I-Simpa, a graphical user interface devoted to host 3D sound propagation numerical codes

Judicaël Picaut, Nicolas Fortin

I-Simpa, a graphical user interface devoted to host 3D sound propagation numerical codes

J. Picaut and N. Fortin

Département Infrastructures et Mobilité (IFSTTAR/IM), Route de Bouaye, CS4, 44344 Bouguenais Cedex
judicael.picaut@ifsttar.fr

Whatever for indoor noise applications (room acoustics, noise in vehicles...) or sound propagation in the environment (open field, urban areas...), many numerical codes have been developed by researchers. Most of them have many common aspects, like the definition of the domain geometry and the materials (boundary conditions, impedance...), the definition of sound sources and of receivers (position, spectrum, directivity...). Moreover, they all have the same objective that is to predict the sound field within the domain, with several acoustics indicators, like acoustic pressure or sound levels, and through several representations (impulse response, spectrum...). In order to make easier the use of such codes (pre and post-processing of data), as well as for facilitating comparisons between models, a specific graphical user interface (I-Simpa) was developed specifically. In addition to previously cited facilities, this tool allows users to implement their own numerical models, as well as to manipulate the interface and to develop specific treatments by creating built-in Python(TM) scripts. As example, the paper will present the implementation of an energetic model for room acoustics predictions. The final objective is to create a community around this tool in order to exchange numerical codes, scripts, information...

1 Introduction

I-Simpa was initiated during a research project funded par the French Environment and Energy Management Agency (ADEME), in collaboration with the University of La Rochelle, the French institute of sciences and technology for transport, development and networks (Ifsttar), the University of Poitiers, and a French Engineering department SerdB. The main objective was to develop a new prediction model for the sound propagation in complex environments (indoor and outdoor). During the project, a graphical user interface (GUI) was developed specifically as a pre and post-processor of the prediction model. At a later time, the developers decided to generalize the GUI, in order to host other numerical calculation codes for 3D complex environments (see section 3.2).

Although I-Simpa is well adapted for energetic models (ray-tracing, sound-particle tracing, theory of reverberation...), it can be extended to use undulatory approaches. Classical applications are room and building acoustics, environmental noise and industrial noise, but it can be easily extend to other applications concerning the sound propagation in 3D environments (interior of vehicle, sound in cavities...).

The main concepts of I-Simpa are the following:

- a functional GUI: elements and components are organized in tree structures, to easily access to all information, parameters and properties. Many features are proposed for helping users;
- an "open" system: all information and data are organized in spreadsheets that can be displayed, exported, copied;
- an "open" tool: each user can integrate its own numerical propagation code, develop its own functionalities within the interface for its own applications.

2 Presentation of I-Simpa

2.1 Graphical user interface

The interface is organized on a main window, with a menu bar and a toolbar (figure 1), which contains a set of dockable/undockable sub-windows:

- The first window, titled "Project", is decomposed into three tabs, which should correspond to the three steps in conducting a project: preparing the 3D scene (tab "Scene"), running the calculation code (tab "Calculation"), showing/processing results (tab "Results"). The first tab "Scene" allows to define all acoustical characteristics that are needed for running the project, such as boundary properties, sound sources, receivers (see section 2.2), etc. In the second tab "Calculation", users can choose the numerical code and define all calculation parameters. In the last tab "Results", all numerical results are showed and can be post-processed if necessary. All data in each tab are organizing in one or more trees.

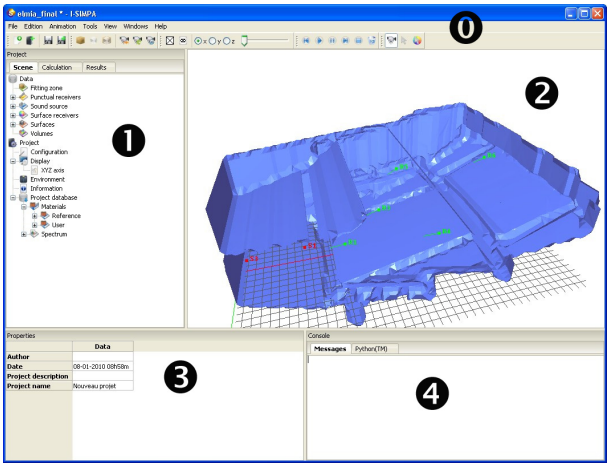


Figure 1: I-Simpa interface (with dockable/undockable windows): illustration with a room acoustics application.

- The second window shows the 3D geometry, with different views (with/without lines, interior/exterior/no surfaces, with/without material colors).
- The third windows, titled "Properties", gives all properties of an element, and is activated by selecting the corresponding element in one of the data trees.
- The fourth window is decomposed in two tabs. The first one ("Console") gives internal messages from I-Simpa or external messages from the calculation codes. The second one ("Python") allows to use Python^{TM} command lines to manipulate data.

2.2 Data project (input data)

As I-Simpa was firstly developed to use calculation codes based on energetic models, main features correspond

to common numerical codes of sound propagation based, for example, on ray-tracing approaches. However, new input data and features can be added using Python™scripts (see section 3.1).

Running a project requires to follow several steps:

1. importing 3D scene geometry (with triangle meshing): user can import 3D models with extension 3DS (3D Studio), STL (stereolithography), PLY (Stanford) and POLY (TetGen, see `tetgen.org`). A manual design of parallelepipedic models is also proposed. Specific importation tools have also been developed in order to correct model defaults (see section 2.3).

2. applying boundary materials: once the model is loaded, users have to assign materials (absorption, transmission loss, scattering, reflection law...) and information (mass density, resistivity...) for each surface element of the 3D scene. Data can been imported from the internal database or from external database. The internal database can also been modified by users.

3. adding punctual sound sources: the next step is to add sound sources in the model and to define all acoustic parameters (spectrum, directivity, delay...);

4. adding punctual and surface receivers: in order to compute several acoustical parameters, it must be necessary to locate receivers in the model. Punctual receivers as well as surface receivers (on the surface model or on arbitrary 2D planes) can be considered.

5. adding fitting zones: for some applications, such as for noise prediction in industrial halls, fitting zones with probabilistic parameters (mean free path, absorption, diffusion law) can be considered.

6. giving environmental data, such as atmospheric conditions and meteorological parameters.

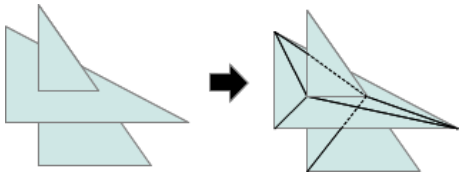
2.3 Model correction

A specific attention has been paid on the model importation. Indeed, some calculation codes require that 3D-models must be closed and without "default", which is not still possible in practice. Two importation tools have been added in I-Simpa in order to correct models that don't respect theses conditions, called Piecewise Linear Complex (PLC) constraints.

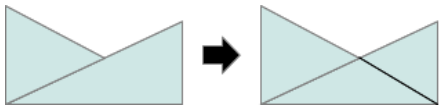
The first one, called "Preprocess" is applied when the model is closed, but contains intersections and discontinuities between faces (figure 2). In this case, the algorithm identifies intersections between faces and produces new faces to match with the PLC constraints. Two cases are considered:

- there is a non-coplanar intersection between two faces (*i.e.* a face intersects another one). Then each side of the intersection of the two faces is split into several faces. The final shape is the same, but the PLC constraints are respected;

- there is a discontinuity between two faces (*i.e.* they don't share the same edge), then, faces are split in two faces at each corresponding vertex.



(a) Intersection between two non-coplanar faces



(b) Discontinuity between two faces

Figure 2: Model corrections: faces intersection and discontinuity.

The second one is applied when the model contains many faces, with unknown holes and coplanar faces in intersection. In this case, user cannot reasonably fix by "hand" such geometry errors with a 3D modeling software. This issue is more complicated, and has involved the development of an advanced algorithm, called "mesh approximation", with five steps:

1. Voxelization: a voxel is the three-dimensional form of a pixel. The voxelization process creates a 3D matrix where cells are defined by their size and position in space. When a cell is in collision with the model surface the value of the cell is modified. This algorithm uses single dimension aggregation for processing speed (in the next step) and memory load reduction. Instead of store x times the same value of matrix on the K dimension the algorithm stores only a single value with an exponent x. The user must specify the resolution of the voxelization in model unit, smaller resolution giving more output faces. To remove holes that are present in the model, the resolution of the voxelization must be slightly higher than the maximum size of holes;

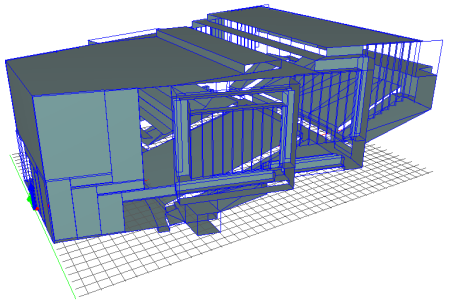
2. Volume identification: the matrix output of the voxelization contains set and unset cells. Unset cells form region separated by set cells. This step updates the matrix with a volume index value equal to the first empty cell found, and propagates through cell neighbors until it reaches non empty cells. This step is repeated until an empty cell cannot be found;

3. Marching cube algorithm: the previous step fills holes, but the volumetric model that is obtained must be translated to the surface model. The marching cube algorithm [1] is a well known method to convert 3D-scalar field into an iso-surface polygonal mesh;

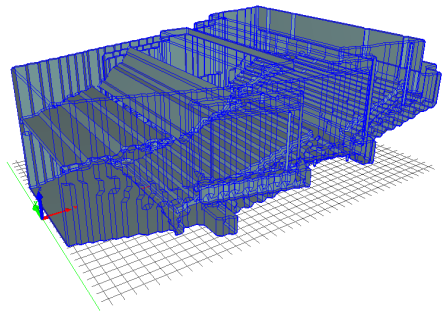
4. Vertex translation: the objective is to refine the output of the marching cube step to get closer to the original model. Each vertex of the marching cube mesh is then translated to the closest original model face.

5. Triangle decimation: this step allows to reduce the number of faces by merging co-planar faces. This is done by the progressive mesh face decimation algorithm [2] that collapses edges and topology-preserves the mesh.

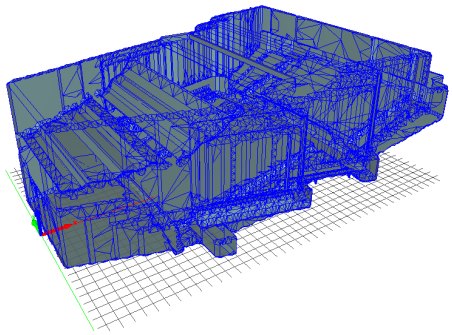
Figure 3 shows an example of the mesh approximation on the Elmia Concert Hall model (see `http://www.ptb.de/en/org/1/16/163/roundrobin/roundrob2_1.htm`) used during the Round Robin II [3]. The original model contains intersections and discontinuity between faces, wrong face orientations and holes. The final step gives a model without default, well oriented faces, close to the original one, which can be used with numerical codes requiring high constrained models.



(a) Original model



(b) Marching cube algorithm



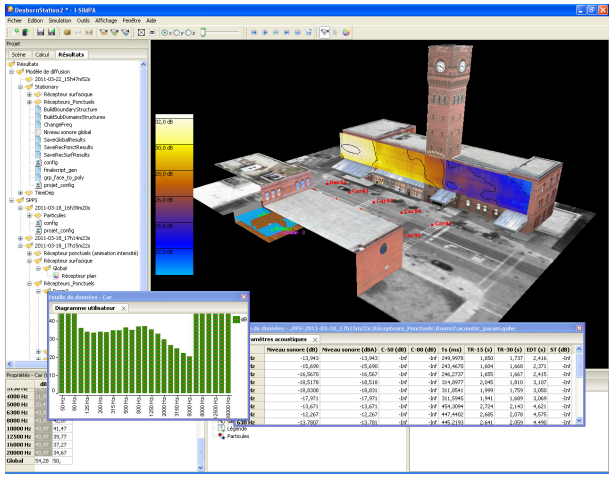
(c) Vertex translation

Figure 3: Generation of an approximate 3D model.

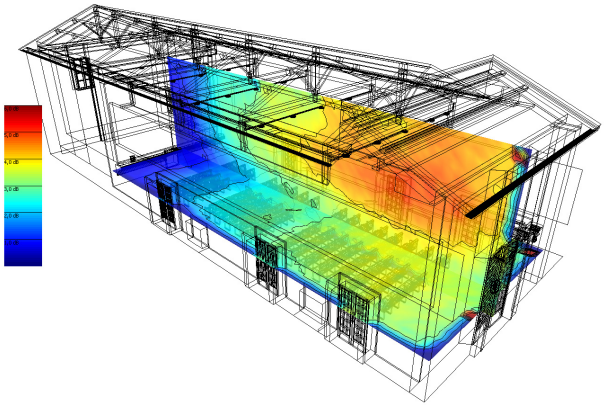
2.4 Results (output data)

Once a numerical code is linked with I-Simpa, user can run calculations. If the format of the output data files follows the I-Simpa I/O format, all results can been displayed in the "Results" tabs. Results are organized on a tree structure that is the exact image of the folders/files tree structure on the hard disk.

In function of the output file extensions, specific post-processing are allowed by I-Simpa. At the present time, since the interface was initially developed for energetic-based models, room acoustics parameters can been calculated according to the ISO 3382-1 standard [4], such as reverberation times, Clarity, Definition, Early support, Centre time, Strength, Early Lateral Energy Fraction, Late Lateral Sound Level. In addition, several results can been displayed such as echogram and Schroeder integration curves, intensity vectors, sound map (stationary, time-varying, cumulated values), iso-contour (for surface receivers), ray-tracing and particle-tracing animations (figure 4) on highly customizable graphics. All results can also be displayed on spreadsheets and are free to be exported out of I-Simpa (by copy/paste or using `CSV` files).



(a)



(b)

Figure 4: Illustration of the use of I-Simpa with a sound particle tracing code (SPPS) for outdoor/indoor applications: graphical and spreadsheet displays.

3 Extensions

One main concept of I-Simpa is to be "open" (figure 5), in order that people can use the GUI for their own applications, which means that new functionalities and other numerical codes can been considered (using user toolboxes). In addition, the GUI can be adapted for each user, by creating new language translation files (for internationalization) or changing the appearance of graphic objects.

3.1 Python™ commands and toolboxes

Users can extend the capabilities of I-Simpa by writing their own Python™ commands and scripts, using a specific Python™ library, called `uictrl`, which contains functions that are interpreted by I-Simpa. This allows to add new

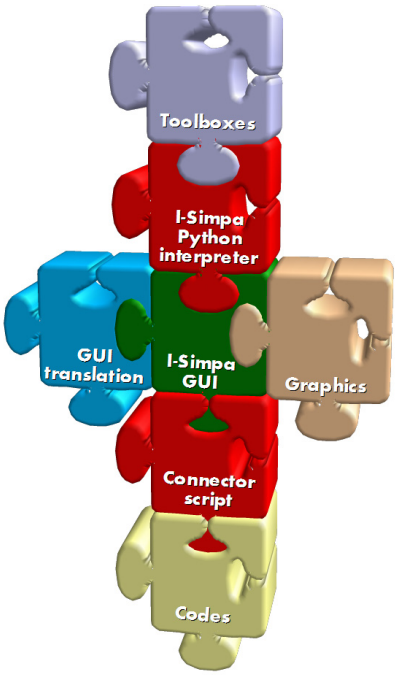


Figure 5: I-Simpa extensions.

functionalities within the interface, manipulate data and I/O files, realize new calculations and representations, create automatic processes, develop specific applications, link I-Simpa to other calculation codes or softwares.

A Python™ console is available within I-Simpa interface: users can write commands within the console to execute specific but simple actions. For long or complicated commands, users can develop and run scripts instead, and create toolboxes.

In practice, users can easily add elements on the data trees and add new functionalities when right-clicking on tree elements (popup menu), etc. After developing their own scripts, they can create toolboxes for running specific actions. By default (and as examples), I-Simpa is given with toolboxes for manipulating sound sources and receivers (translation, rotation...), creating and running a jobs list, and more.

Adding a toolbox within I-Simpa consists only in creating sub-folders in the `UserScript` folder located in I-Simpa installation folder with a file `__ui_startup.py` file inside, containing command line `import toolbox_name` for importing the module named `toolbox_name`. A second file `__init__.py` must be created in the same sub-folder and should contains the specific script of the new toolbox.

For instance the listing at figure 6 shows the script in the `__init__.py` file for adding an action in the popup menu that is activated when right clicking on a group of sound sources. It allows to activate or desactivate all sound sources in the group simultaneously. Others examples are also given in the I-Simpa documentation [5]. In this example, a specific class is creates and then load in I-Simpa using the last command `uctrl.application.register_menu_manager` that links the class with the corresponding tree element (*i.e.* a group of sound sources) with I-Simpa.

3.2 Calculation codes

By default, I-Simpa is given with two calculation codes, which are appropriate for room acoustics applications (TCR and SPPS) as well as environmental and industrial noise (SPPS). The TCR code is a simple implementation of the classical theory of reverberation and can be applied for calculating simple room acoustics parameters (for a single room). The SPPS code is based on particle-tracing method, similar to classical ray-tracing methods, and is well adapting for calculating sound fields in complex environments [6]. In addition, I-Simpa has been designed in order to host any calculation codes of sound propagation in 3D model. Although, I-Simpa is well adapted to energetic sound propagation model (like TCR and SPPS), it can be extend to undulatory approaches. As example, the integration of the transmission line matrix method (TLM) in I-Simpa is currently in progress [7].

The use of external codes in I-Simpa requires (1) that the calculation code can use 3D-models with triangular faces and (2) that the executable file can be called using a bash command line. In practice, in order to add the calculation code in I-Simpa (*i.e.* in the tab "Calculation" of the sub-windows "Project"), users have to declare the new code using a Python™ script in the `UserScript` folder containing the corresponding Python™ commands for importing the code. In addition users have to put the executable file in a new folder within the `core` folder that is located in the I-Simpa installation folder. Lastly, users have to write an connector script in order that I/O files between I-Simpa and the code be compatible. The last action can be done by using two librairies `libsimpa` (for Python™ programming) and `libinterface` (for C++ programming), designed for manipulating I/O I-Simpa files.

3.3 GUI translation (internationalization)

I-Simpa is designed for international users. At the present time, I-Simpa provides translations for French and English users. As I-Simpa uses language files for translation, users can propose new translations (*i.e.* in other languages) or corrections of an existing translation. In addition, users who develop their own Python™ scripts can also use language files in order to internationalize their own applications.

In I-Simpa coding, specific functions have been used in order that expressions can be internationalized. This is for example the case at figure 6 for the "`_("Enable")`" expression that can be translated *a posteriori* in another language. After compiling the code with a specific tool (like `Poedit`, see `www.poedit.net`), it produces a Portable Object file containing translations. Then, the `.po` file is compiled in a Machine Object file (`.mo` file) in order to be readable by I-Simpa. For each language (*i.e.* French, English...), the corresponding `.mo` file is located in a folder language at the root directory of I-Simpa installation.

3.4 Graphics

In spite of the poor interest, I-Simpa allows to replace the existing toolbar and menu icons by user icons, by replacing the corresponding graphics files in the `Bitmaps` folder of the root directory of I-Simpa by new ones. Filenames are sufficiently explicite to identify each icon and button; the first word in the filename define the category of the icon: tree

```
class manager:
    # This class make the user able to enable or disable a group of sound sources with one click only
    def __init__(self):
        #Constructor. Register the two new menu functions
        self.enable_grp_sourcesid=uictrl.application.register_event(self.enable_grp_sources)
        self.disable_grp_sourcesid=uictrl.application.register_event(self.disable_grp_sources)
    def getmenu(self,typeel,idel,menu):
        #Called by the user interface. The list menu structure contains the current implemented functions.
        submenu=[(uictrl._("Enable"),self.enable_grp_sourcesid),(uictrl._("Disable"),self.
                                                                 disable_grp_sourcesid)]
        menu.insert(2,(uictrl._("All emitters"),submenu))
        menu.insert(2,()) #Add a separator
        return True
    def set_grp_src_activation(self,idgrp,newstate):
        grpsrc=uictrl.element(idgrp)
        all_property=grpsrc.getallelementbytype(uictrl.element_type.
                                                ELEMENT_TYPE_SCENE_SOURCES_SOURCE_PROPRIETES)
        for prop in all_property:
            uictrl.element(prop).updateboolconfig("enable",newstate)
    def enable_grp_sources(self,idgrp):
        #Called by user interface when the user click on the enable menu item
        self.set_grp_src_activation(idgrp,True)
    def disable_grp_sources(self,idgrp):
        #Called by user interface when the user click on the disable menu item
        self.set_grp_src_activation(idgrp,False)

#Register the toolbox in I-SIMPA: this toolbox is linked with sources groups
uictrl.application.register_menu_manager(uictrl.element_type.ELEMENT_TYPE_SCENE_SOURCES, manager())
```

Figure 6: Script example: activating/desactivating a group of sound sources.

(an element in the tree structure), popup (an icon in a menu popup), toolbar (an icon in the toolbar).

In addition, new color palettes for the representation of sound maps can be added within I-Simpa. Palettes are defined using the GIMP file format with the extension `.gpl` (see `http://www.gimp.org/`).

4 Support and community

As I-Simpa has been developed "by researchers for researchers", the objective is to create a community around the software, in order to share information, scripts, toolboxes, applications, translation files, etc. For that purpose, a specific web site (`i-simpa.ifsttar.fr`) has been realized with several tools:

- Request and bug tracking: users can report malfunctions, errors, bugs or simply requests using the tracking system;
- Forums: several forums have been opened in order to share information and find answers to problem;
- Showcases: users can share I-Simpa projects and ilustrations/screenshots.

5 Conclusion

With I-Simpa, authors have tried to develop a software for their own need, mainly in order to simplify the use and the development of sound propagation model in 3D complex environments. Since such software can also be of interest for the whole acousticians' community, authors have decided to freely share I-Simpa and to propose tools in order that users can used I-Simpa for their own applications. At this step, the objective is to create a specific community around I-Simpa in order to share works and to develop new versions of I-Simpa that will be enhanced by the community.

References

[1] W.E. Lorensen, H.E. Cline, "Marching Cubes: A high resolution 3D surface construction algorithm", *Computer Graphics* **21**(4), 163-169 (1987)

[2] H. Hoppe, "Progressive meshes", *SIGGRAPH 96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* **30**, ACM, 1996.

[3] I. Bork, "A comparison of room simulation software. The 2nd Round Robin on Room Acoustical Computer Simulation", *Acustica* **86**, 943-956 (2002)

[4] International Organization for Standardization, ISO 3382-1:2009: Acoustics – Measurement of room acoustic parameters – Part 1: Performance spaces" (2009)

[5] J. Picaut, N. Fortin, *Manuel d'utilisation de I-Simpa* (in French), Ifsttar, Université de la Rochelle (2011)

[6] J. Picaut, N. Fortin, "SPPS, a particle-tracing numerical code for indoor and outdoor sound propagation prediction", *Acoustics 2012 Nantes Conference* (2012)

[7] G. Guillaume, J. Picaut, G. Dutilleux, B. Gauvreau, "Time-domain impedance formulation for transmission line matrix modelling of outdoor sound propagation", *Journal of Sound and Vibration* **330**(26), 6467-6481 (2011)