of this user's guide is November 20, 2012, and relates to the 2012.04.26 release of SimSonic2D. This document regularly evolves, in particular based on users feedback. Please feel free to send all relevant comments and questions to simsonic.software@gmail.com

1.3 Quick overview of using SimSonic2D

SimSonic2D consists of a single binary executable file (either for Windows or Linux based systems). To run a simulation, one simply has to call SimSonic2D from a line command, with a simulation directory as argument. The simulation directory contains both input and output files (see [3.1](#) for a detailed description of the various file). Running a simulation consists in the following steps

- Preparing input files. The file "Parameters.ini2D" is a simple text file that contains most of the simulation parameters. The geometry of the various media is coded in a binary file, Geometry.map2D, as an indexed image. Various other binary files may also be needed, to describe input signals for instance. All the input files must be contained in the same directory. A matlab toolbox is provided with functions to write the binary files from matlab data (vectors or matrix)
- Calling SimSonic2D via the command line, with the simulation directory as argument. On windows for instance, the call would simply look like:

```
SimSonic2DPath\SimSonic2D_win64_omp.exe SimulationDirectory\
```

- Analyse output files (signals or snapshots) that are generated in the simulation directory. A matlab toolbox is provided with functions to read the binary files to matlab data (vectors or matrix).

2 Physical Model

2.1 Model Equations

In this section, the vector components of vector \mathbf{a} are noted a_i where subscripts $i = \{1, \dots, d\}$ refer to the direction of space, with d the space dimension ($d = 2$ for SimSonic2D). SimSonic2D computations are based on the following system of elastodynamic equations, expressed in cartesian coordinates :

$$\rho(\mathbf{x}) \frac{\partial v_i}{\partial t}(\mathbf{x}, t) = \sum_{j=1}^d \frac{\partial T_{ij}}{\partial x_j}(\mathbf{x}, t) + f_i(\mathbf{x}, t), \quad (1)$$

$$\frac{\partial T_{ij}}{\partial t}(\mathbf{x}, t) = \sum_{j=1}^d \sum_{i=1}^d c_{ijkl}(\mathbf{x}) \frac{\partial v_k}{\partial x_l}(\mathbf{x}, t) + \theta_{ij}(\mathbf{x}, t). \quad (2)$$

\mathbf{x} and t are the space and time variables. $\rho(\mathbf{x})$ is the mass density and $\mathbf{c}(\mathbf{x})$ is the fourth-order rigidity tensor. The knowledge of these parameters entirely defines the material properties and geometry of the considered media. $\{v_i(\mathbf{x}, t)\}$ are the vector components of the particle velocity field and $\{T_{ij}(\mathbf{x}, t)\}$ are the components of the stress tensor $\mathbb{T}(\mathbf{x}, t)$. These are the unknown quantities that SimSonic computes. $\{f_i$ and θ_{ij} are source terms. $\{f_i\}$ are the vector components of force sources and $\{\theta_{ij}\}$ are the tensor components of strain rate sources. Equations 1 and 2 describe the propagation of mechanical waves in continous media which obeys Hooke's law (Eq.2). This formulation based on the rigidity tensor allows equally taking into account anisotropic solid media and fluid media. Absorption is not taken into account in this model.

The symmetric rigidity tensor is usually given using [Voigt notation](#), which allows formulating Eq. 2 under matrix form. In 2-D, such formulation writes:

$$\begin{bmatrix} \frac{\partial T_{11}}{\partial t} \\ \frac{\partial T_{22}}{\partial t} \\ \frac{\partial T_{12}}{\partial t} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & 0 \\ c_{12} & c_{22} & 0 \\ 0 & 0 & c_{66} \end{bmatrix} \begin{bmatrix} \frac{\partial v_1}{\partial x_1} \\ \frac{\partial v_2}{\partial x_2} \\ \frac{\partial v_2}{\partial x_1} + \frac{\partial v_1}{\partial x_2} \end{bmatrix} \quad (3)$$

The form of the rigidity tensor used above is limited to a number of crystal symmetries (orthorhombic, hexagonal, cubic, isotropic and some tetragonal class of symmetry [4]). SimSonic2D does currently not take into account other symmetries, but straightforward modifications of the code would allow to deal with higher order symmetries if needed.

2.2 FDTD discretization

SimSonic implements a finite-difference time-domain (FDTD) resolution of Eqs. 1 and 2. Briefly, finite-difference methods consist in solving partial differential equations by approximating partial derivatives of continous functions by finite-difference. Following a numerical scheme initially introduced in electromagnetism by Yee in 1966 [7], and later applied in elastodynamics by Virieux [5, 6], SimSonic uses central-difference approximations to the space and time partial derivatives. The FDTD elastodynamic equations are obtain from Equations 1 and 2 by approximating all first-order derivatives based on the following principle:

$$\frac{\partial f}{\partial a}(a) \approx \frac{f(a + \frac{\Delta a}{2}) - f(a - \frac{\Delta a}{2})}{\Delta a}. \quad (4)$$

Following 4, the system of 5 equations given by REF yields the following FDTD approximations:

$$\begin{aligned} v_1(x_1, x_2, t + \Delta t) &= v_1(x_1, x_2, t) + \\ \frac{\Delta t}{\Delta x} \times \frac{1}{\rho(x_1, x_2)} \times [& T_{11}(x_1 + \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2}) - T_{11}(x_1 - \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2}) \\ & + T_{12}(x_1, x_2 + \frac{\Delta x}{2}, t + \frac{\Delta t}{2}) - T_{12}(x_1, x_2 - \frac{\Delta x}{2}, t + \frac{\Delta t}{2})] \\ & + f_1 \cdot \Delta t \end{aligned} \quad (5a)$$

$$\begin{aligned} v_2(x_1, x_2, t + \Delta t) &= v_2(x_1, x_2, t) + \\ \frac{\Delta t}{\Delta x} \times \frac{1}{\rho(x_1, x_2)} \times [& T_{12}(x_1 + \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2}) - T_{12}(x_1 - \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2}) \\ & + T_{22}(x_1, x_2 + \frac{\Delta x}{2}, t + \frac{\Delta t}{2}) - T_{22}(x_1, x_2 - \frac{\Delta x}{2}, t + \frac{\Delta t}{2})] \\ & + f_2 \cdot \Delta t \end{aligned} \quad (5b)$$

$$\begin{aligned} T_{11}(x_1, x_2, t + \Delta t) &= T_{11}(x_1, x_2, t) + \\ \frac{\Delta t}{\Delta x} \times [& c_{11}(x_1, x_2)[v_1(x_1 + \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2}) - v_1(x_1 - \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2})] \\ & + c_{12}(x_1, x_2)[v_2(x_1, x_2 + \frac{\Delta x}{2}, t + \frac{\Delta t}{2}) - v_2(x_1, x_2 - \frac{\Delta x}{2}, t + \frac{\Delta t}{2})] \\ & + \theta_{11} \cdot \Delta t \end{aligned} \quad (6a)$$

$$\begin{aligned} T_{22}(x_1, x_2, t + \Delta t) &= T_{22}(x_1, x_2, t) + \\ \frac{\Delta t}{\Delta x} \times [& c_{12}(x_1, x_2)[v_1(x_1 + \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2}) - v_1(x_1 - \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2})] \\ & + c_{22}(x_1, x_2)[v_2(x_1, x_2 + \frac{\Delta x}{2}, t + \frac{\Delta t}{2}) - v_2(x_1, x_2 - \frac{\Delta x}{2}, t + \frac{\Delta t}{2})] \\ & + \theta_{22} \cdot \Delta t \end{aligned} \quad (6b)$$

$$\begin{aligned} T_{12}(x_1, x_2, t + \Delta t) &= T_{12}(x_1, x_2, t) + \\ \frac{\Delta t}{\Delta x} \times [& c_{66}(x_1, x_2)[v_2(x_1 + \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2}) - v_2(x_1 - \frac{\Delta x}{2}, x_2, t + \frac{\Delta t}{2})] \\ & + c_{66}(x_1, x_2)[v_1(x_1, x_2 + \frac{\Delta x}{2}, t + \frac{\Delta t}{2}) - v_1(x_1, x_2 - \frac{\Delta x}{2}, t + \frac{\Delta t}{2})] \\ & + \theta_{12} \cdot \Delta t \end{aligned} \quad (6c)$$

In Eqs. 5 and 6, Δt and Δx are the time and spatial steps used to approximate time or spatial derivatives according to 4. Accordingly, each variable in SimSonic2D (either velocity component or stress tensor component) is implemented using a regular spatio-temporal mesh with time and spatial steps of constant values Δt and Δx . A careful reading of Eqs. 5 and 6 further point out a fundamental aspect of the Yee/Virieux numerical scheme implemented in SimSonic: the different components of the velocity vector and stress tensor must be defined on staggered grids, both in space and time.

2.2.1 Temporal mesh

Regarding time: all velocity component are computed at the same instants, all stress components are also computed at the same instants, but velocity and stress components are calculated at interleaved instants relatively to each other. More specifically, the computation of a velocity (resp. tensor) component at time $t + \Delta t$ is explicitly derived from its value at time t and from values of the stress (resp. velocity) components at time $t + \Delta t/2$. This type of algorithm is often referred to as leapfrog algorithm. It is illustrated on Fig. 1, which summarize how SimSonic (or any FDTD leapfrog algorithm) works: the algorithm starts its computation from some initial conditions given by the knowledge of the velocity field at time $t = 0$ and of the stress tensor field at time $t = \Delta t/2$. In SimSonic, the first computations corresponds to compute the velocity field at time $t = \Delta t$ from the velocity field at time $t = 0$ and the stress tensor field at time $t = \Delta t/2$.

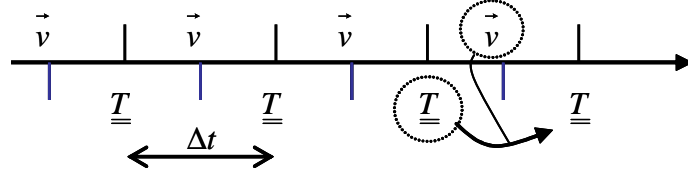


Figure 1: Principle of the leapfrog algorithm: staggered grids in time

2.2.2 Spatial mesh

As discussed in the previous section, the velocity and stress fields are staggered in time. Moreover, the different variables are also staggered in space, such as each spatial finite difference may effectively be centered. Dropping the temporal dimension, the only way to spatially organize the various variables is shown on Fig. 2:

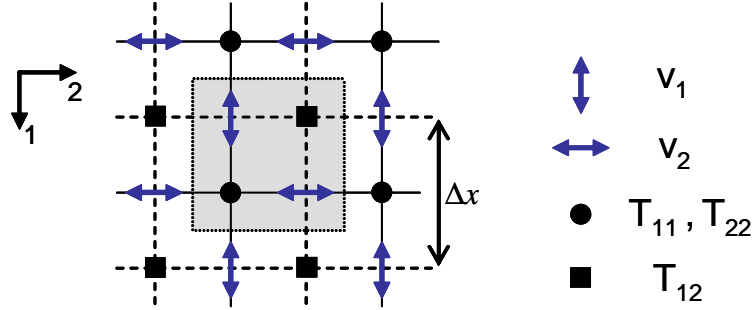


Figure 2: Staggered grids in space

As observed from Fig. 2, only T_{11} and T_{22} happens to be at the same positions. This is coherent with the fact that in a fluid medium, T_{11} and T_{22} must actually be the same values, equal to the opposite of the pressure in the fluid.

2.2.3 Stability condition

Intuitively, both Δt and Δx must be chosen small enough to provide sufficiently smooth representations of the computed field (see next section). The smallness of Δt and Δx conditions the accuracy of the results, that is the degree of approximation introduced by the numerical method. On the other hand, it can be shown that Δt and Δx cannot be chosen independently, and must obey a so-called stability

condition. The stability condition (commonly called CFL condition, from the initials of Courant, Friedrichs and Levy) for the numerical scheme described above is given by

$$\Delta t \leq \frac{1}{\sqrt{d}} \cdot \frac{\Delta x}{c_{max}} \quad (7)$$

where c_{max} is the largest speed of sound amongst all speeds of sound presents in the simulation medium, and d is the space dimension ($d = 2$ for SimSonic2D).

2.2.4 Choice of grid steps

In practice, one usually first chooses the spatial step-size Δx , based on accuracy criteria, and then uses the CFL to derive Δt and ensure stability. Accuracy and stability are completely independent concepts: a simulation may be stable while providing poor accuracy for coarse meshes. On the other hand, even very fine grids will yield instability if the CFL condition is not fulfilled.

The accuracy of a FDTD simulation depends on a number of factors, in addition to the step-size Δx : sources of error not only involve the approximation of derivative by finite difference, but also cumulative errors due to the iterative nature of the method. Therefore, the longer the simulation duration, the larger the errors. Equivalently, the larger the propagation distance, the larger the errors. One major effect generated by most FDTD schemes is numerical dispersion, *i.e.* the dependence of phase velocity on frequency due to the numerical method. As an important consequence, simulated ultrasound pulses are increasingly distorted during propagation. Accuracy criteria in FDTD therefore include tolerance on waveform distortion, as well as on wave amplitude. The obtained accuracy depends both on propagation distances and simulation duration. Note that numerical dispersion is not specific of finite difference schemes but is an artifact to control with most numerical methods, in particular those based on a discretization of the propagation domain. For second-order FDTD schemes such as used in SimSonic, a minimum spatial-step size of typically $\lambda/10$ (*i.e.* ten points per wavelength) is required. For propagation distances over several tens of wavelengths, step size as small as $\lambda/20$ may be required, depending on the desired accuracy. Moreover, for pulsed ultrasound, the accuracy strongly depends on the bandwidth: for a given central frequency, short (*i.e.* broadband) pulses will be more distorted than quasi-harmonic waves, as a value of Δx of one tenth of the central wavelength will correspond to less points per wavelength for the higher frequency content. For pulsed ultrasound, the number of points per wavelength should be determined based on the desired accuracy for the highest significant frequency content, which equivalently corresponds to a waveform distortion criterion. The choice of Δx is therefore highly subjective, and no general rules exist to determine Δx . Ten grid points wavelength should be considered a minimal requirement, that moreover remains rather subjective for pulsed ultrasound. Note that for a homogeneous medium, the CFL condition turns the number of spatial grid points per wavelength into number of temporal grid points per period, with some dimensionless factor close to one. For simulations involving several media with different propagation velocities, one has to consider the *smallest* wavelength (*i.e.* the *smallest* velocity) to choose Δx . On the other hand, the temporal step will be derived by use of the *largest* velocity. For a large range of velocities, such as encountered for simulation in both soft tissue and bone, a consequence is that the number of spatial grid points per smallest wavelength is significantly different from the number of temporal points per period, which increases numerical dispersion. To compensate for this additional dispersion, simulation involving significantly different velocities requires grid steps finer than that for homogeneous media.

Although Δx has to be small enough to fulfill accuracy requirement, it also has to be small enough in order to correctly describe the geometry of propagation media. In FDTD methods, the use of regular

grids leads to “staircases” artifacts when originally smooth interfaces are discretized on such grids. For instance, a plane interface that is not parallel to the coordinates axes, for instance, will have some artificial roughness. In turn, this artificial roughness will create scattering, which amplitude depends on the size of the “staircases” relatively to the wavelength. As for accuracy considerations in homogeneous media, although for a different reason, Δx has to be made small to decrease artificial scattering.

In summary, the spatial-step size Δx of a simulation has to be small enough to both correctly describe the geometry of the medium and minimize numerical dispersion. Practically, it is the computational cost that bounds the value of Δx to some minimal value. For a space dimension d , memory requirements scales as h^d : for fixed spatial physical dimensions, the number of points in the spatial mesh in three dimensions for instance is multiplied by $2^3 = 8$ when h is divided by 2. Moreover, because of the CFL conditions, the computational time scales as Δx^{d+1} : dividing Δx by a factor of 2 multiplies the total number of calculations by $2^{3+1} = 16$ for three-dimensional simulations. From the point of view of computational efficiency, Δx must therefore be kept as large as possible, while being small enough to fulfill accuracy requirements.

2.3 Wave generation

For the time-domain model described above, two approaches may be used to generate ultrasound waves in the simulation domain. On one hand, the user may define sources that are active at some points of the mesh during the simulation. On the other hand, the user may provide initial field values at all grid points that will then evolve in time in source-free media. Note that it is also possible in principle, though less frequent in practice, to use both source terms and initial conditions. The first approach with sources can actually be further separated in two cases. Defining sources in the domain may be done either by:

- forcing field values at some positions in space. At such points, the field is *given* by the user, not *calculated* by the algorithm.
- adding source terms at some positions in space, as described by f_i or θ_{ij} in the model equation. At such points, the field is different from the source term, and is *calculated* using the equations with the sources terms.

These two ways of including sources in the model are very different: on the one hand, forcing field values provides an easy way to generate a wave of known geometry and temporal waveform, but points in space where field values are forced will act as scatterers for waves generated elsewhere. Using this approach thus usually requires that the sources be turned off (the field values are not forced anymore and obey the model equations) before any other wave (such as reflected waves) reach the source region. Forced boundary conditions on part of the mesh boundary is often used to simulate a transducer in contact with an object. On the other hand, a source term added to a field equation allows the linear superposition at the source point of the generated wave with other waves, *i.e.* active regions are transparent to waves generated elsewhere. One drawback of using source terms, except for some simple geometry (such as generation of plane-like wave), is that the field values are usually not related in a simple manner to the values of the source terms. When initial value conditions are used rather than source term, section 2.2.1 indicate that initial conditions must be given both for the velocity fields (at time $t = 0$) and the stress tensor fields (at time $t = \Delta t/2$). The approach based on initial value conditions is well-suited for instance to start a simulation just before an incoming wave propagating in a homogeneous medium (and of analytically known geometrical shape) is about to be scattered in

a complex manner by an object.

To conclude this section on wave generation, let us emphasize that the model equations presented in section 2 do not model wave propagation within ultrasound transducers: in the current version of SimSonic2D, transducers as piezo-electric materials are not taken into account as physically active materials in the simulation domain, but are modeled by regions of space or boundary where field values are forced or source terms are provided.

2.4 Boundary conditions

Handling a mesh in a computer means that meshes necessarily have a finite number of points, and therefore numerical methods such as FDTD only solve the model equations in *bounded* regions of space. Two situations may be considered :

- (1) if the problem involves waves that are indeed physically confined within a bounded region of space, as would be the case for a finite-size object in vacuum (into which no mechanical waves can propagate), the mesh can be designed over the entire region of interest. In this case, the field variables on the mesh boundaries must simply obey conditions that express the physics at the boundary. This has to be done whether the problem is solved numerically on a mesh or analytically on the space continuum;
- (2) on the other hand, one may want to numerically solve wave propagation phenomena in unbounded space, or modeled as such. This is the case for instance in the study of wave scattering by a solid object immersed in an unbounded fluid. The modeling of such unbounded domain requires specific boundary conditions, which role is to make the mesh boundaries transparent to waves incoming from within the simulation region.

In situation (1), Simsonic2D offers four types of boundary conditions: stress-free boundary, rigid boundary, mirror-symmetry boundary, mirror-antisymmetry. How these boundary solution are handled in SimSonic2D is described later in section 3.2 where the various grids are defined. To account for situation (2), SimSonic2D allows defining Perfectly-Matched Layers (PML) on the simulation frontiers. It is out of the scope of this documentation to provide details on PMLs and their FDTD implementation, which can be found in [3] for the algorithm used in SimSonic2D. It is sufficient to say that PML are additional layers leant against the simulation boundaries, as illustrated on Fig. 3 in the case of a simulation grid with PML all around. PML are often referred to as the most convenient way to model unbounded domains [2], while maintaining a reasonable computational cost.

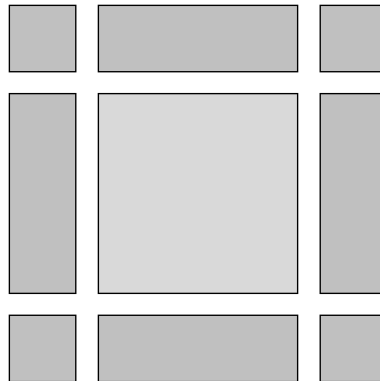


Figure 3: Layout of the Perfectly Matched Layers around the central computation box

3 Description of SimSonic2D

3.1 Overview of input and output files

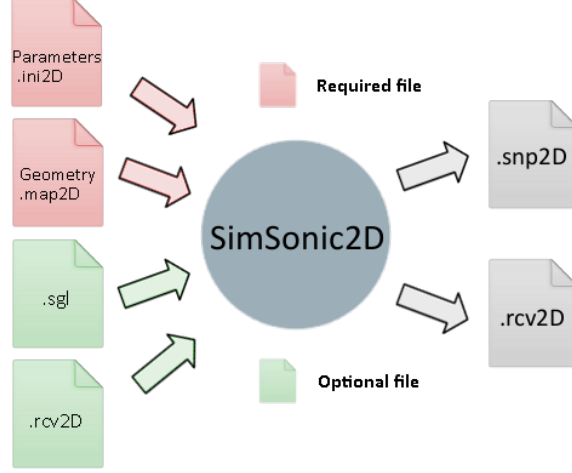


Figure 4: Schematic overview of input (red and green) and output (gray) files involved in SimSonic2D.

As illustrated on Figure 4, SimSonic requires *input* files that entirely define a simulation and computes *output* files. All input files must sit in a unique directory, which will also receive the output files during the computations.

There are four types of input files:

Parameters.ini2D. This file, which name cannot be changed, is a file in raw text format. It contains most of the information input by the user, such as the time and spatial steps, the simulation duration, the types of results to record, information on the boundary conditions, sources, etc. It is described in detail in section 3.4.

Geometry.map2D. This file, which name cannot be changed, is a bitmap file with a SimSonic specific format described in section 3.3. Briefly, it contains a $N_1 \times N_2$ image that uniquely defines the geometry of the materials presents in the simulation. Each material is represented by a 8-bit value, from 0 to 255.

.sgl or .rcv2D files These two types of files, which name can be chosen by the users, contain signals that describe sources in the simulation. The extension .sgl and .rcv2D corresponds to the two possible format used by SimSonic, described in detail in appendix C. Files with extension .sgl contain only a single waveform, that will be used by sources terms described in Parameters.ini2D. The .rcv2D format corresponds to the format of signals received or emitted on 1D-array transducers. A .rcv2D file not only contains the signals for each element of the array, but also all the array parameters such as location, pitch, sampling frequency, etc. There may be as many input signals files as needed to describe all the sources in the simulation.

There are two types of output files:

.snp2D files One .snp2D file contains one "image" of a particular field variable at one given time, referred to as a "snapshot" in this documentation. It contains not only the field values, but also a header with various information such as time, type of variable, spatial grid step, temporal grid step, etc. The format of .snp2D files is given in detail in Appendix C and section 3.6 discusses how to read .snp2D files using the matlab toolbox.

.rcv2D files The .rcv2D format corresponds to the format of signals received or emitted on 1D-array transducers. A .rcv2D file not only contains the signals for each element of the array, but also all the array parameters such as location, pitch, sampling frequency, etc. Section 3.5 discusses how to read .rcv2D files using the matlab toolbox.

The specific formats of each file are described in detail in Appendix C. However, it is not necessary to the users to be aware of those format: the Matlab toolbox provided with SimSonic2D contains all the necessary .m files that allow reading data from SimSonic2D files to Matlab and writing data from Matlab to SimSonic2D.

3.2 Grids in SimSonic2D

Grids layout and sizes. In SimSonic2D, the spatial dimensions of the simulation are entirely and uniquely defined via the Geometry.map2D file. This file contains a $N_1 \times N_2$ image that uniquely defines the geometry of the materials presents in the simulation. Each material is represented by a 8-bit value, from 0 to 255. However, as discussed in section 2.2.2, each field variable has its specific grid. As a consequence, there are in principle different possibilities to define grids from a $N_1 \times N_2$ image. It is absolutely fundamental for SimSonic users to understand precisely how the rectangular simulation grids are defined in SimSonic2D. From a $N_1 \times N_2$ image given in Geometry.map2D, SimSonic defines the following grids for each field variables:

variable	dimensions $X_1 \times X_2$
T_{11}, T_{22}	$N_1 \times N_2$
T_{12}	$(N_1 + 1) \times (N_2 + 1)$
v_1	$(N_1 + 1) \times N_2$
v_2	$N_1 \times (N_2 + 1)$

The rationale for those dimensions is best understood when considering an example: Fig. 5 shows the grids for a Geometry.map2D with a 4×7 image. It can be seen that the simulation boundaries are lines which involve only normal components of the velocity (v_1 or v_2 depending on the orientation of the boundary) and T_{12} . The checkerboard on Fig. 5 represents the 4×7 pixels of the image contained in Geometry.map2D.

Field coordinates. The spatial layout of the grid is something very important that should always be kept in mind when setting up a simulation: in particular, **proper positioning of source terms and measurement points can only be achieved with the grids layout in mind**. The convention used in SimSonic for the points coordinates are the following:

- the first element of a vector has an index of 0 (C language convention, as opposed to Matlab for instance which uses index 1 for the first element of a vector).

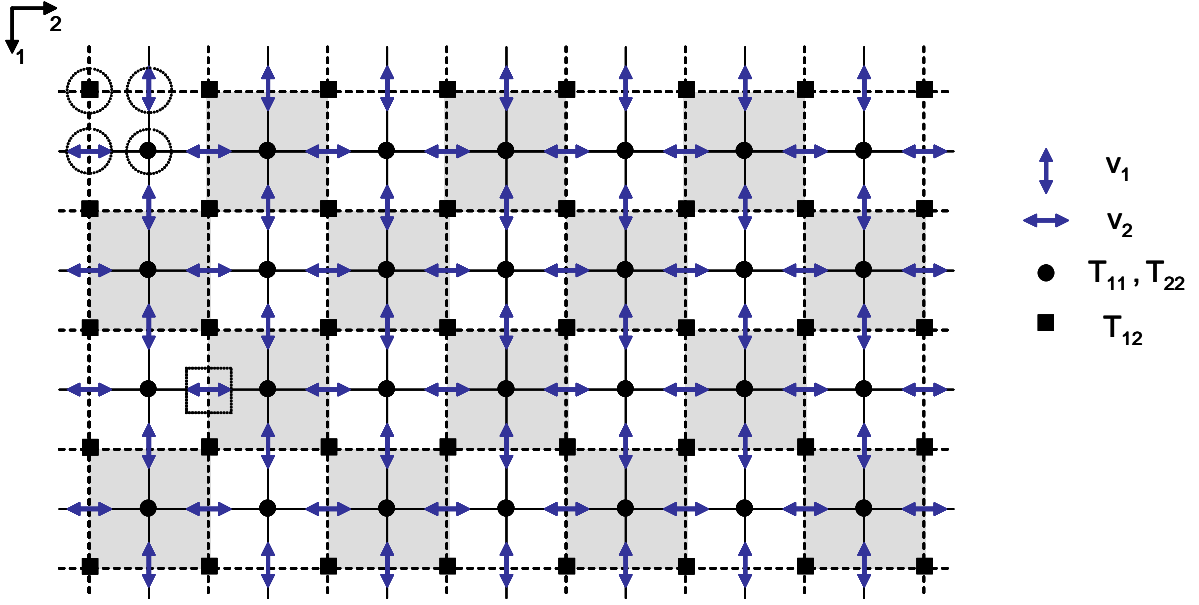


Figure 5: Illustration of the grids dimensions for a "4x7" simulation. The circled variables all have coordinates (0,0). The variable in the square is $v_2(2,1)$. The checkerboard is only intended to help visually identifying pixels of the Geometry.map2D file.

- All coordinates are integer numbers, and refers to a specific grid. As a consequence of the staggered-grids layout, although $T_{11}(0,0)$, $T_{12}(0,0)$, $v_1(0,0)$ and $v_2(0,0)$ all have the *same coordinates*, they corresponds to variables located at *different positions in space* (see circled variables in Fig. 5).

Material properties for heterogeneous domain In SimSonic2D, the users define materials properties via a $N_1 \times N_2$ image. As clear from 5, all variables except T_{11} and T_{22} are located at interfaces between pixels of the $N_1 \times N_2$ image, and cannot be "attached" to one pixel rather than others. It may be seen from Eqs. 6 and 5 that this requires defining averaged properties for values of the density and values of c_{12} .

- Density values are always required at the interface between *two adjacent pixels*, and are defined as the arithmetic average of the density values defined by the user at each of the two pixels.
- c_{12} values are always required at points that belong to *four adjacent pixels*, and are defined as the harmonic average of the c_{12} values defined by the user at each of the four pixels.

3.3 The Geometry.map2D input file

This file, which name must not be changed, is a binary file with a specific format described in appendix C. Briefly, it contains a $N_1 \times N_2$ bitmap image that uniquely defines the geometry of the materials presents in the simulation. Each material is represented by a 8-bit value, from 0 to 255. In practice, the user may use the Matlab SimSonic2D toolbox to create a $N_1 \times N_2$ matrix MAP of class uint8 and call `SimSonic2DWriteMap2D(MAP, 'SimulationDirectory\Geometry.map2D')`

3.4 The Parameters.ini2D input file

3.4.1 Principle

The Parameters.ini2D file, which name must not be changed, is a file in raw text. It can be read and edited with simple text editors such as wordpad under Windows or vi or emacs on Linux systems. As a Matlab toolbox is provided to conveniently SimSonic, Matlab is relevant to read and edit Parameters.ini2D. This file contains most of the information input by the user.

The way SimSonic reads this file is rather crude: SimSonic searches for pre-defined "code" strings in the text file, such as "Grid Step" for instance. Once the line containing the searched string is found, SimSonic reads the parameter found at position 31 on the line: as a general rule, the position of the parameter in Parameters.ini2D that is found after a "code" string on a line should always remain at the same position. **The order in which the various "code" strings and sections appears in the file is not important.** For all input parameters, SimSonic has built-in default values, that are used if fields are missing in the Parameters.ini2D file. As a consequence, a Parameters.ini2D file may have very little information, which helps in terms of readability, but the users must keep in mind default values.

Specifications and requirements for the various parameters are detailed thereafter. Provided that the user follows these requirements, any line of comments may be added to Parameters.ini2D to improve its readability. Although not mandatory, it is recommended that any line of comments begins with %, as it allows to easily discriminate between comments and input data when using Matlab to read and edit Parameters.ini2D.

Important: the natural system of units for MHz ultrasonics is mm, μs and mg. In the following, it is this system of units which is used, along with all the corresponding derived units (MHz, GPa, etc.). However, any system of units consistent with this one (such as stress and velocity numerical values are unchanged) may be used. An example of a such a system, relevant to geophysics, is given in appendix A.

The following sections describe in detail the various parameters contained in Parameters.ini2D.

3.4.2 General parameters

Grid Step is the value of the spatial grid step Δx , expressed in mm. [Default=0.1]

Vmax is a speed of sound value that must be larger than any encountered speed of sound in the simulation domain, expressed in $mm.\mu s^{-1}$. This value is extremely important, as it is used by SimSonic to derive the time step Δt from Δx , in accordance with the CFL stability condition (see **DeltaT Coefficient** thereafter). Note that Vmax may be different from the actual largest speed of sound in the simulation, provided that the CFL condition remains verified. [Default=1.5]

CFL Coefficient is the value of a multiplicative coefficient α used to compute the time step Δt , according to the following equation [Default=0.99]:

$$\Delta t = \alpha \times \frac{\Delta x}{\sqrt{2} \cdot V_{\max}} \quad (8)$$

If V_{\max} does correspond to the actual largest speed of sound in the simulation, then the CFL condition requires that $\alpha \leq 1$. In practice, this value should be strictly less than one, 0.99 for instance.

As a consequence of Eq. 8, the user controls Δt through both α and V_{\max} . In most situation, the user will choose V_{\max} as the exact value for the largest speed of sound present in the simulation, and $\alpha = 0.99$. However, if a user wants to run several simulations with different media (such as a simulation with scatterers and a reference simulation without scatterers), but with the same Δt , he/she should use the same V_{\max} and α for all the simulations. Most importantly, when a user makes a signal files (.sgl file for instance), it should be kept in mind that the sampling frequency used to build the signal must correspond to the value of Δt derived from Eq. 8 with the parameters used in Parameterers.ini2D.

Simulation Length is the duration of the simulated propagation, expressed in μs [Default=0.0]. If this value is not a multiple of the time step Δt , it is automatically rounded by SimSonic.

3.4.3 Boundary conditions

In SimSonic2D, the boundary conditions are given on four lines, named conventionally **X1_low**, **X1_high**, **X2_low** and **X2_high** as illustrated on Fig. 3.4.3

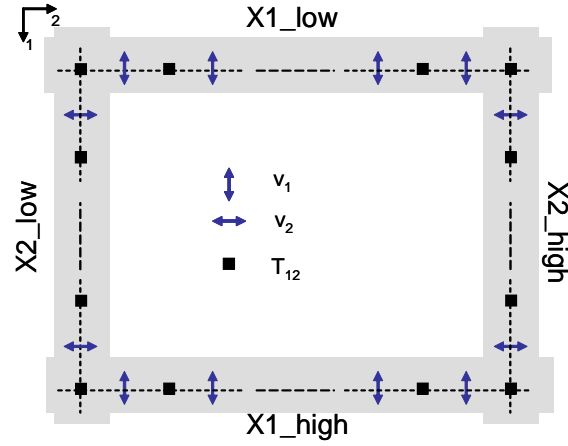


Figure 6: Naming of the four domain boundaries in SimSonic2D

The behavior of each boundary is defined by a number:

- 0 : a PML layer is defined against to the boundary. [Default value]
- 1 : the boundary behaves as a symmetric mirror
- 2 : stress-free boundary
- 3 : rigid boundary
- 4 : the boundary behaves as an anti-symmetric mirror

If one or more PMLs are used, they will all have the same properties defined by the following parameters (if no PML are used, these parameters are just not used by SimSonic2D):

PML Thickness is the PML dimension along the boundary normal, expressed in number of grid step Δx .[Default=40]

Vmax in PML is the highest of speed of sounds values in materials in contact with PML, expressed in $mm.\mu s^{-1}$.[Default=1.5]

PML Efficiency is the requested PML efficiency, expressed in dB. A value of 80 dB means that the wave reflected by the PML is expected to be 80 dB below the amplitude of a incident wave, in the case of normal incidence. [Default=80]

In a continuous world, PMLs are *perfectly* matched layer. Because of the space discretization in numerical methods such as FDTD or FEM (Finite Element Method), the PML loose their perfectness in such approaches. As a consequence, PML in SimSonic have a finite efficiency, that can be expressed as a coefficient of reflection. In practice, the maximum efficiency of a PML depends on the thickness of the PML relatively to the wavelength of the incident wave. As a rule of thumb, a PML should have a thickness of at least one wavelength in order to get an efficiency of several tens of dB for normal incidence. If a user requests a theoretical efficiency larger than that reachable given the PML thickness, the PML will simply not work as efficiently as expected. Moreover, the efficiency of the PML decreases from its maximum in normal incidence to zero for grazing incidence. Setting the PML parameters turns out to be very much based on the user experience. The PML in SimSonic2D are built on the approach described in detail in [3].

3.4.4 Sources

In SimSonic2D, the geometry of a single source object is a line array, which properties are listed below

- The array is aligned in one of the direction of the mesh (direction 1 or 2)
- The array has N identical elements equally distributed
- Each element may consist of a unique grid point or have some width along the array alignment

There are currently two possible ways to define sources in SimSonic2D:

1. The user may define the geometrical parameters of an emitters array in the Parameters.ini2D file. In this case, the signal emitted by all the elements are based on a unique signal defined in a .sgl file, which name is specified in the definition of the array. Options in the definition of the array allows applying delays or apodization coefficients on the elements of the array (see below).
2. The user may provide a .rcv2D file which contains all the information about the array geometry, and as many signals as the number of elements in the array. The .rcv2D format also corresponds to the format of signals received on 1D-array receivers transducers (hence the .rcv2D extension, 2D referring to SimSonic2D).

The parameters related to emission sources defined in the Parameters.ini2D file (case 1.) are listed below:

Type of Source Terms This value is set to 1 or 2 depending on how SimSonic2D considers source signals: 1 corresponds to use input signals as *source terms* in the equations for the corresponding variables. 2 corresponds to use input signals as *forced values* of the corresponding variables.[Default=1]

Number of VAR arrays VAR may be either T11, T22, T12, V1 or V2. [Default=0] This value is an integer indicating how many emitters arrays (for variables VAR) are going to be defined in the following lines of the Parameters.ini2D file. The definition of one array is given on 5 lines, and has the following structure:

-1	Filename			
Array normal				
x1_start	x2_start			
NBElts	Pitch	Width	Apodisation	Focus
Deflection	Velocity			

filename is the name of the .sgl file containing the signal that will be used by the array. -1 is an internal code for SimSonic2D, which must precede the file name. The file must be located in the simulation directory.

Array normal This number, equal to 1 or 2, indicates the direction of the normal to the array. 1 therefore means that the array is aligned along direction 2, and vice versa.

x1_start expressed in *grid coordinates* (integer), is the coordinate along direction 1 of the array element with the smallest coordinate.

x2_start expressed in *grid coordinates* (integer), is the coordinate along direction 2 of the array element with the smallest coordinate.

NBElts Number N of (identical) elements in the array

Pitch expressed in *number of grid steps* (integer), is the array pitch (center to center distance between two consecutive elements)

Width expressed in *number of grid steps* (integer), is the dimension of each element. All the “Width” points belonging to a given element behaves in the same way during signal emission.

Apodization 0 or 1. If set to 1, the signal amplitude on each element is apodized based on a [Hann window](#). This value must be set to 0 if the array only has one element.

Focus expressed in *number of grid steps* (integer). When different from 0, the array will focus on an axis perpendicular to the array, going through the center of the array, at a distance FocalLength from the array. This is done by SimSonic2D by delaying signal. This value may be positive or negative. This value must be set to 0 if the array only has one element.

Deflection expressed in *degrees* (floating point number), controls the emission angle for plane wave emission. It corresponds to add delays varying linearly with the element positions. This value may be positive or negative.

Velocity is the velocity expressed in physical velocity units (floating point number, usually mm/ μ s) used to calculate the delays for focusing or deflection. It should therefore correspond to the velocity of the homogeneous medium in which the array is located.

The best way to understand how the arrays are positioned is to consider the examples provided in section [3.4.5](#) on receivers arrays.

Number of VAR Array Source Files VAR may be either T11, T22, T12, V1 or V2. [Default=0]
This value N is an integer indicating how many emitters arrays (for variables VAR) are going to be defined from .rcv2D files. Directly following this line, there must be N file names corresponding to each array.

IMPORTANT REMARKS ON DEFINING SOURCES

The users has a complete freedom to define sources. However, consistency is not checked by the software. The following points (not an exhaustive list...) should be kept in mind:

- The users should be aware that in a fluid medium, T_{11} always equal T_{22} . Therefore, sources array for T_{11} and T_{22} should always be identical. This is left to the responsibility of the user: if not verified, the code will run, but the results should not be trusted.
- Particular attention should be paid for emitters arrays located on or close to domain boundaries. If a user defines sources on a boundary for which the boundary condition forces values to zero, the emitters array will be overridden by the boundary condition.
- Except for T_{11} and T_{22} in fluids, there should normally be only one array defined at a given location of the domain. In particular, a user will usually define *either* velocity sources *or* stress sources at a given location.

3.4.5 Receivers

As for emitters arrays, the geometry of a single receiver object is a line array, which properties are listed below :

- The array is aligned in one of the direction of the mesh (direction 1 or 2)
- The array has N identical elements equally distributed
- Each element may consist of a unique point or have some width along the array alignment

The information required from the user to define receivers arrays are the following :

Number of VAR Receivers arrays VAR may be either T11, T22, T12, V1 or V2. [Default=0]
This value is an integer indicating how many receivers arrays (for variables VAR) are going to be defined in the following lines of the Parameters.ini2D file. The definition of one array is given on 3 lines, and has the following structure:

```

Filename
Array normal
x1_start      x2_start
NBElts        Pitch      Width

```

Filename is the name of the .rcv2D file in which the signals will be recorded. The file will be written in the simulation directory.

Array normal This number, equal to 1 or 2, indicates the direction of the normal to the array. 1 therefore means that the array is aligned along direction 2, and vice versa.

x1_start expressed in *grid coordinates* (integer), is the coordinate along direction 1 of the array element with the smallest coordinate.

x2_start expressed in *grid coordinates* (integer), is the coordinate along direction 2 of the array element with the smallest coordinate.

NBElts Number N of (identical) elements in the array

Pitch expressed in *number of grid steps* (integer), is the array pitch (center to center distance between two consecutive elements)

Width expressed in *number of grid steps* (integer), is the dimension of each elements. The signal recorded for one element with “Width” points corresponds to the sum of the signals measured on each “Width”. In other words, each element is integrating over its width.

The best way to understand how the arrays are defined in SimSonic2D is to consider the following examples :

Nb of T11 Receivers Arrays : 2		
T11.ONE.rcv2D		
1		
1	0	
2	4	3
T11.TWO.rcv2D		
1		
3	2	
4	1	1
Nb of T12 Receivers Arrays : 1		
T12.rcv2D		
2		
0	0	
2	2	2

Table 1: Example of Receivers section in Parameters.ini2D

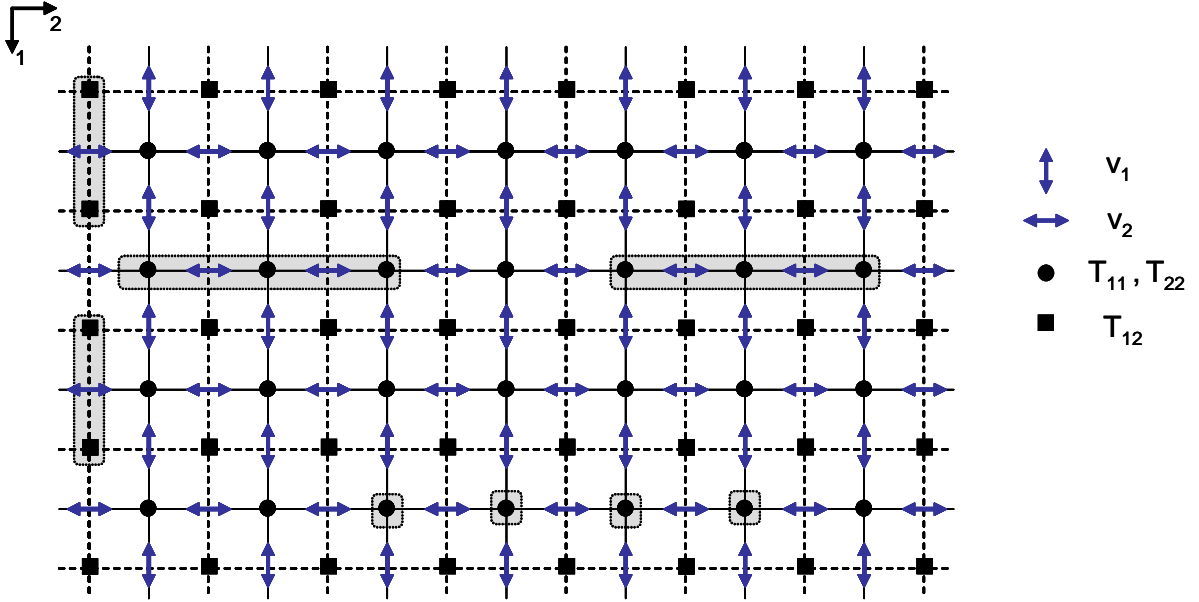


Figure 7: Position of the 3 arrays defined in Table 1

3.4.6 Snapshots

Snapshots Record Period expressed in physical time units (floating point number, usually μs).

This value is the time interval between snapshots.[Default=1]

Record VAR Snapshots 0 or 1. VAR may be either T11, T22, T12, V1, V2 or V. 1 indicates that VAR snapshots will be recorded with the time interval defined by **Snapshots Record Period**. [Default=0]

Remark: The displacement velocity $V = \sqrt{v_1^2 + v_2^2}$ is not a field variable, but is derived in

the software from the computations of v_1 and v_2 . However, as v_1 and v_2 are not defined on the same spatial grids, V is an averaged value over 4 pixels, defined by

$$V(i, j) = \sqrt{\left(\frac{1}{2} [v_1(i, j) + v_1(i + 1, j)]\right)^2 + \left(\frac{1}{2} [v_2(i, j) + v_2(i, j + 1)]\right)^2}.$$

V snapshots have size $N_1 \times N_2$. The value of V may be meaningless at interfaces between different media, in particular fluid/solid interfaces where the tangential velocity is discontinuous.

3.4.7 Definition of material properties

The Geometry.map2D file contains $N_1 \times N_2$ integer values that refer to media which material properties are described in the Parameters.ini2D file. These properties are defined for each material by the users, in a list located between two lines that contains "Starts Materials List" and "Ends Materials List". One material is defined on one line with the following structure:

```
Index  Density  C11  C22  C12  C66
```

Index are integer values ranging from 0 to 255, and all the physical properties are real values defined in consistent physical units (usually mg/mm^3 for density, GPa for $C_{\alpha\beta}$).

By default, all indexes correspond to materials with the following properties, typical of values for water:

```
Index  1  2.25  2.25  2.25  0
```

The user should make sure that all indexes present in Geometry.map2D are defined in the materials list. All indexes with no explicit definition will have the default properties.

3.5 Signals files

As previously introduced in Section 3.1, they are two types of signal files used by SimSonic.

The .sgl files are *input* files that contain a unique signal intended to be used by arrays which parameters are directly defined in the Parameters.ini2D file. The format of .sgl file is described in Appendix C. In practice, the user may use the Matlab SimSonic2D toolbox to create a signal `Waveform` of class double and call `SimSonic2DWriteSgl(Waveform, 'FileName')` to write .sgl files, or call `[Signal, NbPts]=SimSonic2DReadSgl('signal.sgl')` to read .sgl files.

The .rcv2D files are either input or output files. The .rcv2D format corresponds to the format of signals received or emitted on 1D-array transducers. A .rcv2D file contains all the geometrical parameters required to define an array (see sections 3.4.4 and 3.4.5), as well as the temporal grid step, the spatial grid steps, the number of points in the recorded signals, and the signals corresponding to each elements. The format of .rcv file is described in Appendix C.

In practice, the user may use the Matlab SimSonic2D toolbox to create and read .rcv2D files. From a matlab structure `array`, the user may call `SimSonic2DWriteRcv2D(array, 'FileName')` to write a 'FileName' file with .rcv2D format. A Filename file with .rcv2D format may be read into a matlab structure `array` by calling `array=SimSonic2DReadRcv2D('FileName')`.

3.6 Snapshots files

As previously introduced in Section 3.1, SimSonic2D uses .snp2D files to record snapshots of field variables at a given time. The .snp2D format, described in Appendix C, contains not only the field values, but also a header with the following information: dimensions of the snapshot (depending on the recorded variables, see section 3.2), time of the snapshot, temporal grid step and spatial grid step. In practice, the user may use the Matlab SimSonic2D toolbox to read .snp2D files into a matlab structure SNAPSHOT by calling `SNAPSHOT=SimSonic2DReadSnp2D(FILENAME)`.

3.7 Operating Systems and memory requirements

3.7.1 Operating systems

SimSonic is programed in C, and can therefore be compiled and executed on any platform, using operating systems such as Windows or Linux. SimSonic's code includes OpenMP directives, allowing SimSonic to run in parallel on several CPUs sharing a common memory. Users may find several pre-compiled executables on www.simsonic.fr, for both Windows and Linux systems and both 32-bit and 64-bit versions. SimSonic does currently not support MPI, and can therefore be ran only on clusters of CPU sharing a common RAM memory.

3.7.2 Memory requirements

The dimensions of SimSonic simulations are only limited by the available RAM. Simulations that requires more than typically 4 GB of RAM must be ran with 64-bit versions of SimSonic on 64-bit operating systems. Simulations with requirement below typically 2 GB may be run on 32-bit systems with 32-bit version of SimSonic.

The RAM needed for a simulation with grid dimensions $N_1 \times N_2$ and with a PML thickness of W grid points can be approximated by the following formula:

$$\text{RAM(MB)} = \frac{20}{1024^2} \times [N_1 N_2 + 4W^2 + 2W(N_1 + N_2)] \quad (9)$$

This formula holds when SimSonic computes fields variables with float precision (4 bytes). Although it is generally not needed in terms of precision, SimSonic may also used double precision (8 bytes), in which case the memory requirement is twice as that indicated in the above formula.

3.8 How to run a simulation

Running a simulation is straightforward: once all the required input files (Parameters.ini2D, Geometry.map2D, .sgl and/or .rcv2D) have been created and placed in a directory named SimulationDirectory, the user simply has to launch SimSonic2D from a command line with SimulationDirectory as argument, immediatly followed by a forward slash "/" or reverse slash "\", depending on the operating system. On Windows, the command line would look like

```
SimSonic2DPath\SimSonic2D_win64_omp.exe SimulationDirectory\
```

On Linux, the command would look like

SimSonic2DPath/SimSonic2D_gcc64_omp SimulationDirectory/

Windows-compiled versions are provided with or without [OpenMP](#) capabilities. For the OpenMP versions, the user may indicate the number of processors to use by setting the value of `OMP_NUM_THREADS`. For instance, to use two processors, use the following command line BEFORE running SimSonic2D:

```
SET OMP_NUM_THREADS=2
```

If this value is not set explicitly, the program will use the largest number of processors available.

The provided Linux-compiled version is for 64-bit systems with OpenMP. The number of processor to use can also be set by the user, in a way that usually depends on how jobs are launched on the system.

Once a simulation has been successfully launched, the software outputs information, either directly on the screen or in an output file, depending on the operating system. On Windows, the following information appear on the screen right after the computation has started:

```
Running SimulationDirectory\  
Started on : Thu Dec 29 15:36:44 2011
```

When the simulation is completed, the following information is displayed:

```
Running SimulationDirectory\  
Started on : Thu Dec 29 15:36:44 2011  
  
Ended on   : Thu Dec 29 15:36:56 2011  
Total computation time: 0h 0min 12sec
```

It is also possible to get information during the computation by using the following optional argument

SimSonic2DPath/SimSonic2D SimulationDirectory/ info

During the simulation, the following information is displayed, updated at each time step:

```
Running SimulationDirectory\  
Started on : Thu Dec 29 15:36:44 2011  
Computed: 0.5/1.0 us <--> Step: 67/114 Remaining time: 0h 0min 6sec
```

When the simulation is completed, the screen looks like the following:

```
Running SimulationDirectory\  
Started on : Thu Dec 29 15:36:44 2011  
Computed: 1.0/1.0 us <--> Step: 114/114 Remaining time: 0h 0min 0sec  
Ended on   : Thu Dec 29 15:36:56 2011  
Total computation time: 0h 0min 12sec
```

Note that the indicated remaining time is only an estimation, which is quite poor at the beginning of the simulation, and improves during the course of the computation.

4 Tutorial

This section is still under preparation. The reader is invited to check examples available at www.simsonic.fr

References

- [1] E. Bossy, M. Talmant, and P. Laugier. Three-dimensional simulations of ultrasonic axial transmission velocity measurement on cortical bone models. *Journal of the Acoustical Society of America*, 115(5):2314–2324, 2004. [3](#)
- [2] G C Cohen. *Higher-order numerical methods*. Springer, Berlin, 2002. [9](#)
- [3] F. Collino and C. Tsogka. Application of the pml absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media. *Geophysics*, 66(1):294–307, 2001. [9](#), [15](#)
- [4] D Royer and E Dieulesaint. *Elastic waves in solids I*. Springer-Verlag, Berlin, 2 edition, 1999. [4](#)
- [5] J. Virieux. Sh-wave propagation in heterogeneous media - velocity-stress finite-difference method. *Geophysics*, 49(11):1933–1942, 1984. [4](#)
- [6] J. Virieux. P-sv-wave propagation in heterogeneous media - velocity-stress finite-difference method. *Geophysics*, 51(4):889–901, 1986. [4](#)
- [7] K. S. Yee. Numerical solution of initial boundary value problems involving maxwells equations in isotropic media. *Ieee Transactions on Antennas and Propagation*, Ap14(3), 1966. [4](#)

A Physical Units in SimSonic2D

SimSonic has initially been developed to simulate ultrasonic propagation in the MHz frequency range. Accordingly, its system of units is different from the International Systems of Units. One coherent system of units, well suited for MHz ultrasonics, is given in the following table:

	base system			some derived units					
Quantity	length	time	mass	stress	mass density	velocity	frequency	force	viscosity
Unit	mm	μs	mg	GPa	$\text{mg}\cdot\text{mm}^{-3}$	$\text{mm}\cdot\mu\text{s}^{-1}$	MHz	kN	kPl

All examples described in this document or on the website have been designed within this system of units. For geophysics simulations, the following system of units is consistent in SimSonic:

	base system			some derived units					
Quantity	length	time	mass	stress	mass density	velocity	frequency	force	viscosity
Unit	km	s	Gt	GPa	$\text{Gt}\cdot\text{km}^{-3}$	$\text{km}\cdot\text{s}^{-1}$	Hz	PN	GPl

For simulations on the kHz range, the following system of units is consistent in SimSonic:

	base system			some derived units					
Quantity	length	time	mass	stress	mass density	velocity	frequency	force	viscosity
Unit	m	ms	t	GPa	$\text{t}\cdot\text{m}^{-3}$	$\text{m}\cdot\text{ms}^{-1}$	kHz	GN	MPl

For simulations on the μm scale, the following system of units is consistent in SimSonic:

	base system			some derived units					
Quantity	length	time	mass	stress	mass density	velocity	frequency	force	viscosity
Unit	μm	ns	pg	GPa	$\text{pg}\cdot\mu\text{m}^{-3}$	$\mu\text{m}\cdot\text{ns}^{-1}$	GHz	mN	Pl

For simulations on the nanometer scale, the following system of units is consistent in SimSonic:

	base system			some derived units					
Quantity	length	time	mass	stress	mass density	velocity	frequency	force	viscosity
Unit	nm	ps	10^{-21}g	GPa	$10^{-21}\text{g}\cdot\text{nm}^{-3}$	$\text{nm}\cdot\text{ps}^{-1}$	THz	nN	mPl

In all these systems of units, material properties such as mass density, rigidity constants and speeds of sound have the same numerical values (of the order of one).

B SimSonic2D Matlab Toolbox

The SimSonic2D Matlab toolbox contains all the necessary tools to prepare (write functions) and analyse (read functions) a simulation. In particular, this toolbox eliminates the need to know anything about the file formats used by SimSonic2D. The documentation of the functions listed in table 2 below may be found by use of the help command in Matlab, or directly in the .m files.

type	function name
read functions	SimSonic2DReadMap2D
	SimSonic2DReadRcv2D
	SimSonic2DReadSnp2D
	SimSonic2DReadSgl
write functions	SimSonic2DWriteMap2D
	SimSonic2DWriteRcv2D
	SimSonic2DWriteSgl

Table 2: Basic Matlab SimSonic2D functions

C File Formats

This section describes the formats of input and output files used by SimSonic2D. It is intended for users who may want to use their own code to create and read SimSonic files. However, the SimSonic2D matlab toolbox (see Appendix B) provides all the necessary functions to handle SimSonic2D files, and Matlab users should therefore not pay attention to file formats used by SimSonic.

Important: ordering convention for multi-dimensional data. In all relevant files, 2-D $N_1 \times N_2$ data are ordered in the following way: the first element encountered in the file is $(0, 0)$, the last element is $(N_1 - 1, N_2 - 1)$. In between, elements are [row-major ordered](#), i.e. the second dimension (N_2) is contiguous in the file.

- geometry file: .map2D
 - one integer (4 bytes) giving the value of N_1 .
 - one integer (4 bytes) giving the value of N_2 .
 - $N_1 \times N_2$ chars (1 byte per char)
- single signal file: .sgl
 - one integer (4 bytes) giving the number N of signal points.
 - N double (8 bytes per double). The first point in the file corresponds to the first point in time.
- array signal file: rcv2D
 - one char (1 byte) giving the array normal ('1' or '2').
 - one integer (4 bytes) giving the number N_e of array elements.
 - one integer (4 bytes) giving $x1_start$.
 - one integer (4 bytes) giving $x2_start$.
 - one integer (4 bytes) giving the array pitch.
 - one integer (4 bytes) giving the elements width.
 - one double (8 bytes) giving the spatial step Δx .
 - one double (8 bytes) giving the number N_t of signal points.
 - one double (8 bytes) giving the temporal step Δt .
 - $N_e \times N_t$ doubles (8 bytes per double).
- snapshot file: .snp2D
 - one integer (4 bytes) giving the size N_1 .
 - one integer (4 bytes) giving the size N_2 .
 - one double (8 bytes) giving the snapshot time (in physical time unit, usually μs).
 - one double (8 bytes) giving the spatial step Δx .
 - one double (8 bytes) giving the temporal step Δt .
 - $N_1 \times N_2$ floats (4 bytes per double).