A Simple Lattice Infiller

User Manual

(May, 2022)

ASLI website
ASLI github repository

ASLI Version 0.1

KU LEUVEN

Contents

1 Introduction **1**
- 1.1 Authors . . . 1
- 1.2 Licensing . . . 1
- 1.3 Citing ASLI . . . 1
- 1.4 Acknowledgments . . . 1

2 Getting started **2**
- 2.1 Prerequisites . . . 2
- 2.2 Building ASLI in Linux . . . 2
- 2.3 Building ASLI in Windows . . . 3
- 2.4 Parallel mode . . . 4
- 2.5 Graphical user interface . . . 4

3 Using ASLI **4**
- 3.1 Configuration file . . . 4
- 3.2 Graphical user interface . . . 5
- 3.3 Demo . . . 5
  - 3.3.1 Running the demo making use of the configuration file . . . 6
  - 3.3.2 Running the demo making use of the GUI . . . 8
- 3.4 Other examples . . . 9

4 Support **9**

Appendix A: Input parameters **10**
- A.1 Input/output files . . . 10
- A.2 Lattice parameters . . . 11
- A.3 Mesh parameters . . . 12
  - A.3.1 CGAL specific parameters . . . 12
  - A.3.2 Mmg specific parameters . . . 13

Index of input parameters **15**

1 Introduction

ASLI (A Simple Lattice Infiller) is a cross-platform command line open-source tool written in C++ that gives users the ability to provide functionally graded lattice infills to any 3D geometry. The lattice infill is constructed out of unit cells, described by implicit functions, whose type, size and feature can be varied locally to obtain the desired local properties. ASLI also has an optional Graphical User Interface (GUI) available, called QASLI.

Details on the technical aspects of ASLI can be found in "A flexible and easy-to-use open-source tool for designing functionally graded 3D porous structures".

1.1 Authors

ASLI and QASLI are developed by the Biomechanics Research Unit (University of Liège and KU Leuven).

Principal developers

- F. Perez-Boerema [Developer of ASLI] (KU Leuven, Belgium)
- M. Barzegari [Developer of QASLI] (KU Leuven, Belgium)

Principal investigator

- L. Geris (University of Liège and KU Leuven, Belgium)

1.2 Licensing

ASLI is licensed under the terms of the GNU Affero General Public License.

1.3 Citing ASLI

If you use ASLI for your research we kindly ask you to cite:

- F. Perez-Boerema, M. Barzegari and L. Geris. (2022). A flexible and easy-to-use open-source tool for designing functionally graded 3D porous structures, *Virtual and Physical Prototyping*, 17:3, 682-699, DOI: 10.1080/17452759.2022.2048956.

1.4 Acknowledgments

The authors gratefully acknowledge funding from the European Regional Development Fund – Interreg VA Flanders - The Netherlands (PRosPERoS, CCI 2014TC16-RFCB046), the European Union's Horizon 2020 research and innovation programme via the European Research Council (ERC CoG INSITE 772418), the Fund for Scientific Research Flanders (G085018N), the Fédération Wallonie-Bruxelles through the BioWin project BIOPTOS (7560) and the KU Leuven Special Research Fund (C24/17/07).

2 Getting started

The easiest way to get started with ASLI is to use its pre-build binaries. To this end, all you need to do is download the tarballs for your preferred platform (Linux or Windows) from the release page of ASLI repository, unpack them and if using Linux install oneTBB if not available on your system (only required by the parallel version of ASLI). The tarball contains ASLI, including its GUI. For more advanced advanced users, it may be more interesting to build ASLI from scratch. Not only is this likely to increase performance since the program will get optimized for the platform in which it is going to be run, but it will also enable users to customize ASLI in any way they need. The prerequisites to be able to build ASLI are detailed in Section 2.1, while the procedure to build ASLI is detailed in Section 2.2 and 2.3, for Linux and Windows respectively.

2.1 Prerequisites

ASLI makes use of CMake and GNU Make to automate the building process. It also has a series of dependencies, which are listed below.

- AdaptTools
- ALGLIB
- CGAL
  - GMP
  - MPFR
  - Boost
  - oneTBB (optional, required for parallelization)
- Eigen
- MMG
- TETGEN
- Yaml-cpp
- QASLI (optional, the GUI of ASLI)
  - QT (optional, required if compiling the GUI)

Most dependencies are included with ASLI so that the user does not need to worry about them. Not included with ASLI are GMP, MPFR, Boost and TBB (optional). QT, required to compile the GUI of ASLI is not included either. To be able to compile ASLI, depending on the compilation settings, users will need some or all of these libraries available on their system.

2.2 Building ASLI in Linux

If you are missing some required or optional dependencies they can be installed on Debian and Ubuntu with a couple of commands.

- Install build tools:

```
$ sudo apt-get install git cmake
```

- Install compilers:

  ```
  $ sudo apt-get install build-essential
  ```

- Install libraries (GMP, MPFR, Boost):

  ```
  $ sudo apt-get install libgmp-dev libmpfr-dev libboost-all-dev
  ```

- Install libraries (oneTBB):

  ```
  $ sudo apt-get install libtbb-dev
  ```

- Install libraries (QT):

  ```
  $ sudo apt-get install qt3d5-dev
  ```

Once the build tools, compilers and required dependencies are available on the system the main steps to build ASLI in Linux are the following.

1. Retrieve ASLI from the repository:

   ```
   $ git clone https://github.com/tpms-lattice/ASLI.git
   ```

2. Compile:

   ```
   $ cd ASLI
   $ mkdir build
   $ cd build
   $ cmake -DCMAKE_BUILD_TYPE=Release -DMARCH_NATIVE=ON ..
   $ make
   ```

To clean up, simply delete the `build` and `bin` directories found in the source directory.

2.3 Building ASLI in Windows

Windows compilation is supported with MSYS2. Once you have installed MSYS2 in windows the required and optional dependencies can be installed through the MSYS2 MinGW 64-bit terminal with a couple of commands.

- Install build tools:

  ```
  $ pacman -S git mingw-w64-x86_64-cmake
  ```

- Install compilers, and libraries (GMP, MPFR, Boost):

  ```
  $ pacman -S --needed base-devel mingw-w64-x86_64-toolchain
  $ pacman -S msys2-runtime-devel
  ```

- Install libraries (oneTBB):

  ```
  $ pacman -S mingw-w64-x86_64-intel-tbb
  ```

  and add the location of `tbb.dll` and `tbbmalloc.tbb` to the windows environment variables.

- Install libraries (QT):

  ```
  $ pacman -S mingw-w64-x86_64-qt5
  ```

Once the build tools, compilers and required dependencies have been installed the main steps to build ASLI for Windows in MSYS2 are the following.

1. Retrieve ASLI from the repository:

```
$ git clone https://github.com/tpms-lattice/ASLI.git
```

2. Compile:

```
$ cd ASLI
$ mkdir build
$ cd build
$ cmake -G "MSYS Makefiles" -DCMAKE_BUILD_TYPE=Release
↪  -DMARCH_NATIVE=ON ..
$ make
```

To clean up, simply delete the `build` and `bin` directories found in the source directory.

2.4 Parallel mode

To compile ASLI in parallel mode include the tag `-DCGAL_ACTIVATE_CONCURRENT_MESH_3=ON` when calling cmake. Note that parallel mode is currently limited to the CGAL workflow.

2.5 Graphical user interface

To compile ASLI together with its GUI include the tag `-DASLI_GUI=ON` when calling cmake.

3 Using ASLI

After compiling ASLI you will find an executable file named `ASLI` in the bin folder. If you specified you would like the GUI to be compiled along with ASLI you will find a second executable in the bin folder named `QASLI`.

In order to call ASLI from the command line you only need to type the following command in the Linux terminal:

```
./ASLI config.yml
```

or if using windows:

```
ASLI.exe config.yml
```

where `config.yml` points to the configuration file.

If using the GUI of ASLI users will only need to open QASLI, through which all user input parameters can be specified prior to executing ASLI.

3.1 Configuration file

ASLI makes use of a YAML configuration file that can be edited in any text editor. Through this file users can specify the user input parameters. The structure of the configuration file is shown in Fig. 1 while a detailed description of each input parameter can be found in Appendix 4.

```yaml
# ----  CONFIGURATION FILE  ---- #

# Input & output files
files:
  stl: inputs/cube.stl
  tap: inputs/cube.tap
  sap: inputs/cube.sap
  fap: inputs/cube.fap
output: outputs/

# Lattice settings
lt_type: gyroid
lt_type_filterRadius: 1.0
lt_type_correctionFactor: 0.25

lt_size: 0.5

lt_feature: isovalue
lt_feature_val: 0.5
lt_feature_mode: relative

# Mesh settings
me_mesher: CGAL
me_side: scaffold
me_volumeMesh: FALSE
me_nThreads: 1

# Mesh settings (CGAL)
me_facetAngle: 0
me_facetSize: 0
me_facetDistance: 0.015
me_cellRadiusEdgeRatio: 0
me_cellSize: 0
me_preserveEdges: TRUE
me_poissonOffset: 0

# Mesh settings (MMG)
me_hvol: 0
me_hinitial: 0
me_hmin: 0
me_hmax: 0
me_hausd: 0.42
me_hgrad: 0
```

Figure 1: YAML configuration.

3.2 Graphical user interface

QASLI, the GUI of ASLI, is an alternative to using the configuration file. The GUI is shown in Fig. 2. For the sake of simplicity the GUI hides by default most non-essential parameters. Access to these more advanced parameters can be obtained by pressing ctrl + ⇧ shift + e. For more details regarding the "Standard" and "Advanced" parameters see Appendix 4.

3.3 Demo

ASLI includes one demo problem. The files required for the demo are the `cube.stl` file, containing the $1 \times 1 \times 1$ cube shown in Fig. 3a, and the `cube.tap`, `cube.sap` and `cube.fap` files, which contain the local type, size and feature (isovalue) specifications shown in Fig. 3b-d. All four files can be found in the `inputs` folder of ASLI.

The demo files allow users to create a cube with constant infill (Fig. 4a) or a cube with a functionally graded infill that can be hybrid, pseudo-periodic and/or heterogeneous (Fig. 4b-e).

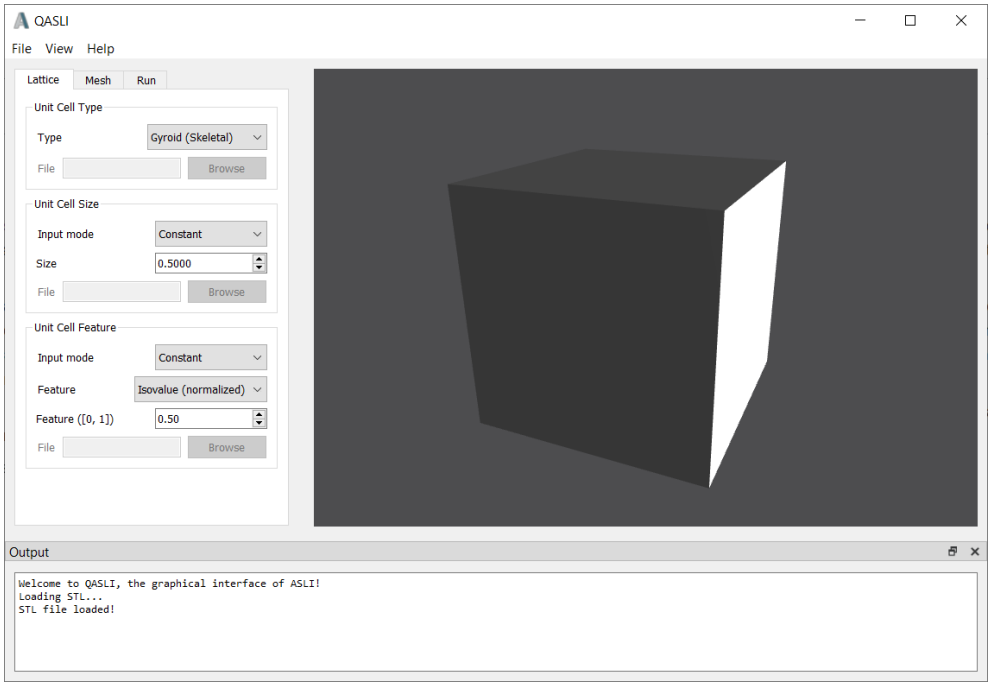


Figure 2: QASLI, the graphical user interface of ASLI.

3.3.1 Running the demo making use of the configuration file

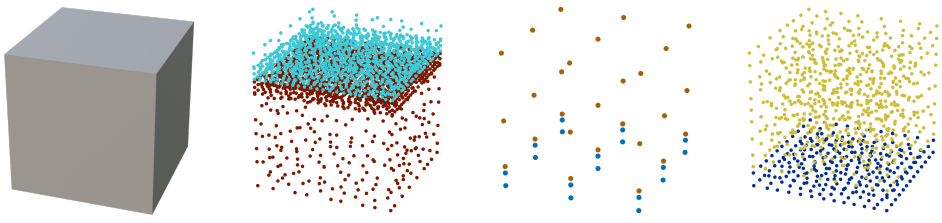
To run the demo problem using the configuration file start by opening the unedited `config.yml` file provided with a text editor and modify the parameters specified below depending on the infill you would like to obtain.

a) Constant infill (Fig. 4a)

- Set `lt_size` to `0.25`

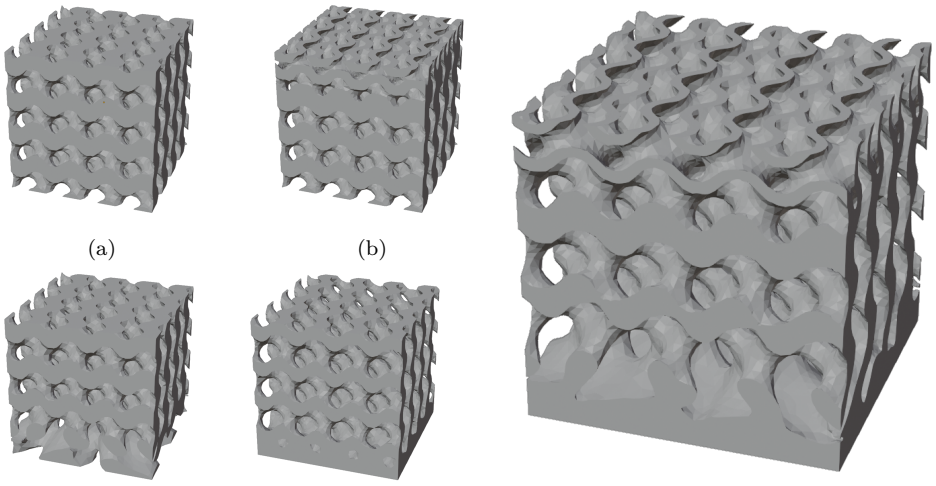
b) Hybrid infill (Fig. 4b)

- Set `tap` to `inputs/cube.tap`
- Set `lt_type to hybrid`



(a) (b) (c) (d)

Figure 3: Input files of the demo problem (a) input geometry, (b) type at point data [• sheet-gyroid • strut-gyroid], (c) size at point data [• 0.25 • 0.5] and (d) feature at point data (isovalue) [• 0.35 • 1].



(a) (b) (c) (d) (e)

Figure 4: $1 \times 1 \times 1$ cube with an (a) constant infill, (b) hybrid infill, (c) pseudo-periodic infill, (d) heterogeneous infill and (e) hybrid pseudo-periodic heterogeneous infill.

- Set `lt_size to 0.25`

c) Pseudo-periodic infill (Fig. 4c)

- Set `sap` to `inputs/cube.sap`
- Set `lt_size` to `0`

d) Heterogeneous infill (Fig. 4d)

- Set `fap` to `inputs/cube.fap`
- Set `lt_size` to `0.25`
- Set `lt_feature_val` to `0`

e) Hybrid pseudo-periodic heterogeneous infill (Fig. 4e)

- Set `tap` to `inputs/cube.tap`
- Set `sap` to `inputs/cube.sap`
- Set `fap` to `inputs/cube.fap`
- Set `lt_type` to `hybrid`
- Set `lt_size` to `0`
- Set `lt_feature_val` to `0`

Save the modifications made and execute ASLI by calling `./ASLI config.yml` if using Linux or `ASLI.exe config.yml` if using windows. The generated lattice will be stored as an `.stl` file in the outputs folder.

3.3.2 Running the demo making use of the GUI

To run the demo using the GUI the first step, after opening QASLI, is to load the `.stl` file containing the geometry to be provided of an infill. To do so go to File > Load Surface (.stl), open the folder `inputs`, select the file named `cube.stl` and click Open.

Next, set the lattice and mesh parameters depending on the infill you would like to obtain following the instructions below.

a) Constant infill (Fig. 4a)

1. Set **Lattice** > Unit Cell Size > Size to `0.25`

b) Hybrid infill (Fig. 4b)

1. Set **Lattice** > Unit Cell Type > Type to '*From file*'
2. Click on the corresponding Browse button, navigate to the `inputs` folder, select the file `cube.tap` and click Open.
3. Set **Lattice** > Unit Cell Size > Size to `0.25`
4. Set **Mesh** > Mesh Engine > Mesher to '*CGAL*'

c) Pseudo-periodic infill (Fig. 4c)

1. Set **Lattice** > Unit Cell Size > Input mode to '*From file*'
2. Click on the corresponding Browse button, navigate to the `inputs` folder, select the file `cube.sap` and click Open.
3. Set **Mesh** > Mesh Engine > Mesher to '*CGAL*'

d) Heterogeneous infill (Fig. 4d)

1. Set **Lattice** > Unit Cell Size > Size to `0.25`
2. Set **Lattice** > Unit Cell Feature > Input mode to '*From file*'
3. Click on the corresponding Browse button, navigate to the `inputs` folder, select the file `cube.fap` and click Open.
4. Set **Mesh** > Mesh Engine > Mesher to '*CGAL*'

e) Hybrid pseudo-periodic heterogeneous infill (Fig. 4e)

1. Set **Lattice** > Unit Cell Type > Type to '*From file*'
2. Click on the corresponding Browse button, navigate to the `inputs` folder, select the file `cube.tap` and click Open.
3. Set **Lattice** > Unit Cell Size > Input mode to '*From file*'
4. Click on the corresponding Browse button, navigate to the `inputs` folder, select the file `cube.sap` and click Open.
5. Set **Lattice** > Unit Cell Feature > Input mode to '*From file*'
6. Click on the corresponding Browse button, navigate to the `inputs` folder, select the file `cube.fap` and click Open.
7. Set **Lattice** > Mesh Engine > Mesher to '*CGAL*'

Finally, go to the **Run** tab and click Run. Once finished you will be notified, dismiss the notice by clicking on Ok. The generated lattice will be displayed automatically in the viewer and stored as an `.stl` file in the outputs folder.

3.4 Other examples

A few examples that showcase ASLI capabilities are shown in Fig. 5.

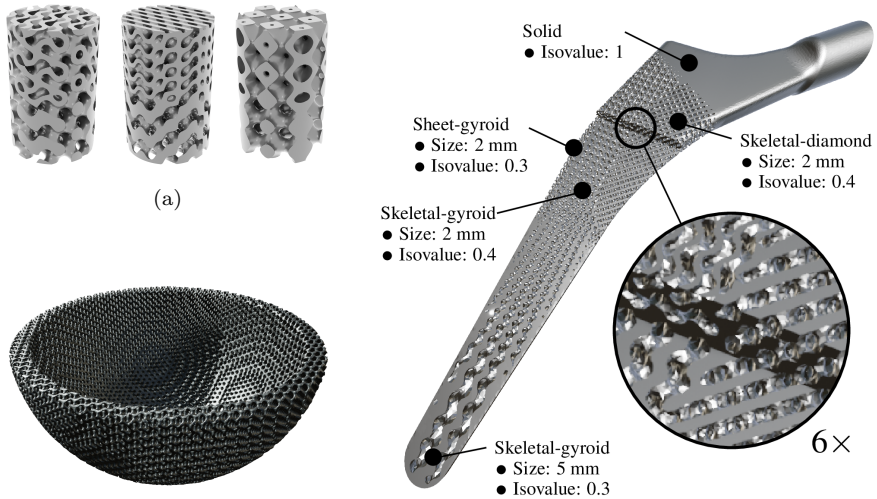


Figure 5: Examples of lattice structures generated with ASLI: (a) functionally graded cylinder test samples, (b) functionally graded acetabular cup and (c) functionally graded femoral implant with stem cut in half to show internal structure.

4 Support

If you have questions not addressed in this manual, there are a number of resources listed below you can use.

- You can create a post on stackoverflow with the tag 'ASLI'.
- You can report bugs using ASLI issue tracker on GitHub. You are also welcome to use the issue tracker to ask questions or post suggestions.
- If you have specific questions about ASLI that are not suitable for public and archived mailing lists you can contact us at `asli@kuleuven.be`.
- ASLI relies on CGAL and MMG in order to discretize the scaffolds. If you have specific questions about CGAL or MMG, you can contact their communities at `https://www.cgal.org/support.html` and `https://forum.mmgtools.org`.

Appendix A: Input parameters

ASLI requires multiple parameters to be specified by the user. A description of all input parameters is found below. Some of the parameters have default values. These default values work well for many use cases and are therefore considered a good starting point. In the description of the input parameters there is also a "Standard/Advanced" label included. "Advanced" parameters are only required for specific cases and be can be ignored otherwise. For convenience all input parameters are indexed in the Section "Index of input parameters" found at the end of the manual.

A.1 Input/output files

- *Parameter name:* `stl`

  Default: Should be provided

  Description: [Standard] STL file containing the object to be provided of an infill.

  Possible values: String value

- *Parameter name:* `tap`

  Default: Should be provided

  Description: [Standard] Type at point file containing local unit cell type specifications.

  Possible values: String value

- *Parameter name:* `sap`

  Default: Should be provided

  Description: [Standard] Size at point file containing local unit cell size specifications.

  Possible values: String value

- *Parameter name:* `fap`

  Default: Should be provided

  Description: [Standard] Feature at point file containing local unit cell feature specifications.

  Possible values: String value

- *Parameter name:* `output`

  Default: ./outputs/

  Description: [Standard] Location were the output file(s) will be stored.

  Possible values: String value

A.2 Lattice parameters

- *Parameter name:* `lt_type`

  Default: Should be provided

  Description: [Standard] Parameter specifying the unit cell type, i.e. *gyroid*, *sheet_gyroid*, *diamond*, *sheet_diamond*, *primitive*, *sheet_primitive*, *IWP* or *sheet_IWP*. To specify multiple unit cell types set to *hybrid* and provide a .tap file (see `tap` parameter).

  Possible values: String value

- *Parameter name:* `lt_type_filterRadius`

  Default: 1.0

  Description: [Advanced] Radius specifying the size of hybrid regions. The radius scales automatically with local unit cell size. Only active if `lt_type` is set to *hybrid*.

  Possible values: Non-negative floating point number

- *Parameter name:* `lt_type_correctionFactor`

  Default: 0.25

  Description: [Advanced] Correction factor to compensate for the thinning effect of hybridization. Only active if `lt_type` is set to *hybrid*.

  Possible values: Non-negative floating point number

- *Parameter name:* `lt_size`

  Default: Should be provided

  Description: [Standard] Parameter specifying the unit cell size. Set to a value larger than 0 for a constant unit cell size. To prescribe a variable unit cell size set to 0 and provide a .sap file (see `sap` parameter).

  Possible values: Non-negative floating point number

- *Parameter name:* `lt_feature`

  Default: Should be provided

  Description: [Standard] Parameter specifying the unit cell feature that will be specified, i.e. *isovalue*, *volumeFraction*, *wallSize* or *poreSize*.

  Possible values: String value

- *Parameter name:* `lt_feature_val`

  Default: Should be provided

  Description: [Standard] Parameter specifying the unit cell feature value. Set to a value larger than 0 for a constant feature value. To prescribe a variable feature value set to 0 and provide a .fap file (see `fap` parameter). If the feature value being provided is an isovalue, it should be specified as a normalized isovalue

  Possible values: Non-negative floating point number

- *Parameter name:* `lt_feature_mode`

  Default: relative

  Description: [Advanced] Parameter specifying the feature size mode, i.e. *absolute* or *relative*. Only active if `lt_feature` is set to *wallSize* or *poreSize*. If set to relative, provide wall and pore sizes relative to a $1 \times 1 \times 1$ unit cell.

  Possible values: String value

A.3 Mesh parameters

- *Parameter name:* `me_mesher`

  Description: [Standard] Parameter specifying the mesh library to use, i.e. *CGAL* or *MMG*.

  Default: Should be provided

  Possible values: String value

- *Parameter name:* `me_side`

  Default: scaffold

  Description: [Advanced] Parameter specifying the side of the scaffold to discretize, i.e. *scaffold* or *void*.

  Possible values: String value

- *Parameter name:* `me_volumeMesh`

  Default: FALSE

  Description: [Advanced] Parameter specifying if the volume should also be discretized. Set to TRUE to discretize the volume, otherwise only the surface mesh will be generated.

  Possible values: Boolean value (TRUE or FALSE)

- *Parameter name:* `me_nThreads`

  Default: 1

  Description: [Standard] Number of threads to use. (Parallel mode is currently only available when `me_mesher` is set to CGAL)

  Possible values: Positive integer value

A.3.1 CGAL specific parameters

- *Parameter name:* `me_facetAngle`

  Default: 30

  Description: [Standard] Surface facet shape, i.e. lower bound for the surface facets angle in degrees.

  Possible values: Positive floating point number

- *Parameter name:* `me_facetSize`

  Default: [The local unit cell size]

  Description: [Standard] Surface facet size, i.e. upper bound for the radii of surface Delaunay balls. Scales with unit cell size.

  Possible values: Positive floating point number

- *Parameter name:* `me_facetDistance`

  Default: Should be provided

  Description: [Standard] Surface approximation error, i.e. upper bound for the distance between the circumcenter of a surface facet and the center of the surface Delaunay ball of this facet. Scales with the unit cell size.

  Possible values: Positive floating point number

- *Parameter name:* `me_cellRadiusEdgeRatio`

  Default: 3.0

  Description: [Standard] Tetrahedron shape quality measure, i.e. upper bound for the ratio between the circumradius of a mesh tetrahedron and its shortest edge. Only active if `me_volumeMesh` is set to TRUE.

  Possible values: Positive floating point number larger than 2

- *Parameter name:* `me_cellSize`

  Default: [The local wall size]

  Description: [Standard] Tetrahedra size, i.e. upper bound on the circumradii of the mesh tetrahedra. Only active if `me_volumeMesh` is set to TRUE. Scales with wall size.

  Possible values: Positive integer number

- *Parameter name:* `me_preserveEdges`

  Default: TRUE

  Description: [Advanced] Set to TRUE to preserve the edges, otherwise set to FALSE.

  Possible values: Boolean value (TRUE or FALSE)

- *Parameter name:* `me_poissonOffset`

  Default: 0.5

  Description: [Advanced] Poisson reconstruction offset. Scales with largest unit cell size in the design. Only active if `me_preserveEdges` is set to TRUE.

  Possible values: Positive floating point number larger or equal than 0.1

A.3.2 Mmg specific parameters

- *Parameter name:* `me_hvol`

  Default: 1.5

Description: [Standard] Maximum volume constraint of the intermediate TetGen mesh. Scales with the the smallest unit cell size in the design.

Possible values: Positive floating point number

- *Parameter name:* `me_hinitial`

  Default: 0.36

  Description: [Standard] Mesh size of the mesh used to compute the level-set. Scales with the smallest feature size.

  Possible values: Positive floating point number

- *Parameter name:* `me_hmin`

  Default: [Inactive]

  Description: [Standard] Minimum edge size. Scales with the smallest feature size in the design.

  Possible values: Non-negative floating point number

- *Parameter name:* `me_hmax`

  Default: [Inactive]

  Description: [Advanced] Maximum edge size. Scales with the largest feature size in the design.

  Possible values: Non-negative floating point number

- *Parameter name:* `me_hausd`

  Default: Should be provided

  Description: [Advanced] Maximal Hausdorff distance for the boundaries approximation. Scales with the mean feature size.

  Possible values: Positive floating point number

- *Parameter name:* `me_hgrad`

  Default: 1.3

  Description: [Advanced] Gradation value. Only active if `me_volumeMesh` is set to TRUE.

  Possible values: Positive floating point number

Index of input parameters

IO files
- fap file, 10
- output location, 10
- sap file, 10
- stl file, 10
- tap file, 10

Lattice
- correction factor, 11
- feature, 11
- feature mode, 12
- feature value, 11
- filter radius, 11
- lattice type, 11
- size, 11

Mesh
- CGAL
  - cell radius edge ratio, 13
  - cell size, 13
  - facet angle, 12
  - facet distance, 13
  - facet size, 13
  - poisson offset, 13
  - preserve edges, 13
- mesher, 12
- MMG
  - hausd, 14
  - hgrad, 14
  - hinitial, 14
  - hmax, 14
  - hmin, 14
  - hvol, 13
- number of threads, 12
- side, 12
- volume mesh, 12