Neper Reference Manual

The documentation for Neper 3.5.1
A software package for polycrystal generation and meshing

15 November 2019

Romain Quey

Table of Contents

Conditions of Use

Copying Conditions

Neper is "free software"; this means that everyone is free to use it and to redistribute it on a free basis. Neper is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of Neper that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of Neper, that you receive source code or else can get it if you want it, that you can change Neper or use pieces of Neper in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of Neper, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for Neper. If Neper is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the license for Neper are found in the General Public License that accompanies the source code (see Appendix E [GNU General Public License], page 95). Further information about this license is available from the GNU Project webpage `http://www.gnu.org/copyleft/gpl-faq.html`.

User Guidelines

If you use Neper for your own work, please, mention it explicitly in your reports (papers, books, talks, ...) and cite one of several of the following papers (depending of your use of Neper):

- *R. Quey, P.R. Dawson, F. Barbe. Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing. Comput. Methods Appl. Mech. Engrg., vol. 200, pp. 1729–1745, 2011* (for Voronoi tessellation, regularization, meshing);
- *R. Quey and L. Renversade, Optimal polyhedral description of 3D polycrystals: Method and application to statistical and synchrotron X-ray diffraction data, Comput. Methods Appl. Mech. Engrg., vol. 330, pp. 308-333, 2018* (for tessellation from experimental properties with option `-morpho`);
- *R. Quey, A. Villani and C. Maurice, Nearly uniform sampling of crystal orientations, J. Appl. Crystallogr., vol. 51, pp. 1162-1173, 2018* (for uniform crystal orientation distribution with option `-ori uniform`).

1 Introduction

1.1 The Neper Project

1.1.1 Description

Neper is a software package for polycrystal generation and meshing. The polycrystals can be 2D or 3D. Neper is built around three modules:

- Module -T generates polycrystals as tessellations. The two main capabilities are: *(i)* the generation of tessellations from cell properties (e.g. a size distribution) and *(ii)* the generation of multiscale tessellations (i.e. including cell subdivisions). These capabilities can also be used together. Tessellations are Laguerre (or Voronoi) tessellations and are therefore composed of convex cells. Finally, the tessellations can be "regularized" by removing their smallest features (edges and faces), which then enables good-quality meshing with module -M. Periodicity conditions can be prescribed. Crystal orientations are provided for the grains. The output is a tessellation file written at a scalar (vectorial) or raster format. Scalar tessellations are intended to be passed to modules -M and -V while raster tessellations can be used by FFT solvers.
- Module -M meshes polycrystals described as tessellation files. Two meshing techniques are available: free (or unstructured) meshing, which generates triangular elements (in 2D) or tetrahedral elements (in 3D) that follow the grain shapes, and mapped meshing, which generates regular, square elements (in 2D) or regular, cubic elements (in 3D) that do not necessarily follow the grain shapes. Free meshing into good-quality elements is ensured by optimized meshing rules, and multimeshing—a complementary use of several meshing algorithms. Remeshing is also available and is similar to meshing except that it takes a mesh as input. Cohesive elements can be inserted at interfaces. The output is a mesh file that can be written under several formats.
- Module -V generates publication-quality images of tessellations and meshes. Colouring and transparency of the different entities can be set up in detail, and mesh nodes can be displaced. Mesh slicing is also available. The output is a PNG image file.

Neper aims to be an easy-to-use, efficient and robust tool. All the input data are prescribed non-interactively, using command lines and / or ASCII files. All treatments can therefore be automated.

1.1.2 Resources and Support

Several, complementary resources describing Neper are available:

- The Neper reference manual, which is this document, describes all Neper's capabilities. Each module is dedicated a specific chapter, which describes the available commands and result files, and provides examples. The manual is available at the PDF and info formats.[1]
- The Neper website, `http://neper.info` gives a general introduction to Neper (including examples with illustrations) and is where Neper can be downloaded from.
- The three Neper reference papers,
  - "*R. Quey, P.R. Dawson and F. Barbe, Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing, Comput. Methods Appl. Mech. Engrg., vol. 200, pp. 1729-1745, 2011*" describes the regularization / meshing methodologies. It can be downloaded from the Neper homepage or directly from this link: `http://neper.info/docs/neper-reference-paper.pdf`;

[1] Provided that the info file is properly installed at your site, it can be accessed by the command: `info neper`.

- "*R. Quey and L. Renversade, Optimal polyhedral description of 3D polycrystals: Method and application to statistical and synchrotron X-ray diffraction data, Comput. Methods Appl. Mech. Engrg., vol. 330, pp. 308-333, 2018*" describes tessellation generation from experimental properties;
- "*R. Quey, A. Villani and C. Maurice, Nearly uniform sampling of crystal orientations, J. Appl. Crystallogr., vol. 51, pp. 1162-1173, 2018.*" describes uniform sampling of crystal orientations.

The best way to report bugs is directly through the GitHub issue tracker, `http://github.com/rquey/neper/issues`. When reporting bugs, please help us helping you by providing a minimal working example. Neper also has two dedicated mailing lists (the GitHub issue tracker should be preferred, especially for bug reports): `neper-announce` (`https://lists.sourceforge.net/lists/listinfo/neper-announce`), the "read-only" list for release announcement and `neper-users` (`https://lists.sourceforge.net/lists/listinfo/neper-users`), the "read-write" list for users. As of October 2019, Neper also has an IRC channel, `#neper`, on the `freenode` network.

1.2 Installing Neper

Neper is written in (mostly ANSI) C and a little C++, and it can run on any Unix-like system. Neper is multithreaded (using openMP) and runs, by default, on all threads (as read in the *`OMP_NUM_THREADS`* environment variable). Neper must be compiled using CMake, as follows:

- Create a `build` directory, for instance as a subdirectory of Neper's `src` directory,
  ```
  $ mkdir build
  ```
- Run CMake from within the `build` directory, pointing to Neper's `src` directory,
  ```
  $ cd build
  $ cmake ..
  ```
- Build Neper,
  ```
  $ make
  ```
  Use option '`-j`' for a multithreaded compilation.
- Install Neper on your system (as root),
  ```
  $ make install
  ```

This procedure uses the default configuration options and should work out-of-the-box if the required dependencies are available in standard system locations. If needed, a finer configuration can be achieved, before building Neper, by running

`$ ccmake ..` (for a command-line tool)

or

`$ cmake-gui ..` (for a graphical tool)

Several of Neper's dependencies can be managed at this stage. Some of them are optional and can be enabled or disabled through variables `HAVE_`*`LIBRARY`*:

- The GNU Scientific Library (GSL, mandatory). It is likely to be available on your system or from your system package manager (binary and development packages); alternatively, the source code version can be downloaded from `http://www.gnu.org/software/gsl`.
- The NLopt library (optional, enabled by default). It is needed by module -T for tessellation generation from cell properties. It is likely to be available on your system or from your system package manager (binary and development packages); alternatively, the source code version can be downloaded from `http://ab-initio.mit.edu/wiki/index.php/NLopt`.
- The OpenMP library (optional, enabled by default). It is likely to be available on your system or from your system package manager.

- The libScotch library (version `6.0.0` or higher, optional, disabled by default). It is needed by module -M for mesh partitioning. The source code can be downloaded from `www.labri.fr/perso/pelegrin/scotch`.
- The pthread library (mandatory if libScotch enabled). It is likely to be available on your system or from your system package manager.

Other dependencies are needed at run time only (they are not linked against Neper's binary):

- The Gmsh program (version `2.4.2` or higher, excluding version `2.5.1`, mandatory for module -M). Both binary and source-code versions can be downloaded from `http://www.geuz.org/gmsh` (compiling from the source code significantly speeds up meshing). Gmsh must be available at the terminal as the command `gmsh`, or the path to its binary must be specified with option `-gmsh` (in module -M).
- The POV-Ray program (mandatory for module -V). It is likely to be available on your system or from your system package manager (binary package); alternatively, a binary, or the source code, can be downloaded from `http://www.povray.org`. POV-Ray must be available at the terminal as the command `povray`, or the path to its binary must be specified with option `-povray` (in module -V).

Neper has a few more dependencies, which are directly included in the source code (see directory `src/contrib`): nanoflann (`https://github.com/jlblancoc/nanoflann`), muparser (`http://beltoforion.de/article.php?a=muparser`) and openGJK (`https://github.com/MattiaMontanari/openGJK`).

Finally, the Neper installation can be tested out by running

```
$ make test
```

or (equivalently)

```
$ ctest
```

1.3 Getting Started

The '`neper`' binary must be run in a terminal, followed by a list of arguments,

```
$ neper list_of_arguments
```

Neper returns messages in the terminal and results in ASCII files.

The list of arguments describes the problem to solve. There are several general-purpose, self-explanatory arguments:

```
$ neper --help
$ neper --version
$ neper --license
```

This section provides information on how to call Neper's modules, properly format option arguments, and set up an initialization file.

1.3.1 Modules

To call a module, run

```
$ neper module_name module_arguments
```

where the module name can be `-T`, `-M` or `-V`, and the module arguments can include both required input data and options. Input data (when not a file name) and options start by '`-`'. Options can be provided in arbitrary order, each of them being followed by a single argument (containing no space). American-english variants of options can be used. String completion is available for all options, so they may be abbreviated as long as the abbreviation is not ambiguous. For instance, in module `-T`, option `-regularization` can be abbreviated to `-reg`. Logical options can be enabled or disabled by providing argument values of '`1`' or '`0`', respectively. Integer or real

argument values can be written as such, or as mathematical or logical expressions (for details, see Section A.1 [Mathematical and Logical Expressions], page 59). For instance, in module -T, option `-rcl 0.5` can also be written as `-rcl 1/2` or `-rcl "cos(pi/3)"`. Module -V shows some exceptions with respect to these rules: the argument cannot be listed in arbitrary order, string completion is not available, and option `-loop` takes several arguments.

1.3.2 Argument Separators

Some options may take several argument values. These values can be combined using *separators* (see Chapter 2 [Tessellation Module (-T)], page 9, Chapter 3 [Meshing Module (-M)], page 27, and Chapter 4 [Visualization Module (-V)], page 41, for details on each option). There are three possible separators:

- The '`,`' separator combines uncorrelated arguments, i.e. arguments of the same type, which can be processed independently from each other. Such arguments can be output file formats, for instance.
- The '`:`' separator is used to combine correlated arguments, i.e. arguments of different types, which cannot be processed independently from each other. Such arguments can be values of a variable in different directions, for instance.
- The '`::`' separator is used in module -T (and a little in module -M) for assigning argument values to the different scales of a multiscale tessellation. It is a "super-separator" that takes precedence over the '`,`' and '`:`' standard separators.

1.3.3 Initialization File

When Neper is run, it starts by reading commands from an initialization file, `$HOME/.neperrc`, if that file exists. Another initialization file can be specified with option `--rcfile`, before calling a module,

```
$ neper --rcfile my_file module_name module_arguments
```

To disable the reading of an initialization file, use option `--rcfile none`.

When a module of Neper is called, Neper looks for the occurrence of '`neper module_name`' in the initialization file and then reads all arguments until the next occurrence of '`neper`' (which should denote the beginning of another module option field) or the end of the file. The arguments may be any legal arguments but are typically limited to frequently-used options. Moreover, any field of comments can be preceded by '`neper comments`'.

Here is an example of initialization file:

```
neper comments ------------------------------------------------------
This is my initialization file.
neper -T -reg 1
neper -M -gmsh my_gmsh_path
neper comments ------------------------------------------------------
```

If the initialization file is not found, or if '`neper module_name`' is not found inside it, Neper will consider only the command line arguments. Also, if an argument is initialized several times (for instance, both in the initialization file and at the command line), only the last specified value is used.

1.4 Reading this Manual

This manual is maintained as a Texinfo manual. Here are the writing conventions used in the document:

- A command that can be typed in a terminal is printed like `this`, or, in the case of a major command, like

  ```
  $ this ;
  ```

- A program (or command) option is printed like `this`;
- The name of a variable is printed like `this`;
- A meta-syntactic variable (i.e. something that stands for another piece of text) is printed like *this*;
- Literal examples are printed like '`this`';
- File names are printed like `this`.

Module arguments are tagged by type and importance level:

- Prerequisites are tagged '[Prerequisite]'—they should be placed in the initialization file;
- Input data are tagged '[Input Data]';
- Standard options are tagged '[Option]';
- Secondary options, which should be used only for fine-tuning and if you really know what you are doing, are tagged '[Secondary option]';
- Post-processing options are tagged '[Post-processing]'.

Some abbreviations are used consistently for options and contribute to Neper's jargon:

`algo`	algorithm
`arch`	architecture
`aspratio`	aspect ratio
`cl`	characteristic length
`col`	colour
`conv`	convergence
`coo`	coordinate
`crysym`	crystal symmetry
`csys`	coordinate system
`dim`	dimension
`dis`	distribution or distortion
`dof`	degree of freedom
`dup`	duplicate(d)
`elset`	element set
`elt`	element
`expr`	expression
`fact`	factor
`faset`	element face set
`geo`	geometry
`id`	identifier
`ini`	initial
`inf`	infinity
`inter`	interpolation
`iter`	iteration
`max`	maximum
`min`	minimum
`morpho`	morphology
`neigh`	neighbour
`nset`	node set
`opti`	optimization
`ori`	orientation
`part`	partition
`poly`	polyhedron
`pov`	POV-Ray file
`qual`	quality

`rad`	radius
`rcl`	relative characteristic length
`res`	resolution
`rmax`	relative maximum
`sing`	singular
`stat`	statistics
`surf`	surface
`tesr`	raster tessellation
`tess`	scalar tessellation
`tmp`	temporary
`trs`	transparency
`val`	value
`var`	variable
`ver`	vertex

2 Tessellation Module (-T)

Module -T generates *tessellations* and *multiscale tessellations* of a bounded *domain* of space, in 2D or 3D. The domain must be convex, but once generated the tessellation can be cut by various shapes, e.g. to include a notch (experimental feature). Periodicity and semi-periodicity conditions can be prescribed. Module -T also enables to *regularize* the tessellations for better-quality meshing. The tessellations are provided in scalar (vectorial) or raster formats. Module -T also generates crystal orientations for the tessellation cells.

Tessellations can be generated from various types of morphological cell properties (option `-morpho`). Several predefined properties are available, such as those obtained by grain growth in metals (which are described by cell size and sphericity distributions). Custom properties can be specified using various metrics, including the size, the sphericity, the centroid or even the actual shape (using a raster tessellation), in terms of either distributions (when applicable) or individual cell values. Global morphological properties, such as a cell aspect ratio or a columnar axis, can also be specified. The generated *tessellations* are Laguerre (or Voronoi) tessellations whose seed attributes are set by optimization to obtain the desired cell properties. Of course, it is also possible to generate standard tessellations (e.g. Poisson-Voronoi or regular tessellations).

Multiscale tessellations are characterized by the subdivision of the cells of a primary tessellation into secondary tessellations (and so on) and are obtained by combining into one, using the '`::`' separator, the option arguments that apply at the successive scales. The same value can be used for defining the tessellations at a given scale, or different values can be loaded using '`msfile(`*`filename`*`)`', where *`filename`* is a *multiscale cell file* (see Section B.3 [Multiscale Cell File], page 78).[1] So, all capabilities available for generating a standard (single-scale) tessellations are available for generating the tessellations at the different scales of a multiscale tessellation. Examples are provided in the following.

The *domain* of space in which the tessellation is created can be of any convex shape. In 3D, cuboidal, cylindrical and spherical shapes (and a few other, exotic shapes) are directly supported while other morphologies can be defined from a set of planes (option `-domain`). Non convex domain shapes can be obtained by cutting the tessellation by different geometrical primitives once generated (option `-tranform cut`, experimental).[2] Periodicity or semi-periodicity conditions can be applied to the tessellation (option `-periodicity`).

Crystal orientations can be randomly distributed (according to a uniform frequency distribution), either in the 3D space or along a specific orientation fibre, or uniformly distributed (option `-ori`). Uniform crystal orientation distributions ensure that all possible crystal orientations are equally represented (no orientation clustering). Crystal orientations can be provided according to different descriptors (option `-oridescriptor`).

Regularization can be applied to the tessellations and consists in removing their small edges and faces (option `-regularization`) which would otherwise be detrimental to good-quality meshing with module -M (see Chapter 3 [Meshing Module (-M)], page 27). It it not available for periodic tessellations yet.

Output files describe the tessellation either at the scalar format `.tess` or at the raster format `.tesr` (see Appendix B [File Formats], page 71). Both are input files of module -M (see Chapter 3 [Meshing Module (-M)], page 27) and module -V (see Chapter 4 [Visualization Module (-V)], page 41). Third-party software file formats are also available.

Here is what a typical run of module -T looks like,

```
$ neper -T -n 10 -id 1 -reg 1
```

[1] As of version 3.5.0, '`msfile(`*`filename`*`)`' should be preferred over '`file(`*`filename`*`)`' to load a multiscale cell file.

[2] Only non-convexities larger than the typical cell size are supported.

```
=======================    N   e   p   e   r    ======================
Info   : A software package for polycrystal generation and meshing.
Info   : Version 3.5.0
Info   : Built with: gsl nlopt muparser
Info   : <http://neper.info> <http://neper.info/community>
Info   : Copyright (C) 2003-2019, and GNU GPL'd, by Romain Quey.
Info   : Loading initialization file '/home/rquey/.neperrc'...
Info   : ---------------------------------------------------------------
Info   : MODULE  -T loaded with arguments:
Info   : [ini file]
Info   : [com line] -n 10 -id 1 -morpho gg -reg 1
Info   : ---------------------------------------------------------------
Info   : Reading input data...
Info   : Creating domain...
Info   : Creating tessellation...
Info   :   - Running tessellation...
Info   :     > Entering optimization...
Info   :     > Setting seeds... 100%
Info   :     > Initial solution: f   =3.623581896
Info   :     > Iteration   2864: fmin=0.077875850 f=0.077875946
Info   :     > Reached 'reps' criterion.
Info   : Regularizing tessellation...
Info   :   - loop 1/2: 100% del=2
Info   : Writing tessellation...
Info   :     [o] Writing file 'n10-id1.tess'...
Info   :     [o] Wrote file 'n10-id1.tess'.
Info   : Elapsed time: 3.333 secs.
=========================================================================
```

2.1 Arguments

2.1.1 Input Data

`-n` *integer or char_string* [Input data]

Specify the number of cells of the tessellation. The argument can be an integer, an expression based on the tessellation cell variables (see Section A.2 [Tessellation Keys], page 61), or '`from_morpho`' to set the value from the morphology (see `-morpho`).
Possible values: `any`. Default value: `none`.

`-id` *integer* [Input data]

Specify the identifier of the tessellation. It defines the seed used by the random number generator to compute the (initial) seed positions.
Possible values: `any`. Default value: `1`.

`-dim` *integer* [Option]

Specify the dimension of the tessellation.
Possible values: `2 or 3`. Default value: `3`.

`-domain` *char_string* [Option]

Specify the domain morphology. In 3D, for a cuboidal shape, provide '`cube(`*size_x*`,`*size_y*`,`*size_z*`)`', for a cylindrical shape, provide '`cylinder(`*height*`,`*diameter*`)`', and for a spherical shape, provide '`sphere(`*diameter*`)`'. In 2D, for a rectangular shape, provide '`square(`*size_x*`,`*size_y*`)`' and for a

circular shape, provide '`circle(diameter)`'. To specify the number of facets, `facet_nb`, of a circle, cylinder or sphere domain, use '`circle(diameter,facet_nb)`', '`cylinder(height,diameter,facet_nb)`' or '`sphere(diameter,facet_nb)`'. For a Rodrigues space fundamental region, provide '`rodrigues(crysym)`', where `crysym` is the crystal symmetry (see Section A.9 [Crystal Symmetries], page 67). For a standard stereographic triangle, provide '`stdtriangle(segment_nb)`', where `segment_nb` is the number of segments on the [011]–[111] line. For an arbitrary convex 3D shape, provide '`planes(file_name)`', where `file_name` is the name of a file containing the total number of planes then, for each plane, the 4 parameters of its equation (d, a, b and c, for an equation of the form $a\,x + b\,y + c\,z = d$). The plane normal, (a, b, c), must be an outgoing vector of the domain. For a tessellation cell, provide '`cell(file_name,cell_id)`', where `file_name` is the name of the tessellation file and `cell_id` is the cell identifier. To transform the domain, append a transformation to the domain name using the '`:`' separator. Available transformations are: '`rotate(axis_x,axis_y,axis_z,angle)`' for a rotation of angle *angle* about axis (*axis_x*, *axis_y*, *axis_z*), '`scale(x_factor,y_factor,z_factor)`' for scaling, '`translate(x_delta,y_delta,z_delta)`' for a translation and '`split(dir)`' for splitting the domain in half along direction '`dir`' ('`x`', '`y`' or '`z`'), which can be used to apply symmetries. For a 2D tessellation, the axis parameters can be omitted in '`rotate`' (default $(0, 0, 1)$), and the z component can be omitted in '`scale`' (default 1) and '`translate`' (default 0). An example is '`sphere(1,100):translate(-0.5,-0.5,-0.5):scale(0.5,1,2)`'.
Possible values: `see above list`. Default value: `cube(1,1,1) in 3D and square(1,1) in 2D`.

`-periodicity` *char_string* [Option]
Specify the periodicity conditions that apply to the domain (and therefore to the tessellation). Provide as argument '`0`' (or '`none`') for no periodicity, '`1`' (or '`all`') for full periodicity, or a list of periodicity directions (among '`x`', '`y`' and '`z`') combined with '`,`' for semi-periodicity.
Possible values: `see above list`. Default value: `0`.

Is it also possible to load a tessellation or a raster tessellation from a file,

`-loadtess` *file_name* [Input data]
Load a tessellation from a file. Provide as argument the file name.
Possible values: `any`. Default value: `none`.

`-loadtesr` *file_name* [Input data]
Load a raster tessellation from a file. Provide as argument the file name. To load only a subregion of a raster tessellation, use the syntax '`file_name:crop(xmin,xmax,ymin,ymax,zmin,zmax)`', where `xmin`, `xmax`, `ymin`, `ymax`, `zmin` and `zmax` are the minimum and maximum positions along `x`, `y` and `z`, respectively. For 2D raster tessellations, the `z` values can be omitted. To scale the number of voxels of a raster tessellation, use the syntax '`file_name:rasterscale(factor)`', where `factor` is the scaling factor, or '`file_name:scale(factor_x,factor_y,factor_z)`', where `factor_x`, `factor_y` and `factor_z` are the scaling factor along x, y and z, respectively. For 2D raster tessellations, the `z` value can be omitted.
Possible values: `any`. Default value: `none`.

Finally, it is possible to load a set of points. These points are used only for statistics, in option `-statpoint`; they are not seed points of the tessellation (see option `-morphooptiini` instead).

`-loadpoint` *file_name* [Input data]
Load points from a file. See Section B.4 [Position File], page 79, for the file format. Provide as argument the file name.
Possible values: `any`. Default value: `none`.

2.1.2 Morphology Options

These options can be used to set the cell morphology. If you want to set seeds attributes instead, use '`-morphooptiini ... -morpho voronoi`'.

`-morpho` *`char_string`* [Option]

Specify morphological properties of the cells. It can be done either by using a special morphology string (as defined below), or by specifying custom cell properties such as sizes, sphericities, centroids, or even exact shapes (using a raster tessellation). Global properties (sometimes refered to as *morphological texture*) can be added. Alternatively, a complete tessellation can also be loaded (as a tessellation file).

(i) The special morphology strings are the following (mutually-exclusive):

• '`voronoi`' for a standard Poisson-Voronoi tessellation;

• '`graingrowth`' or '`gg`' for grain-growth statistical properties, which correspond to a wider grain size distribution and higher grain sphericities than in a Voronoi tessellation (it actually is an alias for '`diameq:lognormal(1,0.35),1-sphericity:lognormal(0.145,0.03)`', see below); the '`graingrowth(`*`mean`*`)`' and '`gg(`*`mean`*`)`' variants can be used to provide an absolute mean grain size, *`mean`* (in which case `-n from_morpho` must be used, see below);

• '`centroidal`' for a centroidal tessellation[3] (it actually is an alias for '`centroid:seed`', see below);

• '`cube(`*`N`*`)`' / '`square(`*`N`*`)`' for a regular tessellation into cubic / square cells, where '*`N`*' is the number of cells along a direction, or '`cube(`*`N1,N2,N3`*`)`' / '`square(`*`N1,N2`*`)`' for a regular tessellation into cubic / square cells, where '*`N1`*', '*`N2`*' and '*`N3`*' are the number of cells along the different directions;

• '`tocta(`*`N`*`)`' for a regular tessellation into truncated octahedra, where '*`N`*' is the number of cells along a direction;

• '`lamellar(w=`*`w`*`,v=`*`v`*`,pos=`*`pos`*`)`' for a lamellar morphology. Argument '`w=`*`w`*' is mandatory, and arguments '`v=`*`v`*' and '`pos=`*`pos`*' are optional. Argument '`w=`*`w`*' enables to specify the absolute lamella width *`w`*. For specifying several widths, combine them with '`:`'. Argument '`v=`*`v`*' enables to specify the lamella plane normals *`v`*. For randomly-distributed normals taken from a uniform distribution, use '`random`' (the default). For a direction of space, use '`(`*`dir_x,dir_y,dir_z`*`)`', where (dir_x, dir_y, dir_z) is the direction. For a direction of the parent crystal, use '`crysdir(`*`crysdir_x,crysdir_y,crysdir_z`*`)`', where ($crysdir\_x, crysdir\_y, crysdir\_z$) is the crystal direction. Argument '`pos=`*`pos`*' enables to specify the position of the lamellaes, *`pos`*. It can be '*`random`*' for a random position (the default) or '*`start`*' for the first lamella starting (full-width) from the start point of the domain (along direction *`dir`*). In the case of a multiscale tessellation, multiscale cell files can be provided as values of '*`w`*', '*`v`*', and '*`pos`*', see Section B.3 [Multiscale Cell File], page 78.

(ii) Custom morphological properties can be defined by providing as argument the cell property and its value, combined with the '`:`' separator. The available properties are the following:

• '`size`' for the size (volume in 3D and area in 2D) and '`diameq`' for the equivalent diameter;

• '`sphericity`' for the sphericity and '`1-sphericity`' for 1 − the sphericity[4] • '`centroid`' for the centroid;

• '`centroidtol`' for the centroid with a tolerance — centroids with a tolerance higher than 1000 times the minimum tolerance are disregarded;

• '`centroidsize`' for combined centroid and size, and '`centroiddiameq`' for combined centroid and equivalent diameter;

---

[3] `centroidal` is not recommended as it does not correspond to a morphological property *per se*; size and/or sphericity properties should be used instead.

[4] The reason behind the '`1-sphericity`' variable is that, for a grain growth microstructure, 1 − the sphericity follows a lognormal distribution; see *R. Quey and L. Renversade, Optimal polyhedral description of 3D polycrystals: Method and application to statistical and synchrotron X-ray diffraction data, Comput. Methods Appl. Mech. Engrg., vol. 330, pp. 308-333, 2018.*

• '`tesr`' for cells of a raster tessellation.

Sizes, diameqs and sphericities can be defined by statistical distributions or on a per-cell basis, while centroids can be defined only on a per-cell basis. The list of available statistical distributions is provided in Section A.1 [Mathematical and Logical Expressions], page 59. If the number of cells is defined (see option `-n`), the size or diameq distribution is scaled to get the specified number of cells; otherwise, i.e. `-n` is set to '`from_morpho`', the number of cells is determined from the size or diameq distribution. An interval of possible values can also be provided using '`interval(`*`min`*`,`*`max`*`)`'. A unique value can be assigned to all cells using '*`value`*', where '*`value`*' is the value. Cell-by-cell values can be provided using '`file(`*`file_name`*`)`', where '*`file_name`*' is the name of the file containing the cell values (see Section B.4 [Position File], page 79, for '`centroid`'; add a tolerance to each position for '`centroidtol`'). For '`centroid`', provide '`seed`' to get a centroidal tessellation. For '`tesr`', '*`file_name`*' is the name of the raster tessellation file. If `-n` is set to '`from_morpho`', the number of cells is set to the number of cells of the raster tessellation.

(iii) The global cell properties are the following (mutually-exclusive):

• '`columnar(`*`dir`*`)`' for a columnar morphology along coordinate axis '*`dir`*', where '*`dir`*' can be '`x`', '`y`' or '`z`';

• '`bamboo(`*`dir`*`)`' for a bamboo morphology along coordinate axis '*`dir`*', where '*`dir`*' can be '`x`', '`y`' or '`z`';

• '`aspratio(`*`r1`*`,`*`r2`*`,`*`r3`*`)`', where '*`r1`*', '*`r1`*' and '*`r3`*' represents relative length along the coordinate axes. For a 2D tessellation, '*`r3`*' can be omitted. When provided, other properties, such as the equivalent diameter or the sphericity, are considered to apply to the cells as if they had no aspect ratio.

(iv) A tessellation can be loaded using '`file(`*`file_name`*`)`', where '*`file_name`*' is the name of the tessellation file.

To specify several properties, combine them with '`,`' (centroids and sizes/equivalent diameters should be seen as one property and specified with `centroidsize`/`centroiddiameq`).

Possible values: `any`. Default value: `voronoi`.

`-morphooptiini` *`char_string`* [Option]

Specify the initial positions and/or weights of the seeds. The general form of the argument is '`coo:`*`coo_char_string`*`,weight:`*`weight_char_string`*', but it is also possible to read the seeds of a tessellation file (provide its name as argument). Different values of '*`coo_char_string`*' and '*`weight_char_string`*' are available, depending on the value of option `-morpho`:

• '*`weight_char_string`*' can be a real value, any mathematical expression based on variables '`radeq`' and '`diameq`' (see Section A.2 [Tessellation Keys], page 61) and their average values, '`avradeq`' and '`avdiameq`', respectively, or '`file(`*`file_name`*`)`' to load values from a file. The default depends on the value of option '`-morpho`': for '`voronoi`', it is '`0`', for a cell-size statistical distribution, it is '`avradeq`', and for cell-based size values (including `-morpho tesr`) , it is '`radeq`'.

• '*`coo_char_string`*' can be '`random`' for random positions, '`packing`' for positions set by (rough) dense sphere packing using the weights as sphere radii, '`centroid`' for cell centroids, '`LLLFP2011`' for Lyckegaard et al.'s method[5], or '`file(`*`file_name`*`)`' to load values from a file (see Section B.4 [Position File], page 79). The default depends on the value of option '`-morpho`': for '`voronoi`', it is '`random`', for a cell-size statistical distribution, it is '`none`', and for cell-based coordinate values (including `-morpho tesr`), it is '`centroid`'.

It is also possible to provide a tessellation file in '`file(`*`file_name`*`)`' (from which seed coordinates or weights are read).

Possible values: `see above list`. Default value: `default`.

[5] A. Lyckegaard, E.M. Lauridsen, W. Ludwig, R.W. Fonda, and H.F. Poulsen. On the Use of Laguerre Tessellations for Representations of 3D Grain Structures. Advanced Engineering Materials, vol. 13, pp. 165–170, 2011.

`-morphooptiobjective` *char_string* [Secondary option]
Specify the objective function. The argument should be of the form '*prop1*:*objective_function1*,*prop2*:*objective_function2*,...', where *propx* are properties as defined in option `-morpho`, and *objective_functionx* are their objective functions. An objective function depends on the property and its definition.
• For properties defined by a statistical distribution (which can be '`size`', '`diameq`' or '`sphericity`'), the available values are '`chi2`' (Chi-square test), '`ks`' (Kolmogorov-Smirnov test), '`kuiper`' (Kuiper's test), '`cmv`' (Cramér-Von Mises test), '`ad`' (Anderson-Darling test), '`FL2`' (L^2-norm on F), '`FL2w`' (weighted L^2-norm on F)[6], '`FL2wu`' (weighted L^2-norm on F corresponding to '`FL2w`' for a unimodal distribution), and the default value is '`FL2w`'.
• For '`centroid`', a Minkowski distance between the seeds and centroids is used, and can be '`L1`', '`L2`' or '`Linf`'.
• For '`tesr`', the definition of the objective function includes several factors. First, preprocessing operations to the raster tessellation can be applied using '`transform(`*operations*`)`', where '*operations*' can include '`scale`' to scale the tessellation to correct for a global cell elongation or '`rasterscale`' to scale the raster itself to correct for a global voxel elongation (which may result from operation '`scale`'); combine with '`,`'. Second, control points can be defined using '`pts(`*args*`)`' where '`args`' can include '`region=`*region*' where '*region*' can be '`surf`' for surface voxels or '`all`' for all voxels, and '`res=`*res*' where '*res*' is the resolution, i.e. the average number of control points along a direction of a grain; combine with '`,`'. Third, the expression of the objective function *per se* can be specified using '`val(`*expr*`)`', where '*expr*' can be '`bounddist`' for minimizing the distance between the raster tessellation and tessellation cell boundaries, or '`intervol`' for maximizing the volume intersection between the raster tessellation and tessellation (both provide similar results). To define the objective function, combine the above factors using '`+`'. The default value is '`pts(region=surf,res=5)+val(bounddist)`'. It is finally possible to define how the different parts of the objective function are combined into the objective function (in the case where several properties are specified), using '`general:`*objective_function*', where '*objective_function*' can be '`L1`', '`L2`' or '`Linf`'; the default is '`L2`'.
Possible values: `any`. Default value: `default`.

`-morphooptidof` *char_string* [Secondary option]
Specify the degrees of freedom. The available values are '`x`', '`y`' and '`z`' for the 3 coordinates, and '`w`' for the weights. Combine with '`,`'.
Possible values: `see above list`. Default value: `x,y,z,w`.

`-morphooptistop` *char_string* [Secondary option]
Specify the stopping criteria of the optimization process. Note that you do not have to define all criteria; in most cases, only one or two are needed. A stopping expression must be of the form '*var*=*val*', where '*var*' is a variable and '*val*' is its value. The available variables are: an absolute or relative error on the value of the objective function evaluated on a number of degrees of freedom basis, '*eps*' or '*reps*' (use '*nlopt_eps*' or '*nlopt_reps*' for the NLopt iteration-based values), an absolute or relative error on the components of the solution vector, '*xeps*' or '*xreps*', a value of the objective function, '*val*', a maximum number of iterations, '*itermax*', a maximum computation time, '*time*', or a maximum number of iteration loops, '*loopmax*' (see option '`-morphooptialgomaxiter`'). Combine them with '`,`'. Optimization stops as soon as one stopping criterion is verified. Optimization can also be stopped anytime using the Ctrl+C command.

[6] *R. Quey and L. Renversade, Optimal polyhedral description of 3D polycrystals: Method and application to statistical and synchrotron X-ray diffraction data, Comput. Methods Appl. Mech. Engrg., vol. 330, pp. 308-333, 2018.*

Possible values: `any`. Default value: `eps=1e-6 (val=1e-4,iter=10000 for -morpho centroidal)`.

`-morphooptialgo` *char_string* [Secondary option]
Specify the optimization algorithm. The available values are '`subplex`' (Subplex), '`praxis`' (Praxis), '`neldermead`' (Nelder-Mead), '`cobyla`' (Cobyla), '`bobyqa`' (Bobyqa) and '`newuoa`' (Newuoa) — only '`subplex`' and '`praxis`' are recommended; it is also possible to provide several algorithms combined with '`,`', in which case the additional algorithms may be used if previous ones fail. In the case of `-morpho centroidal`, another available value is '`lloyd`' (Lloyd's algorithm); to specify the seed displacement factor (from the seed to the centroid), use '`lloyd(`*`factor`*`)`' (the default value is 1.9). Another available value (use only if you know what you are doing) is '`random(`*`seednb`*`,`*`dimnb`*`,`*`min`*`,`*`max`*`,`*`id`*`)`', for which, at each odd iteration, each of '*`seednb`*' seeds sees '*`dimnb`*' of its attributes (among those specified by option `-morphooptidof`) being randomly perturbed, the norm of the total perturbation vector ranging from '*`min`*'–'*`max`*'; '*`id`*' is the identifier of the distribution (similarly to option `-id`); variables can be any mathematical expression based on '*`seednb`*' (the total number of seeds), '*`dim`*' (the tessellation dimension), '*`avdiameq`*' (the average equivalent cell diameter) and '*`inistep`*' (the value of `-morphooptiinistep`); at each next (even) iteration, the attributes of the seeds are reverted to their original values.
Possible values: `any`. Default value: `subplex,praxis (lloyd for -morpho centroidal)`.

`-morphooptialgoneigh` *char_string* [Secondary option]
Specify the neighbour search algorithm. The available values are '`nanoflann`' (nanoflann) and '`qsort`' (direct computation with qsort). '`nanoflann`' is much faster.
Possible values: `any`. Default value: `nanoflann`.

`-morphooptigrid` *char_string* [Secondary option]
Specify the grids used to discretize the distributions. Provide the values using the syntax '*`var`*`:`*`grid`*', where *var* is the variable, among `diameq`, `size` and `sphericity` and *grid* is the grid. The grid must be of the form '`regular(`*`min`*`,`*`max`*`,`*`bin_nb`*`)`', where *`min`* and *`max`* are the minimum and maximum values of the grid interval, respectively, and *`bin_nb`* is the number of bins.
Possible values: `any`. Default value: `diameq:regular(-1,10,1100),size:regular(-1,10,1100),sphericity:regular(-0.1,1.1,1200)`.

`-morphooptismooth` *char_string* [Secondary option]
Specify the standard deviations of the Gaussian distributions which are assigned to each cell data to compute the distributions. Provide the values using the syntax '*`var`*`:`*`val`*', where *var* is the variable, among `diameq`, `size` and `sphericity` and *val* is the value. It is also possible to specify how the convolution functions should be treated: provide '`analytical`' for analytical functions or '`numerical`' for numerical functions (the default, recommended). Combine with '`,`'.
Possible values: `any`. Default value: `diameq:0.05,size:0.05,sphericity:0.005,numerical`.

`-morphooptideltamax` *real* [Secondary option]
Specify the maximal value by which each variable is allowed to change during optimization.
Possible values: `any>=0`. Default value: `HUGE_VAL`.

`-morphooptiinistep` *real* [Secondary option]
Specify the step used to perturb the seed positions and weights when optimization begins. The argument can be a function of *avdiameq*, the average equivalent cell diameter.
Possible values: `any>0`. Default value: `avdiameq/10`.

`-morphooptilogtime` *char_string* [Secondary option]
Log the time taken during the optimization process. The keys are provided in Section A.4 [Tessellation Optimization Keys], page 63. The log file has extension `.logtime`.
Possible values: `any`. Default value: `none`.

`-morphooptilogvar` *char_string* [Secondary option]
Log the variables, i.e. the seed attributes, during the optimization process. The keys are provided in Section A.4 [Tessellation Optimization Keys], page 63. The log file has extension `.logvar`.
Possible values: `any`. Default value: `none`.

`-morphooptilogval` *char_string* [Secondary option]
Log the value of the objective function during the optimization process. The keys are provided in Section A.4 [Tessellation Optimization Keys], page 63. The log file has extension `.logval`.
Possible values: `any`. Default value: `none`.

`-morphooptilogdis` *char_string* [Secondary option]
Log the distributions during the optimization process. The keys are provided in Section A.4 [Tessellation Optimization Keys], page 63. The log files have extension `.logdis*`.
Possible values: `any`. Default value: `none`.

`-morphooptilogtesr` *char_string* [Secondary option]
Log the raster tessellation voxel data during the optimization process. The keys are provided in Section A.4 [Tessellation Optimization Keys], page 63. The log file has extension `.logtesr`.
Possible values: `any`. Default value: `none`.

2.1.3 Crystal Orientation Options

`-ori` *char_string* [Option]
Specify the type of crystal orientation distribution. It can be:
• '`random`' for randomly-distributed orientations in 3D space;
• '`uniform`' for uniformly-distributed orientations in 3D space (the crystal symmetry must be specified using `-oricrysym`);
• '`fibre(`*`dirs_x`*`,`*`dirs_y`*`,`*`dirs_z`*`,`*`dirc_x`*`,`*`dirc_y`*`,`*`dirc_z`*`)`' for randomly-distributed orientations about the fibre defined by crystal direction (*`dirc_x`*, *`dirc_y`*, *`dirc_z`*) parallel to sample direction (*`dirs_x`*, *`dirs_y`*, *`dirs_z`*);
• an ideal orientation, see Section A.8 [Rotations and Orientations], page 67;
• a discrete orientation defined using an orientation descriptor:
'`g(`*`g11`*`,`*`g12`*`,`*`g13`*`,`*`g21`*`,`*`g22`*`,`*`g23`*`,`*`g31`*`,`*`g32`*`,`*`g33`*`,)`' for a rotation matrix, '`e(`*`e1`*`,`*`e2`*`,`*`e3`*`)`' for Euler angles in Bunge convention, '`ek(`*`ek1`*`,`*`ek2`*`,`*`ek3`*`)`' for Euler angles in Kocks convention, '`er(`*`er1`*`,`*`er2`*`,`*`er3`*`)`' for Euler angles in Roe convention, '`rtheta(`*`r1`*`,`*`r2`*`,`*`r3`*`,`*`theta`*`)`' for an axis / angle (in degrees) pair, '`q(`*`q1`*`,`*`q2`*`,`*`q3`*`,`*`q4`*`)`' for a quaternion, '`R(`*`R1`*`,`*`R2`*`,`*`R3`*`)`' for a Rodrigues vector and '`m(`*`h`*`,`*`k`*`,`*`l`*`,`*`u`*`,`*`v`*`,`*`w`*`)`' for Miller indices;
• '`equal`' for orientations equal to the one of the parent cell;
• '`spread(`*`thetam`*`)`' for orientations randomly distributed about the one of the parent cell, according to a 3-variate normal distribution, and so that the average misorientation angle with respect to the average orientation is equal to *`thetam`*;
• '`file(`*`filename`*`):`*`des`*' for reading orientations from file *`filename`* and written using the descriptor *`des`* (see Section A.8 [Rotations and Orientations], page 67); if Euler angles (`e`), the descriptor can be omitted.
Possible values: `see above list`. Default value: `random`.

`-oricrysym` *char_string* [Option]
Specify the crystal symmetry (see Section A.9 [Crystal Symmetries], page 67). This is used by option `-ori uniform` and to reduce the domain of definition of the orientation descriptors.
Possible values: `see above`. Default value: `triclinic`.

`-orioptiini` *char_string* [Secondary option]
Specify the initial crystal orientations. Provide '`random`' for random orientations, or '`file(`*`filename`*`):`*`des`*' for reading orientations from file *`filename`* and written using the descriptor *`des`* (see Section A.8 [Rotations and Orientations], page 67); if Euler angles (`e`), the descriptor can be omitted.
Possible values: `any`. Default value: `random`.

`-orioptifix` *char_string* [Secondary option]
Fix some orientations during optimization. Provide '`none`' for none, or '`file(`*`filename`*`)`', where *`filename`* contains a logical value for each orientation: '`0`' if the orientation is not fixed and '`1`' if the orientation is fixed.
Possible values: `any`. Default value: `none`.

`-orioptistop` *char_string* [Secondary option]
Specify the stopping criterion of the optimization process, under the form of a logical expression based on the following variables: the relative error on the forces at orientations, '*`reps`*', and the iteration number, '*`iter`*'.
Possible values: `any`. Default value: `reps<1e-3||iter>=1e3`.

`-orioptineigh` *char_string* [Secondary option]
Use only the close neighbourhood of orientations to compute their forces (for '`-ori uniform`'). Specify the radius of the neighbourhood, which can be any mathematical expression based on *`dr`* (the average radius of an orientation). *`Nstar`* is the grand number of orientations (i.e., taking crystal symmetry into account).
Possible values: `any`. Default value: `Nstar<10000?pi:20*dr`.

`-orioptilogvar` *char_string* [Secondary option]
Log the variables, i.e. the orientations, during the optimization process. The keys are provided in Section A.5 [Orientation Optimization Keys], page 65. The log file has extension `.logorivar`.
Possible values: `any`. Default value: `none`.

2.1.4 Transformation Options

`-transform` *char_string(...)* [Option]
Apply transformations to a tessellation (if scalar and raster tessellations are written in output, they are transformed independently; i.e., rasterization preceeds transformation).

The following transformations apply to a scalar tessellation:
• '`scale(`*`x_fact`*`,`*`y_fact`*`,`*`z_fact`*`)`' scales a tessellation by *`x_fact`*, *`y_fact`* and *`z_fact`* along directions x, y and z, respectively. For a 2D tessellation, *`z_fact`* can be omitted.
• '`rotate(`*`axis_x`*`,`*`axis_y`*`,`*`axis_z`*`,`*`angle`*`)`' for a rotation about the centre and by an axis/angle pair.
• '`translate(`*`dist_x`*`,`*`dist_y`*`,`*`dist_z`*`)`' for a translation by distances *`dist_x`*, *`dist_y`* and *`dist_z`* along directions x, y and z, respectively.
• '`cut(`*`primitive1`*`,`*`primitive2`*`,...)`' for cutting by a series of geometrical primitives (experimental). The primitives can be:
(i) '`hspace(d,a,b,c)`' for the half-space of equation $a\,x + b\,y + c\,z \geq d$;
(ii) '`sphere(`*`centre_x`*`,`*`centre_y`*`,`*`centre_z`*`,`*`rad`*`)`' for a sphere of centre

(*centre_x*, *centre_y*, *centre_z*) and radius *rad*;
(iii) '`cylinder(basis_x,basis_y,basis_z,axis_x,axis_y,axis_z,rad)`' for a cylinder of basis point (*basis_x*, *basis_y*, *basis_z*), axis (*axis_x*, *axis_y*, *axis_z*) and radius *rad*;
(iv) '`ecylinder(basis_x,basis_y,basis_z,axis_x,axis_y,axis_z,esaxis1_x,esaxis1_y,esaxis1_z,esaxis2_x,esaxis2_y,esaxis2_z,srad1,srad2)`' for an elliptic cylinder of basis point (*basis_x*, *basis_y*, *basis_z*), axis (*axis_x*, *axis_y*, *axis_z*), ellipse section first axis (*esaxis1_x*, *esaxis1_y*, *esaxis1_z*), ellipse section second axis (*esaxis2_x*, *esaxis2_y*, *esaxis2_z*), ellipse section first radius *esrad1* and ellipse section second radius *esrad2*.
Append '`i`' to a primitive name (to get '`cylinderi`', etc.) for the complementary shape.
• '`planecut(d,a,b,c)`' for cutting by the (oriented) plane of equation $a\,x + b\,y + c\,z = d$.
• '`slice(d,a,b,c)`' for slicing a 3D tessellation by the (oriented) plane of equation $a\,x + b\,y + c\,z = d$ (yielding to a 2D tessellation).
• '`mergecell(expr1,expr2,...)`' for merging cells matching expression *`expr1`*, *`expr2`*, etc., where expressions are based on the variables defined in Section A.2 [Tessellation Keys], page 61. Cell merging operates incrementally through expressions.
• '`rmcell(expr1,expr2,...)`' for removing cells matching expression *`expr1`*, *`expr2`*, etc., where expressions are based on the variables defined in Section A.2 [Tessellation Keys], page 61. Cell removing operates incrementally through expressions.
• '`resetcellid`' for resetting cell ids (contiguous numbering from 1).

The following transformations apply to a raster tessellation:
• '`scale(x_fact,y_fact,z_fact)`' scales a tessellation by *`x_fact`*, *`y_fact`* and *`z_fact`* along directions *x*, *y* and *z*, respectively. For a 2D tessellation, *`z_fact`* can be omitted.
• '`rotate(axis_x,axis_y,axis_z,angle)`' for a rotation about the centre and by an axis/angle pair.
• '`translate(dist_x,dist_y,dist_z)`' for a translation by distances *`dist_x`*, *`dist_y`* and *`dist_z`* along directions *x*, *y* and *z*, respectively.
• '`renumber`' renumbers a tessellation to remove cells that are empty or have a zero id.
• '`oriaverage`' sets the cell orientations (field '`**cell/*ori`') as the averages of the cell voxel orientations (field '`**oridata`').
• '`crop:shape(parameters)`' crops the raster tessellation by a shape, where '*`shape(parameters)`*' can be '`cube(xmin,xmax,ymin,ymax,zmin,zmax)`' or '`cylinder(centre_x,centre_y,diameter)`' (the axis is z). The values can be any mathematical expression based on the variables defined in Section A.3 [Raster Tessellation Keys], page 63, (see general).
• '`autocrop`' reduces the raster to its minimal size.
• '`rasterscale(x_fact,y_fact,z_fact)`' scales the number of voxels of the raster by factors *`x_fact`*, *`y_fact`* and *`z_fact`* along directions *x*, *y* and *z*, respectively. For a 2D tessellation, *`z_fact`* can be omitted.
• '`rmsat`' removes the cell "satellites", i.e. parts disconnected from the cell bulk.
• '`grow`' grows the cells to fill the domain.
• '`tessinter(tess_file)`' computes the intersection with tessellation *`tess_file`*.
• '`addbuffer(buffx,buffy,buffz)`' adds a buffer of *`buffx`* void voxels on both sides in the x direction, a buffer of *`buffy`* void voxels on both sides in the y direction and a buffer of *`buffz`* void voxels on both sides in the z direction.
• '`2d`' transforms a 3D tessellation with 1 voxel along z into a 2D tessellation.
Several transformations can be applied successively by combining them with '`,`'.
Possible values: `any`. Default value: `none`.

`-sort` *`char_string`* [Secondary option]
Sort the tessellation cells (typically to facilitate data post-processing). Provide as argument the mathematical expression used for sorting (see Section A.1 [Mathematical and Logical

Expressions], page 59).
Possible values: `any`. Default value: `none`.

2.1.5 Regularization Options

`-regularization` *logical* [Option]
Regularize a tessellation, that is, removes the small edges and, indirectly, the small faces. In return, faces can become non-planar (in 3D). This is controlled by options `-fmax`, `-sel` and `-mloop`. Using regularization enables to get better-quality meshes using module -M (see Chapter 3 [Meshing Module (-M)], page 27).
Possible values: `0 or 1`. Default value: `0`.

`-fmax` *real* [Option]
Specify the maximum allowed face flatness fault (in degrees). The flatness fault is the maximum angle between the normals at two locations on a face.
Possible values: `0 to 180`. Default value: `20`.

`-sel or -rsel` *real* [Secondary option]
Specify the absolute or relative small edge (maximum) length. `rsel` is defined relative to the average cell size (volume in 3D and area in 2D). The default `-rsel 1` leads to a length of 0.25 for a unit volume cell in 3D and 0.125 for a unit area cell in 2D. The value also enables to avoid mesh refinement with the default meshing parameters (see Chapter 3 [Meshing Module (-M)], page 27). It is also possible to specify values on a per-cell basis. The first way is to use the syntax *default_sel*`,`*cell_expr1*`:`*cell_sel1*`,`*cell_expr2*`:`*cell_sel2*`...`, where *default_sel* is the default small edge length, *cell_expri* is an expression defining the set of cells *i* and *cell_seli* is the corresponding small edge length. '*cell_expri*' can be any expression based on variables provided in Section A.2 [Tessellation Keys], page 61. The expressions are processed one after the other. When processing expression *cell_expri*, the matching cells are assigned *cell_seli* as small edge length. Typically, option `-rsel` should be passed the same argument than option `-rcl` of module -M, see Chapter 3 [Meshing Module (-M)], page 27. The second way is to load values from an external file using the syntax '`file(`*file_name*`)`', where *file_name* is the name of the file containing the length values.
Possible values: `any`. Default value: `-rsel 1`.

`-mloop` *integer* [Secondary option]
Specify the maximum number of regularization loops. During each loop, the small edges are considered in turn from the shortest to the largest. Regularization stops when the maximum number of loops is reached or no edges are deleted during a loop.
Possible values: `any`. Default value: `2`.

2.1.6 Output Options

`-o` *file_name* [Option]
Specify the output file name.
Possible values: `any`. Default value: `none`.

`-format` *char_string* [Option]
Specify the format of the output file(s). For scalar tessellations, the available formats are the Neper '`tess`', the Gmsh '`geo`', the Ply '`ply`', the STL '`stl`' (for a separate file for each cell, append '`:bycell)`', the Wavefront '`obj`', the 3dec '`3dec`' and the Surface Evolver '`fe`'. For raster tessellations, the available formats are the Neper '`tesr`', the Orilib map format '`olmap`', and the Kitware '`vtk`'. Orientations for the cells can be obtained using '`ori`' (see also options starting by '`-ori`'). Combine the values with '`,`'.
Possible values: `tess, geo, ply, stl(:bycell), obj, 3dec, fe, tesr, olmap, vtk, ori`.
Default value: `tess`.

`-tesrformat` *char_string* [Option]
Specify the format of the raster output file(s). The available formats are ASCII ('`ascii`'), 8-bit binary / unsigned char-type ('`binary8`'), 16-bit binary / short-type ('`binary16`' and '`binary16_big`') and 32-bit binary / int-type ('`binary32`' and '`binary32_big`'). Formats '`binary16`' and '`binary32`' mean little endianness while formats '`binary16_big`' and '`binary32_big`' mean big endianness.[7]
Possible values: `ascii, binary8, binary16, binary16_big, binary32, binary32_big`. Default value: `binary16 or binary_big (depending on the system)`.

`-tesrsize` *integer* [Option]
Specify the number of points of a raster tessellation along a direction of the domain. In case of a domain of different lengths along the different directions, the argument stands for the geometrical average of the number of points along the different directions, so that the raster points are as cubic as possible. To specify different values along the x, y and z directions, combine the values with '`:`'.
Possible values: `any`. Default value: `20`.

`-oridescriptor` *char_string* [Option]
Specify the orientation descriptor used in the `.tess`, `.tesr` and `.ori` files. It can be Euler angles in Bunge, Kocks or Roe convention (`e`, `ek`, `er`), rotation matrix (`g`), axis / angle of rotation (`rtheta`), angle of rotation (`theta`), Rodrigues vector (`R`) or quaternion (`q`).
Possible values: `above-mentioned values`. Default value: `e`.

`-oriformat` *char_string* [Option]
Specify the format of the `.ori` output file. The available formats are: the Neper-native `plain` (i.e. only the descriptors on successive lines), the Zset/Zébulon `geof` and the FEpX `fepx`.
Possible values: `above-mentioned values`. Default value: `plain`.

2.1.7 Post-Processing Options

The first options apply to the cells and seeds of a tessellation or a raster tessellation, independently of its dimension,

`-statcell` *char_string* [Post-processing]
Provide statistics on the tessellation cells. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 61, for a tessellation and Section A.3 [Raster Tessellation Keys], page 63, for a raster tessellation (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stcell`.

`-statseed` *char_string* [Post-processing]
Provide statistics on the tessellation seeds. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 61, for a tessellation and Section A.3 [Raster Tessellation Keys], page 63, for a raster tessellation (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stseed`.

For a tessellation, it is also possible to get statistics on an entity-basis,

`-statver` *char_string* [Post-processing]
Provide statistics on the tessellation vertices. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 61, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stver`.

[7] Endianness is both written in the tesr file and tested on the system when reading the tesr file, so that the user normally does not have to care about it (even when transferring files across systems).

`-statedge` *char_string* [Post-processing]
Provide statistics on the tessellation edges. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 61, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stedge`.

`-statface` *char_string* [Post-processing]
Provide statistics on the tessellation faces. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 61, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stface`.

`-statpoly` *char_string* [Post-processing]
Provide statistics on the tessellation polyhedra. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 61, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stpoly`.

For a tessellation, it is also possible to get statistics on an voxel-basis, ww

`-statvox` *char_string* [Post-processing]
Provide statistics on the tessellation voxels. Give as argument the keys as described in Section A.3 [Raster Tessellation Keys], page 63, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stpoly`.

Finally, it is possible to get statistics for a particular set of points. The option applies to a tessellation.

`-statpoint` *char_string* [Post-processing]
Provide statistics on points. The points must be loaded with option `-loadpoint`. Give as argument the keys as described in Section A.7 [Point Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stpoint`.

2.1.8 Advanced Options

These advanced options set running conditions for the optimization libraries,

`-morphooptialgomaxiter` *real* [Secondary option]
Specify the maximum number of iterations allowed to the optimization library for running without decreasing the objective function. The expression can be any mathematical expression based on variable '*varnb*' (the total number of optimization variables).
Possible values: `any`. Default value: `max(varnb,1000)`.

2.1.9 Debugging Options

`-checktess` *file_name* [Input data]
Check a tessellation file. Provide as argument the file name. Use this option if the tessellation file fails to load using option `-loadtess` or in other modules.
Possible values: `any`. Default value: `none`.

2.2 Output Files

2.2.1 Tessellation

- Neper tessellation file: `.tess`

  It contains a scalar description of the tessellation. See Appendix B [File Formats], page 71, for the file syntax.
- Neper raster tessellation file: `.tesr`

  It contains a raster description of the tessellation. See Appendix B [File Formats], page 71, for the file syntax.
- Orilib map file: `.olmap`

  It contains a description of a 2D raster tessellation as an orientation map and written under the Orilib format `.olmap`.
- Gmsh geometry file: `.geo`

  It contains a minimal description of the tessellation written under the Gmsh geometry file format `.geo`. This file can be opened with Gmsh for visualization.
- Ply file: `.ply`

  It contains a description of the tessellation written under the standard "Polygon File Format" `.ply`.
- STL file: `.stl`

  It contains a description of the tessellation written under the STL (STereoLithography) format `.stl`. If '`-format stl:bycell`' was used, a separate file is written for each cell, whose name ends in `-`*`id`*`.stl`, where '*id*' is the cell identifier written with leading zeros.
- Wavefront geometry file: `.obj`

  It contains a description of the tessellation written under the Wavefront geometry format `.obj`.
- 3dec file: `.3dec`

  It contains a description of the tessellation written under the 3dec format `.3dec`.
- VTK file: `.vtk`

  It contains a description of the raster tessellation written under the VTK format `.vtk` as supported by Amitex_ffpt. Binary data are always written using BigEndians.
- Orientation file: `.ori`

  It contains crystal orientations for the tessellation cells. The orientations are written on successive lines, using the descriptor specified by option `-oridescriptor` (see also Section A.8 [Rotations and Orientations], page 67) and the writing convention specified by option `-oriformat`.

2.2.2 Statistics

Statistics files are provided for cells, seeds, vertices, edges, faces, polyhedra and points. They are formatted with one entity per line. Each line contains the data specified to the corresponding `-stat` option and described in Section A.2 [Tessellation Keys], page 61, and Section A.3 [Raster Tessellation Keys], page 63, (files `.stcell` and `.stseed` only).

- Tessellation cell statistics file, `.stcell`.
- Tessellation seed statistics file, `.stseed`.
- Tessellation vertex statistics file, `.stver`.
- Tessellation edge statistics file, `.stedge`.
- Tessellation face statistics file, `.stface`.

- Tessellation polyhedron statistics file, `.stpoly`.
- Point statistics file, `.stpoint`.

2.2.3 Tessellation Optimization Log Files

Log files are provided for the time, variables, statistical distributions and objective function value. The files contain the data specified to the corresponding `-morphooptilog` option and described on Section A.4 [Tessellation Optimization Keys], page 63.

- Time file, `.logtime`.
- Variables, i.e. seed attributes, file, `.logvar`.
- Statistical distribution files, `.logdisid`, where *id* is the identifier of the distribution.
- Objective function value file, `.logval`.
- Target raster tessellation file, `-obj.tesr`.

2.2.4 Orientation Optimization Log Files

A log file is provided for the orientation variables. The files contain the data specified to the `-orioptilogvar` option and described on Section A.5 [Orientation Optimization Keys], page 65.

- Variables, i.e. orientations, `.logorivar`.

2.3 Examples

Below are some examples of use of neper -T.

1. Generate a Voronoi tessellation containing 100 cells (with identifier = 1).
   ```
   $ neper -T -n 100 -id 1
   ```
2. Use an elongated domain and generate a Voronoi tessellation containing 100 cells.
   ```
   $ neper -T -n 100 -id 1 -domain "cube(3,1,0.33)"
   ```
3. Generate a Voronoi tessellation containing 100 cells and apply regularization.
   ```
   $ neper -T -n 100 -id 1 -reg 1
   ```
4. Generate a 2D Voronoi tessellation containing 100 cells.
   ```
   $ neper -T -n 100 -id 1 -dim 2
   ```
5. Generate a tessellation containing 100 cells with an x columnar axis.
   ```
   $ neper -T -n 100 -id 1 -morpho "columnar(x)"
   ```
6. Generate a tessellation containing 100 cells with a bamboo structure along x.
   ```
   $ neper -T -n 100 -id 1 -morpho "bamboo(x)"
   ```
7. Generate a tessellation containing 100 cells with experimental grain-growth morphological properties.
   ```
   $ neper -T -n 100 -id 1 -morpho gg
   ```
8. Generate a tessellation containing 100 cells with experimental grain-growth morphological properties and an aspect ratio of 2:1:0.5.
   ```
   $ neper -T -n 100 -id 1 -morpho "gg,aspratio(2,1,0.5)"
   ```
9. Generate a tessellation containing 100 cells with experimental grain-growth morphological properties, and get the equivalent diameters and sphericities of the cells.
   ```
   $ neper -T -n 100 -id 1 -morpho gg -statcell diameq:rel,sphericity
   ```
10. Generate a tessellation of specified absolute grain size distribution (and determine the number of cells accordingly).
    ```
    $ neper -T -n from_morpho -id 1 -morpho
      "diameq:lognormal(0.1,0.03),1-sphericity:lognormal(0.145,0.03)"
    ```

11. Generate a tessellation in a non-convex domain (by cutting the tessellation once generated).

```
$ neper -T -n 100 -id 1 -morpho gg
  -transform "cut(cylinder(1.2,0.5,0.5,0,1,0,0.4))"
```

12. Generate a 2-scale Voronoi tessellation containing 100×10 cells.

```
$ neper -T -n 100::10 -id 1::1
```

13. Generate a 2-scale tessellation containing 10 primary cells with grain-growth morphological properties, each one divided into lamellae of width 0.1.

```
$ neper -T -n 10::from_morpho -id 1::1 -morpho "gg::lamellar(w=0.1)"
```

14. Generate a 2-scale Voronoi tessellation containing 10 primary cells with grain-growth morphological properties, each one divided into lamellae of widths loaded from file `lam_width` and plane normals loaded from file `lam_normal`.

```
$ neper -T -n 10::from_morpho -id 1::1 -morpho "gg::lamellar(w=file(lam_
width),v=file(lam_normal))"
```

```
lam_width:
 1 0.05
 2 0.10
 3 0.05
 4 0.10
 5 0.05
 6 0.10
 7 0.05
 8 0.10
 9 0.05
10 0.10
```

```
lam_normal:
 1 1.000000 0.000000 0.000000
 2 0.000000 1.000000 0.000000
 3 1.000000 0.000000 0.000000
 4 0.000000 1.000000 0.000000
 5 1.000000 0.000000 0.000000
 6 0.000000 1.000000 0.000000
 7 1.000000 0.000000 0.000000
 8 0.000000 1.000000 0.000000
 9 1.000000 0.000000 0.000000
10 0.000000 1.000000 0.000000
```

15. Generate a 2-scale Voronoi tessellation containing 3 primary cells divided into 1, 10 and 100 secondary cells, respectively.

```
$ neper -T -n "3::file(myfile)" -id 1::1
```

```
myfile:
1 1
2 10
3 100
```

16. Generate a 2-scale Voronoi tessellation containing 2×3 cells with specific seed coordinates at both scales (files `coo1` and `coo2`).

```
$ neper -T -n 2::3 -id 1::1 \
  -morphooptiini "coo:file(coo1),weight:0::coo:msfile(coo2),weight:0" \
  -morpho voronoi
```

```
coo1:
0.25 0.50 0.50
```

```
0.75 0.50 0.50
coo2:
1 0.25 0.10 0.50
1 0.25 0.50 0.50
1 0.25 0.90 0.50
2 0.75 0.50 0.10
2 0.75 0.50 0.50
2 0.75 0.50 0.90
```

Note that `coo1` is a simple position file (see Section B.4 [Position File], page 79) while `coo2` is a multiscale cell file (see Section B.3 [Multiscale Cell File], page 78).

17. Generate a Voronoi tessellation containing 100 cells (with identifier = 1) with uniformly distributed crystal orientations and cubic crystal symmetry.

    ```
    $ neper -T -n 100 -oricrysym cubic -ori uniform
    ```

18. Generate 100 uniformly distributed crystal orientations with cubic crystal symmetry (no tessellation).

    ```
    $ neper -T -n 100 -oricrysym cubic -ori uniform -for ori
    ```

3 Meshing Module (-M)

Module -M is the module for meshing scalar and raster tessellations. Two meshing strategies are available. *Free (or unstructured) meshing* creates a conforming mesh into tetrahedral elements (triangular in 2D). *Mapped (or structured) meshing* generates a non-conforming mesh into regular hexahedral elements (quadrangular in 2D). Free meshing is carried out so that the elements have sizes as close as possible to the target value, and show high quality, that is, equilateral shapes. The input file is a tessellation file (`.tess`) or a raster tessellation file (`.tesr`), as provided by module -T. Standard tessellations, multiscale tessellations and periodic (or semi-periodic) tessellations are supported. Free meshing of raster tessellations works for 2D tessellations only. The output mesh can be written in various formats.

The target element size of the mesh can be specified through the element *characteristic length (*`cl`*')*. It stands for the length of a 1D element, the length of the edge of a triangular or quadrangular element (2D) and the length of an edge of a tetrahedral or hexahedral element (3D). For convenience, a *relative characteristic length (*`rcl`*)* is also defined, whose value is relative to the average cell size and provides a medium number of elements. It is also possible to specify `cl` (or `rcl`) values on a per-cell basis, or to specify different values along the three coordinate axes.

For free meshing, mesh quality is ensured to the greatest extent possible using several advanced capabilities,

- Optimized meshing rules. The mesh properties are controlled by size parameters (options `-cl`, `-rcl`, etc.) and a size gradient parameter used for 1D meshing (option `-pl`).
- Multimeshing. Each tessellation face and volume is meshed separately of the others, with several meshing algorithms, until a target mesh quality is reached. This is controlled by options starting by `-meshqual`, and options `-mesh2dalgo` and `-mesh3dalgo`.

Note that, in general, tessellation *regularization* is also necessary to ensure good-quality meshing, see Chapter 2 [Tessellation Module (-T)], page 9.

Remeshing can also be applied to generate a new, good-quality mesh from a mesh containing poor-quality elements. The variables defined on the parent mesh can be transported on the child mesh (options starting by `-transport`).

For mapped meshing, mesh cleaning options enable to remove isolated elements or duplicate nodes, or to duplicate nodes subjected to singularity behaviour (options `-clean`, `-dupnodemerge` and `-singnodedup`).

Mesh partitioning enables to divide the mesh nodes and elements into several sets while minimizing the interfaces between them[1], for parallel finite element calculations. Partitioning can return any number of partitions, or more efficiently, can be carried out according to a given parallel computer architecture (option `-part`).

In the output mesh, the individual entities of the tessellations (the vertices, edges, faces and polyhedra) can be described by element sets (option `-dim`). Node sets of the vertices, edges and faces of the boundary of the tessellation are also provided for prescribing the boundary conditions (option `-nset`). The surface element sets are also provided (option `-faset`). Element sets other than those corresponding to the tessellation cells can be defined (option `-elset`). The mesh order can be 1 or 2 (option `-order`). Statistical data can be obtained on the meshes (options starting by `-stat`).

[1] Each partition being assigned to a processor in the finite element calculation, the minimization of the interfaces between the partitions is done in terms of the number of necessary communications between processors.

Here is what a typical run of module -M looks like,

```
$ neper -M n10-id1.tess

=======================    N   e   p   e   r    ======================
Info   : A software package for polycrystal generation and meshing.
Info   : Version 3.5.0
Info   : Built with: gsl nlopt muparser
Info   : <http://neper.info> <http://neper.info/community>
Info   : Copyright (C) 2003-2019, and GNU GPL'd, by Romain Quey.
Info   : Loading initialization file '/home/rquey/.neperrc'...
Info   : ---------------------------------------------------------------
Info   : MODULE  -M loaded with arguments:
Info   : [ini file] -gmsh /foo/bar/bin/gmsh
Info   : [com line] n10-id1.tess
Info   : ---------------------------------------------------------------
Info   : Reading input data...
Info   :   - Reading arguments...
Info   : Loading input data...
Info   :   - Loading tessellation...
Info   :     [i] Parsing file 'n10-id1.tess'...
Info   :     [i] Parsed file 'n10-id1.tess'.
Info   : Meshing...
Info   :   - Preparing... (cl = 0.2321) 100%
Info   :   - 0D meshing... 100%
Info   :   - 1D meshing... 100%
Info   :   - 2D meshing... 100% (0.72|0.87/83%|11%| 6%)
Info   :     > Checking 2D mesh for pinching out...
Info   :   - 3D meshing... 100% (0.89|0.91/100%| 0%| 0%)
Info   : Searching nsets...
Info   : Writing mesh results...
Info   :   - Preparing mesh...
Info   :   - Mesh properties:
Info   :     > Node number:      287
Info   :     > Elt  number:     1006
Info   :     > Mesh volume:    1.000
Info   :   - Writing mesh...
Info   :     [o] Writing file 'n10-id1.msh'...
Info   :     [o] Wrote file 'n10-id1.msh'.
Info   : Elapsed time: 8.537 secs.
=========================================================================
```

3.1 Arguments

3.1.1 Prerequisites

`-gmsh` *`path_name`* [Prerequisite]
Specify the path of the Gmsh binary (for meshing into triangle and tetrahedral elements).
Possible values: `any`. Default value: `gmsh`.

`-tmp` *`path_name`* [Prerequisite]
Specify the path of the temporary directory (used by Gmsh).
Possible values: `any`. Default value: `"."`.

3.1.2 Input Data

In normal use, the input data is a tessellation file, a raster tessellation file or a mesh file,

`file_name` [Input data]
Specify the name of the input file. It can be a tessellation file (`.tess`), a raster tessellation file (`.tesr`) or a mesh file for remeshing (`.msh`). To load several of them (namely, both a tessellation file and a mesh file for remeshing), combine them with '`,`'. To overwrite the coordinates of the nodes of a mesh, use the syntax '*`file_name`*`:`*`nodecoo_file_name`*', where *`file_name`* is the name of the mesh file and *`nodecoo_file_name`* is the name of the file containing the coordinates of the nodes (see Section B.4 [Position File], page 79). To load only a subregion of a raster tessellation, use the syntax '*`file_name`*`:crop(`*`xmin,xmax,ymin,ymax,zmin,zmax`*`)`', where '*`xmin`*', '*`xmax`*', '*`ymin`*', '*`ymax`*', '*`zmin`*' and '*`zmax`*' are the minimum and maximum positions along x, y and z, respectively. For 2D raster tessellations, the z boundaries can be omitted. To scale the number of points of a raster tessellation, use the syntax '*`file_name`*`:scale(`*`factor`*`)`', where *`factor`* is the scaling factor, or '*`file_name`*`:scale(`*`factor_x,factor_y,factor_z`*`)`', where *`factor_x`*, *`factor_y`* and *`factor_z`* are the scaling factor along x, y and z, respectively. For 2D raster tessellations, the z factor can be omitted.
Possible values: `any`. Default value: `none`.

It is also possible to load a result mesh from a file (use option `-o` not to overwrite the file). If the file contains meshes of dimensions lower than the tessellation's dimension, these meshes are used and only higher-dimension meshes are generated.

`-loadmesh` *`file_name`* [Input data]
Load a mesh from a file (`.msh` format).
Possible values: `any`. Default value: `none`.

Finally, it is possible to load a set of points. These points are used only for statistics, in option `-statpoint`,

`-loadpoint` *`file_name`* [Input data]
Load points from a file. See Section B.4 [Position File], page 79, for the file format. Provide as argument the file name.
Possible values: `any`. Default value: `none`.

3.1.3 Meshing Options

`-elttype` *`char_string`* [Option]
Specify the type of elements, among '`tri`' for triangular elements, '`quad`' for quadrangular elements, '`quad9`' for 9-node quadrangular elements, '`tet`' for tetrahedral elements and '`hex`' for hexahedral elements. (In 2D, '`tet`' and '`hex`' are allowed and interprated as '`tri`' and '`quad`', respectively.)
Possible values: `see above`. Default value: `tet in 3D and tri in 2D`.

`-cl or -rcl` *real* [Option]
Specify the absolute or relative characteristic length of the elements. `rcl` is defined relative to the average cell size. The default, `-rcl 1`, leads to a mesh with about 100 elements per cell in average (64 in 2D). For free meshing, it is also possible to define the characteristic length on a per-cell basis, using any mathematical expression based on the variables defined in Section A.2 [Tessellation Keys], page 61, (or Section A.3 [Raster Tessellation Keys], page 63, for a 2D raster tessellation). A typical use is '`-rcl (body>0)?`*`val1`*`:`*`val2`*' to get interior cells meshed with *`rcl`*`=`*`val1`* and boundary cells meshed with *`rcl`*`=`*`val2`*. The second way is to load values from an external file using the syntax '`file(`*`file_name`*`)`', where *`file_name`* is the name of the file containing the characteristic length values.
Possible values: `any`. Default value: `-rcl 1`.

`-dim` *char_string* [Option]
Specify the meshing dimension. By default, it is equal to the input data dimension (`'inputdim'`). To get meshes of several dimensions in output, provide the values combined with '`,`'. Provide `'all'` for all and `'none'` for none. Note that the meshes of all dimensions are always written into a `.msh` mesh file unless '`:msh`' is appended to the option argument. If a mesh dimension of 3 is required, but the input data is 2D, the 3D mesh is obtained by extrusion of the 2D mesh. With '`-format geof`', use '`1,inputdim`' to get the 1D mesh written as lisets (for visualization).
Possible values: `0 to 3, all, none, inputdim`. Default value: `inputdim`.

`-order` *integer* [Option]
Specify the mesh order. 1 means 2-node linear elements, 3-node triangular elements, 4-node quadrangular elements, 4-node tetrahedral elements and 8-node hexahedral elements. 2 means 3-node linear elements, 6-node triangular elements, 8-node or 9-node quadrangular elements, 10-node tetrahedral elements and 20-node hexahedral elements.
Possible values: `1 or 2`. Default value: `1`.

`-clface or -rclface` *real* [Secondary Option]
Specify the absolute or relative characteristic length of the elements, on a per-face basis. Provide as argument any mathematical expression based on the variables defined in Section A.2 [Tessellation Keys], page 61, or '`default`' (the value inherited from the parent polyhedra). A typical use is '`-rcl (domface==`*`id`*`)?`*`val`*`:default`' to get a finer mesh at domain surface *`id`*. The second way is to load values from an external file using the syntax '`file(`*`file_name`*`)`', where *`file_name`* is the name of the file containing the characteristic length values.
Possible values: `any`. Default value: `none`.

`-cledge or -rcledge` *real* [Secondary Option]
Specify the absolute or relative characteristic length of the elements, on a per-edge basis. Provide as argument any mathematical expression based on the variables defined in Section A.2 [Tessellation Keys], page 61, or '`default`' (the value inherited from the parent faces). A typical use is '`-rcl (domedge==`*`id`*`)?`*`val`*`:default`' to get a finer mesh at domain edge *`id`*. The second way is to load values from an external file using the syntax '`file(`*`file_name`*`)`', where *`file_name`* is the name of the file containing the characteristic length values.
Possible values: `any`. Default value: `none`.

`-clver or -rclver` *real* [Secondary Option]
Specify the absolute or relative characteristic length of the elements, on a per-vertex basis. Provide as argument any mathematical expression based on the variables defined in Section A.2 [Tessellation Keys], page 61, or '`default`' (the value inherited from the parent edges). A typical use is '`-rcl (domver==`*`id`*`)?`*`val`*`:default`' to get a finer mesh at domain vertex *`id`*. The second way is to load values from an external file using the syntax '`file(`*`file_name`*`)`', where *`file_name`* is the name of the file containing the characteristic

length values.
Possible values: `any`. Default value: `none`.

`-pl` *real* [Secondary option]
Specify the progression factor for the element characteristic lengths. This value is the maximum ratio between the lengths of two adjacent 1D elements.
Possible values: `any >= 1`. Default value: `2`.

`-clratio` *char_string* [Secondary option]
Specify the ratios between the *cl*-values along the different coordinate axes. Provide the values combined with '`:`'. For example, '`2:1:1`' leads to elements twice as long in the x direction as in the y and z directions.
Possible values: `none`. Default value: `any`.

`-clmin` *real* [Not recommended option]
Specify the minimum characteristic length of the elements.
Possible values: `any`. Default value: `none`.

The following options define the multimeshing algorithm (for 2D and 3D free meshings). *Multimeshing* consists in using several meshing algorithms concurrently, for each face or polyhedron, until a minimum, target mesh quality is reached. The mesh quality factor, O, accounts for both the element sizes and aspect ratios. It is given by $O = {O_{dis}}^{\alpha} \times {O_{size}}^{1-\alpha}$, where O_{dis} and O_{size} range from 0 (poor quality) to 1 (high quality) and α is a factor equal to 0.8. Therefore, O also ranges from 0 (poor quality) to 1 (high quality). See the Neper reference paper for more information. The minimum quality value can be modified using option `-meshqualmin`. The values of O and O_{dis} can be modified using options `-meshqualexpr` and `-meshqualdisexpr`. The value of the target mesh quality significantly influences meshing speed and output mesh quality. A value of 0 provides the fastest meshing while a value of 1 provides the best-quality meshing. The default value provides an effective balance. The meshing algorithms are taken from the Gmsh[1] and Netgen[2] libraries (options `-mesh2dalgo` and `-mesh3dalgo`).

`-meshqualmin` *real* [Option]
Specify the minimum, target value of mesh quality, O, as defined by option `-meshqualexpr`.
Possible values: `0 to 1`. Default value: `0.9`.

`-meshqualexpr` *char_string* [Option]
Specify the expression of mesh quality, O, as a function of *Odis* and *Osize*.
Possible values: `any`. Default value: `Odis^0.8*Osize^0.2`.

`-meshqualdisexpr` *char_string* [Secondary option]
Specify the expression of the mesh element distortion parameter, *Odis*, as a function of the element distortion parameter *dis* (see the Neper reference paper).
Possible values: `any`. Default value: `dis^(exp((dis^0.1)/(dis^0.1-1)))`.

`-mesh2dalgo` *char_string* [Secondary option]
Specify the 2D meshing algorithms (combine with '`,`'). The available values are `mead` (MeshAdapt), `dela` (Delaunay) and `fron` (Frontal).
Possible values: `mead, dela, fron`. Default value: `mead,dela,fron`.

[1] *Ch. Geuzaine and J.-F. Remacle, Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, International Journal for Numerical Methods in Engineering, 79, 1309–1331, 2009.*

[2] *J. Schoberl, Netgen, an advancing front 2d/3d-mesh generator based on abstract rules. Comput. Visual. Sci., 52, 1–41, 1997.*

`-mesh3dalgo` *char_string* [Secondary option]
Specify the 3D meshing algorithms (combine with '`,`'). Each algorithm has format '*`mesh`*`:`*`opti`*', where *`mesh`* and *`opti`* stand for the meshing and mesh optimization algorithms. The available values of *`mesh`* are currently limited to `netg` (Netgen). The available values of *`opti`* are '`gmsh`' (Gmsh), '`netg`' (Netgen) and '`gmne`' (Gmsh `+` Netgen). Use '`none`' for none. Possible values: `netg:none, netg:gmsh, netg:netg, netg:gmne`. Default value: `netg:gmsh,netg:netg,netg:gmne`.

`-interface` *char_string* [Secondary option]
Specify the type of interface meshing. Provide '`continuous`' for a continuous mesh at interfaces, with shared nodes between neighbour element sets (which are associated to the tessellation cells). Provide '`discontinuous`' for a discontinuous mesh at interfaces, with distinct nodes for the neighbour element sets. Provide '`cohesive`' for cohesive elements at interfaces, joining the neigbour element sets. In the case of a multiscale tessellation, it is possible to provide a different value for each scale using the '`::`' separator (if fewer values than tessellation scales are provided, the last provided value is used for all higher scales). See option `-faset` for the output format.
Possible values: `see above`. Default value: `continuous`.

`-mesh2dpinchfix` *logical* [Secondary option]
Apply 2D-mesh pinches correction after 2D meshing. Disable only if you really know what you are doing.
Possible values: `0 or 1`. Default value: `1`.

3.1.4 Raster Tessellation Meshing Options

Raster tessellation meshing implies interface reconstruction, interface mesh smoothing then remeshing. The following options enable to control interface smoothing.

`-tesrsmooth` *char_string* [Secondary option]
Method for smoothing the interface meshes reconstructed from raster tessellations. Laplacian smoothing ('`laplacian`') is an iterative method that modifies the coordinates of a node using the coordinates of the neighbouring nodes. At iteration i, the position of a node, X_i, is calculated from its previous position, X_{i-1}, and the position of the barycentre of the neighbouring nodes (weighted barycentre, considering the inverse of the distance between the node and the neighbour), X^n_{i-1}, as follows: $X_i = (1 - A)\, X_{i-1} + A\, X^n_{i-1}$. $A \in [0, 1]$ is an adjustable parameter (see option `-tesrsmoothfact`). The number of iterations is set by option `-tesrsmoothitermax`. There is no stop criterion, so `itermax` will always be reached.
Possible values: `laplacian or none`. Default value: `laplacian`.

`-tesrsmoothfact` *real* [Secondary option]
Specify the factor used for the interface mesh smoothing (A in option `-tesrsmooth`).
Possible values: `0 to 1`. Default value: `0.5`.

`-tesrsmoothitermax` *integer* [Secondary option]
Specify the number of iterations used for interface mesh smoothing.
Possible values: `any >= 0`. Default value: `5`.

3.1.5 Mesh Cleaning Options

The following options are specific to mapped meshing of raster tessellations containing voids.

`-clean` *integer* [Secondary option]
Clean the mesh so that it consists of a set of connected elements. Provide as argument the level of cleaning. A value of `1` indicates that two elements should be considered connected to each other if they share at least a vertex. A value of `2` indicates that two elements should

be considered connected to each other if they share at least a face. Using this option, the elements (or bunches of elements) that are not connected to the main skeleton are removed. Possible values: `0 to 2`. Default value: `0`.

`-singnodedup` *logical* [Secondary option]
Duplicate nodes which are the subject of singularity. Such a node belongs to several elements which share only a node or an edge, which provides a singularity behaviour. In Mechanics, it corresponds to imposing a common displacement, while the point can carry no stress. In Thermics, it corresponds to imposing a given temperature at a particular location shared by two bodies, but not enabling heat flux to operate at that location. When this option is enabled, such a node is duplicated, so that each body has its own node. Option `-dupnodemerge` enables to merge back duplicate nodes.
Possible values: `0 or 1`. Default value: `0`.

`-dupnodemerge` *real* [Secondary option]
Merge duplicate nodes. Provide as argument the distance between nodes below which two nodes are merged. Note that Neper does not generate meshes with duplicate nodes, except using options `-singnodedup` or `-interface`.
Possible values: `0`. Default value: `any>0`.

3.1.6 Transformation Options

`-transform` *char_string(...)* [Option]
Apply transformations to a mesh. The following transformations can be applied:
• '`scale(`*x_fact*`,`*y_fact*`,`*z_fact*`)`' scales a mesh by *x_fact*, *y_fact* and *z_fact* along directions *x*, *y* and *z*, respectively. For a 2D mesh, *z_fact* can be omitted.
• '`rotate(`*axis_x*`,`*axis_y*`,`*axis_z*`,`*angle*`)`' for a rotation about the centre and by an axis/angle pair.
• '`translate(`*dist_x*`,`*dist_y*`,`*dist_z*`)`' translates a mesh by distances *dist_x*, *dist_y* and *dist_z* along directions *x*, *y* and *z*, respectively.
• '`smooth(`*fact*`,`*itermax*`,`*type*`)`' smooths the interfaces of a mesh by Laplacian smoothing (use only if you know what you are doing). Laplacian smoothing is an iterative method that modifies the coordinates of a node using the coordinates of the neighbouring nodes. At iteration *i*, the position of a node, X_i, is calculated from its previous position, X_{i-1}, and the position of the barycentre of the neighbouring nodes (weighted barycentre, considering the inverse of the distance between the node and the neighbour), X^n_{i-1}, as follows: $X_i = (1 - fact)\, X_{i-1} + fact\, X^n_{i-1}$; $0 \leq fact \leq 1$. *itermax* iterations are applied. *type* defines the nodes used for smoothing and can be '`all`' for all nodes or '`interior`' for interior nodes.
Possible values: `see above`. Default value: `none`.

3.1.7 Mesh Partitioning Options

Mesh partitioning is achieved using the libScotch library[3]. The principle of mesh partitioning is to create partitions of same size while minimizing the interfaces between them. This affects the same load to all computation units and minimizes communications between them, therefore minimizes the total computation time. There are two available strategies for mesh partitioning. The first one creates partitions and arranges them independently of each other, while the second one consists in optimizing the size and arrangment of the partitions based on a given computer cluster architecture to minimize computation time further. For those clusters that contain processors made of several cores, the communication time between cores of a processor is much lower than the communication time between cores of different processors. To minimize the

[3] *F. Pellegrini, Scotch and libScotch 5.1 User's Guide, INRIA Bordeaux Sud-Ouest, ENSEIRB & LaBRI, UMR CNRS 5800, 2008.*

global communication time, partitions which are processed by cores of the same processor can be grouped together. Partitioning is applied to the higher-dimension mesh. On top of defining the partitions, it renumbers the nodes and elements by increasing partition identifier and writes partitions as element and node sets (`vtk`, `inp` and `geof` formats). This can be managed using option `-part`.

`-part` *integer or char_string* [Option]
Specify the number of partitions or a computer cluster architecture. Using the first option, the number of partition can be any. At the opposite, for a computer cluster architecture, the total number of partitions must be a power of 2. An architecture can be specified in two ways. First, for clusters that contain processors made of several cores, the number of processors and the number of cores per processor can be combined using the '`:`' separator. A ratio of 10 is considered between the computation time between cores located on different processors and the one between cores of the same processor. Second, the name of a file describing the cluster architecture at the Scotch format can be provided.
Possible values: `any`. Default value: `none`.

`-partbalancing` *real* [Secondary option]
Specify the rate of element partition balancing. The partitioning algorithm applies to the nodes; the element partitions are determined afterwards and can be somewhat unbalanced. This option enables to enforce balancing, but getting a full balancing is highly CPU-demanding.
Possible values: `0 to 1`. Default value: `0.5`.

`-partmethod` *char_string* [Secondary option]
Specify the partitioning method. Provide the partitioning expression in Scotch's jargon, or '`none`' for none.
Possible values: `any`. Default value: `see_the_source`.

3.1.8 Field Transport Options

`-transport` *char_string:char_string:file_name,...* [Option]
Transport data from a parent mesh to a child mesh. The parent mesh is the input mesh and the child mesh is the result mesh (created by remeshing or loaded with `-loadmesh`). A transport entry must have format '*entity_type*:*data_type*:*file_name*', where '*entity_type*' must be '`node`' or '`elt`', '*data_type*' is the type of data, under format '`integer`*X*' or '`real`*X*', where *X* is the dimension, and *file_name* is the name of the file containing the parent data. For several data transports, combine the transport entries with '`,`'. Nodal data are transported by interpolation using the shape functions (2D only). Elemental data are transported as specified by `-transporteltmethod`.
Possible values: `any`. Default value: `none`.

`-transportfepx` *char_string:char_string...* [Option]
Transport FEpX data from a parent mesh to a child mesh. The parent mesh is the input mesh and the child mesh is the result mesh (created by remeshing or loaded with `-loadmesh`). A transport entry must have format '*orientation_file*:*tau_file*', where '*orientation_file*' is the orientation file (a list of Euler angles in Kocks convention) and '*tau_file*' is the critical resolved shear strength file (a list of scalars). These elemental data are transported as specified by `-transporteltmethod`. Moreover, the `.kocks` and `.opt` FEpX files are generated from these data and can be used to run a new simulation.
Possible values: `any`. Default value: `none`.

`-transporteltmethod` *char_string:char_string...* [Secondary option]
Specify the method to transport elemental data from the parent mesh to the child mesh. For each element of the child mesh, its centre, c, is considered. Use '`distance`' to choose, for

each element of the child mesh (of centre c), the element of the parent mesh whose centre is the closest to c, or '`location`' to choose the element c belongs to.
Possible values: `see above`. Default value: `distance`.

3.1.9 Output Options

`-o` *`file_name`* [Option]
Specify the output file name.
Possible values: `any`. Default value: `none`.

`-format` *`char_string`* [Option]
Specify the format of the output file(s). Mesh formats are: the Gmsh '`msh`', the VTK '`vtk`', the Abaqus '`inp`', the Zset/Zébulon '`geof`' and the FEpX '`fepx`' (for the FEpX legacy format, provide '`fepx:legacy`'). For '`msh`', append '`:`*`format`*' to specify the format, which can be '`ascii`' or '`binary`' (default '`ascii`'). The tessellation file format '`tess`' is also available. Combine arguments with '`,`'.
Possible values: `see above`. Default value: `msh`.

`-nset` *`char_string`* [Option]
Specify the node sets to provide, among: `faces`, `edges`, `vertices` for all domain faces, edges and vertices, and `facebodies` and `edgebodies` for all face and edge bodies. Provide `all` for all and `none` for none. To get the node sets corresponding to individual entities, provide their labels. For a cuboidal domain, they are `[x-z][0,1]` for the domain faces, `[x-z][0,1][x-z][0,1]` for the edges, and `[x-z][0,1][x-z][0,1][x-z][0,1]` for the vertices. For a cylindrical domain, they are `z[0,1]` for the z faces, and `f[1,2,...]` for the faces on the circular part of the domain. For other domains, they are `f[1,2,...]` for the faces. For cylindrical and other types of domains, the edge and vertex labels are obtained from the face labels as for cuboidal domains. For a 2D mesh (generated from a 2D tessellation), the labels are `[x-y][0,1]` for the edges and `[x-y][0,1][x-y][0,1]` for the vertices. Append '`body`' to a label to get only the body nodes of the set. Combine labels with '`,`'.
Possible values: `see above`. Default value: `faces in 3D and edges in 2D`.

`-faset` *`char_string`* [Option]
Specify the element surface meshes (edge meshes in 2D) to provide. Use '`faces`' for all domain faces. To get the meshes of individual faces, provide their labels (see option `-nset`). For internal mesh faces (edges in 2D) as created by '`-interface discontinuous`', provide '`internal`'.[4] Combine them with '`,`'. Provide `none` for none.
Possible values: `see above`. Default value: `none`.

`-elset` *`char_string`* [Secondary option]
Specify the element sets to provide. The argument can be: `default` for the default element sets (those corresponding to the input tessellation cells), or an expression of the form '*`elset_label`*`:`*`elset_definition`*', where *`elset_label`* is a custom elset label and *`elset_definition`* is an expression defining the elements belonging to the elset, defined from the variables provided in Section A.6 [Mesh Keys], page 66. Combine arguments with '`,`'.
Possible values: `see above`. Default value: `default`.

`-oriformat` *`char_string`* [Option]
Specify the format of the `.per` output file. The available formats are: the Neper-native `plain` and the Zset/Zébulon `geof`.
Possible values: `see above`. Default value: `plain`.

[4] In a `.msh` file, the elements are assigned to fasets using a 4th element tag, and the fasets are defined as physical entities.

3.1.10 Post-Processing Options

The following options provide statistics on the nodes ('`nodes`'), 0D elements ('`elt0d`') and element sets ('`elset0d`'), 1D elements ('`elt1d`') and element sets ('`elset1d`'), 2D elements ('`elt2d`') and element sets ('`elset2d`') and 3D elements ('`elt3d`') and element sets ('`elset3d`'). Also note that the '*`elt`*' and '*`elset`*' labels can be used in place of '`eltnd`' and '`elsetnd`', where *`n`* is the higher mesh dimension. This enables to use the same command whatever the higher mesh dimension is.

`-statnode` *`char_string`* [Post-processing]
Provide statistics on the nodes. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stnode`.

`-statelt0d` *`char_string`* [Post-processing]
Provide statistics on the 0D elements. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stelt0d`.

`-statelt1d` *`char_string`* [Post-processing]
Provide statistics on the 1D elements. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stelt1d`.

`-statelt2d` *`char_string`* [Post-processing]
Provide statistics on the 2D elements. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stelt2d`.

`-statelt3d` *`char_string`* [Post-processing]
Provide statistics on the 3D elements. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stelt3d`.

`-statelset0d` *`char_string`* [Post-processing]
Provide statistics on the 0D element sets. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stelset0d`.

`-statelset1d` *`char_string`* [Post-processing]
Provide statistics on the 1D element sets. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stelset1d`.

`-statelset2d` *`char_string`* [Post-processing]
Provide statistics on the 2D element sets. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stelset2d`.

`-statelset3d` *char_string* [Post-processing]
Provide statistics on the 3D element sets. Provide as argument the keys as described in Section A.6 [Mesh Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stelset3d`.

`-statpoint` *char_string* [Post-processing]
Provide statistics on points. The points must be loaded with option `-loadpoint`. Provide as argument the keys as described in Section A.7 [Point Keys], page 66, (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension `.stpoint`.

3.1.11 Advanced Options

These advanced options set running conditions for the meshing libraries (triangle and tetrahedral meshing),

`-mesh3dclreps` *real* [Secondary option]
Specify the relative tolerance on the average element characteristic length of each polyhedron (compared to `cl`). The value can also be defined on a per-cell basis, using any mathematical expression based on the variables defined in Section A.2 [Tessellation Keys], page 61, or by loading values from an external file, using the syntax '`file(`*file_name*`)`', where *file_name* is the name of the file containing the values. As Neper proceeds iteratively to adjust the average element characteristic length, this is a highly CPU-sensitive capability, and low values will increase computation time significantly.
Possible values: `any`. Default value: `0.02`.

`-mesh2dmaxtime` *real* [Secondary option]
Specify the maximum processing time allowed to the meshing library for meshing a tessellation face (in seconds).
Possible values: `any`. Default value: `1000`.

`-mesh2drmaxtime` *real* [Secondary option]
Specify a factor used to determine the maximum processing time allowed to the meshing library for meshing a tessellation face. This option is similar to `-mesh2dmaxtime`, but the actual maximum time is the product of the maximum processing time of the previous meshings by the value provided in argument.
Possible values: `any`. Default value: `100`.

`-mesh2diter` *integer* [Secondary option]
Specify the maximum number of 2D meshing attempts for a particular face (in case of failure).
Possible values: `any`. Default value: `3`.

`-mesh3dmaxtime` *real* [Secondary option]
Specify the maximum processing time allowed to the meshing library for meshing a tessellation polyhedron (in seconds).
Possible values: `any`. Default value: `1000`.

`-mesh3drmaxtime` *real* [Secondary option]
Specify a factor used to determine the maximum processing time allowed to the meshing library for meshing a tessellation polyhedron. This option is similar to `-mesh3dmaxtime`, but the actual maximum time is the product of the maximum processing time of the previous meshings by the value provided in argument.
Possible values: `any`. Default value: `100`.

`-mesh3diter` *integer* [Secondary option]
Specify the maximum number of 3D meshing attempts for a particular polyhedron (in case of failure).
Possible values: `any`. Default value: `3`.

3.2 Output Files

3.2.1 Mesh

The mesh can be written in the following formats,

- Gmsh format: file `.msh`
- VTK format: file `.vtk`
- Abaqus format: file `.inp`
- Zset/Zébulon format: file `.geof`
- FEpX format: files `.mesh`, `.grain` and `.bcs` (`.parms`, `.mesh`, `.surf`, `.opt`, `.bcs` and `.kocks` in legacy mode)

3.2.2 Periodicity

- File `.per`.

  It contains the mesh periodicity conditions, written following the convention specified by option `-performat`. For format '`plain`', for each slave node, the file provides (one slave node per line): the node id, the master node id and the shift vector of the slave. For format '`geof`', the file provides the displacement relationships between master and slaves nodes.

3.2.3 Interfaces

- File `.intl`. For each 2D interface, the file provide (one 2D interface per line): the labels of the two element sets of the interface.

3.2.4 Statistics

Statistics files are provided for nodes, elements, element sets and points. They are formatted with one entity per line. Each line contains the data specified to the corresponding `-stat` option and described in Section A.6 [Mesh Keys], page 66.

- Node statistics file, `.stnode`.
- (Higher-dimension) element statistics file, `.stelt`.
- (Higher-dimension) element set statistics file, `.stelset`.
- 0D element statistics file, `.stelt0d`.
- 1D element statistics file, `.stelt1d`.
- 2D element statistics file, `.stelt2d`.
- 3D element statistics file, `.stelt3d`.
- 0D element set statistics file, `.stelset0d`.
- 1D element set statistics file, `.stelset1d`.
- 2D element set statistics file, `.stelset2d`.
- 3D element set statistics file, `.stelset3d`.
- Point statistics file, `.stpoint`.

3.3 Examples

Below are some examples of use of neper -M,

1. Mesh tessellation `n100-id1.tess`.
   ```
   $ neper -M n100-id1.tess
   ```
2. Mesh 2D raster tessellation `n100-id1.tesr`.
   ```
   $ neper -M n100-id1.tesr
   ```
3. Mesh tessellation `n100-id1.tess` with a mesh size of rcl = 0.5 and in 2nd-order elements.
   ```
   $ neper -M n100-id1.tess -rcl 0.5 -order 2
   ```
4. Mesh tessellation `n100-id1.tess` with small elements for the interior cells and bigger elements for the boundary cells.
   ```
   $ neper -M n100-id1.tess -rcl "0.2,body==0:0.5"
   ```
5. Remesh mesh `n150_def.msh` (comprising poor-quality elements) into a clean, new mesh. Transport the scalar data of file `n150_def.data` from the deformed mesh to the new mesh.
   ```
   $ neper -M n150.tess,n150_def.msh -transport elt:real1:n150_def.data
     -rcl 0.5 -o n150_new
   ```
6. Mesh tessellation `n100-id1.tess` and divide the mesh into 8 partitions.
   ```
   $ neper -M n100-id1.tess -part 8
   ```
7. Mesh tessellation `n100-id1.tess` into regular hexahedral elements (non-conformal mesh).
   ```
   $ neper -M n100-id1.tess -elt hex
   ```
8. Mesh tessellation `n100-id1.tess` and get, for each element, its radius ratio and its volume.
   ```
   $ neper -M n100-id1.tess -statelt rr,vol
   ```

4 Visualization Module (-V)

Module -V is the Neper visualization module, with which the tessellations and meshes can be printed as publication-quality images. It is also possible to visualize data on them using colours and transparency, or displacements of the nodes and to plot data on slices of the mesh. Points of specific size and colour can also be shown. The output is a PNG image file. The POV-Ray ray-tracing library is used for generating the images.

Contrary to other modules, module -V processes the command arguments one after the other. Typically, using module -V consists in loading a tessellation or a mesh, then data fields to render them. The data can apply to the tessellation entities: polyhedra, faces, edges and vertices, to the mesh entities: 3D, 2D, 1D and 0D elements and nodes, and to points (options starting by `-data`). The entities that are to be visible, for example particular tessellation cells, element sets or elements, can also be specified (options starting by `-show`). The way they are plotted: camera position and angle, projection type, image size, etc., can be set up too (options starting by `-camera` or `-image`). Finally, the coordinate system can be added.

Here is what a typical run of module -V looks like,

```
$ neper -V n10-id1.tess,n10-id1.msh -dataelsetcol id -print img

========================    N   e   p   e   r    ======================
Info   : A software package for polycrystal generation and meshing.
Info   : Version 3.5.0
Info   : Built with: gsl nlopt muparser
Info   : <http://neper.info> <http://neper.info/community>
Info   : Copyright (C) 2003-2019, and GNU GPL'd, by Romain Quey.
Info   : Loading initialization file '/home/rquey/.neperrc'...
Info   : ---------------------------------------------------------------
Info   : MODULE  -V loaded with arguments:
Info   : [ini file]
Info   : [com line] n10-id1.tess,n10-id1.msh -dataelsetcol id -print img
Info   : ---------------------------------------------------------------
Info   : Loading tessellation...
Info   :     [i] Parsing file 'n10-id1.tess'...
Info   :     [i] Parsed file 'n10-id1.tess'.
Info   : Loading mesh...
Info   :     [i] Parsing file 'n10-id1.msh'...
Info   :     [i] Parsed file 'n10-id1.msh'.
Info   : Reconstructing mesh...
Info   : Reading data (elset3d, col)...
Info   : Printing image...
Info   :     [o] Writing file 'img.pov'...
Info   :   - Printing mesh...
Info   :     > Reducing data...
Info   :       . Number of 3D elt faces reduced by 90% (to 410).
Info   :       . Number of 3D elt edges reduced by 50% (to 615).
Info   :       . Number of 0D elts    reduced by 100% (to 0).
Info   :     [o] Wrote file 'img.pov'.
Info   :   - Generating png file (1080x1080 pixels)...
Info   :     [o] Writing file 'img.png'...
Info   :     [o] Wrote file 'img.png'.
Info   : Printing scale...
```

```
Info   : Elapsed time: 1.620 secs.
================================================================================
```

4.1 Arguments

4.1.1 Prerequisites

`-povray` *path_name* [Prerequisite]
Specify the path of the POV-Ray binary (for generating PNG images).
Possible values: `any`. Default value: `povray`.

4.1.2 Input Data

file_name [Input data]
Specify the name of the input file. It can be a tessellation file (`.tess`), a raster tessellation file (`.tesr`), a mesh file (`.msh`) or a point file (see Section B.4 [Position File], page 79). To load several of them, combine them with '`,`'.
Possible values: `any`. Default value: `none`.

4.1.3 Tessellation Data Loading and Rendering

The following options enable to define the properties (colour and size) of the tessellation cells or entities (polyhedra, faces, edges and vertices). This can be done either directly, by specifying the property values (e.g. the RGB channel values for colour) or indirectly, e.g. using scalar values that are converted in colour using a given *colour scheme*. In this case, a scale image is generated in addition to the tessellation image. The scale properties can be set up (minimum, maximum and tick values).

The following options apply to the cells of a tessellation or a raster tessellation, independently of its dimension,

`-datacellcol` *char_string* [Option]
Set the colours of the tessellation cells. The argument can be one of the following: *(i)* '`id`' for colouring based on the identifier, using a colour palette (see Section A.10 [Colours], page 68), *(ii)* '`ori`' for colouring based on the crystal orientations, *(iii)* the name of a colour that will be used for all cells (see Section A.10 [Colours], page 68), *(iv)* the name of a file containing a list of colours (provided as RGB channel values), or *(v)* a string indicating how the colours can be obtained. The string has the format '*var*`:`*file_name*', where *var* can be '`id`' for identifiers, '`ori`' for crystal orientations or '`scal`' for scalar values, and '*file_name*' is the name of the file containing the data. The colour schemes used to derive the colours from the data can be specified with option `-datacellcolscheme`.
Possible values: `any`. Default value: `white`.

`-datacellcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the tessellation cells loaded with option `-datacellcol`. The type of colour scheme depends on the type of data.
• For crystal orientations (cubic symmetry is assumed), the colour scheme can be: '`R`' for Rodrigues vector colouring (default is inside the fundamental region, can be bounded using '`R(`*max*`)`', where *max* is the maximum extent), '`r`' for rotation axis colouring, '`theta`' for rotation angle colouring (can be bounded using '`theta(`*max*`)`', where *max* is the maximum angle (in radian)) and '`rtheta`' for rotation axis / angle colouring (can be bounded using '`rtheta(`*max*`)`', where *max* is the maximum angle (in radian)).
• For scalars, the colour scheme can be any list of colours.
Possible values: `see above list`. Default value: `"R" for crystal orientations and "blue,cyan,yellow,red" for scalars`.

`-datacelltrs` *`real`* [Option]
Set the transparency of the tessellation cells. Provide as argument a value that applies to all cells or '`file(`*`file_name`*`)`' to load values from a file.
Possible values: `0 to 1`. Default value: `0`.

`-datacellscale` *`char_string`* [Option]
Set the scale relative to the '`-datacellcol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datacellscaletitle` *`char_string`* [Option]
Set the title of the scale relative to the '`-datacellcol scal`' data.
Possible values: `any`. Default value: `none`.

For tessellations, it is also possible to set data on a per-entity basis,

`-datapolycol` *`char_string`* [Option]
Set the colours of the tessellation polyhedra. The argument can be one of the following: *(i)* '`id`' for colouring based on the identifier, using a colour palette (see Section A.10 [Colours], page 68), *(ii)* '`scaleid(`*`scale`*`)`' for colouring based on the identifier of the *scale* tessellation the polyhedron belongs to, using a colour palette (see Section A.10 [Colours], page 68), *(iii)* the name of a colour that will be used for all polyhedra (see Section A.10 [Colours], page 68), *(iv)* the name of a file containing a list of colours (provided as RGB channel values), or *(v)* a string indicating how the colours can be obtained. The string has the format '*`var`*`:`*`file_name`*', where *`var`* can be '`ori`' for crystal orientations or '`scal`' for scalar values, and '*`file_name`*' is the name of the file containing the data. The colour schemes used to derive the colours from the data can be specified with option `-datapolycolscheme`.
Possible values: `any`. Default value: `white`.

`-datapolycolscheme` *`char_string`* [Option]
Set the colour scheme used to get colours from the data of the tessellation faces loaded with option `-datafacecol`. See option `-datacellcolscheme` for the argument format.
Possible values: `see option -datacellcolscheme`. Default value: `see option -datacellcolscheme`.

`-datapolytrs` *`real`* [Option]
Set the transparency of the tessellation polyhedra. Provide as argument a value that applies to all polyhedra or '`file(`*`file_name`*`)`' to load values from a file.
Possible values: `0 to 1`. Default value: `0`.

`-datapolyscale` *`char_string`* [Option]
Set the scale relative to the '`-datapolycol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datapolyscaletitle` *`char_string`* [Option]
Set the title of the scale relative to the '`-datapolycol scal`' data.
Possible values: `any`. Default value: `none`.

`-datafacecol` *`char_string`* [Option]
Set the colours of the tessellation faces. See option `-datapolycol` for the argument format.
Possible values: `any`. Default value: `white`.

`-datapolycolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the tessellation faces loaded with option `-datafacecol`. See option `-datacellcolscheme` for the argument format.
Possible values: `see option -datacellcolscheme`. Default value: `see option -datacellcolscheme`.

`-datafacecolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the tessellation faces loaded with option `-datafacecol`. See option `-datacellcolscheme` for the argument format.
Possible values: `see option -datacellcolscheme`. Default value: `see option -datacellcolscheme`.

`-datafacetrs` *real* [Option]
Set the transparency of the tessellation faces. Provide as argument a value that applies to all faces or '`file(`*file_name*`)`' to load values from a file.
Possible values: `0 to 1`. Default value: `0`.

`-datafacescale` *char_string* [Option]
Set the scale relative to the '`-datafacecol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datafacescaletitle` *char_string* [Option]
Set the title of the scale relative to the '`-datafacecol scal`' data.
Possible values: `any`. Default value: `none`.

`-dataedgerad` *char_string* [Option]
Set the radii of the tessellation edges. The argument can be one of the following: a real value that will be used for all entities or the name of a file containing a list of radii.
Possible values: `any`. Default value: `tessellation dependent`.

`-dataedgecol` *char_string* [Option]
Set the colours of the tessellation edges. See option `-datapolycol` for the argument format.
Possible values: `any`. Default value: `black`.

`-dataedgecolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the tessellation edges loaded with option `-dataedgecol`. See option `-datacellcolscheme` for the argument format.
Possible values: `see option -datacellcolscheme`. Default value: `see option -datacellcolscheme`.

`-dataedgetrs` *real* [Option]
Set the transparency of the tessellation edges. Provide as argument a value that applies to all edges or '`file(`*file_name*`)`' to load values from a file.
Possible values: `0 to 1`. Default value: `0`.

`-dataedgescale` *char_string* [Option]
Set the scale relative to the '`-dataedgecol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataedgescaletitle char_string` [Option]
Set the title of the scale relative to the '`-dataedgecol scal`' data.
Possible values: `any`. Default value: `none`.

`-dataverrad char_string` [Option]
Set the radii of the tessellation vertices. See option `-dataedgerad` for the argument format.
Possible values: `any`. Default value: `tessellation dependent`.

`-datavercol char_string` [Option]
Set the colours of the tessellation vertices. See option `-datapolycol` for the argument format.
Possible values: `any`. Default value: `black`.

`-datavercolscheme char_string` [Option]
Set the colour scheme used to get colours from the data of the tessellation vertices loaded with option `-datavercol`. See option `-datacellcolscheme` for the argument format.
Possible values: `see option -datacellcolscheme`. Default value: `see option -datacellcolscheme`.

`-datavertrs real` [Option]
Set the transparency of the tessellation vertices. Provide as argument a value that applies to all vertices or '`file(file_name)`' to load values from a file.
Possible values: `0 to 1`. Default value: `0`.

`-dataverscale char_string` [Option]
Set the scale relative to the '`-datavercol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataverscaletitle char_string` [Option]
Set the title of the scale relative to the '`-datavercol scal`' data.
Possible values: `any`. Default value: `none`.

`-dataseedrad char_string` [Option]
Set the radii of the tessellation seeds. See option `-dataedgerad` for the argument format.
Possible values: `any`. Default value: `tessellation dependent`.

`-dataseedcol char_string` [Option]
Set the colours of the tessellation seeds. See option `-datapolycol` for the argument format.
Possible values: `any`. Default value: `grey`.

`-dataseedcolscheme char_string` [Option]
Set the colour scheme used to get colours from the data of the tessellation seeds loaded with option `-dataseedcol`. See option `-datacellcolscheme` for the argument format.
Possible values: `see option -datacellcolscheme`. Default value: `see option -datacellcolscheme`.

`-dataseedscale char_string` [Option]
Set the scale relative to the '`-dataseedcol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataseedscaletitle` *char_string* [Option]
Set the title of the scale relative to the '`-dataseedcol scal`' data.
Possible values: `any`. Default value: `none`.

Below are options specific to raster tessellations, For a raster tessellation, it is also possible to set data on a per-voxel basis,

`-datavoxcol` *char_string* [Option]
Set the colours of the voxels. The argument can be one of the following: *(i)* '`id`' for colouring based on the identifier, using a colour palette (see Section A.10 [Colours], page 68), *(ii)* '`ori`' for colouring based on the crystal orientations, *(iii)* '`disori`' for colouring based on the disorientations with respect to the corresponding mean cell orientation, *(iv)* the name of a colour that will be used for all cells (see Section A.10 [Colours], page 68), *(v)* the name of a file containing a list of colours (provided as RGB channel values), or *(vi)* a string indicating how the colours can be obtained. The string has the format '*var*`:`*file_name*', where *var* can be '`id`' for identifiers, '`ori`' for crystal orientations or '`scal`' for scalar values, and '*file_name*' is the name of the file containing the data. The colour schemes used to derive the colours from the data can be specified with option `-datavoxcolscheme`.
Possible values: `any`. Default value: `white`.

`-datavoxcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the voxels loaded with option `-datavoxcol`. See option `-datacellcolscheme` for the argument format.
Possible values: `see option -datacellcolscheme`. Default value: `see option -datacellcolscheme`.

`-datavoxscale` *char_string* [Option]
Set the scale relative to the '`-datavoxcol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datavoxscaletitle` *char_string* [Option]
Set the title of the scale relative to the '`-datavoxcol scal`' data.
Possible values: `any`. Default value: `none`.

`-datavoxedgerad` *real* [Option]
Set the radius of the edges of the raster points.
Possible values: `any`. Default value: `proportional to the raster point size`.

`-datavoxedgecol` *char_string* [Option]
Set the colour of the edges of the raster points. Provide as argument the name of a colour that will be used for all points (see Section A.10 [Colours], page 68).
Possible values: `any`. Default value: `black`.

4.1.4 Mesh Data Loading and Rendering

The following options enable to define the properties (colour, size, etc.) of the mesh entities (3D, 2D, 1D and 0D elements, and nodes). This can be done either directly, by specifying the property values (e.g. the RGB channel values for colour) or indirectly, e.g. using scalar values that are rendered in colour using a given *colour scheme*. In this case, a scale image is generated in addition to the mesh image. The scale properties can be set up (start and end values, tick values).

The options are listed below for 3D elements ('`elt3d`') and element sets ('`elset3d`'), 2D elements ('`elt2d`') and element sets ('`elset2d`'), 1D elements ('`elt1d`') and element sets ('`elset1d`'), 0D elements ('`elt0d`') and element sets ('`elset0d`') and nodes ('`nodes`'). Also note that the '*`elt`*' and '*`elset`*' labels can be used in place of '`eltnd`' and '`elsetnd`', where *`n`* is the highest mesh dimension. This enables to use the same command whatever the highest mesh dimension is.

The following options enable to load data relative to the 3D mesh elements. Note that the options can be applied to element sets by changing '`elt`' to '`elset`'.

`-dataelt3dcol` *`char_string`* [Option]
Set the colours of the 3D elements. The argument can be one of the following: *(i)* '`id`' for the default colour palette (see Section A.10 [Colours], page 68), *(ii)* the name of a colour that will be used for all elements (see Section A.10 [Colours], page 68), *(iii)* the name of a file containing a list of colours (provided as RGB channel values), *(iv)* a string indicating how the colours can be obtained, or *(v)* '`from_nodes`' to derive the colours of the elements from the colours of the nodes (the node colours must be loaded using option `-datanodecol`). In case *(iv)*, the string has the format '*`var:file_name`*', where *`var`* can be '`ori`' for crystal orientations or '`scal`' for scalar values, and '*`file_name`*' is the name of the file containing the data. The colour schemes used to derive the colours from the data can be specified with option `-dataelt3dcolscheme`.
Possible values: `any`. Default value: `white`.

`-dataelt3dcolscheme` *`char_string`* [Option]
Set the colour scheme used to get colours from the data of the 3D elements loaded with option `-dataelt3dcol`. The type of colour scheme depends on the type of data. For crystal orientations, the colour scheme can be: '`R`' for Rodrigues vector colouring; for scalar data, the colour scheme can be any list of colours.
Possible values: `"R" for crystal orientations and any list of colours for scalars`. Default value: `"R" for crystal orientations and "blue,cyan,yellow,red" for scalars`.

`-dataelt3dscale` *`char_string`* [Option]
Set the scale relative to the '`-dataelt3dcol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataelt3dscaletitle` *`char_string`* [Option]
Set the title of the scale relative to the '`-dataelt3dcol scal`' data.
Possible values: `any`. Default value: `none`.

`-dataelt3dedgerad` *`real`* [Option]
Set the radius of the edges of the 3D elements.
Possible values: `any`. Default value: `mesh dependent`.

`-dataelt3dedgecol` *`char_string`* [Option]
Set the colour of the edges of the 3D elements. Provide as argument the name of a colour that will be used for all elements (see Section A.10 [Colours], page 68).
Possible values: `any`. Default value: `black`.

The following options enable to load data relative to the 2D elements. Note that the options can be applied to element sets by changing '`elt`' to '`elset`'.

`-dataelt2dcol` *char_string* [Option]
Set the colours of the 2D elements. See option `-dataelt3dcol` for the argument format.
Possible values: `any`. Default value: `white`.

`-dataelt2dcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the 2D elements loaded with option `-dataelt2dcol`. See option `-dataelt3dcolscheme` for the argument format.
Possible values: `see option -dataelt3dcolscheme`. Default value: `see option -dataelt3dcolscheme`.

`-dataelt2dscale` *char_string* [Option]
Set the scale relative to the '`-dataelt2dcol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataelt2dscaletitle` *char_string* [Option]
Set the title of the scale relative to the '`-dataelt2dcol scal`' data.
Possible values: `any`. Default value: `none`.

`-dataelt2dedgerad` *real* [Option]
Set the radius of the edges of the 2D elements.
Possible values: `any`. Default value: `mesh dependent`.

`-dataelt2dedgecol` *char_string* [Option]
Set the colours of the edges of the 3D elements. See option `-dataelt3dedgecol` for the argument format.
Possible values: `any`. Default value: `black`.

The following options enable to load data relative to the 1D elements. Note that the options can be applied to element sets by changing '`elt`' to '`elset`'.

`-dataelt1dcol` *char_string* [Option]
Set the colours of the 1D elements. See option `-dataelt3dcol` for the argument format.
Possible values: `any`. Default value: `black`.

`-dataelt1dcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the 1D elements loaded with option `-dataelt1dcol`. See option `-dataelt3dcolscheme` for the argument format.
Possible values: `see option -dataelt3dcolscheme`. Default value: `see option -dataelt3dcolscheme`.

`-dataelt1dscale` *char_string* [Option]
Set the scale relative to the '`-dataelt1dcol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataelt1dscaletitle` *char_string* [Option]
Set the title of the scale relative to the '`-dataelt1dcol scal`' data.
Possible values: `any`. Default value: `none`.

`-dataelt1drad` *char_string* [Option]
Set the radius of the 1D elements.
Possible values: `any`. Default value: `mesh dependent`.

The following options enable to load data relative to the 0D mesh elements. Note that the options can be applied to element sets by changing '`elt`' to '`elset`'.

`-dataelt0dcol` *char_string* [Option]
Set the colours of the 0D elements. See option `-dataelt3dcol` for the argument format.
Possible values: `any`. Default value: `black`.

`-dataelt0dcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the 0D elements loaded with option `-dataelt0dcol`. See option `-dataelt3dcolscheme` for the argument format.
Possible values: `see option -dataelt3dcolscheme`. Default value: `see option -dataelt3dcolscheme`.

`-dataelt0dscale` *char_string* [Option]
Set the scale relative to the '`-dataelt0dcol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataelt0dscaletitle` *char_string* [Option]
Set the title of the scale relative to the '`-dataelt0dcol scal`' data.
Possible values: `any`. Default value: `none`.

`-dataelt0drad` *char_string* [Option]
Set the radius of the 0D elements.
Possible values: `any`. Default value: `mesh dependent`.

The following options enable to load data relative to the nodes.

`-datanodecoo` *char_string* [Option]
Set the coordinates of the nodes. The argument can be the name of a file containing a list of coordinates, or a string indicating how the coordinates can be obtained. The string has the format '*var*`:file_name`', where *var* can be '`disp`' for displacements, and `file_name` is the name of the file containing the data.
Possible values: `any`. Default value: `none`.

`-datanodecoofact` *real* [Option]
Set the value of the scaling factor to apply to the displacements of the nodes.
Possible values: `any`. Default value: `1`.

`-datanoderad` *file_name* [Option]
Set the radius of the nodes.
Possible values: `any`. Default value: `mesh dependent`.

`-datanodecol` *file_name* [Option]
Set the colours of the nodes. See option `-dataelt3dcol` for the argument format.
Possible values: `any`. Default value: `black`.

`-datanodecolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the nodes loaded with option `-datanodecol`. See option `-dataelt3dcolscheme` for the argument format.
Possible values: `see option -dataelt3dcolscheme`. Default value: `see option -dataelt3dcolscheme`.

`-datanodescale` *char_string* [Option]
Set the scale relative to the '`-datanodecol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datanodescaletitle` *char_string* [Option]
Set the title of the scale relative to the '`-datanodecol scal`' data.
Possible values: `any`. Default value: `none`.

4.1.5 Point Data Loading and Rendering

The following options enable to define the properties (colour, shape, size, etc.) of points loaded as input. This can be done either directly, by specifying the property values (e.g. the RGB channel values for colour) or indirectly, e.g. using scalar values that are rendered in colour using a given *colour scheme*. In this case, a scale image is generated in addition to the image. The scale properties can be set up (start and end values, tick values).

`-datapointcoo` *char_string* [Option]
Set the coordinates of the points. The argument can be the name of a file containing a list of coordinates, or a string indicating how the coordinates can be obtained. The string has the format '*var*`:file_name`', where *var* can be '`disp`' for displacements, and `file_name` is the name of the file containing the data.
Possible values: `any`. Default value: `none`.

`-datapointcoofact` *real* [Option]
Set the value of the scaling factor to apply to the displacements of the points.
Possible values: `any`. Default value: `1`.

`-datapointrad` *char_string* [Option]
Set the radius (and shape) of the points. The argument can be a value that applies to all points, a file containing a list of radii, or a string indicating how the radii can be obtained. The string has the format '*var*`:file_name`', where *var* stands for the morphology of the points and `file_name` is the name of the file containing the morphology parameters. For cube shape, *var* must be '`cube`' and the file must contain, for each point, the radius (half of the edge length) and then the coordinates of the three axes (which also is the rotation matrix that brings the reference axes into coincidence with the cube axes). For cylinder shape, *var* must be '`cyl`' and the file must contain, for each point, the radius, the length, and then the coordinates of the axis. For arrow shape, *var* must be '`arr`' and the file must contain, for each point, the radius, the length, and then the coordinates of the axis. For disc (2D) shape, *var* must be '`disc`' and the file must contain, for each point, the radius, then the coordinates of the axis. For ellipsoidal shape, *var* must be '`ell`' and the file must contain, for each point, the three radii then the coordinates of the three axes. For torus shape, *var* must be '`tor`' and the file must contain, for each point, the major radius (centre to centre line), the minor radius, and then the coordinates of the normal axis. The last capability is very specific: if the points are plotted in Rodrigues space, appending '`:rodrigues`' to the option argument enables to account for space distortion.
Possible values: `any`. Default value: `point set dependent`.

`-datapointcol` *char_string* [Option]
Set the colours of the points. The argument can be one of the following: *(i)* '`id`' for the default colour palette (see Section A.10 [Colours], page 68), *(ii)* the name of a colour that will be used for all points (see Section A.10 [Colours], page 68), *(iii)* the name of a file containing a list of colours (provided as RGB channel values) or *(iv)* a string indicating how the colours can be obtained,. In case *(iv)*, the string has the format '*`var:file_name`*', where *`var`* can be '`id`' for identifiers, '`ori`' for crystal orientations or '`scal`' for scalar values, and '*`file_name`*' is the name of the file containing the data. The colour schemes used to derive the colours from the data can be specified with option `-datapointcolscheme`.
Possible values: `any`. Default value: `grey`.

`-datapointcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the points loaded with option `-datapoint`. The type of colour scheme depends on the type of data. For crystal orientations, the colour scheme can be: '`R`' for Rodrigues vector colouring; for scalar data, the colour scheme can be any list of colours.
Possible values: `"R" for crystal orientations and any list of colours for scalars`. Default value: `"R" for crystal orientations and "blue,cyan,yellow,red" for scalars`.

`-datapointtrs` *real* [Option]
Set the transparency of the points. Provide as argument a value that applies to all points or '`file(`*`file_name`*`)`' to load values from a file.
Possible values: `0 to 1`. Default value: `0`.

`-datapointscale` *char_string* [Option]
Set the scale relative to the '`-datapointcol scal`' data. Provide as argument the start and end values, combined with '`:`'. To specify the intermediate values, provide as argument the start value, the intermediate values, and then the end value, combined with '`:`'. The labels of the scale follow the format used for the start value.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datapointscaletitle` *char_string* [Option]
Set the title of the scale relative to the '`-datapointcol scal`' data.
Possible values: `any`. Default value: `none`.

4.1.6 Coordinate System Rendering

`-datacsyscoo` *char_string* [Option]
Set the coordinates of the origin of the coordinate system. Combine the coordinates with '`:`'.
Possible values: `any`. Default value: `0:0:0`.

`-datacsyslength` *real* [Option]
Set the length of the coordinate system axes.
Possible values: `any`. Default value: `0.2`.

`-datacsysrad` *real* [Option]
Set the radius of the coordinate system axes.
Possible values: `any`. Default value: `0.01`.

`-datacsyslabel` *char_string* [Option]
Set the labels of the coordinate system axes. Combine the labels with '`:`'.
Possible values: `any`. Default value: `X1:X2:X3`.

`-datacsyscol` *char_string* [Option]
Set the colour of the coordinate system. Provide as argument any colour as detailed in Section A.10 [Colours], page 68.
Possible values: `any`. Default value: `32|32|32`.

4.1.7 Slice Settings

`-slicemesh` *char_string* [Option]
plot one (or several) slice(s) of the mesh. Provide as argument the equation(s) of the plane(s), under the form '`a*x+b*y+c*z=d`' or any equivalent mathematical expression. Combine with '`,`'.
Possible values: `any`. Default value: `none`.

4.1.8 Show Settings

The following options apply to the full tessellations or mesh.

`-showtess` *logical* [Option]
Show or hide the tessellation.
Possible values: `0 or 1`. Default value: `1 if tess loaded and no mesh`.

`-showtesr` *logical* [Option]
Show or hide the raster tessellation.
Possible values: `0 or 1`. Default value: `1 if tesr loaded and no mesh`.

`-showmesh` *logical* [Option]
Show or hide the mesh.
Possible values: `0 or 1`. Default value: `1 if mesh loaded and no slice`.

`-showmeshslice` *logical* [Option]
Show or hide the mesh slice(s).
Possible values: `0 or 1`. Default value: `1 if existing slice(s)`.

`-showpoint` *logical or char_string* [Option]
Show or hide the points. To show only specific points, provide '`file(`*file_name*`)`' to load point numbers from a file.
Possible values: `any`. Default value: `none`.

The following option applies to the cells of a tessellation or a raster tessellation, independently of its dimension,

`-showcell` *char_string* [Option]
Specify the cells to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load polyhedron identifiers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 61, or Section A.3 [Raster Tessellation Keys], page 63.
Possible values: `any`. Default value: `all`.

For a tessellation, it is also possible to set visibility on a per-entity basis,

`-showpoly` *char_string* [Option]
Specify the polyhedra to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load polyhedron identifiers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 61.
Possible values: `any`. Default value: `all`.

`-showface` *char_string* [Option]
Specify the faces to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load face identifiers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 61. The following specific keys are also available: '`cell_shown`' and '`poly_shown`'.
Possible values: `any`. Default value: `none`.

`-showedge` *char_string* [Option]
Specify the edges to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load edge numbers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 61. The following specific keys are also available: '`cell_shown`', '`poly_shown`' and '`face_shown`'.
Possible values: `any`. Default value: `cell_shown`.

`-showver` *char_string* [Option]
Specify the vertices to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load vertex numbers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 61. The following specific keys are also available: '`cell_shown`', '`poly_shown`', '`face_shown`' and '`edge_shown`'.
Possible values: `any`. Default value: `none`.

`-showseed` *char_string* [Option]
Specify the seeds to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load seed numbers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 61. The following specific key is also available: '`cell_shown`'.
Possible values: `any`. Default value: `none`.

`-showfaceinter` *logical* [Secondary option]
Show the interpolations of the tessellation faces (if any). The interpolation edges are printed in grey with a radius equal to the radius of the face edges.
Possible values: `0 or 1`. Default value: `0`.

For a raster tessellation, it is possible to set visibility of the individual voxels,

`-showvox` *char_string* [Option]
Specify the voxels to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load voxel identifiers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 61.
Possible values: `any`. Default value: `all`.

The following options apply to the entities of the mesh. The options apply to 3D elements ('`elt3d`') and element sets ('`elset3d`'), 2D elements ('`elt2d`') and element sets ('`elset2d`'), 1D elements ('`elt1d`') and element sets ('`elset1d`'), 0D elements ('`elt0d`') and element sets ('`elset0d`'), and nodes ('`nodes`'). Also note that the '`elt`' and '`elset`' labels can be used in place of '`elt`*n*`d`' and '`elset`*n*`d`', where *n* is the highest mesh dimension. This enables to use the same command whatever the highest mesh dimension is.

In the following option descriptions, note that any options can be applied to element *sets* by changing '`elt`' to '`elset`'.

`-showelt3d` *char_string* [Option]
Specify the 3D elements to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load element identifiers from a file, or any expression based on the keys listed in Section A.6 [Mesh Keys], page 66.
Possible values: `any`. Default value: `all if highest mesh dim. is 3 and none otherwise`.

`-showelt2d` *char_string* [Option]
Specify the 2D elements to show. The argument can be: '`all`' for all, '`none`' for none, '`file(`*file_name*`)`' to load element identifiers from a file, or any expression based on the keys listed in Section A.6 [Mesh Keys], page 66. The following specific key is also available: '`elt3d_shown`'.
Possible values: `any`. Default value: `all if highest mesh dim. is 2 and none otherwise`.

`-showelt1d` *char_string* [Option]
Specify the 1D elements to show. The argument can be: '`all`' for all , '`none`' for none, '`file(`*file_name*`)`' to load element numbers from a file, or any expression based on the keys listed in Section A.6 [Mesh Keys], page 66. The following specific keys are also available: '`elt2d_shown`' and '`elt3d_shown`'.
Possible values: `any`. Default value: `all if highest mesh dim. is 1 and none otherwise`.

`-showelt0d` *char_string* [Option]
Specify the 0D elements to show. The argument can be: '`all`' for all '`none`' for none, '`file(`*file_name*`)`' to load element numbers from a file, or any expression based on the keys listed in Section A.6 [Mesh Keys], page 66. The following specific keys are also available: '`elt1d_shown`', '`elt2d_shown`' and '`elt3d_shown`'.
Possible values: `any`. Default value: `all if highest mesh dim. is 0 and none otherwise`.

`-shownode` *char_string* [Option]
Specify the nodes to show. The argument can be: '`all`' for all , '`none`' for none, '`file(`*file_name*`)`' to load node numbers from a file, or any expression based on the keys listed in Section A.6 [Mesh Keys], page 66. The following specific keys are also available: '`elt0d_shown`', '`elt1d_shown`', '`elt2d_shown`' and '`elt3d_shown`'.
Possible values: `any`. Default value: `none`.

`-showcsys` *logical* [Option]
Show the coordinate system.
Possible values: `0 or 1`. Default value: `0`.

`-showshadow` *logical* [Option]
Show the shadows. If you want colours not affected by shadowing, switch this option off.
Possible values: `0 or 1`. Default value: `1 in 3D and 0 in 2D`.

4.1.9 Camera Settings

`-cameracoo` *char_string:char_string:char_string* [Option]
Specify the camera coordinates. By default, the camera is shifted by *length* times a vector v from the centre of the bounding box of the tessellation or mesh. The variable *length* is the average length of the bounding box (1 for a unit cube), and the coordinates of vector v are denoted as vx, vy and vz (= 3.462, -5.770 and 4.327, respectively, in 3D, and 0, 0 and 8, respectively, in 2D). The coordinates of the tessellation or mesh centre are denoted as x, y and z (if both a tessellation and a mesh have been loaded, the mesh is considered). Provide as argument the expression for the 3 coordinates, combined with '`:`'.
Possible values: `any`. Default value: `x+length*vx:y+length*vy:z+length*vz`.

`-cameralookat` *char_string:char_string:char_string* [Option]
Specify the location the camera looks at. By default, the camera looks at the tessellation or mesh centre. The coordinates of the tessellation or mesh centre are denoted as x, y and z (if both a tessellation and a mesh have been loaded, the mesh is considered). Provide as argument the expression for the 3 coordinates, combined with '`:`'.
Possible values: `any`. Default value: `x:y:z`.

`-cameraangle` *real* [Option]
Specify the opening angle of the camera along the horizontal direction (in degrees). The opening angle along the vertical direction is determined from the opening along the horizontal direction and the image size ratio.
Possible values: `any`. Default value: `25`.

`-camerasky` *real:real:real* [Option]
Specify the sky vector of the camera (vertical direction). Provide as argument the coordinates combined with '`:`'.
Possible values: `any`. Default value: `0:0:1`.

`-cameraprojection` *char_string* [Option]
Specify the type of projection of the camera.
Possible values: `perspective or orthographic`. Default value: `perspective for 3D and orthographic for 2D`.

4.1.10 Output Image Settings

`-imagesize` *int:int* [Option]
Specify the size of the image (in pixels). Provide as argument the width and height, combined with '`:`'.
Possible values: `any`. Default value: `1200:900`.

`-imagebackground` *char_string* [Option]
Specify the colour of the background. Provide as argument any colour as detailed in Section A.10 [Colours], page 68.
Possible values: `any`. Default value: `white`.

`-imageantialias` *logical* [Option]
Use antialiasing to produce a smoother image. Switch antialiasing off for faster image generation or smaller image file.
Possible values: `0 or 1`. Default value: `1`.

`-imageformat` *char_string* [Option]
Specify the output image format. It can be the PNG format (`png`) or the POV-Ray format (`pov`). Use '`pov:objects`' to get a POV-Ray file containing only the objects[1]. Combine with '`,`'.
Possible values: `see above list`. Default value: `png`.

4.1.11 Scripting

`-loop char_string real real real ... -endloop` [Option]
Create a loop of commands. Provide as argument the name of the loop variable, its initial value, the loop increment value, the final value, and then the commands to execute. An example of use of the `-loop` / `-endloop` capability is provided in the Examples Section.
Possible values: `any`. Default value: `none`.

4.1.12 Advanced Options

`-includepov` *char_string:char_string:...* [Option]
Use this option to include additional objects to the image, under the form of a POV-Ray file. Provide as first argument the name of the POV-Ray file and as next arguments successive transformations to apply to the objects of the POV-Ray file. The

[1] Not compatible with '`png`'; the resulting file can be loaded with `-includepov`.

transformations can be '`translate(vx,vy,vz)`' for a translation of vector (vx, vy, vz), '`scale(sx,sy,sz)`' for scaling by factors sx, sy and sz in the three directions of space, and '`rotate(thetax,thetay,thetaz)`' for a rotation of angles $thetax$, $thetay$ and $thetaz$ about axes x, y and z[2].
Possible values: `any`. Default value: `none`.

4.2 Output Files

The output files are

- PNG file, `.png`: a bitmapped image (the alpha channel is off).
- POV-Ray file, `.pov`: a POV-Ray script file.

A PNG image can be obtained from a POV-Ray file by invoquing POV-Ray as follows (see the POV-Ray documentation for details and further commands), `povray file.pov +Wimage_width +Himage_height -D +A0.2`.

4.3 Examples

Below are some examples of use of neper -V.

1. Print out tessellation `n10-id1.tess` with cells coloured from their identifiers and an image size of 900×450 pixels.

```
$ neper -V n10-id1.tess -datacellcol id -imagesize 900:450 -print img
```

2. Print out tessellation `n10-id1.tess` with cells coloured from crystal orientations and semi-transparency.

```
$ neper -V n10-id1.tess -datacellcol ori -datacelltrs 0.5 -print img
```

3. Print out mesh `n10-id1.msh` with elements coloured from scalar values written in file `v` and a scale ranging from 0 to 100.

```
$ neper -V n10-id1.msh -dataeltcol scal:v -dataeltscale 0:100 \
        -print img
```

4. Print out mesh `n10-id1.msh` with elements coloured from nodal scalar values written in file `v` and a scale ranging from 0 to 100.

```
$ neper -V n10-id1.msh -datanodecol scal:v -dataeltcol from_nodes \
        -dataeltscale 0:100 -print img
```

5. Print out the 10 first cells of a 100-cell tessellation, coloured from their identifiers and semi-transparency, and with edges shown in red and vertices shown as green spheres of radius 0.01.

```
$ neper -V n100-id1.tess -showcell "id<=10" \
        -datacellcol id -datacelltrs 0.5 \
        -showedge cell_shown -showver cell_shown \
        -dataverrad 0.01 -dataedgecol red -datavercol green \
        -print img
```

[2] The rotation is read in POV-Ray style; i.e., according to the POV-Ray documentation: *Note that the order of the rotations does matter. Rotations occur about the x-axis first, then the y-axis, then the z-axis. If you are not sure if this is what you want then you should only rotate on one axis at a time using multiple rotation statements to get a correct rotation. Rotation is always performed relative to the axis. Thus if an object is some distance from the axis of rotation it will not only rotate but it will orbit about the axis as though it was swinging around on an invisible string. POV-Ray uses a left-handed rotation system. Using the famous "Computer Graphics Aerobics" exercise, you hold up your left hand and point your thumb in the positive direction of the axis of rotation. Your fingers will curl in the positive direction of rotation. Similarly if you point your thumb in the negative direction of the axis your fingers will curl in the negative direction of rotation.*

6. Print out the interior element sets of mesh `n100-id1.msh` and show the 1D elements.

```
$ neper -V n100-id1.tess,n100-id1.msh -dataelsetcol id \
        -showelset 'body>0' -showelt1d elt3d_shown -print img
```

7. Print out 3 slices of mesh `n100-id1.msh`.

```
$ neper -V n100-id1.msh -dataelsetcol id \
        -slicemesh "x=0.5,y=0.5,z=0.5" -print img
```

8. Print out slices of mesh `n100-id1.msh`, at z coordinates ranging from 0.1 to 0.9 by step of 0.1, each slice being printed in a separate file.

```
$ neper -V n100-id1.msh -dataelsetcol id \
        -loop Z 0.1 0.1 0.9 \
        -slicemesh "z=Z" -print imgZ \
        -endloop
```

Appendix A Expressions and Keys

A.1 Mathematical and Logical Expressions

Neper can handle mathematical expressions. It makes use of the `muparser` library. The expression must contain no space, tabulation or new-line characters, and match the following syntax[1].

A.1.1 Functions

The following table gives an overview of the functions supported by the default implementation. It lists the function names, the number of arguments and a brief description.

Name	**Description**
`sin`	sine function
`cos`	cosine function
`tan`	tangens function
`asin`	arcus sine function
`acos`	arcus cosine function
`atan`	arcus tangens function
`sinh`	hyperbolic sine function
`cosh`	hyperbolic cosine
`tanh`	hyperbolic tangens function
`asinh`	hyperbolic arcus sine function
`acosh`	hyperbolic arcus tangens function
`atanh`	hyperbolic arcur tangens function
`log2`	logarithm to the base 2
`log10`	logarithm to the base 10
`log`	logarithm to the base 10
`ln`	logarithm to base e (2.71828...)
`exp`	e raised to the power of x
`sqrt`	square root of a value
`sign`	sign function: -1 if $x < 0$; 1 if $x > 0$
`rint`	round to nearest integer
`abs`	absolute value
`min`	min of all arguments
`max`	max of all arguments
`sum`	sum of all arguments
`avg`	mean value of all arguments

A.1.2 Binary Operators

The following table lists the default binary operators supported by the parser.

Operator	**Description**	**Priority**
`&&`	logical and	1
`||`	logical or	2
`<=`	less or equal	4
`>=`	greater or equal	4
`!=`	not equal	4
`==`	equal	4

[1] Taken from the `muparser` documentation, see `http://beltoforion.de/article.php?a=muparser` for more information.

`>`	greater than	4
`<`	less than	4
`+`	addition	5
`-`	subtraction	5
`*`	multiplication	6
`/`	division	6
`^`	raise x to the power of y	7

A.1.3 Ternary Operators

The parser has built in support for the if-then-else operator. It uses lazy evaluation in order to make sure only the necessary branch of the expression is evaluated.

Operator	**Description**
`?:`	if-then-else operator, following the C/C++ syntax: `(test)?value_if_true:value_if_false`.

A.1.4 Statistical Distributions

The following table lists the statistical distributions. Arguments enclosed in square brackets ('`[`' and '`]`') are optional. Custom endpoints (not indicated) can also be added as arguments, as described in the following.

Operator	**Description**	**Information**
`normal(`*mean*`,`*sigma*`)`	normal	
`lognormal(`*mean*`,`*sigma*`)`	lognormal	
`dirac(`*mean*`)`	Dirac	
`beta(`*x*`,`*y*`)`	beta function	$x > 0$, $y > 0$
`lorentzian(`*mean*`,`*sigma*`)`	Lorentzian	
`studentst(`*mean*`,`*sigma*`)`	Student's t	
`weibull(`*k*`,`*sigma*`)`	Weibull	$k > 0$ represents the shape
`breitwigner(`*mean*`,`*sigma*`[,`*gamma*`])`	Breit-Wigner	$gamma \geq 0$, default 1
`expnormal(`*mean*`,`*sigma*`[,`*gamma*`])`	exp-normal	$gamma > 0$ default $sigma$
`moffat(`*mean*`,`*sigma*`[,`*gamma*`])`	Moffat	$gamma > 0$, default 1
`pearson7(`*mean*`,`*sigma*`[,`*gamma*`])`	Pearson type VII	default $gamma = 1.5$
`pseudovoigt(`*mean*`,`*sigma*`[,`*gamma*`])`	Pseudo-Voigt	$gamma \in [0, 1]$, default 0.5
`skewnormal(`*mean*`,`*sigma*`[,`*gamma*`])`	skewed normal	default $gamma = sigma$

'`mean`' represents the mean (or centre), and '`sigma`' (> 0) represents the standard deviation (or scale). '`gamma`' depends on the distribution function (see the above table). For all distributions, custom endpoints can be added as last arguments, '*`from`*`,`*`to`*', where '*`from`*' is the lower end point and '*`to`*' is the upper endpoint. If the parameter keywords are used, the parameters can be given in any order; e.g., '`moffat(`*gamma*`=1,`*from*`=0,`*to*`=1,`*sigma*`=0.1,`*mean*`=0.5)`'. Otherwise, the parameters are read based on their positions; e.g., '`lognormal(0.25,0.01)`' yields $mean = 0.25$ and $sigma = 0.01$. It is also possible (but not recommended) to specify some variables by keyword and let others be read by position. Endpoints are considered inclusive by default, but exclusive endpoints can be specified using '`fromexclusive=`*from*' and '`toexclusive=`*to*' ('`frominclusive=`*from*' and '`toinclusive=`*to*' can be used for inclusive endpoints). String completion is available for the keywords. Finally, a sum of distributions of increasing averages can be provided; e.g., '`0.3*lognormal(0.5,0.1)+0.7*normal(1,0.1)`'.

A.2 Tessellation Keys

Available keys for tessellation seeds, vertices, edges, faces and polyhedra are provided below. Also note that the descriptors apply to *cells* if they are tagged to apply to *polyhedra* and the tessellation is 3D and *faces* and the tessellation is 2D.

To turn a key value into a value relative to the mean over all entities (e.g. the relative cell size), append the key expression with the '`:rel`' modifier. To turn a key value into a value which holds for a unit cell size, append the key expression with the '`:uc`' modifier. To use as a reference only the *body* or *true* entities (see below), append '`b`' or '`t`' to the modifiers, respectively.

Key	**Descriptor**	**Apply to**
`id`	identifier	seed, ver, edge, face, poly
`x`	x coordinate	seed, ver, edge, face, poly
`y`	y coordinate	seed, ver, edge, face, poly
`z`	z coordinate	seed, ver, edge, face, poly
`xmin`	minimum x coordinate	edge, face, poly
`ymin`	minimum y coordinate	edge, face, poly
`zmin`	minimum z coordinate	edge, face, poly
`xmax`	maximum x coordinate	edge, face, poly
`ymax`	maximum y coordinate	edge, face, poly
`zmax`	maximum z coordinate	edge, face, poly
`w`	weight (width for a lamella tessellation)	seed, cell
`true`	true level	ver, edge, face, poly
`body`	body level	ver, edge, face, poly
`state`	state	ver, edge, face, poly
`domtype`	type of domain (0 if on a domain vertex, 1 if on a domain edge and 2 if on a domain face)	ver, edge, face
`domface`	domain face (-1 if undefined)	face
`domedge`	domain edge (-1 if undefined)	edge
`domver`	domain vertex (-1 if undefined)	ver
`scale`	scale	face[2]
`length`	length	edge
`area`	area	face, poly
`vol`	volume	poly
`size`	size (area/volume in 2D/3D)	cell
`diameq`	equivalent diameter[3]	face, poly
`radeq`	equivalent radius[4]	face, poly
`circularity`	circularity (2D counterpart of sphericity)	face
`sphericity`	sphericity[5]	poly
`convexity`	convexity[6]	face[7], poly
`dihangleav`	average dihedral angle	face, poly

[2] Applies only to a 3D tessellation and relevant for multiscale tessellations. The scale of a face is the scale at which the face was created, and it ranges from 0 (for domain faces) to the number of scales of the tessellation (for the last created faces).

[3] Equivalent diameter = diameter of the circle of equivalent area/volume in 2D/3D.

[4] Equivalent radius = radius of the circle of equivalent area/volume in 2D/3D.

[5] Sphericity of a polyhedron = ratio of the surface area of the sphere of equivalent volume to the surface area of the polyhedron.

[6] Convexity of a polyhedron (face) = ratio of the volume (area) of the polyhedron (face) to the volume (area) of the convex hull of the polyhedron (face).

[7] Applies only to a 2D tessellation.

`dihanglemin`	minimum dihedral angle	face, poly
`dihanglemax`	maximum dihedral angle	face, poly
`dihangles`	dihedral angles	face, poly
`ff`	flatness fault (in degrees)	face
`theta`	disorientation angle (in degrees)	edge (in 2D), face (in 3D)
`cyl`	whether or not is used to describe the circular part of a cylinder domain	edge
`vernb`	number of vertices	edge, face, poly
`vers`	vertices	edge, face, poly
`edgenb`	number of edges	ver, face, poly
`edges`	edges	ver, face, poly
`facenb`	number of faces	ver, edge, poly
`faces`	faces	ver, edge, poly
`polynb`	number of polyhedra	ver, edge, face
`polys`	polyhedra	ver, edge, face
`nfacenb`	number of neighbouring faces	face
`nfaces`	neighbouring faces	face
`npolynb`	number of neighbouring polyhedra	poly
`npolys`	neighbouring polyhedra	poly
`nseednb`	number of neighbouring seeds	poly
`nseeds`	neighbouring seeds[8]	poly
`vercoos`	vertex coordinates	face, poly
`faceareas`	face surface areas	poly
`faceeqs`	face equations[9]	poly
`scaleid(`*`scale`*`)`	identifier of the corresponding cell at scale *scale*	cell
`lamid`	lamella width id[10]	face, poly
`modeid`	mode id[11]	face, poly
`e`	Euler angles (Bunge convention)	seed, face (in 2D), poly

Variables consisting of several values ('`vers`', etc.) are not available for sorting (option `-sort`).

For a cell, the `body` and `true` variables are defined as follows,

- *`body`* is an integer equal to `0` if the cell is at the domain boundary, i.e. if it shares at least one face with it (edge in 2D), and is equal to `1` or higher otherwise. This is determined as follows: if a cell is surrounded by cells with *`body`* values equal to or higher than *`n`*, its *`body`* value is equal to *`n`* `+ 1`. Therefore, `body` tends to increase with the distance to the domain boundary and can be used to define cells that may suffer from boundary effects.
- *`true`* is an integer equal to `0` it the cell shape is biased by the domain boundary, and is equal to `1` or higher otherwise. A value higher than `0` is achieved if and only if any seed that would have been located outside the domain (where it could not be) would not have affected the shape of the cell. This condition is fulfilled if the distance between the seed of the cell and any of its vertices is lower than the minimum distance between a vertex of the cell and the domain boundary. *`true`* is extended to values higher than `1` in the same way as body: if a cell is surrounded by cells with *`true`* values equal to or higher than *`n`*, its *`true`* value is equal to *`n`* `+ 1`. As *`body`*, *`true`* tends to increase with the distance to the domain

[8] If a polyhedron has no neighbour on a face, a negative value is returned.

[9] A face equation is specified by the parameters d, a, b and c, with the equation being: $ax + by + cz = d$. The vector (a, b, c) is pointing outwards of the polyhedron.

[10] In the case of a lamellar tessellation with several lamella widths, *lamid* stands for the actual lamellar width of the cell (starting from 1).

[11] In the case of a multimodal tessellation (e.g. in terms of cell size), *modeid* stands for the actual mode (starting from 1).

boundary, and *true* $\leq$ *body*. `true` is especially useful for statistics on the cells (morphology, mesh, etc.), for which only cells with *true* ≥ 1 should be considered.

For entities of lower dimension than cells (vertices, edges and faces), `body` and `true` are equal to the maximum `body` or `true` values of the cells they belong to.

A.3 Raster Tessellation Keys

Available keys for raster tessellation seeds and cells are provided below. Mathematical and logical expressions based on these keys can also be used. To turn a key value into a value relative to the mean over all entities (e.g. the relative cell size), append the key expression with the '`:rel`' modifier. To turn a key value into a value which holds for a unit cell size, append the key expression with the '`:uc`' modifier.

Key	Descriptor	Apply to
`id`	identifier	seed, cell, voxel
`x`	x coordinate	seed, cell, voxel
`y`	y coordinate	seed, cell, voxel
`z`	z coordinate	seed, cell, voxel
`xyz`	x, y and z coordinates	seed, cell, voxel
`vx`	x coordinate (in voxel)	voxel
`vy`	y coordinate (in voxel)	voxel
`vz`	z coordinate (in voxel)	voxel
`vxyz`	x, y and z coordinates (in voxel)	voxel
`e`	Euler angles (Bunge convention)	cell, voxel
`cell`	cell	voxel
`w`	Laguerre weight	seed
`size`	size (area/volume in 2D/3D)	cell
`diameq`	equivalent diameter[12]	cell
`radeq`	equivalent radius[13]	cell
`convexity`	convexity[14]	cell
`voxsizex`	Voxel size in direction x	general
`voxsizey`	Voxel size in direction y	general
`voxsizez`	Voxel size in direction z	general
`domsizex`	Domain size in direction x	general
`domsizey`	Domain size in direction y	general
`domsizez`	Domain size in direction z	general

A.4 Tessellation Optimization Keys

A.4.1 Time Keys

The available keys for option `-morphooptilogtime` are provided below. Use '`iter(factor)`', where '*factor*' is an integer reduction factor, to log values only at specific iteration numbers.

Key	Descriptor
`iter`	iteration number
`varupdateqty`	number of updated variables
`seedupdateqty`	number of updated seeds

[12] Equivalent diameter = diameter of the circle of equivalent area/volume in 2D/3D.

[13] Equivalent radius = radius of the circle of equivalent area/volume in 2D/3D.

[14] Convexity of a cell = ratio of the volume of the cell to the volume of the convex hull of the cell.

`seedupdatelist`	list of updated seeds
`cellupdateqty`	number of updated cells
`cellupdatelist`	list of updated cells
`var`	time for variable update
`seed`	time for seed update
`cell_init`	time for cell update initialization
`cell_kdtree`	time for cell update kdtree computation
`cell_shift`	time for cell update shift computation
`cell_neigh`	time for cell update neigh. computation
`cell_cell`	time for cell update cell computation
`cell_other`	time for cell update others
`cell_total`	total time for cell update
`val`	time for (objective function) value update
`val_init`	time for (objective function) value update / initialization
`val_penalty`	time for (objective function) value update / penalty computation
`val_val`	time for (objective function) value update / value computation
`val_val_cellval`	time for (objective function) value update / value computation / cell values
`val_val_comp`	time for (objective function) value update / value computation / computation
`val_comp`	time for (objective function) value update / computation
`total`	total time
`cumtotal`	cumulative total time

A.4.2 Variable Keys

The available keys for option `-morphooptilogvar` are provided below. Use '`iter(`*`factor`*`)`', where '*`factor`*' is an integer reduction factor, to log values only at specific iteration numbers.

Key	**Descriptor**	**Apply to**
`iter`	iteration number	n/a
`id`	identifier	seed
`x`	x coordinate	seed
`y`	y coordinate	seed
`z`	z coordinate	seed
`w`	weight	seed

A.4.3 Objective Function Value Keys

The available keys for option `-morphooptilogval` are provided below. Use '`iter(`*`factor`*`)`', where '*`factor`*' is an integer reduction factor, to log values only at specific iteration numbers.

Key	**Descriptor**	**Apply to**
`iter`	iteration number	n/a
`val`	value	n/a
`valmin`	minimal value	n/a
`val0`	value, without smoothing	n/a
`valmin0`	minimal value, without smoothing	n/a
`val(`*`i`*`)`	*i*th subvalue	n/a
`val0(`*`i`*`)`	*i*th subvalue, without smoothing	n/a
`eps`	error on the objective function (see option `-morphooptistop`)	n/a

`reps`	relative error on the objective function (see option `-morphooptistop`)	n/a
`loop`	optimization loop	n/a
`plateaulength`	current plateau length[15]	n/a

A.4.4 Statistical Distribution Keys

The available keys for option `-morphooptilogdis` are provided below. PDF stands for *probability density function* and CDF stands for *cumulative probability density function*. Use '`iter(factor)`', where '*`factor`*' is a reduction factor, to log values only at specific iteration numbers.

Key	Descriptor	Apply to
`iter`	iteration number	n/a
`x`	x coordinate	n/a
`tarpdf`	target PDF	n/a
`tarcdf`	target CDF	n/a
`curpdf`	current PDF	n/a
`curcdf`	current CDF	n/a
`tarpdf0`	target PDF, not smoothed	n/a
`tarcdf0`	target CDF, not smoothed	n/a
`curcdf0`	current CDF, not smoothed	n/a

A.4.5 Raster Tessellation Voxel Keys

The available keys for option `-morphooptilogtesr` are provided below. Values are written for each voxel used to compute the objective function. Use '`iter(factor)`', where '*`factor`*' is a reduction factor, to log values only at specific iteration numbers.

Key	Descriptor	Apply to
`iter`	iteration number	n/a
`id`	cell identifier	n/a
`x`	x coordinate	n/a
`y`	y coordinate	n/a
`z`	z coordinate	n/a
`dist`	distance to the cell	n/a

A.5 Orientation Optimization Keys

A.5.1 Variable Keys

The available keys for option `-orioptilogvar` are provided below. For all orientation descriptors but '`q`', the returned orientation are located in the fundamental region.

Key	Descriptor	Apply to
`iter`	iteration number	n/a
`id`	identifier	seed
`g`	rotation matrix	seed
`rtheta`	rotation axis / angle pair	seed
`R`	Rodrigues vector	seed
`q`	quaternion	seed

[15] Number of iterations during which the objective function did not decrease.

`e`	Euler angles (Bunge convention)	seed
`ek`	Euler angles (Kocks convention)	seed
`er`	Euler angles (Roe convention)	seed

A.6 Mesh Keys

Available keys for mesh node, elements and element sets (of all dimensions) and points are provided below.

Key	**Descriptor**	**Apply to**
`id`	identifier	node, *n*D elt, *n*D elset
`x`	x coordinate	node, *n*D elt, *n*D elset
`y`	y coordinate	node, *n*D elt, *n*D elset
`z`	z coordinate	node, *n*D elt, *n*D elset
`dim`	dimension (= lowest parent elt dimension)	node
`elset0d`	0D elset	0D elt
`elset1d`	1D elset	1D elt
`elset2d`	2D elset	2D elt
`elset3d`	3D elset	3D elt
`part`	partition	*n*D elt, node
`cyl`	whether or not is used to describe the circular part of a cylinder domain	1D elt, 1D elset
`vol`	volume	3D elt, 3D elset
`area`	area	2D elt
`length`	length	1D elt, 3D elt, 1D elset
`elsetvol`	elset volume	3D elt
`elsetarea`	elset area	2D elt
`elsetlength`	elset length	1D elt
`rr`	radius ratio	3D elt
`rrav`, `rrmin`, `rrmax`	average, min and max radius ratios	3D elset
`Osize`	Osize	3D elset
`eltnb`	number of elements	*n*D elset
`true`	true level	*n*D elt, *n*D elset
`body`	body level	*n*D elt, *n*D elset
`domtype`	type of domain (0 if on a domain vertex, 1 if on a domain edge and 2 if on a domain face)	2D elset, 1D elset, 0D elset, 2D elt, 1D elt, 0D elt
`2dmeshp`	coordinates of the closest point of the 2D mesh	node, 3D elt
`2dmeshd`	distance to '`2dmeshp`'	node, 3D elt
`2dmeshv`	vector to '`2dmeshp`'	node, 3D elt
`2dmeshn`	outgoing normal vector of the 2D mesh at '`2dmeshp`'	node, 3D elt
`e`	Euler angles (Bunge convention)	2D elt, 3D elt, 2D elset, 3D elset

nD stands for an arbitrary dimension (from 0D to 3D). Variables starting by '`2dmesh`' are only available for statistics (options starting by `-stat` of module -M); for elements, they apply to the centroids.

A.7 Point Keys

Available keys for points are provided below.

Key	Descriptor	Apply to	Require
`id`	identifier	point	
`x`	x coordinate	point	
`y`	y coordinate	point	
`z`	z coordinate	point	
`cell`	cell	point	tessellation
`elt`	containing element	point	mesh
`elset`	containing elset	point	mesh
`2dmeshp`	coordinates of the closest point of the 2D mesh	point	3D mesh
`2dmeshd`	distance to '`2dmeshp`'	point	3D mesh
`2dmeshv`	vector to '`2dmeshp`'	point	3D mesh
`2dmeshn`	outgoing normal vector of the 2D mesh at '`2dmeshp`'	point	3D mesh

A.8 Rotations and Orientations

Rotations and orientations can be described using the following descriptors (see Orilib, `http://orilib.sourceforge.net`, for more information).

Key	Descriptor	Number of variables
`g`	rotation matrix	9
`rtheta`	rotation axis / angle pair	4
`R`	Rodrigues vector	3
`q`	quaternion	4
`e`	Euler angles (Bunge convention)	3
`ek`	Euler angles (Kocks convention)	3
`er`	Euler angles (Roe convention)	3

Keys are available for ideal orientations:

Cube	$(0\,0\,1)\,[1\,0\,0]$
Goss	$(0\,1\,1)\,[1\,0\,0]$
U	$(1\,0\,1)\,[\bar{1}\,0\,1]$
45NDCube	$(0\,0\,1)\,[1\,\bar{1}\,0]$
S1	$(1\,2\,3)\,[6\,3\,\bar{4}]$
S2	$(\bar{1}\,2\,3)\,[6\,\bar{3}\,4]$
S3	$(1\,\bar{2}\,3)\,[6\,\bar{3}\,\bar{4}]$
S4	$(\bar{1}\,\bar{2}\,3)\,[6\,3\,4]$
Brass1	$(1\,1\,0)\,[1\,\bar{1}\,2]$
Brass2	$(\bar{1}\,1\,0)\,[1\,1\,\bar{2}]$
Copper1	$(1\,1\,2)\,[1\,1\,\bar{1}]$
Copper2	$(\bar{1}\,1\,2)\,[1\,\bar{1}\,1]$

A.9 Crystal Symmetries

Crystal symmetries can be described using the following descriptors.

Key	Descriptor	Number of operators
`triclinic`	triclinic (Laue group $\bar{1}$)	24
`cubic`	cubic	24

`hexagonal`	hexagonal	1
`-1`	Laue group $\overline{1}$	1
`2/m`	Laue group $2/m$	2
`mmm`	Laue group mmm	4
`4/m`	Laue group $4/m$	4
`4/mmm`	Laue group $4/mmm$	8
`-3`	Laue group $\overline{3}$	3
`-3m`	Laue group $\overline{3}m$	6
`6/m`	Laue group $6/m$	6
`6/mmm`	Laue group $6/mmm$	12
`m-3`	Laue group $m\overline{3}$	12
`m-3m`	Laue group $m\overline{3}m$	24

Keys are available for ideal orientations:

Cube	$(0\,0\,1)\,[1\,0\,0]$
Goss	$(0\,1\,1)\,[1\,0\,0]$
U	$(1\,0\,1)\,[\overline{1}\,0\,1]$
45NDCube	$(0\,0\,1)\,[1\,\overline{1}\,0]$
S1	$(1\,2\,3)\,[6\,3\,\overline{4}]$
S2	$(\overline{1}\,2\,3)\,[6\,\overline{3}\,4]$
S3	$(1\,\overline{2}\,3)\,[6\,\overline{3}\,\overline{4}]$
S4	$(\overline{1}\,\overline{2}\,3)\,[6\,3\,4]$
Brass1	$(1\,1\,0)\,[1\,\overline{1}\,2]$
Brass2	$(\overline{1}\,1\,0)\,[1\,1\,\overline{2}]$
Copper1	$(1\,1\,2)\,[1\,1\,\overline{1}]$
Copper2	$(\overline{1}\,1\,2)\,[1\,\overline{1}\,1]$

A.10 Colours

The available colours are provided below, with their corresponding RGB channel values. Any other colour can be defined from the RGB channel values, under format '*R_value*|*G_value*|*B_value*'.

(0, 0, 0)	`black`	(255, 0, 0)	`red`
(0, 255, 0)	`green`	(0, 0, 255)	`blue`
(255, 255, 0)	`yellow`	(255, 0, 255)	`magenta`
(0, 255, 255)	`cyan`	(255, 255, 255)	`white`
(128, 0, 0)	`maroon`	(0, 0, 128)	`navy`
(127, 255, 0)	`chartreuse`	(0, 255, 127)	`springgreen`
(128, 128, 0)	`olive`	(128, 0, 128)	`purple`
(0, 128, 128)	`teal`	(128, 128, 128)	`grey`
(0, 191, 255)	`deepskyblue`	(124, 252, 0)	`lawngreen`
(64, 64, 64)	`darkgrey`	(255, 69, 0)	`orangered`
(192, 192, 192)	`silver`	(255, 250, 250)	`snow`
(139, 0, 0)	`darkred`	(0, 0, 139)	`darkblue`
(255, 140, 0)	`darkorange`	(240, 255, 255)	`azure`
(248, 248, 255)	`ghostwhite`	(255, 255, 240)	`ivory`
(0, 0, 205)	`mediumblue`	(255, 182, 193)	`lightpink`
(245, 255, 250)	`mintcream`	(75, 0, 130)	`indigo`
(240, 128, 128)	`lightcoral`	(255, 192, 203)	`pink`
(255, 127, 80)	`coral`	(250, 128, 114)	`salmon`
(255, 250, 240)	`floralwhite`	(127, 255, 212)	`aquamarine`

(255, 250, 205)	lemonchiffon
(0, 100, 0)	darkgreen
(240, 248, 255)	aliceblue
(255, 255, 224)	lightyellow
(0, 139, 139)	darkcyan
(70, 130, 180)	steelblue
(255, 245, 238)	seashell
(72, 61, 139)	darkslateblue
(255, 160, 122)	lightsalmon
(135, 206, 250)	lightskyblue
(240, 255, 240)	honeydew
(255, 218, 185)	peachpuff
(255, 99, 71)	tomato
(255, 105, 180)	hotpink
(255, 235, 205)	blanchedalmond
(255, 228, 181)	moccasin
(60, 179, 113)	mediumseagreen
(238, 130, 238)	violet
(255, 239, 213)	papayawhip
(188, 143, 143)	rosybrown
(139, 69, 19)	saddlebrown
(30, 144, 255)	dodgerblue
(222, 184, 135)	burlywood
(250, 240, 230)	linen
(64, 224, 208)	turquoise
(72, 209, 204)	mediumturquoise
(255, 228, 225)	mistyrose
(250, 235, 215)	antiquewhite
(50, 205, 50)	limegreen
(176, 196, 222)	lightsteelblue
(152, 251, 152)	palegreen
(245, 222, 179)	wheat
(230, 230, 250)	lavender
(144, 238, 144)	lightgreen
(47, 79, 79)	darkslategrey
(46, 139, 87)	seagreen
(138, 43, 226)	blueviolet
(107, 142, 35)	olivedrab
(244, 164, 96)	sandybrown
(102, 205, 170)	mediumaquamarine
(238, 232, 170)	palegoldenrod
(25, 25, 112)	midnightblue
(211, 211, 211)	lightgrey
(100, 149, 237)	cornflowerblue
(178, 34, 34)	firebrick
(205, 92, 92)	indianred
(173, 216, 230)	lightblue
(165, 42, 42)	brown
(220, 220, 220)	gainsboro
(95, 158, 160)	cadetblue
(255, 215, 0)	gold
(255, 165, 0)	orange
(224, 255, 255)	lightcyan
(139, 0, 139)	darkmagenta
(205, 133, 63)	peru
(255, 240, 245)	lavenderblush
(0, 250, 154)	mediumspringgreen
(184, 134, 11)	darkgoldenrod
(255, 228, 196)	bisque
(250, 250, 210)	lightgoldenrodyellow
(255, 248, 220)	cornsilk
(245, 245, 245)	whitesmoke
(112, 128, 144)	slategrey
(253, 245, 230)	oldlace
(189, 183, 107)	darkkhaki
(0, 206, 209)	darkturquoise
(199, 21, 133)	mediumvioletred
(173, 255, 47)	greenyellow
(143, 188, 143)	darkseagreen
(255, 20, 147)	deeppink
(148, 0, 211)	darkviolet
(119, 136, 153)	lightslategrey
(255, 222, 173)	navajowhite
(123, 104, 238)	mediumslateblue
(135, 206, 235)	skyblue
(245, 245, 220)	beige
(210, 180, 140)	tan
(216, 191, 216)	thistle
(233, 150, 122)	darksalmon
(65, 105, 225)	royalblue
(220, 20, 60)	crimson
(186, 85, 211)	mediumorchid
(240, 230, 140)	khaki
(175, 238, 238)	paleturquoise
(153, 50, 204)	darkorchid
(154, 205, 50)	yellowgreen
(219, 112, 147)	palevioletred
(147, 112, 219)	mediumpurple
(85, 107, 47)	darkolivegreen
(106, 90, 205)	slateblue
(34, 139, 34)	forestgreen
(32, 178, 170)	lightseagreen
(218, 112, 214)	orchid
(160, 82, 45)	sienna
(176, 224, 230)	powderblue
(105, 105, 105)	dimgrey
(210, 105, 30)	chocolate
(218, 165, 32)	goldenrod
(221, 160, 221)	plum

The default colour palette (used for options `-datacellcol`, `-dataelt3dcol`, etc.) is defined from the above colour list, by excluding colours of brightness below 0.2 and above 0.8. The brightness is defined as the average of the channel values divided by 255. The resulting list of colours is: `red`, `green`, `blue`, `yellow`, `magenta`, `cyan`, `chartreuse`, `springgreen`, `olive`, `purple`, `teal`, `grey`, `deepskyblue`, `lawngreen`, `darkgrey`, `orangered`, `silver`, `darkorange`, `mediumblue`, `indigo`, `lightcoral`, `coral`, `salmon`, `aquamarine`, `gold`, `orange`, `darkmagenta`, `darkcyan`, `peru`, `steelblue`, `mediumspringgreen`, `darkslateblue`, `darkgoldenrod`, `lightsalmon`, `lightskyblue`, `tomato`, `slategrey`, `hotpink`, `darkkhaki`, `darkturquoise`, `mediumseagreen`, `mediumvioletred`, `violet`, `greenyellow`, `darkseagreen`, `rosybrown`, `deeppink`, `saddlebrown`, `darkviolet`, `dodgerblue`, `lightslategrey`, `burlywood`, `mediumslateblue`, `turquoise`, `skyblue`, `mediumturquoise`, `tan`, `limegreen`, `darksalmon`, `lightsteelblue`, `royalblue`, `palegreen`, `crimson`, `mediumorchid`, `khaki`, `lightgreen`, `darkslategrey`, `darkorchid`, `seagreen`, `yellowgreen`, `blueviolet`, `palevioletred`, `olivedrab`, `mediumpurple`, `sandybrown`, `darkolivegreen`, `mediumaquamarine`, `slateblue`, `forestgreen`, `midnightblue`, `lightseagreen`, `orchid`, `cornflowerblue`, `sienna`, `firebrick`, `indianred`, `dimgrey`, `chocolate`, `brown`, `goldenrod`, `plum` and `cadetblue`.

Appendix B File Formats

B.1 Tessellation File (.tess)

Here are details on the .tess file format version 2.0. Developers should note that read and write functions are available as 'neut_tess_fscanf' and 'neut_tess_fprintf', defined in directories neut/neut_tess/neut_tess_fscanf and neut/neut_tess/neut_tess_fprintf.

```
***tess
 **format
   format
 **general
   dim type
 **cell
   number_of_cells
 [*id
   cell1_id cell2_id ... ]
 [*seed
   seed_id seed_x seed_y seed_z seed_weight
   ... ]
 [*ori
   descriptor
   cellid_param1 cellid_param2 ...
   ... ]
 [*lamid
   cell1_lamid cell2_lamid ... ]
 [*modeid
   cell1_modeid cell2_modeid ... ]
 [*crysym
   crysym]
 **vertex
   total_number_of_vertices
   ver_id  ver_x ver_y ver_z ver_state
   ...
 **edge
   total_number_of_edges
   edge_id  ver_1 ver_2 edge_state
   ...
 **face
   total_number_of_faces
   face_id number_of_vertices ver_1   ver_2   ...
           number_of_edges    edge_1* edge_2* ...
           face_eq_d face_eq_a face_eq_b face_eq_c
           face_state face_point face_point_x face_point_y face_point_z
   ...
 **polyhedron
   total_number_of_polyhedra
   poly_id number_of_faces face_1* face_2* ...
   ...
 **domain
  *general
```

```
   dom_type
  *vertex
   total_number_of_dom_vertices
   dom_ver_id  dom_ver_x dom_ver_y dom_ver_z dom_ver_label
               number_of_dom_tess_vertices ver_1
   ...
  *edge
   total_number_of_dom_edges
   dom_edge_id number_of_dom_vertices [dom_ver_1 dom_ver_2]
               dom_edge_label
               number_of_dom_tess_edges edge_1 edge_2 ...
   ...
  *face
   total_number_of_dom_faces
   dom_face_id number_of_dom_vertices dom_ver_1  dom_ver_2  ...
               number_of_dom_edges    dom_edge_1 dom_edge_2 ...
               dom_face_type
               number_of_params dom_face_param1 dom_face_param2 ...
               dom_face_label
               number_of_dom_tess_faces dom_tess_face_1 dom_tess_face_2 ...
   ...
 **periodic
  *general
   per_x per_y per_z
   per_dist_x per_dist_y per_dist_z
  *seed
   slave_seed_qty
   slave_seed_id master_seed_id per_shift_x per_shift_y per_shift_z
   ...
  *vertex
   slave_ver_qty
   slave_ver_id master_ver_id per_shift_x per_shift_y per_shift_z
   ...
  *edge
   slave_edge_qty
   slave_edge_id master_edge_id per_shift_x per_shift_y per_shift_z slave_edge_
ori
   ...
  *face
   slave_face_qty
   slave_face_id master_face_id per_shift_x per_shift_y per_shift_z slave_face_
ori
   ...
 **scale
  *general
   number_of_scales
  *cellid
   cell1_id cell1_scale1 cell1_scale2 ... cell1_scalenumber_of_scales
   ...
***end
```

where (with identifiers being integer numbers),

- `***tess` denotes the beginning of a tessellation file.
- `**format` denotes the beginning of the format field.
- *`format`* is the file format, currently '`2.0`' (character string).
- `**general` denotes the beginning of the general information field.
- *`dim`* is the dimension of the tessellation (1, 2 or 3).
- *`type`* is the type of tessellation (always '`standard`').
- `**cell` denotes the beginning of the cell field.
- *`number_of_cells`* is the number of cells.
- `*id` denotes the beginning of an optional identifier field. It the field is not present, the cells are considered to be numbered contiguously from 1.
- *`cell1_id`*, *`cell2_id`*, ... are the actual identifiers of the cells.
- `*lamid` denotes the beginning of an optional lamella identifier field.
- *`cell1_lamid`*, *`cell2_lamid`*, ... are the lamella identifiers of the cells.
- `*modeid` denotes the beginning of an optional mode identifier field.
- *`cell1_modeid`*, *`cell2_modeid`*, ... are the mode identifiers of the cells.
- *`crysym`* is the crystal symmetry (`triclinic`, `cubic` or `hexagonal`).
- `*seed` denotes the beginning of a seed field.
- *`seed_id`* is the identifier of a seed and ranges from `1` to `number_of_cells`.
- *`seed_x`*, *`seed_y`* and `seed_z` are the three coordinates of a seed (real numbers).
- *`seed_weight`* is the weight of a seed (real number).
- `*ori` denotes the beginning of an optional crystal orientation field.
- `descriptor` is the descriptor used to parametrize the crystal orientations. See Section A.8 [Rotations and Orientations], page 67, for the list of available descriptors.
- *`cellid_param1`*, *`cellid_param2`*, ... are the values of the orientation descriptor of cell *id*.
- `**vertex` denotes the beginning of the vertex field.
- *`total_number_of_vertices`* is the total number of vertices.
- *`ver_id`* is the identifier of a vertex and ranges from `1` to `total_number_of_vertices`.
- *`ver_x`*, *`ver_y`* and `ver_z` are the three coordinates of a vertex (real numbers).
- *`ver_state`* is an integer indicating the state of a vertex. For a standard tessellation (no regularization), it equals `0`. For a regularized tessellation, it equals `0` if the vertex has not been modified by regularization and is higher than `0` otherwise.
- `**edge` denotes the beginning of the edge field.
- *`total_number_of_edges`* is the total number of edges.
- *`edge_id`* is the identifier of an edge and ranges from `1` to `total_number_of_edges`.
- *`ver_1`*, *`ver_2`*, ... are identifiers of vertices.
- *`edge_state`* is an integer indicating the state of an edge (always `0`).
- `**face` denotes the beginning of the face field. It is present for a tessellation of dimension 2 or 3.
- *`total_number_of_faces`* is the total number of faces.
- *`face_id`* is the identifier of a face and ranges from `1` to `total_number_of_faces`.
- *`number_of_vertices`* is the number of vertices of a face.
- *`number_of_edges`* is the number of edges of a face.

- *edge_1**, *edge_2**, ... are identifiers of the edges of a face, signed according to their orientation in the face.
- *face_eq_a*, *face_eq_b*, *face_eq_c* and *face_eq_d* are the parameters of the equation of a face: $face\_eq\_a\,x + face\_eq\_b\,y + face\_eq\_c\,z = face\_eq\_d$. The parameters are scaled so that $face\_eq\_a^2 + face\_eq\_b^2 + face\_eq\_c^2 = 1$.
- *face_state* is an integer indicating the state of a face. For a standard tessellation (no regularization), it equals 0. For a regularized tessellation, it equals 0 if it has not been modified by regularization and 1 otherwise.
- *face_point* is an integer indicating the point used for the interpolation of a face. For a standard tessellation (no regularization), it equals 0. For a regularized tessellation: if the point is the face barycentre, it equals 0; if the point is one of the face vertices, it equals to the position of the vertex in the list of vertices of the face. It equals -1 if the point is undefined.
- *face_point_x*, *face_point_y* and *face_point_z* are the coordinates of the point used for the interpolation of a face (equal 0 if undefined).
- **polyhedron denotes the beginning of the polyhedron field. It is present for a tessellation of dimension 3.
- *total_number_of_polyhedra* is the total number of polyhedra.
- *poly_id* is the identifier of a polyhedron and ranges from 1 to total_number_of_polyhedra.
- *number_of_faces* is the number of faces of a polyhedron.
- *face_1**, *face_2**, ... are identifiers of the faces of a polyhedron, signed according to their orientations in the polyhedron (positive if the normal of the face is pointing outwards and negative if it is pointing inwards).
- **domain denotes the beginning of the domain field.
- *general denotes the beginning of the domain general information field.
- *dom_type* is the type of the domain (one of cube, cylinder, square, circle, poly and planes).
- *vertex denotes the beginning of the domain vertex field.
- *total_number_of_dom_vertices* is the total number of domain vertices.
- *dom_ver_id* is the identifier of a domain vertex and ranges between 1 to total_number_of_dom_vertices.
- *dom_ver_x*, *dom_ver_y* and dom_ver_z are the three coordinates of a domain vertex (real numbers).
- *dom_ver_label* is the label of a domain vertex.
- *number_of_dom_tess_vertices* is the number of tessellation vertices of a domain vertex (must be 1).
- *edge denotes the beginning of the domain edge field (for a tessellation of dimension 2 or 3).
- *total_number_of_dom_edges* is the total number of domain edges.
- *dom_edge_id* is the identifier of a domain edge and ranges between 1 to total_number_of_dom_edges.
- *number_of_dom_vertices* is the number of domain vertices of a domain edge or a domain face.
- *dom_ver_1*, *dom_ver_2*, ... are identifiers of the domain vertices of a domain edge or face.
- *dom_edge_label* is the label of a domain edge.

- *number_of_dom_tess_edges* is the number of tessellation edges of a domain edge.
- *face denotes the beginning of the domain face field (for a tessellation of dimension 3).
- *total_number_of_dom_faces* is the total number of domain faces.
- *dom_face_id* is the identifier of a domain face and ranges from 1 to total_number_of_dom_faces.
- *number_of_dom_edges* is the number of domain edges of a domain face.
- *dom_edge_1*, *dom_edge_2*, ... are identifiers of the domain edges of a domain face.
- *dom_face_type* is the type of a face, among 'plane', 'cylinder' or 'sphere'.
- *number_of_params* is the number of parameters of a domain face.
- *dom_face_param1*, *dom_face_param2*, ... are the parameters of a domain face. For a planar face, they are the parameters of the equation of the face, listed in the order *face_eq_d*, *face_eq_a*, face_eq_b and *face_eq_c*. For a cylindrical face, they are the coordinates of the base point, the axis and the radius. For a spherical face, they are the coordinates of the centre and the radius.
- *dom_face_label* is the label of a domain face. If dom_type is 'cube', it is one of 'x0', 'x1', 'y0', 'y1', 'z0' or 'z1'. If dom_type is 'cylinder', it is one of 'z0', 'z1', 'f1', 'f2', ... Otherwise, it is one of 'f1', 'f2', ...
- *number_of_dom_tess_faces* is the number of tessellation faces of a domain face.
- *dom_tess_face_1*, *dom_tess_face_2*, ... are the identifiers of the tessellation faces of a domain face.
- ***end denotes the end of a tessellation file.
- **periodicity denotes the beginning of the periodicity field.
- *general denotes the beginning of the periodicity general information field.
- *per_x*, *per_y* and *per_z* are booleans indicating x, y, and z periodicity.
- *per_dist_x*, *per_dist_y* and *per_dist_z* are the periodicity distances along x, y, and z.
- *seed denotes the beginning of the periodicity seed field.
- *slave_seed_qty* is the number of slave seeds.
- *slave_seed_id* is the identifier of a slave seed.
- *master_seed_id* is the identifier of the master of a slave seed.
- *per_shift_x*, *per_shift_y* and *per_shift_z* are the shifts of a slave seed (or vertex, etc.) relative to its master, along x, y and z. The values can be -1, 0 or 1.
- *vertex denotes the beginning of the periodicity vertex field.
- *slave_vertex_qty* is the number of slave vertices.
- *slave_vertex_id* is the identifier of a slave vertex.
- *master_vertex_id* is the identifier of the master of a slave vertex.
- *edge denotes the beginning of the periodicity edge field.
- *slave_edge_qty* is the number of slave edges.
- *slave_edge_id* is the identifier of a slave edge.
- *master_edge_id* is the identifier of the master of a slave edge.
- *slave_edge_ori* is the orientation of the slave edge with respect to the master edge: 1 if identical, -1 if opposite.
- *face denotes the beginning of the periodicity face field (for a tessellation of dimension 3).
- *slave_face_qty* is the number of slave faces.
- *slave_face_id* is the identifier of a slave face.

- *master_face_id* is the identifier of the master of a slave face.
- *slave_face_ori* is the orientation of the slave face with respect to the master face: `1` if identical, `-1` if opposite.
- *number_of_scales* is the number of scales.
- *cell1_scale1*, *cell1_scale2*, ... are the identifiers of the cells of the scale-1, scale-2, ... tessellations which the cell belongs to.

B.2 Raster Tessellation File (`.tesr`)

Here are details on the `.tesr` file format version `2.0`. Developers should note that read and write functions are available as '`neut_tesr_fscanf`' and '`neut_tesr_fprintf`', defined in directories `neut/neut_tesr/neut_tesr_fscanf` and `neut/neut_tesr/neut_tesr_fprintf`. Compared to a tessellation file (`.tess`), a raster tessellation file can include cell morphological properties such as their centroids or volumes. This is motivated by the fact that, for a raster tessellation, these properties are both in small number and relatively expensive to compute, and so are stored once known. Square brackets (`[ ]`) denote optional fields.

```
***tesr
 **format
   format data_format
 **general
   dimension
   size_x size_y [size_z]
   voxsize_x voxsize_y [voxsize_z]
 [*origin
   origin_x origin_y [origin_z]]
 [*hasvoid has_void]
[**cell
   number_of_cells
 [*id
   cell1_id cell2_id ...]
 [*seed
   seed_id seed_x seed_y [seed_z] seed_weight
   ...]
 [*ori
   descriptor
   cell1_param1 cell1_param2 ...
   cell2_param1 cell2_param2 ...
   ...]
 [*coo
   cell1_x cell1_y [cell1_z]
   cell2_x cell2_y [cell2_z]
   ...]
 [*vol
   cell1_vol
   cell2_vol
   ...]
 [*convexity
   cell1_convexity
   cell2_convexity
   ...]
```

```
]
[*crysym
   crysym
]
 **data
   vox1_cell vox2_cell ...
 or
  *file data_file_name
[**oridata
   descriptor
   vox1_param1 vox1_param2 ...
   vox2_param1 vox2_param2 ...
 or
   descriptor
  *file oridata_file_name
]
***end
```

where,

- `***tesr` denotes the beginning of a raster tessellation file.
- `**format` denotes the beginning of the format field.
- *format* is the file format, currently '`2.0`' (character string).
- *data_format* is the format of the data in field `**data`. It can be either `ascii`, `binary8` (8-bit binary), `binary16` (16-bit binary, LittleEndian), `binary16_big` (16-bit binary, BigEndian), `binary32` (32-bit binary, LittleEndian) or `binary32_big` (32-bit binary, BigEndian).
- `**general` denotes the beginning of the general information field.
- *dimension* is the dimension of the raster tessellation.
- *size_x*, *size_y* and *size_z* are the raster sizes along the coordinate axes. The number of sizes must match *dimension*.
- *voxsize_x*, *voxsize_y* and *voxsize_z* are the voxel (pixel, in 2D) sizes along the coordinate axes. The number of sizes must match *dimension*.
- `*origin` denotes the beginning of an optional origin field.
- *origin_x*, *origin_y* and *origin_z* are the (absolute) coordinates of the origin of the raster tessellation along the coordinate axes. The number of coordinates must match *dimension*.
- `*hasvoid` denotes the beginning of an optional has-void field.
- *has_void* is a boolean indicating whether the tessellation contains void voxels (which have a cell id of 0).
- `**cell` denotes the beginning of an optional cell field.
- *number_of_cells* is the number of cells.
- `*id` denotes the beginning of an optional identifier field. If the field is present, the cell identifiers listed under `**data` are supposed to be numbered contiguously from 1 (or 0 in case of void), and their actual identifiers are considered to be the ones provided in the list. The actual identifiers are used in output files.
- *cell1_id*, *cell2_id*, ... are the actual identifiers of the cells.
- `*seed` denotes the beginning of a seed field.
- *seed_id* is the identifier of a seed and ranges from `1` to `number_of_cells`.
- *seed_x*, *seed_y* and `seed_z` are the three coordinates of a seed (real numbers).

- *seed_weight* is the weight of a seed (real number).
- *ori denotes the beginning of an optional crystal orientation field.
- descriptor is the descriptor used to parametrize the crystal orientations. See Section A.8 [Rotations and Orientations], page 67, for the list of available descriptors.
- *cellid_param1*, *cellid_param2*, ... are the values of the orientation descriptor of cell *id*.
- *coo denotes the beginning of an optional centroid field.
- *cellid_x*, *cellid_y* and *cellid_z* are the coordinates of the centroids of cell *id*.
- *vol denotes the beginning of an optional volume field.
- *cellid_vol* is the volume of cell *id*.
- *convexity denotes the beginning of an optional convexity field.
- *cellid_convexity* is the convexity of cell *id*.
- *crysym denotes the beginning of an optional crystal symmetry field.
- *crysym* is the crystal symmetry (triclinic, cubic or hexagonal).
- **data denotes the beginning of the data field. Data can be provided in the .tesr file or in a separate file, using *file, see below.
- *voxid_cell* is the cell voxel *id* belongs to. Voxels must be listed in column-major order (x varying first, y varying second and z varying last). The cell identifiers should start from 1. Use 0 for voids.
- *file denotes the beginning of a file field.
- data_file_name is the name of a file that contains the data. It must be located in the same directory as the parent tesr file, or its path relative to the parent tesr file must be provided. Typically, it is a .raw file.
- **oridata denotes the beginning of the orientation data field. Data can be provided in the .tesr file or in a separate file, using *file, see below.
- *voxid_param1*, *voxid_param2*, ... are the values of the orientation descriptor of voxel *id*. Orientations must be listed in column-major order (x varying first, y varying second and z varying last). Arbitrary orientations can be used for void voxels (*voxid_cell* = 0). These data must be written under format *data_format*, in terms of ASCII or binary. In the case of binary format, double-precision numbers are considered.
- oridata_file_name is the name of a file that contains the orientation data. It must be located in the same directory as the parent tesr file, or its path relative to the parent tesr file must be provided. Typically, it is a .raw file.

B.3 Multiscale Cell File

A multiscale cell file provides cell-by-cell values for a multiscale tessellation and can be loaded using '**msfile(*filename*)**'[1]. The file contains, for each cell, its *multiscale identifier*, *mid*, and the value(s). A *cell multiscale identifier* (*mid*) is a character string identifying a cell at a specific scale. For a given cell, C, *mid* combines the identifiers of the cells that C belongs to, at successive scales, to its own id, separated by '::'. For a 1-scale tessellation, *mid* equals *id*. For a 2-scale tessellation composed of 2×3 cells, the *mid*s are equal to 1::1, 1::2, 1::3, 2::1, 2::2 and 2::3. The domain (which can be considered as a cell at scale 0), *mid* is nil. An example of a multiscale cell file that could be used to define the amounts of cells of a 3-scale tessellation is:

```
2
1 2
2 4
```

[1] As of version 3.5.0, '**msfile(*filename*)**' should be preferred over '**file(*filename*)**'

```
1::1 3
1::2 4
2::1 5
2::2 6
2::3 7
2::4 8
```

The file could be used in `-T` as: `-n msfile(`*`filename`*`)::msfile(`*`filename`*`)::msfile(`*`filename`*`)`. The first instance of '`msfile(`*`filename`*`)`' reads the number of scale-1 cells in line 1, the second instance of '`msfile(`*`filename`*`)`' reads the number of scale-2 cells in lines 2–3, and the third instance of '`msfile(`*`filename`*`)`' reads the number of scale-3 cells in lines 4–7.

B.4 Position File

A position file lists the coordinates of a given number of points. The file must contain 1 coordinate per point in 1D, 2 coordinates per point in 2D and 3 coordinates per point in 3D. While the dimension can be known from the context in which the file is read, it can also be specified by appending '`:`*`dim`*' to the name of the position file, where *dim* is the dimension. A coordinate can be an integer or real number. A real number can have an arbitrary number of digits, but the decimal mark must be '`.`'. The coordinates can be separated from each other by spaces, tabulators or newlines (any number as well as arbitrary combinations of them are supported). However, a good practice is to format the file with one line per point. An example of a position file containing 5 points in 3D is:

```
2.1235 9.4544 5.2145
5.9564 3.6884 9.2145
2.2547 3.2658 8.2514
8.2515 9.4157 2.9454
0.5874 4.2848 2.4874
```

Appendix C Developer's Guide

This chapter provides information useful to anyone who plans to contribute to Neper or wishes to better understand how it works. The code structure is detailed and information are given on how to efficiently contribute to it. If you are missing information, complain!

C.1 Code Structure

The Neper root directory content is as follows (the slash character '`/`' denotes directories),

- `COPYING`: license terms;
- `README`: information about other files and directories in the directory;
- `VERSIONS`: information on the versions of Neper;
- `src/`: source code directory;
- `doc/`: documentation directory.

Details on the '`src/`' and '`doc/`' directories are provided in the following.

C.1.1 Source Code

Neper's source code is located in directory `src/` and consists of roughly 120,000 lines shared between 250 directories and 1000 text files. The '`src/`' directory contains the following files and directories,

- `neper_.h` and `neper.c`

  These are the main source code header file and source code file of Neper. '`neper.c`' contains Neper's '`main`' function. It reads the arguments passed at the command line and runs the corresponding functions, which can be one of the program module.

- `neper_t/`, `neper_m/` and `neper_v/`

  These are directories that contain the source code of each of the program modules. The modules aim to be independent from each other; that is, a function of one module will never calls a function of another module (with a few exceptions).

- `neut/`

  '`neut`' stands for *Neper utilities*. The directory contains utility functions specific to Neper and used by several modules.

- `contrib/`

  This directory contains utility functions not specific to Neper. The first one is '`ut`', which is a collection of general-purpose, low-level C functions (memory allocation, etc.). The second one is '`orilib`', which is a collection of routines for orientation manipulation (see `http://orilib.sourceforge.net`). The third one is nanoflann, a library for nearest neighbour searching (see `https://github.com/jlblancoc/nanoflann`), the fourth one is openGJK, a library for fast point-simplex distance computations (`https://github.com/MattiaMontanari/openGJK`), and the last one is muparser, a library for interprating mathematical and logical expressions (see `http://beltoforion.de/article.php?a=muparser`). Although these libraries are also distributed alone (and might be already installed on your system), they are included into Neper instead of being considered as dependencies (contrary to the GSL, ...) to make installation easier.

- `CMakeLists.txt`, `neper_config.h.in` and `cmake/`

  These files and directories are specific to the building system, CMake. `CMakeLists.txt` is the CMake source file, which tells CMake where to find the program source files, how to manage dependencies, where to install Neper, etc. `neper_config.h.in` is a small configuration file that is useful to CMake for managing dependencies and program version numbers.

`cmake/` contains `.cmake` files which help CMake locating the dependencies on the system (library and header files).

A module directory, `neper_X/`, where '*X*' stands for the module letter (one of '`t`', '`m`' or '`v`'), is structured as follows,

- `neper_X.h`, `neper_X_.h` and `neper_X.c`

  These are the source code header files and source code file of the module. `neper_X.c` contains the module function, '`neper_X`'. `neper_X_.h` is the source code header file, which is `#include`'ed in `neper_X.c` and contains a bunch of `#include`s to all necessary library header files. `neper_X.h` contains the prototype of the module function and is `#include`'ed in `neper_.h`. Hence, files `_.h` are local header files while files `.h` are header files `#include`'ed into a upper-level source code header file. This is true everywhere in the source code. Moreover, any function specific to module *X* is prefixed '`neX_`'.

- `neX_input/` and `structIn_X.h`

  The '`neX_input/`' directory contains functions for reading the value of the arguments passed to module *X* from the command line. The information are recorded into an '`IN_X`' C structure, which is declared in file '`structIn_X.h`'.

- `neX_foo/`, `neX_bar/`, etc.

  Each of these directories is associated to a specific task of the module and contains a function of the same name ('`neX_foo`', etc.) which is called from function '`neper_X`'. Each directory contains a directory tree structure.

- `CMakeLists.txt`

  This file tells CMake where to find the source files and how to manage dependencies in the module. It is used by the upper-level `CMakeLists.txt` file (there is no lower-level `CMakeLists.txt` file).

The `neut` directory is roughly structured as follows,

- `CMakeLists.txt`

  This file tells CMake where to find the source files and how to manage dependencies in the module. It is used by the upper-level `CMakeLists.txt` file (there is no lower-level `CMakeLists.txt` file).

- `neut.h`, `neut_t.h`, `neut_m.h` and `neut_v.h`

  These files are source code header files that `#include` header files of `neut` (which contain function prototypes) and are `#include`d in the modules. `neut.h` `#include`s all header files while the three others `#include` header files only necessary to the corresponding module (this speeds up compilation at development stage).

- `neut_structs/`

  This directory contains header files which defines all C structures used in the program.

- `neut_foo/`, `neut_bar/`, etc.

  Each of these directories contain functions specific to a particular C structure. For example, `neut_tess` contains functions relative to the '`TESS`' structure, which describes a tessellation.

C.1.2 Documentation

Neper's documentation is located in directory `doc/`. It is written in Texinfo, the GNU software documentation system. The documentation consists of a collection of `.texi` files (text files). The documentation may be compiled in PDF, info or html format by running `make pdf`, `make info` or `make html`, respectively. In official releases, both the PDF and info documentation files are built and included in the archive.

C.2 Contributing to Neper

The Neper development repository is hosted on GitHub: `https://github.com/rquey/neper`. Code contributions to be included in Neper's official (public) version should be submitted as pull requests on this repository.

C.2.1 Coding Conventions

Neper is written following the GNU Coding Standards (`http://www.gnu.org/prep/standards`), with the exception that braces are not indented (because there is so often 3+ loop levels in Neper). Please follow this convention. Here are a few tips and other remarks,

- For Vim, put the following commands in file `$HOME/.vimrc`:

```
:set sw=2
:set cindent
:syntax enable
:set textwidth=72
```

- You can run `indent -bli0` *`source_files`* for automatic formatting.
- Break up the code into meaningful chunks using blank lines. Always use a single blank line to separate parts of the code.
- Neper admits no compilation warnings. Please fix up all of them.
- Please help us maintaining good documentation by documenting any capability you may add.

C.2.2 Adding a New Option

In modules -T and -M, adding a new option can be done by following these steps:

- Add a variable to the '`IN_X`' structure to record the value of the option (file `structIn_X.h`).
- If necessary, allocate / free the variable in the `neX_in_set_zero` and `neX_in_free` functions (file `neX_input1.c`) Assign it a default value in `net_input_options_default` (file `neX_input3.c`).
- Add the option to the option list in `net_input_options_set` (file `neX_input3.c`), taking as an example another option of the same type (integer, etc.).
- Where appropriate in the source code, add a new function for the new option (if necessary in a new file or directory). The function should be executed depending on the value of the option.
- If adding one or several files or directories, add the source file(s) to the source file list in the `CMakeLists.txt` file of the corresponding module.
- Make sure the whole thing compiles and runs properly for your purpose.
- Make sure your own changes did not break anything in the rest of the code by running full testing, using `make test` (or `ctest`). You may also want to add a test specific to the new option.

In module -V, options are processed differently. Instead of being recorded in a C structure, they are read one after the other and associated functions are executed along the way. To add a new option, take an existing option as an example.

C.2.3 Compilation Options

For development, several compilation options can be changed from their default values. This must be done at configuration stage, using commands '`ccmake ..`' or '`cmake-gui ..`'. The compilation options are

- `DEVEL_DEBUGGING_FLAG`

  Setting the option to `ON` turns on the debugging compilation flag '`-g`', which is required for debugging with `gdb` and `valgrind`, and turns on the compilation flag '`-Werror`', which makes all compilation warnings into errors.

- `DEVEL_DEBUGGING_TEST`

  Setting the option to `ON` runs internal tests during Neper execution at places where the code is otherwise considered robust.

- `DEVEL_OPTIMIZATION`

  Setting this option to `OFF` disables code optimization, which is useful for debugging with `gdb` and `valgrind`.

- `DEVEL_PROFILING`

  Setting this option to `ON` turns on the code profiling compilation flag '`-pg`', which is required for profiling with `gprof`. This is a highy CPU-sensitive option, which should be used only when profiling is actually carried out.

C.2.4 Testing

The code can be tested using `CTest`. The usual way is as follows:

```
$ ctest
```

It is also possible to run only some of the tests.

- Option `-R` selects the tests whose name contains a character string:

  ```
  $ ctest -R string
  ```

  As test names start by the letter of the module they refer to, followed by character '`_`', it becomes handy to run tests on a specific module, for example:

  ```
  $ ctest -R T_
  ```

- Option `-E` selects the tests that do not contain a character string:

  ```
  $ ctest -E string
  ```

- Option `-I` selects the tests from their numbers, for example:

  ```
  $ ctest -I 3,5
  ```

Appendix D Versions

New in 3.5.1 (15 Nov 2019):
- module -T: fixed -morpho from wide grain size distributions, sped up -morpho tesr using openGJK, added olmap format.
- module -M: added -elt quad9, added -transform rotate, added fasets to .msh file, fixed -elt hex for non-cubic tessellations.
- module -V: fixed scale labels.
- general: fixed compilation without openMP.

New in 3.5.0 (01 Oct 2019):
- module -T: added -transform slice and oriaverage, added new statistical distributions to -morpho: beta, lorentzian, studentst, weibull, breitwigner, expnormal, moffat, pearson7, pseudovoigt and skewnormal, added endpoints to statistical distributions, added variable 1-sphericity to -morpho, added tessellation keys, added -orioptilogvar, improved argument reading (multiscale tessellation), improved tesr format, improved -transform renumber, improved -ori fibre, improved -statcell convexity, replaced -transform plane with -transform hspace, fixed -loadtesr file.tesr:..., fixed -statvox e, fixed -for ori, fixed -domain, fixed -morpho centroidal, fixed tesr reading, fixed vtk output, enrich tesr file, made minor fixes.
- module -M: added -transform scale, translate and smooth, added -transporteltmethod, added -transportfepx, added mesh keys, added -performat, added kocks file to -for fepx:legacy, fixed -interface cohesive for Abaqus, fixed -elt hex, merged -scale into -transform, speed up -transport, made minor fixes.
- module -V: added -showvox, added -datavoxcol disori, added colour schemes R(), r, theta and rtheta, added -imageformat pov:objects, improved -includepov, fixed visualization of periodic meshes, made small fixes.
- documentation: made minor fixes and improvements.
- general: fixed compilation when GSL and NLopt locally installed, website now at http://neper.info.

New in 3.4.0 (15 Apr 2019):
- module -T: added tessellation cutting (to get non-convex domains) using -transform cut, added cell merging and removal using -transform mergecell and rmcell, added -transform resetcellid, added -ori spread, improved -morpho lamellar, improved -morpho diameq, added -format stl:bycell, added tessellation keys, fixed -morpho aspratio and rasterization of periodic tessellations, fixed -domain circle:split, made small fixes and improvements.
- module -M: made -interface mutliscale, renamed -mesh3dclconv into -mesh3dclreps, improved -cl, -rcl and -mesh3dclreps argument expressions, speed up -transport, fixed 2D remeshing, speed up -statelt and -statelset, added -[r]clface, -[r]cledge and -[r]clver.
- module -V: improved visualization of non-convex cells, fixed visualization of hex meshes, minor other fixes.

New in 3.3.0 (25 Jul 2018):
- New paper: "R. Quey, A. Villani, and C. Maurice. Nearly uniform sampling of crystal orientations. J. Appl. Crystallogr., vol. 51, doi:10.1107/S1600576718009019, 2018."

```
- module -T: added -ori uniform for uniform orientation distribution,
  added sister options -orioptiini, -orioptifix, -orioptistop and
  -orioptineigh, added -morpho tocta, columnar, bamboo and aspratio,
  fixed -morpho multimodal, improved -morphooptiini, added -transform
  translate, fixed tesr output for periodic tessellations, fixed tess to
  tesr conversion, added several tessellation keys, merged option -scale
  into -transform, made minor improvements.
- module -M: fixed meshing of periodic tessellation with openMP, allowed
  Gmsh development versions.
- module -V: improved camera parameters, fixed -slicemesh for hex
  meshes.

New in 3.2.0 (23 May 2018):
- general:  Neper is now fully multithreaded.
- module -T: improved -morpho, added -morpho variable:value, added -morpho
  centroidtol, improved -morphooptiobjective, improved -transform crop, fixed
  -n from_morpho, added -transform addbuffer, renamed -datarpt* into -datavox*,
  added -transform renumber, added -domain stdtriangle, added **oridata to the
  tesr file, added -statvox, speed up tesr generation, added -morphooptilogtime
  val_ keys, minor fixes.
- module -M: improved inp format.
- module -V: improved msh reading, added -datapointrad arr, added -datavox
  options.
- documentation:  small fixes.

New in 3.1.1 (20 Feb 2018):
- New paper:  "R. Quey and L. Renversade, Optimal polyhedral description of 3D
  polycrystals:  Method and application to statistical and synchrotron X-ray
  diffraction data, Comput.  Methods Appl.  Mech.  Engrg., vol.  330, pp.  308-333,
  2018."
- module -T: fixed -scale, added -domain rodrigues.
- module -M: fixed mesh of regularized tessellation through 2D-mesh pinches
  correction, fixed -stat when a tesr is meshed, improved -loadmesh, added
  -mesh2dpinchfix, added test on Gmsh's version.
- module -V : fixed -datapolyscale.
- general:  fixed use without nlopt or openmp.

New in 3.1.0 (05 Sep 2017):
- module -T: made all options applicable to multiscale tessellations,
  improved -morpho for all types of data, improved -morpho*
  options, added -n from_morpho to set the number of cells from all
  -morpho inputs, added -format stl, added -transform crop and 2d, added
  -morphooptilogtesr, added tessellation key 'scale', included openMP
  parallelization for -morpho tesr.
- module -M: allowed -interface cohesive -periodic in 2D, improved -dim
  for -format geof, improved -format geof, minor fixes.
- module -V: added -datapointrad tor and disc.
- documentation:  small fixes.
- general:  support OpenMP.

New in 3.0.2 (16 Feb 2017):
- module -T: added -transform scale and transform rotate for scalar
```

```
  tessellations (.tess), fixed -morpho with -periodic, fixed -morpho
  with non-unitary domains, fixed -statedge/-statface theta with
  -oricrysym hexagonal, improved -ori, added -morpho
  lamellar(v=crysdir), added ***tess/**cell/*crysym section to .tess
  file, fixed -morphooptialgo random, small fixes.
- module -M: fixed -interface cohesive in 2D and 3D (for Abaqus), enable
  -interface discontinuous and cohesive for periodic tessellations,
  added -format msh:binary, fixed faset internal for 3D, added
  -transport node:..., changed point keys "elt3d" and "elset3d" into "elt"
  and "elset" (now apply to 3D or 2D), small fixes.
- documentation:  minor fixes.

New in 3.0.1 (01 Dec 2016):
- module -T: fixed up options -periodic and -morphooptialgoneigh qsort,
  added tessellation key 'modeid' (for cells), added -morpho
  cube(N1,N2,N3) and -morpho square(N1,N2), enabled -loadtess -scale,
  added -oricrysym hexagonal, added tessellation keys 'nfacelist' and
  'theta', cleaned code, fixed documentation of .tess file, added tests
  on conflicing options.
- module -M: added option -scale, fixed hexahedral meshing of periodic
  tessellations, added physical groups to .msh, fixed 2D meshes in
  Abaqus (.inp) file, fixed up option -faset internal.
- module -V: fixed options -dataseed*.
- general:  removed libmatheval dependency, fixed testing.

New in 3.0.0 (12 Sep 2016):
- module -T: added 3 major capabilities:  (i) tessellation generation
  from morphological cell properties (options starting by -morpho),
  (ii) multiscale tessellation generation (using the ::  separator), and
  (iii) periodic and semi-periodic tessellation generation (option
  -periodic); made some other improvements and some clean up all over
  the place.
- module -M: added ability to mesh the new tessellations, including
  periodic tessellations; added interface meshing using cohesive
  elements (option -interface); made small other improvements.
- module -V: made small improvements.
- module -D: replaced by "make test" using CTest.
- new development website:  https://github.com/rquey/neper.

New in 2.0.5 (06 Feb 2016):
- module -T: fixed up -domain planes in 3D, added -domain sphere, added
  'rotate', 'translate' and 'scale' arguments to -domain, added normal
  specification in -morpho lamella, minor other improvements.
- module -M: fixed up vtk output.
- module -V: added -datacellcol id:filename, fixed up options -data*col
  id.
- module -D: minor improvements.

New in 2.0.4 (22 Jun 2015):
- module -T: fixed up regularization in 2D, extended -morpho planes to
  2D, added semi-periodicity for raster tessellations, minor fixes.
- module -M: fixed up 2D mesh output in Abaqus format, minor fixes.
```

```
- module -V: fixed up -datacellcolscheme, improved -cameracoo to account
  for the tessellation/mesh dimensions.

New in 2.0.3 (27 Nov 2014):
- module -M: fixed up bug on Mac OS X, fixed up Ctrl-C, fixed up and
  speed up option -statpoint 2dmesh*.
- module -T: improved options for 2-scale tessellations, added option
  '-clip'.

New in 2.0.2 (29 Sep 2014):
- module -T: fixed up regularization of cylinder tessellations, fixed up
  option '-domain planes', added tessellation cell domain, fixed up 3dec
  and ply support, added Wavefront obj format, added / fixed up
  tessellation keys.
- module -M: added vtk mesh format, fixed up fepx and geof mesh formats,
  added extrusion of a 2D mesh to get a 3D mesh (option -dim), fixed up
  topology reconstruction.
- module -V: added points plotting as cubes, spheres, cylinders or
  ellipsoids (options -showpoint and -datapoint*).

New in 2.0.1 (12 Mar 2014):
- Fixed up compilation on some systems, added support for libscotch
  version 6.0, small fixes and cleanups.
- module -T: enabled square and cube tessellations in .tess format,
  fixed up cell sorting, made option -id mandatory, improved
  regularization of 2D tessellations, added bunch of tessellation
  keys, small fixes.
- module -M: added 'domtype' mesh key.
- module -V: fixed up simultaneous tess and mesh printing, fixed up
  colouring based on id, improved camera positioning for 2D and 1D
  inputs, added coordinate system, improved option -slicemesh, added
  options -data*scaletitle, improved -data*scale options.

New in 2.0.0 (10 Jan 2014):
- General: Full restructuring and added many new features. Neper now
  has 3 main modules: tessellation module (-T), meshing module (-M) and
  visualization module (-V); details are provided below. Added
  developer's guide and module (-D). Documentation has been much
  improved.
- module -T: added several tessellation algorithms (hardcore Voronoi
  and Laguerre Voronoi); added orientation generation (was previously in
  -O); significantly sped up tessellation; included and significantly
  sped up regularization (was previously in -FM); added 2-scale
  polycrystal generation; added 2D and 1D supports; improved statistics;
  enabled both scalar (tess) and raster (tesr) outputs; cleaned up tess
  file.
- module -M: module for free and mapped meshings (merging of -FM and
  -MM). Removed regularization (now in -T); added per-cell mesh size
  definition; sped up multimeshing; improved statistics.
- module -V: full restructuring; added support for 2D and 1D
  tessellations and meshes; the way all entities are shown (cells,
  polyhedra, faces, edges, vertices, germs, 3D/2D/1D/0D element sets and
```

```
  elements, nodes) can be set in great detail; added transparency.

New in 1.10.3 (26 Nov 2012):
- module -T: added 3dec geometry format, added option -checktess, minor
  improvements, added individual file extension support in -stattess,
  changed option -neigh 1 to -statp i,f,npl,fal,feql.
- module -FM: added 3dec geometry format; changed "top" and "bot" nset
  names for cylindrical domains to "z0" and "z1"; minor bug fixes;
  improved fev format support; added individual file extension support in
  options -stattess and -statmesh.
- module -O:  minor bug fixes.
- module -MM: sped up meshing; fixed -domain, -scale and -nset options,
  add .nper file for periodicity conditions; fixed msh output for meshes
  with different element dimensions; minor other bug fixes.
- module -VS: sped up meshing reconstruction and PNG file generation,
  added option '-camerasky', added option '-showeltedge', sped up mesh
  reconstruction, minor fixes
- documentation:  minor fixes.
- General:  minor fixes.

New in 1.10.2 (09 Aug 2012):
- module -T: fixed -centroid option.
- module -FM: fixed list of available meshing algorithms.    Added
  tests.
- module -MM: fixed nset syntax in inp (Abaqus) files.
- module -VS: added capability to plot mapped meshes.
- General:  various minor improvements, code cleaning.

New in 1.10.1 (08 June 2012):
- Bug fix to get Neper working after invoquing `make install'.

New in 1.10.0 (04 June 2012):
- General:  New (hopefully simpler) installation procedure based on Cmake.
  Added support for domains of any convex polyhedral shape.
- module -VS: major code rewriting and option changes.  New capabilities
  for defining the colours and sizes of the tessellation / mesh
  (including gradients).    Added options to show only specific parts of the
  tessellation / mesh and to view slices of a mesh.  Other small
  enhancements.
- module -T : added option `-domain' to specify the shape of the domain
  (cuboidal, cylindrical or of any convex shape), small bug fixes, added
  centroid Voronoi tessellation generation (option -centroid), merged
  option -centrecoo into option -morpho, added polyhedron centroid
  coordinates in file .stt3, changed option -load to -loadtess, added
  output format `.ply' (thanks Ehsan!).
- module -FM: mesh partitionning needs libscotch version 5.1.12 or later,
  small bug fixes, changed default value of -faset to "" (i.e.  no faset
  in output), fixed bug for Abaqus output, added polyhedron centroid
  coordinates in file .stt3, added output format `.ply' (geometry only).
- module -MM: new options -dsize and -scale, new option -loadmesh, new
  option -outdim, changed arguments of -ttype, changed default value of
  -faset to "" (i.e.  no faset in output), fixed bug for Abaqus output,
```

```
  small bug fixes.

New in 1.9.2 (Sep 2011):
- module -T: added option -morpho for specifying the type of grain
  structure (equiaxed, columnar or bamboo), merged option -regular
  with -morpho, added post-processing -neighbour option for information
  on the polyhedron neighbours, added geo (Gmsh geometry) output format
  (mostly for visualization), fixed bugs.
- module -MM: proper processing of the input tess files, added msh
  (Gmsh) and inp (Abaqus) output formats, added options -morpho and
  -centrecoo (as in module -T), small bug fixes, code cleaning.
- module -FM: added geo (Gmsh geometry) output format (mostly for
  visualization), small bug fixes.
- documentation:  small corrections.

New in 1.9.1 (May 2011):
- module -FM: fixed bug occurring when -mesh3dalgo is not set by the
  user.    Small other bug fixes.
- module -MM: small bug fixes.

New in 1.9.0 (Apr 2011):
This is a major release.    Neper now has its own paper:
"R.Quey, P.R. Dawson and F. Barbe.    Large-scale 3D random polycrystal
 for the finite element method:  Generation, meshing and remeshing.
 Computer Methods in Applied Mechanics and Engineering,
 Vol.  200, pp.  1729--1745, 2011."
Please cite it in your works if you use Neper.
- General:  added option --rcfile to disregard / change the
  initialization file; big distribution and source clean up; bug fixes.
- module -T: added capability to generate regular morphologies
  (truncated octahedra), tess file format bumped to 1.9; big clean up.
- module -FM: included multimeshing, remeshing and mesh partitioning
  capabilities; big clean up.    Neper now uses the *standard* Gmsh
  distribution for 2D and 3D meshings (versions >=2.4.2).    Strongly
  reduced memory usage.
- module -O: added capability to handle different orientation
  descriptors.
- module -VS: new visualization module to generate publication-quality
  images (PNG format) of the tessellations, meshes and more...

New in 1.8.1 (Aug 2009):
- upgraded website at http://neper.sourceforge.net
- module -T: new file format ***tess1.8, new option -restart to load an
  existing tessellation (not through std input any more), new option
  -printformat, bug fixes.
- module -MM: bug fixes.
- module -FM: new output format mae, new option -restart to restart from
  an existing geometry or mesh (options -mesh and -conv removed); new
  options -printformat and -maeextension; better mesh numbering (+ new
  options -elementfirstid and -nodefirstid), new way to choose the node
  sets to output (-nset 4), fixed option -estat, renamed -bwcy-clmin to
  -clmin, cleaned bunch of options, bug fixes.
```

```
- module -O: added option -eulerangleconvention (Bunge, Roe & Kocks);
  new output formats mae and geof (option -format).
- manual:  some corrections.

New in 1.8.0 (Jul 2009):
- First GPL-distributed version of Neper.
```

Appendix E GNU General Public License

GNU General Public License

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a devel copy. Propagation includes copying, distribution (with or without modification), making available to the distrib, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the distrib in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms

that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general distrib at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is distribly documented (and with an implementation available to the distrib in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for distribity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a distribly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

    If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's distrib statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRIT-

ING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the distrib, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
```

```
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

Option Index