Gmsh

Gmsh Reference Manual

The documentation for Gmsh 1.56
A finite element mesh generator with built-in pre- and post-processing facilities

Edition 1.27 (18 November 2004)

Christophe Geuzaine
Jean-François Remacle

Short Contents

Table of Contents

Copying conditions

Gmsh is "free software"; this means that everyone is free to use it and to redistribute it on a free basis. Gmsh 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 Gmsh that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of Gmsh, that you receive source code or else can get it if you want it, that you can change Gmsh or use pieces of Gmsh 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 Gmsh, 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 Gmsh. If Gmsh 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 Gmsh are found in the General Public License that accompanies the source code (see Appendix C [License], page 161). Further information about this license is available from the GNU Project webpage `http://www.gnu.org/copyleft/gpl-faq.html`. Detailed copyright information can be found in Section 11.3 [Credits], page 147.

The source code and various pre-compiled versions of Gmsh (for Unix, Windows and Mac OS) can be downloaded from the webpage `http://www.geuz.org/gmsh/`.

If you use Gmsh, we would appreciate that you mention it in your work. References, as well as the latest news about Gmsh development, are always available on `http://www.geuz.org/gmsh/`. Please send all Gmsh-related questions to the public Gmsh mailing list at `gmsh@geuz.org`.

If you want to integrate Gmsh into a closed-source software, or want to sell a modified closed-source version of Gmsh, please contact one of the authors. You can purchase a version of Gmsh under a different license, with "no strings attached" (for example allowing you to take parts of Gmsh and integrate them into your own proprietary code).

1 Overview

Gmsh is an automatic three-dimensional finite element mesh generator, primarily Delaunay, with built-in pre- and post-processing facilities. Its design goal is to provide a simple meshing tool for academic problems with parametric input and up to date visualization capabilities. One of its strengths is the ability to respect a characteristic length field for the generation of adapted meshes on lines, surfaces and volumes, and to mix these meshes with simple structured (transfinite, extruded, etc.) grids.

Gmsh is built around four modules: geometry, mesh, solver and post-processing. All geometrical, mesh, solver and post-processing instructions are prescribed either interactively using the graphical user interface (GUI) or in ASCII data files using Gmsh's own scripting language. Interactive actions generate language bits in the input files, and vice versa. This makes it possible to automate all treatments, using loops, conditionals and external system calls. A brief description of the four modules is given hereafter.

1.1 Geometry: geometrical entity definition

Geometries are created in a bottom-up flow by successively defining points, oriented lines (line segments, circles, ellipses, splines, ...), oriented surfaces (plane surfaces, ruled surfaces, ...) and volumes. Compound groups of geometrical entities can be defined, based on these elementary geometric entities. Gmsh's scripting language allows all geometrical entities to be fully parameterized.

1.2 Mesh: finite element mesh generation

A finite element mesh is a tessellation of a given subset of the three-dimensional space by elementary geometrical elements of various shapes (in Gmsh's case: lines, triangles, quadrangles, tetrahedra, prisms, hexahedra and pyramids), arranged in such a way that if two of them intersect, they do so along a face, an edge or a node, and never otherwise. All the finite element meshes produced by Gmsh are considered as "unstructured", even if they were generated in a "structured" way (e.g., by extrusion). This implies that the elementary geometrical elements are defined only by an ordered list of their nodes but that no predefined order relation is assumed between any two elements.

The mesh generation is performed in the same bottom-up flow as the geometry creation: lines are discretized first; the mesh of the lines is then used to mesh the surfaces; then the mesh of the surfaces is used to mesh the volumes. In this process, the mesh of an entity is only constrained by the mesh of its boundary[1]. This automatically assures the

[1] For example, in three dimensions:

- the triangles discretizing a surface will be forced to be faces of tetrahedra in the final 3D mesh only if the surface is part of the boundary of a volume;
- the line elements discretizing a curve will be forced to be edges of tetrahedra in the final 3D mesh only if the curve is part of the boundary of a surface, itself part of the boundary of a volume;
- a single node discretizing a point in the middle of a volume will be forced to be a vertex of one of the tetrahedra in the final 3D mesh only if this point is connected to a curve, itself part of the boundary of a surface, itself part of the boundary of a volume...

conformity of the mesh when, for example, two surfaces share a common line. But this also implies that the discretization of an "isolated" (n-1)-th dimensional entity inside an n-th dimensional entity does *not* constrain the n-th dimensional mesh. Every meshing step is constrained by the characteristic length field, which can be uniform, specified by characteristic lengths associated with elementary geometrical entities, or associated with another mesh (the background mesh).

For each meshing step, all structured mesh directives are executed first, and serve as additional constraints for the unstructured parts. The implemented Delaunay algorithm is subdivided in the following five steps for surface/volume discretization:

1. trivial meshing of a box including the convex polygon/polyhedron defined by the boundary nodes resulting from the discretization of the lines/surfaces;
2. creation of the initial mesh by insertion of all the nodes on the lines/surfaces thanks to the Bowyer algorithm;
3. boundary restoration to force all the edges/faces of the lines/surfaces to be present in the initial mesh;
4. suppression of all the unwanted triangles/tetrahedra (in particular those containing the nodes of the initial box);
5. insertion of new nodes by the Bowyer algorithm until the characteristic size of each simplex is lower or equal to the characteristic length field evaluated at the center of its circumscribed circle/sphere.

1.3 Solver: external solver interface

External solvers can be interfaced with Gmsh through Unix sockets, which permits to easily launch external computations and to collect and exploit the simulation results within Gmsh's post-processing module. The default solver interfaced with Gmsh is GetDP (http://www.geuz.org/getdp/).

1.4 Post-processing: scalar, vector and tensor field visualization

Multiple post-processing scalar, vector or tensor maps can be loaded and manipulated (globally or individually) along with the geometry and the mesh. Scalar fields are represented by iso-value lines/surfaces or color maps, while vector fields are represented by three-dimensional arrows or displacement maps. Post-processing functions include section computation, offset, elevation, boundary and component extraction, color map and range modification, animation, vector graphic output, etc. All post-processing options can be accessed either interactively or through the input ASCII text files. Scripting permits to automate all post-processing operations, e.g., for the creation of animations. User-defined operations can also be performed on post-processing views through dynamically loadable plugins.

1.5 What Gmsh is pretty good at . . .

Gmsh is a (relatively) small program, and was principally developed "in academia, to solve academic problems". . . Nevertheless, over the years, many people outside universities have found Gmsh useful in their day-to-day jobs. Here is a tentative list of what Gmsh does best:

- quickly describe simple and/or "repetitive" geometries, thanks to user-defined functions, loops, conditionals and includes (see Section 2.4 [User-defined functions], page 14, Section 2.5 [Loops and conditionals], page 14, and Section 2.6 [General commands], page 15);
- parameterize these geometries. Gmsh's scripting language enables all commands and command arguments to depend on previous calculations (see Section 2.1 [Expressions], page 9, and Section 3.1 [Geometry commands], page 35);
- generate 1D, 2D and 3D simplicial (i.e., using line segments, triangles and tetrahedra) finite element meshes. The performance of the 1D and 2D algorithms is pretty good; the 3D algorithm is still experimental and slow (see Chapter 4 [Mesh module], page 45, and Chapter 7 [Tutorial], page 95);
- specify target element sizes accurately. Gmsh provides several mechanisms to control the size of the elements in the final mesh: through interpolation from geometrical point characteristic lengths or geometrical attractors, or from user-defined background meshes (see Section 4.2 [Mesh commands], page 45);
- create simple extruded geometries and meshes (see Section 3.1 [Geometry commands], page 35, and Section 4.2 [Mesh commands], page 45);
- interact with external solvers. Gmsh provides C/C++ and Perl interfaces, and others can be easily added (see Chapter 5 [Solver module], page 59);
- visualize computational results in a great variety of ways. Gmsh can display scalar, vector and tensor data sets, and can perform various operations on the resulting post-processing views (see Chapter 6 [Post-processing module], page 79);
- export plots in many different formats: vector PostScript or encapsulated PostScript, LaTeX, PNG, JPEG, . . . (see Section 2.7 [General options], page 18);
- generate complex animations (see Chapter 2 [General tools], page 9, and Section 7.8 [t8.geo], page 114);
- run on low end machines and/or machines with no graphic system. Gmsh can be compiled with or without the graphical user interface, and all versions can be used either interactively or not, directly from the command line (see Chapter 8 [Running Gmsh], page 119);
- configure your preferred options. Gmsh has a large number of configuration options that can be set interactively using the GUI, scattered inside command files, changed on the fly in scripts, set in per-user configuration files, or specified on the command-line (see Section 2.7 [General options], page 18, Section 3.2 [Geometry options], page 40, Section 4.3 [Mesh options], page 49, Section 6.3 [Post-processing options], page 86, and Chapter 8 [Running Gmsh], page 119);
- and do all the above on various platforms (Windows, Mac and Unix), for free (see [Copying conditions], page 1), using clear-text ASCII files and/or a small but powerful graphical user interface.

1.6 . . . and what Gmsh is not so good at

Due to its historical background and limited developer manpower, Gmsh has also some (a lot of?) weaknesses:

- the bottom-up approach for describing geometries can become inconvenient for complex models;
- there is no support for NURBS and only very limited support for trimmed surfaces;
- Gmsh is not primarily a structured mesh generator: no automatic quadrilateral or hexahedral meshing algorithm is provided. If you want quadrangles, you have to use transfinite, elliptic or extruded meshes or recombine unstructured triangular meshes. For hexahedra, your only choice is transfinite or extruded meshes;
- Gmsh is not a multi-bloc generator: all meshes produced by Gmsh are conforming in the sense of finite element meshes;
- the 2D anisotropic and the 3D unstructured algorithms are still experimental and not very robust. If these algorithms fail, try to change some characteristic lengths to generate meshes that better suit the geometrical details of the structures;
- Gmsh was designed to solve academic "test cases", not industrial-size problems. You may find that Gmsh is too slow for large problems (with thousands of geometric primitives, or millions of mesh/post-processing elements).

If you have the skills and some free time, feel free to join the project! We gladly accept any code contributions (see Chapter 10 [Programming notes], page 137) to remedy the aforementioned (and all other) shortcommings...

1.7 Syntactic rules used in this document

Here are the rules we tried to follow when writing this user's guide. Note that metasyntactic variable definitions stay valid throughout the manual (and not only in the sections where the definitions appear).

1. Keywords and literal symbols are printed like `this`.
2. Metasyntactic variables (i.e., text bits that are not part of the syntax, but stand for other text bits) are printed like *this*.
3. A colon (`:`) after a metasyntactic variable separates the variable from its definition.
4. Optional rules are enclosed in `< >` pairs.
5. Multiple choices are separated by `|`.
6. Three dots (. . .) indicate a possible (multiple) repetition of the preceding rule.

1.8 Comments

All Gmsh ASCII text input files support both C and C++ style comments:

1. any text comprised between `/*` and `*/` pairs is ignored;
2. the rest of a line after a double slash `//` is ignored.

These commands won't have the described effects inside double quotes or inside keywords. Also note that 'white space' (spaces, tabs, new line characters) is ignored inside all expressions.

2 General tools

This chapter describes the general commands and options that can be used in Gmsh's ASCII text input files. By "general", we mean "not specifically related to one of the geometry, mesh, solver or post-processing modules". Commands peculiar to these modules will be introduced in Chapter 3 [Geometry module], page 35, Chapter 4 [Mesh module], page 45, Chapter 5 [Solver module], page 59, and Chapter 6 [Post-processing module], page 79, respectively.

Note that, if you are just beginning to use Gmsh, or just want to see what Gmsh is all about, you really don't need to read this chapter and the four next ones. Just have a quick look at Chapter 8 [Running Gmsh], page 119, and go play with the graphical user interface, running the tutorials and demonstration files bundled in the distribution! Most of the commands and options described in the following chapters are available interactively in the GUI, so you don't need to worry about Gmsh's internals for creating your first geometries, meshes and post-processing plots. Once you master the tutorial (read the source files: they are heavily commented—see Chapter 7 [Tutorial], page 95), you might want to come back here to learn more about the specific syntax of Gmsh's commands and esoteric options.

2.1 Expressions

The two constant types used in Gmsh are *real* and *string* (there is no integer type). These types have the same meaning and syntax as in the C or C++ programming languages.

2.1.1 Floating point expressions

Floating point expressions (or, more simply, "expressions") are denoted by the metasyntactic variable *expression* (remember the definition of the syntactic rules in Section 1.7 [Syntactic rules], page 6), and are evaluated during the parsing of the data file:

expression:
  real |
  string |
  string `[` *expression* `]` |
  `#` *string* `[ ]` |
  `(` *expression* `)` |
  operator-unary-left expression |
  expression operator-unary-right |
  expression operator-binary expression |
  expression operator-ternary-left expression operator-ternary-right expression |
  built-in-function |
  real-option

Such *expressions* are used in most of Gmsh's commands. The operators *operator-unary-left*, *operator-unary-right*, *operator-binary*, *operator-ternary-left* and *operator-ternary-right* are defined in Section 2.2 [Operators], page 11. For the definition of *built-in-functions*, see Section 2.3 [Built-in functions], page 13. The various *real-options* are listed in Section 2.7 [General options], page 18, Section 3.2 [Geometry options], page 40, Section 4.3 [Mesh

options], page 49, Section 5.1 [Solver options], page 59, and Section 6.3 [Post-processing options], page 86.

List of expressions are also widely used, and are defined as:

expression-list:
  expression-list-item `<,` *expression-list-item*`>` ...

with

expression-list-item:
  expression `|`
  expression `:` *expression* `|`
  expression `:` *expression* `:` *expression* `|`
  string `[ ] |`
  string `[ {` *expression-list* `} ] |`
  `Point {` *expression* `} |`
  transform `|`
  extrude

The second case in this last definition permits to create a list containing the range of numbers comprised between two *expression*s, with a unit incrementation step. The third case also permits to create a list containing the range of numbers comprised between two *expression*s, but with a positive or negative incrementation step equal to the third *expression*. The fourth case permits to reference an expression list. The fifth case permits to reference an expression sublist (whose elements are those corresponding to the indices provided by the *expression-list*). The sixth case permits to retrieve the coordinates of a given geometry point (see Section 3.1.1 [Points], page 35). The last two cases permit to retreive the indices of entities created through geometrical transformations adn extrusions (see Section 3.1.6 [Transformations], page 39, and Section 3.1.5 [Extrusions], page 38).

To see the practical use of such expressions, have a look at the first couple of examples in Chapter 7 [Tutorial], page 95. Note that, in order to lighten the syntax, you can always omit the braces `{}` enclosing an *expression-list* if this *expression-list* only contains a single item. Also note that a braced *expression-list* can be preceded by a minus sign in order to change the sign of all the *expression-list-item*s.

2.1.2 Character expressions

Character expressions are defined as:

char-expression:
  `"`*string*`"` `|`
  `StrPrefix (` *char-expression* `) |`
  `StrCat (` *char-expression* `,` *char-expression* `) |`
  `Sprintf (` *char-expression* `,` *expression-list* `) |`
  char-option

The second case in this definition permits to take the prefix of a string (e.g., for removing the extension from a file name). The third case permits to concatenate two character expressions, and the fourth is an equivalent of the `sprintf` C function (where *char-expression* is a format string that can contain floating point formatting characters: `%e`, `%g`, etc.). The last case permits to use the value of a *char-option* as a *char-expression*. The various *char-option*s

are listed in Section 2.7 [General options], page 18, Section 3.2 [Geometry options], page 40, Section 4.3 [Mesh options], page 49, Section 5.1 [Solver options], page 59, and Section 6.3 [Post-processing options], page 86.

Character expressions are mostly used to specify non-numeric options and input/output file names. See Section 7.8 [t8.geo], page 114, for an interesting usage of *char-expression*s in an animation script.

2.1.3 Color expressions

Colors expressions are hybrids between fixed-length braced *expression-list*s and *string*s:

color-expression:
  string |
  `{` *expression*`,` *expression*`,` *expression* `}` |
  `{` *expression*`,` *expression*`,` *expression*`,` *expression* `}` |
  color-option

The first case permits to use the X Windows names to refer to colors, e.g., `Red`, `SpringGreen`, `LavenderBlush3`, ... (see '`Common/Colors.h`' in Gmsh's source tree for a complete list). The second case permits to define colors by using three expressions to specify their red, green and blue components (with values comprised between 0 and 255). The third case permits to define colors by using their red, green and blue color components as well as their alpha channel. The last case permits to use the value of a *color-option* as a *color-expression*. The various *color-option*s are listed in Section 2.7 [General options], page 18, Section 3.2 [Geometry options], page 40, Section 4.3 [Mesh options], page 49, Section 5.1 [Solver options], page 59, and Section 6.3 [Post-processing options], page 86.

See Section 7.3 [t3.geo], page 99, for an example of the use of color expressions.

2.2 Operators

Gmsh's operators are similar to the corresponding operators in C and C++. Here is the list of the unary, binary and ternary operators currently implemented.

operator-unary-left:

`-` Unary minus.

`!` Logical not.

operator-unary-right:

`++` Post-incrementation.

`--` Post-decrementation.

operator-binary:

`^` Exponentiation.

`*` Multiplication.

`/` Division.

`%` Modulo.

`+` Addition.

`-` Subtraction.

`==` Equality.

`!=` Inequality.

`>` Greater.

`>=` Greater or equality.

`<` Less.

`<=` Less or equality.

`&&` Logical 'and'.

`||` Logical 'or'. (Warning: the logical 'or' always implies the evaluation of both arguments. That is, unlike in C or C++, the second operand of `||` is evaluated even if the first one is true).

operator-ternary-left:

`?`

operator-ternary-right:

`:` The only ternary operator, formed by *operator-ternary-left* and *operator-ternary-right*, returns the value of its second argument if the first argument is non-zero; otherwise it returns the value of its third argument.

The evaluation priorities are summarized below[1] (from stronger to weaker, i.e., `*` has a highest evaluation priority than `+`). Parentheses `()` may be used anywhere to change the order of evaluation:

1. `()`, `[]`, `.`, `#`
2. `^`
3. `!`, `++`, `--`, `-` (unary)
4. `*`, `/`, `%`
5. `+`, `-`
6. `<`, `>`, `<=`, `>=`
7. `==`, `!=`
8. `&&`
9. `||`
10. `?:`
11. `=`, `+=`, `-=`, `*=`, `/=`

[1] The affectation operators are introduced in Section 2.6 [General commands], page 15.

2.3 Built-in functions

A built-in function is composed of an identifier followed by a pair of parentheses containing an *expression-list* (the list of its arguments)[2]. Here is the list of the built-in functions currently implemented:

build-in-function:

`Acos ( expression )`
: Arc cosine (inverse cosine) of an *expression* in [-1,1]. Returns a value in [0,Pi].

`Asin ( expression )`
: Arc sine (inverse sine) of an *expression* in [-1,1]. Returns a value in [-Pi/2,Pi/2].

`Atan ( expression )`
: Arc tangent (inverse tangent) of *expression*. Returns a value in [-Pi/2,Pi/2].

`Atan2 ( expression, expression )`
: Arc tangent (inverse tangent) of the first *expression* divided by the second. Returns a value in [-Pi,Pi].

`Ceil ( expression )`
: Rounds *expression* up to the nearest integer.

`Cos ( expression )`
: Cosine of *expression*.

`Cosh ( expression )`
: Hyperbolic cosine of *expression*.

`Exp ( expression )`
: Returns the value of e (the base of natural logarithms) raised to the power of *expression*.

`Fabs ( expression )`
: Absolute value of *expression*.

`Fmod ( expression, expression )`
: Remainder of the division of the first *expression* by the second, with the sign of the first.

`Floor ( expression )`
: Rounds *expression* down to the nearest integer.

`Hypot ( expression, expression )`
: Returns the square root of the sum of the square of its two arguments.

`Log ( expression )`
: Natural logarithm of *expression* (*expression* > 0).

`Log10 ( expression )`
: Base 10 logarithm of *expression* (*expression* > 0).

`Modulo ( expression, expression )`
: see `Fmod( expression, expression )`.

[2] For compatibility with GetDP (http://www.geuz.org/getdp/), parentheses can be replaced by brackets `[]`.

`Rand (` *expression* `)`
: Random number between zero and *expression*.

`Sqrt (` *expression* `)`
: Square root of *expression* (*expression* >= 0).

`Sin (` *expression* `)`
: Sine of *expression*.

`Sinh (` *expression* `)`
: Hyperbolic sine of *expression*.

`Tan (` *expression* `)`
: Tangent of *expression*.

`Tanh (` *expression* `)`
: Hyperbolic tangent of *expression*.

2.4 User-defined functions

User-defined functions take no arguments, and are evaluated as if a file containing the function body was included at the location of the `Call` statement.

`Function` *string*
: Begins the declaration of a user-defined function named *string*. The body of the function starts on the line after '`Function` *string*', and can contain any Gmsh command.

`Return`
: Ends the body of the current user-defined function. Function declarations cannot be imbricated.

`Call` *string*`;`
: Executes the body of a (previously defined) function named *string*.

See Section 7.5 [t5.geo], page 103, for an example of a user-defined function.

2.5 Loops and conditionals

Loops and conditionals are defined as follows, and can be imbricated:

`For (` *expression* `:` *expression* `)`
: Iterates from the value of the first *expression* to the value of the second *expression*, with a unit incrementation step. At each iteration, the commands comprised between '`For (` *expression* `:` *expression* `)`' and the matching `EndFor` are executed.

`For (` *expression* `:` *expression* `:` *expression* `)`
: Iterates from the value of the first *expression* to the value of the second *expression*, with a positive or negative incrementation step equal to the third *expression*. At each iteration, the commands comprised between '`For (` *expression* `:` *expression* `:` *expression* `)`' and the matching `EndFor` are executed.

`For` *string* `In {` *expression* `:` *expression* `}`
: Iterates from the value of the first *expression* to the value of the second *expression*, with a unit incrementation step. At each iteration, the value of the iterate is affected to an expression named *string*, and the commands comprised between '`For` *string* `In {` *expression* `:` *expression* `}`' and the matching `EndFor` are executed.

`For` *string* `In {` *expression* `:` *expression* `:` *expression* `}`
: Iterates from the value of the first *expression* to the value of the second *expression*, with a positive or negative incrementation step equal to the third *expression*. At each iteration, the value of the iterate is affected to an expression named *string*, and the commands comprised between '`For` *string* `In {` *expression* `:` *expression* `:` *expression* `}`' and the matching `EndFor` are executed.

`EndFor`
: Ends a matching `For` command.

`If (` *expression* `)`
: The body enclosed between '`If (` *expression* `)`' and the matching `Endif` is evaluated if *expression* is non-zero.

`EndIf`
: Ends a matching `If` command.

See Section 7.5 [t5.geo], page 103, for an example of `For` and `If` commands. Gmsh does not provide any `Else` (or similar) command at the time of this writing.

2.6 General commands

The following commands can be used anywhere in a Gmsh ASCII text input file:

string `=` *expression*`;`
: Defines a new expression identifier *string*, or affects *expression* to an existing expression identifier. Eight expression identifiers are predefined (hardcoded in Gmsh's parser):

  `Pi` Returns 3.1415926535897932.

  `MPI_Size` Returns the number of processors on which Gmsh is running (always 1, except if you compiled Gmsh's parallel extensions).

  `MPI_Rank` Returns the rank of the current processor.

  `newp` Returns the next available point number. As explained in Chapter 3 [Geometry module], page 35, a unique number must be associated with every geometrical point: `newp` permits to know the highest number already attributed (plus one). This is mostly useful when writing user-defined functions (see Section 2.4 [User-defined functions], page 14) or general geometric primitives, when one does not know *a priori* which numbers are already attributed, and which ones are still available.

  `newl` Returns the next available line number.

  `news` Returns the next available surface number.

`newv` Returns the next available volume number.

`newreg` Returns the next available region number. That is, `newreg` returns the maximum of `newp`, `newl`, `news`, `newv` and all physical entity numbers[3].

string `[ ] = {` *expression-list* `};`
Defines a new expression list identifier *string*`[]`, or affects *expression-list* to an existing expression list identifier.

string `[ {` *expression-list* `} ] = {` *expression-list* `};`
Affects each item in the right hand side *expression-list* to the elements (indexed by the left hand side *expression-list*) of an existing expression list identifier. The two *expression-list*s must contain the same number of items. Remember the remark made when defining *expression-list*s: the braces enclosing an *expression-list* are optional if the list only contains a single item.

real-option `=` *expression*`;`
Affects *expression* to a real option.

char-option `=` *char-expression*`;`
Affects *char-expression* to a character option.

color-option `=` *color-expression*`;`
Affects *color-expression* to a color option.

string `|` *real-option* `+=` *expression*`;`
Adds and affects *expression* to an existing expression identifier or to a real option.

string `|` *real-option* `-=` *expression*`;`
Subtracts and affects *expression* to an existing expression identifier or to a real option.

string `|` *real-option* `*=` *expression*`;`
Multiplies and affects *expression* to an existing expression identifier or to a real option.

string `|` *real-option* `/=` *expression*`;`
Divides and affects *expression* to an existing expression identifier or to a real option.

string `[ {` *expression-list* `} ] += {` *expression-list* `};`
Adds and affects, item per item, the right hand side *expression-list* to an existing expression list identifier.

string `[ {` *expression-list* `} ] -= {` *expression-list* `};`
Subtracts and affects, item per item, the right hand side *expression-list* to an existing expression list identifier.

string `[ {` *expression-list* `} ] *= {` *expression-list* `};`
Multiplies and affects, item per item, the right hand side *expression-list* to an existing expression list identifier.

[3] For compatibility purposes, the behavior of `newl`, `news`, `newv` and `newreg` can be modified with the `Geometry.OldNewReg` option (see Section 3.2 [Geometry options], page 40).

string `[ {` *expression-list* `} ] /= {` *expression-list* `};`
: Divides and affects, item per item, the right hand side *expression-list* to an existing expression list identifier.

`Exit;`
: Aborts the current script.

`Printf (` *char-expression* `,` *expression-list* `);`
: Prints a character expression in the information window and/or on the terminal. `Printf` is equivalent to the `printf` C function: *char-expression* is a format string that can contain formatting characters (`%f`, `%e`, etc.). Note that all *expression*s are evaluated as floating point values in Gmsh (see Section 2.1 [Expressions], page 9), so that only valid floating point formatting characters make sense in *char-expression*. See Section 7.5 [t5.geo], page 103, for an example of the use of `Printf`.

`Merge` *char-expression*`;`
: Merges a file named *char-expression*. This command is equivalent to the 'File->Merge' menu in the graphical user interface. If the path in *char-expression* is not absolute, *char-expression* is appended to the path of the current file.

`Draw;`
: Redraws the scene.

`BoundingBox;`
: Recomputes the bounding box of the scene (which is normally computed only after new geometrical entities are added or after files are included or merged). The bounding box is computed as follows:

  1. If there is a mesh (i.e., at least one mesh vertex), the bounding box is taken as the box enclosing all the mesh vertices;
  2. If there is no mesh but there is a geometry (i.e., at least one geometrical point), the bounding box is taken as the box enclosing all the geometrical points;
  3. If there is no mesh and no geometry, but there are some post-processing views, the bounding box is taken as the box enclosing all the primitives in the views.

`BoundingBox {` *expression*`,` *expression*`,` *expression*`,` *expression*`,` *expression*`,` *expression* `};`
: Forces the bounding box of the scene to the given *expression*s (X min, X max, Y min, Y max, Z min, Z max).

`Delete All;`
: Deletes all geometrical entities and all currently loaded meshes.

`Print` *char-expression*`;`
: Prints the graphic window in a file named *char-expression*, using the current `Print.Format` (see Section 2.7 [General options], page 18). If the path in *char-expression* is not absolute, *char-expression* is appended to the path of the current file.

`Sleep` *expression*`;`
: Suspends the execution of Gmsh during *expression* seconds.

`System` *char-expression*`;`
: Executes a system call.

`Include` *char-expression*`;`
: Includes the file named *char-expression* at the current position in the input file. The include command should be given on a line of its own. If the path in *char-expression* is not absolute, *char-expression* is appended to the path of the current file.

2.7 General options

Here is the list of the general *char-option*s, *real-option*s and *color-option*s (in that order—check the default values to see the actual types). Most of these options are accessible in the graphical user interface, but not all of them. When running Gmsh interactively, changing an option in the ASCII text input file will modify the option in the GUI in real time. This permits for example to resize the graphical window in a script, or to interact with animations in the script and in the GUI at the same time.

Gmsh's default behavior is to save some of these options in a per-user "session resource" file (`General.SessionFileName`) every time Gmsh is shut down. This permits for example to automatically remember the size and location of the windows or which fonts to use. Other options can be saved in a per-user "option" file (`General.OptionsFileName`), automatically loaded by Gmsh every time it starts up, by using the 'Tools->Options->Save' menu.

`General.DefaultFileName`
: Default project file name
Default value: `"untitled.geo"`
Saved in: `General.OptionsFileName`

`General.Display`
: X server to use (only for Unix versions)
Default value: `""`
Saved in: `-`

`General.ErrorFileName`
: File into which the log is saved if a fatal error occurs
Default value: `".gmsh-errors"`
Saved in: `General.OptionsFileName`

`General.GraphicsFont`
: Font used in the graphic window
Default value: `"Helvetica"`
Saved in: `General.OptionsFileName`

`General.OptionsFileName`
: Option file created with 'Tools->Options->Save'; automatically read on startup
Default value: `".gmsh-options"`
Saved in: `General.SessionFileName`

`General.SessionFileName`
: Option file into which session specific information is saved; automatically read on startup
Default value: `".gmshrc"`
Saved in: `-`

`General.Scheme`
: FLTK user interface scheme (try e.g. plastic)
Default value: `""`
Saved in: `General.OptionsFileName`

`General.TextEditor`
: System command to launch a text editor
Default value: `"open -e %s"`
Saved in: `General.OptionsFileName`

`General.TmpFileName`
: Temporary file used by the geometry module
Default value: `".gmsh-tmp"`
Saved in: `General.SessionFileName`

`General.WebBrowser`
: System command to launch a web browser
Default value: `"open %s"`
Saved in: `General.OptionsFileName`

`General.AlphaBlending`
: Enable alpha blending (transparency) in post-processing views
Default value: `1`
Saved in: `General.OptionsFileName`

`General.ArrowHeadRadius`
: Relative radius of arrow head
Default value: `0.12`
Saved in: `General.OptionsFileName`

`General.ArrowStemLength`
: Relative length of arrow stem
Default value: `0.56`
Saved in: `General.OptionsFileName`

`General.ArrowStemRadius`
: Relative radius of arrow stem
Default value: `0.02`
Saved in: `General.OptionsFileName`

`General.Axes`
: Display the axes linked to the model
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Clip0`
: Enable clipping plane 0 (Geometry=2^0, Mesh=2^1, View[i]=2^(2+i))
Default value: `0`
Saved in: `-`

`General.Clip0A`
First coefficient in equation for clipping plane 0 ('A' in 'AX+BY+CZ+D=0')
Default value: `1`
Saved in: `-`

`General.Clip0B`
Second coefficient in equation for clipping plane 0 ('B' in 'AX+BY+CZ+D=0')
Default value: `0`
Saved in: `-`

`General.Clip0C`
Third coefficient in equation for clipping plane 0 ('C' in 'AX+BY+CZ+D=0')
Default value: `0`
Saved in: `-`

`General.Clip0D`
Fourth coefficient in equation for clipping plane 0 ('D' in 'AX+BY+CZ+D=0')
Default value: `0`
Saved in: `-`

`General.Clip1`
Enable clipping plane 1 (Geometry=2^0, Mesh=2^1, View[i]=2^(2+i))
Default value: `0`
Saved in: `-`

`General.Clip1A`
First coefficient in equation for clipping plane 1
Default value: `1`
Saved in: `-`

`General.Clip1B`
Second coefficient in equation for clipping plane 1
Default value: `0`
Saved in: `-`

`General.Clip1C`
Third coefficient in equation for clipping plane 1
Default value: `0`
Saved in: `-`

`General.Clip1D`
Fourth coefficient in equation for clipping plane 1
Default value: `0`
Saved in: `-`

`General.Clip2`
Enable clipping plane 2 (Geometry=2^0, Mesh=2^1, View[i]=2^(2+i))
Default value: `0`
Saved in: `-`

`General.Clip2A`
First coefficient in equation for clipping plane 2
Default value: `1`
Saved in: `-`

`General.Clip2B`
Second coefficient in equation for clipping plane 2
Default value: `0`
Saved in: `-`

`General.Clip2C`
Third coefficient in equation for clipping plane 2
Default value: `0`
Saved in: `-`

`General.Clip2D`
Fourth coefficient in equation for clipping plane 2
Default value: `0`
Saved in: `-`

`General.Clip3`
Enable clipping plane 3 (Geometry=2^0, Mesh=2^1, View[i]=2^(2+i))
Default value: `0`
Saved in: `-`

`General.Clip3A`
First coefficient in equation for clipping plane 3
Default value: `1`
Saved in: `-`

`General.Clip3B`
Second coefficient in equation for clipping plane 3
Default value: `0`
Saved in: `-`

`General.Clip3C`
Third coefficient in equation for clipping plane 3
Default value: `0`
Saved in: `-`

`General.Clip3D`
Fourth coefficient in equation for clipping plane 3
Default value: `0`
Saved in: `-`

`General.Clip4`
Enable clipping plane 4 (Geometry=2^0, Mesh=2^1, View[i]=2^(2+i))
Default value: `0`
Saved in: `-`

`General.Clip4A`
First coefficient in equation for clipping plane 4
Default value: `1`
Saved in: `-`

`General.Clip4B`
Second coefficient in equation for clipping plane 4
Default value: `0`
Saved in: `-`

`General.Clip4C`
Third coefficient in equation for clipping plane 4
Default value: `0`
Saved in: `-`

`General.Clip4D`
Fourth coefficient in equation for clipping plane 4
Default value: `0`
Saved in: `-`

`General.Clip5`
Enable clipping plane 5 (Geometry=2^0, Mesh=2^1, View[i]=2^(2+i))
Default value: `0`
Saved in: `-`

`General.Clip5A`
First coefficient in equation for clipping plane 5
Default value: `1`
Saved in: `-`

`General.Clip5B`
Second coefficient in equation for clipping plane 5
Default value: `0`
Saved in: `-`

`General.Clip5C`
Third coefficient in equation for clipping plane 5
Default value: `0`
Saved in: `-`

`General.Clip5D`
Fourth coefficient in equation for clipping plane 5
Default value: `0`
Saved in: `-`

`General.ClipPositionX`
Horizontal position (in pixels) of the upper left corner of the clipping planes window
Default value: `650`
Saved in: `General.SessionFileName`

`General.ClipPositionY`
: Vertical position (in pixels) of the upper left corner of the clipping planes window
Default value: `150`
Saved in: `General.SessionFileName`

`General.ColorScheme`
: Default color scheme (0, 1 or 2)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.ConfirmOverwrite`
: Ask confirmation before overwriting files?
Default value: `1`
Saved in: `General.OptionsFileName`

`General.ContextPositionX`
: Horizontal position (in pixels) of the upper left corner of the contextual windows
Default value: `650`
Saved in: `General.SessionFileName`

`General.ContextPositionY`
: Vertical position (in pixels) of the upper left corner of the contextual windows
Default value: `150`
Saved in: `General.SessionFileName`

`General.DoubleBuffer`
: Use a double buffered graphic window (on Unix, should be set to 0 when working on a remote host without GLX)
Default value: `1`
Saved in: `General.OptionsFileName`

`General.DrawBoundingBoxes`
: Draw bounding boxes
Default value: `0`
Saved in: `General.OptionsFileName`

`General.FakeTransparency`
: Use fake transparency (cheaper than the real thing, but incorrect)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.FastRedraw`
: Draw simplified model while rotating, panning and zooming
Default value: `1`
Saved in: `General.OptionsFileName`

`General.FileChooserPositionX`
: Horizontal position (in pixels) of the upper left corner of the file chooser windows
Default value: `200`
Saved in: `General.SessionFileName`

`General.FileChooserPositionY`
: Vertical position (in pixels) of the upper left corner of the file chooser windows
Default value: `200`
Saved in: `General.SessionFileName`

`General.FontSize`
: Size of the font in the user interface
Default value: `12`
Saved in: `General.OptionsFileName`

`General.GraphicsFontSize`
: Size of the font in the graphic window
Default value: `14`
Saved in: `General.OptionsFileName`

`General.GraphicsHeight`
: Height (in pixels) of the graphic window
Default value: `600`
Saved in: `General.SessionFileName`

`General.GraphicsPositionX`
: Horizontal position (in pixels) of the upper left corner of the graphic window
Default value: `50`
Saved in: `General.SessionFileName`

`General.GraphicsPositionY`
: Vertical position (in pixels) of the upper left corner of the graphic window
Default value: `50`
Saved in: `General.SessionFileName`

`General.GraphicsWidth`
: Width (in pixels) of the graphic window
Default value: `600`
Saved in: `General.SessionFileName`

`General.InitialModule`
: Module launched on startup (0=automatic, 1=geometry, 2=mesh, 3=solver, 4=post-processing)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light0`
: Enable light source 0
Default value: `1`
Saved in: `General.OptionsFileName`

`General.Light0X`
: X position of light source 0
Default value: `0.65`
Saved in: `General.OptionsFileName`

`General.Light0Y`
: Y position of light source 0
Default value: `0.65`
Saved in: `General.OptionsFileName`

`General.Light0Z`
: Z position of light source 0
Default value: `1`
Saved in: `General.OptionsFileName`

`General.Light0W`
: Divisor of the X, Y and Z coordinates of light source 0 (W=0 means infinitely far source)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light1`
: Enable light source 1
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light1X`
: X position of light source 1
Default value: `0.5`
Saved in: `General.OptionsFileName`

`General.Light1Y`
: Y position of light source 1
Default value: `0.3`
Saved in: `General.OptionsFileName`

`General.Light1Z`
: Z position of light source 1
Default value: `1`
Saved in: `General.OptionsFileName`

`General.Light1W`
: Divisor of the X, Y and Z coordinates of light source 1 (W=0 means infinitely far source)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light2`
: Enable light source 2
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light2X`
: X position of light source 2
Default value: `0.5`
Saved in: `General.OptionsFileName`

`General.Light2Y`
Y position of light source 2
Default value: `0.3`
Saved in: `General.OptionsFileName`

`General.Light2Z`
Z position of light source 2
Default value: `1`
Saved in: `General.OptionsFileName`

`General.Light2W`
Divisor of the X, Y and Z coordinates of light source 2 (W=0 means infinitely far source)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light3`
Enable light source 3
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light3X`
X position of light source 3
Default value: `0.5`
Saved in: `General.OptionsFileName`

`General.Light3Y`
Y position of light source 3
Default value: `0.3`
Saved in: `General.OptionsFileName`

`General.Light3Z`
Z position of light source 3
Default value: `1`
Saved in: `General.OptionsFileName`

`General.Light3W`
Divisor of the X, Y and Z coordinates of light source 3 (W=0 means infinitely far source)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light4`
Enable light source 4
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light4X`
X position of light source 4
Default value: `0.5`
Saved in: `General.OptionsFileName`

`General.Light4Y`
: Y position of light source 4
Default value: `0.3`
Saved in: `General.OptionsFileName`

`General.Light4Z`
: Z position of light source 4
Default value: `1`
Saved in: `General.OptionsFileName`

`General.Light4W`
: Divisor of the X, Y and Z coordinates of light source 4 (W=0 means infinitely far source)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light5`
: Enable light source 5
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Light5X`
: X position of light source 5
Default value: `0.5`
Saved in: `General.OptionsFileName`

`General.Light5Y`
: Y position of light source 5
Default value: `0.3`
Saved in: `General.OptionsFileName`

`General.Light5Z`
: Z position of light source 5
Default value: `1`
Saved in: `General.OptionsFileName`

`General.Light5W`
: Divisor of the X, Y and Z coordinates of light source 5 (W=0 means infinitely far source)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.LineWidth`
: Display width of lines (in pixels)
Default value: `1`
Saved in: `General.OptionsFileName`

`General.MenuPositionX`
: Horizontal position (in pixels) of the upper left corner of the menu window
Default value: `800`
Saved in: `General.SessionFileName`

`General.MenuPositionY`
: Vertical position (in pixels) of the upper left corner of the menu window
Default value: `50`
Saved in: `General.SessionFileName`

`General.MessagePositionX`
: Horizontal position (in pixels) of the upper left corner of the message window
Default value: `650`
Saved in: `General.SessionFileName`

`General.MessagePositionY`
: Vertical position (in pixels) of the upper left corner of the message window
Default value: `150`
Saved in: `General.SessionFileName`

`General.MessageHeight`
: Height (in pixels) of the message window
Default value: `350`
Saved in: `General.SessionFileName`

`General.MessageWidth`
: Width (in pixels) of the message window
Default value: `450`
Saved in: `General.SessionFileName`

`General.OptionsPositionX`
: Horizontal position (in pixels) of the upper left corner of the option window
Default value: `650`
Saved in: `General.SessionFileName`

`General.OptionsPositionY`
: Vertical position (in pixels) of the upper left corner of the option window
Default value: `150`
Saved in: `General.SessionFileName`

`General.Orthographic`
: Orthographic projection mode (0=perspective projection)
Default value: `1`
Saved in: `General.OptionsFileName`

`General.PointSize`
: Display size of points (in pixels)
Default value: `3`
Saved in: `General.OptionsFileName`

`General.QuadricSubdivisions`
: Number of subdivisions used to draw points or lines as spheres or cylinders
Default value: `8`
Saved in: `General.OptionsFileName`

`General.RotationX`
: First Euler angle (used if Trackball=0)
Default value: `0`
Saved in: `-`

`General.RotationY`
: Second Euler angle (used if Trackball=0)
Default value: `0`
Saved in: `-`

`General.RotationZ`
: Third Euler angle (used if Trackball=0)
Default value: `0`
Saved in: `-`

`General.RotationCenterGravity`
: Rotate around the (pseudo) center of mass instead of (RotationCenterX, RotationCenterY, RotationCenterZ)
Default value: `1`
Saved in: `General.OptionsFileName`

`General.RotationCenterX`
: X coordinate of the center of rotation
Default value: `0`
Saved in: `-`

`General.RotationCenterY`
: Y coordinate of the center of rotation
Default value: `0`
Saved in: `-`

`General.RotationCenterZ`
: Z coordinate of the center of rotation
Default value: `0`
Saved in: `-`

`General.SaveOptions`
: Automatically save current options in General.OptionsFileName each time you quit Gmsh?
Default value: `0`
Saved in: `General.SessionFileName`

`General.SaveSession`
: Automatically save session specific information in General.SessionFileName each time you quit Gmsh?
Default value: `1`
Saved in: `General.SessionFileName`

`General.ScaleX`
: X-axis scale factor
Default value: `1`
Saved in: `-`

`General.ScaleY`
: Y-axis scale factor
Default value: `1`
Saved in: `-`

`General.ScaleZ`
: Z-axis scale factor
Default value: `1`
Saved in: `-`

`General.Shininess`
: Material shininess
Default value: `0.4`
Saved in: `General.OptionsFileName`

`General.ShininessExponent`
: Material shininess exponent (between 0 and 128)
Default value: `40`
Saved in: `General.OptionsFileName`

`General.SmallAxes`
: Display the small axes
Default value: `1`
Saved in: `General.OptionsFileName`

`General.SmallAxesPositionX`
: X position of small axes (use negative values for right alignment)
Default value: `-60`
Saved in: `General.OptionsFileName`

`General.SmallAxesPositionY`
: Y position of small axes (use negative values for bottom alignment)
Default value: `-40`
Saved in: `General.OptionsFileName`

`General.SolverPositionX`
: Horizontal position (in pixels) of the upper left corner of the solver windows
Default value: `650`
Saved in: `General.SessionFileName`

`General.SolverPositionY`
: Vertical position (in pixels) of the upper left corner of the solver windows
Default value: `150`
Saved in: `General.SessionFileName`

`General.StatisticsPositionX`
: Horizontal position (in pixels) of the upper left corner of the statistic window
Default value: `650`
Saved in: `General.SessionFileName`

`General.StatisticsPositionY`
: Vertical position (in pixels) of the upper left corner of the statistic window
Default value: `150`
Saved in: `General.SessionFileName`

`General.SystemMenuBar`
: Use the system menu bar on Mac OS X?
Default value: `1`
Saved in: `General.SessionFileName`

`General.Terminal`
: Should information be printed on the terminal (if available)?
Default value: `0`
Saved in: `General.OptionsFileName`

`General.Tooltips`
: Show tooltips in the user interface
Default value: `1`
Saved in: `General.OptionsFileName`

`General.Trackball`
: Use trackball rotation mode
Default value: `1`
Saved in: `General.OptionsFileName`

`General.TrackballQuaternion0`
: First trackball quaternion component (used if General.Trackball=1)
Default value: `0`
Saved in: `-`

`General.TrackballQuaternion1`
: Second trackball quaternion component (used if General.Trackball=1)
Default value: `0`
Saved in: `-`

`General.TrackballQuaternion2`
: Third trackball quaternion component (used if General.Trackball=1)
Default value: `0`
Saved in: `-`

`General.TrackballQuaternion3`
: Fourth trackball quaternion component (used if General.Trackball=1)
Default value: `1`
Saved in: `-`

`General.TranslationX`
: X-axis translation (in model units)
Default value: `0`
Saved in: `-`

`General.TranslationY`
: Y-axis translation (in model units)
Default value: `0`
Saved in: `-`

`General.TranslationZ`
: Z-axis translation (in model units)
Default value: `0`
Saved in: `-`

`General.VectorType`
: Default vector display type (for normals, etc.)
Default value: `4`
Saved in: `General.OptionsFileName`

`General.Verbosity`
: Level of information printed during processing (0=no information)
Default value: `3`
Saved in: `General.OptionsFileName`

`General.VisibilityMode`
: Default mode for the visibility browser (0=Geometry+Mesh, 1=Geometry, 2=Mesh)
Default value: `0`
Saved in: `General.OptionsFileName`

`General.VisibilityPositionX`
: Horizontal position (in pixels) of the upper left corner of the visibility window
Default value: `650`
Saved in: `General.SessionFileName`

`General.VisibilityPositionY`
: Vertical position (in pixels) of the upper left corner of the visibility window
Default value: `150`
Saved in: `General.SessionFileName`

`General.ZoomFactor`
: 'Speed' of the middle mouse button zoom
Default value: `1.1`
Saved in: `General.OptionsFileName`

`General.Color.Background`
: Background color
Default value: `{0,0,0}`
Saved in: `General.OptionsFileName`

`General.Color.Foreground`
: Foreground color
Default value: `{255,255,255}`
Saved in: `General.OptionsFileName`

`General.Color.Text`
Text color
Default value: `{255,255,255}`
Saved in: `General.OptionsFileName`

`General.Color.Axes`
Axes color
Default value: `{255,255,0}`
Saved in: `General.OptionsFileName`

`General.Color.SmallAxes`
Small axes color
Default value: `{255,255,255}`
Saved in: `General.OptionsFileName`

`General.Color.AmbientLight`
Ambient light color
Default value: `{25,25,25}`
Saved in: `General.OptionsFileName`

`General.Color.DiffuseLight`
Diffuse light color
Default value: `{255,255,255}`
Saved in: `General.OptionsFileName`

`General.Color.SpecularLight`
Specular light color
Default value: `{255,255,255}`
Saved in: `General.OptionsFileName`

`Print.EpsBackground`
Save image background in PostScript/PDF output
Default value: `1`
Saved in: `General.OptionsFileName`

`Print.EpsBestRoot`
Try to minimize primitive splitting in BSP tree sorted PostScript/PDF output
Default value: `1`
Saved in: `General.OptionsFileName`

`Print.EpsCompress`
Compress PostScript/PDF output using zlib
Default value: `0`
Saved in: `General.OptionsFileName`

`Print.EpsLineWidthFactor`
Width factor for lines in PostScript/PDF output
Default value: `0.5`
Saved in: `General.OptionsFileName`

`Print.EpsOcclusionCulling`
Cull occluded primitives (to reduce PostScript/PDF file size)
Default value: `1`
Saved in: `General.OptionsFileName`

`Print.EpsPointSizeFactor`
Size factor for points in PostScript/PDF output
Default value: `1`
Saved in: `General.OptionsFileName`

`Print.EpsPS3Shading`
Enable PostScript Level 3 shading
Default value: `0`
Saved in: `General.OptionsFileName`

`Print.EpsQuality`
PostScript/PDF quality (1=simple sort, 2=BSP tree sort)
Default value: `1`
Saved in: `General.OptionsFileName`

`Print.Format`
File format (10=automatic)
Default value: `10`
Saved in: `General.OptionsFileName`

`Print.GifDither`
Apply dithering to GIF output
Default value: `0`
Saved in: `General.OptionsFileName`

`Print.GifInterlace`
Interlace GIF output
Default value: `0`
Saved in: `General.OptionsFileName`

`Print.GifSort`
Sort the colormap in GIF output
Default value: `1`
Saved in: `General.OptionsFileName`

`Print.GifTransparent`
Output transparent GIF image
Default value: `0`
Saved in: `General.OptionsFileName`

`Print.JpegQuality`
JPEG quality (between 1 and 100)
Default value: `100`
Saved in: `General.OptionsFileName`

3 Geometry module

Gmsh's geometry module provides a simple CAD engine, using a bottom-up approach: you need to first define points (using the `Point` command: see below), then lines (using `Line`, `Circle`, `Spline`, ..., commands or by extruding points), then surfaces (using for example the `Plane Surface` or `Ruled Surface` commands, or by extruding lines), and finally volumes (using the `Volume` command or by extruding surfaces).

These geometrical entities are called "elementary" in Gmsh's jargon, and are assigned identification numbers when they are created:

1. each elementary point must possess a unique identification number;
2. each elementary line must possess a unique identification number;
3. each elementary surface must possess a unique identification number;
4. each elementary volume must possess a unique identification number.

Elementary geometrical entities can then be manipulated in various ways, for example using the `Translate`, `Rotate`, `Scale` or `Symmetry` commands.

Compound groups of elementary geometrical entities can also be defined and are called "physical" entities. These physical entities cannot be modified by geometry commands: their only purpose is to assemble elementary entities into larger groups, possibly modifying their orientation, so that they can be referred to by the mesh module as single entities. As is the case with elementary entities, each physical point, physical line, physical surface or physical volume must be assigned a unique identification number. See Chapter 4 [Mesh module], page 45, for more information about how physical entities affect the way meshes are saved.

3.1 Geometry commands

The next subsections describe all the available geometry commands. These commands can be used anywhere in a Gmsh ASCII text input file. Note that the following general syntax rule is followed for the definition of geometrical entities: "If an *expression* defines a new entity, it is enclosed between parentheses. If an *expression* refers to a previously defined entity, it is enclosed between braces."

3.1.1 Points

`Point (` *expression* `) = {` *expression*`,` *expression*`,` *expression*`,` *expression* `};`

Creates an elementary point. The *expression* inside the parentheses is the point's identification number; the three first *expression*s inside the braces on the right hand side give the three X, Y and Z coordinates of the point in the three-dimensional Euclidean space; the last *expression* sets the characteristic mesh length at that point. See Section 4.2.1 [Characteristic lengths], page 46, for more information about how this characteristic length information is used in the meshing process.

`Physical Point (` *expression* `) = {` *expression-list* `};`

Creates a physical point. The *expression* inside the parentheses is the physical point's identification number; the *expression-list* on the right hand side should

contain the identification numbers of all the elementary points that need to be grouped inside the physical point.

3.1.2 Lines

`Bezier (` *expression* `) = {` *expression-list* `};`
: Creates a Bezier curve. The *expression* inside the parentheses is the Bezier curve's identification number; the *expression-list* on the right hand side should contain the identification numbers of all the curve's control points.

`BSpline (` *expression* `) = {` *expression-list* `};`
: Creates a B-spline curve. The *expression* inside the parentheses is the B-spline curve's identification number; the *expression-list* on the right hand side should contain the identification numbers of all the B-spline's control points. Repeating control points has the expected effect.

`Circle (` *expression* `) = {` *expression*`,` *expression*`,` *expression* `};`
: Creates a circle arc (strictly) smaller than Pi. The *expression* inside the parentheses is the circle arc's identification number; the first *expression* inside the braces on the right hand side gives the identification number of the start point of the arc; the second *expression* gives the identification number of the center of the circle; the last *expression* gives the identification number of the end point of the arc.

`CatmullRom (` *expression* `) = {` *expression-list* `};`
: `CatmullRom` is a synonym for `Spline`.

`Ellipse (` *expression* `) = {` *expression*`,` *expression*`,` *expression*`,` *expression* `};`
: Creates an ellipse arc. The *expression* inside the parentheses is the ellipse arc's identification number; the first *expression* inside the braces on the right hand side gives the identification number of the start point of the arc; the second *expression* gives the identification number of the center of the ellipse; the third *expression* gives the identification number of any point located on the major axis of the ellipse; the last *expression* gives the identification number of the end point of the arc.

  (A deprecated synonym for `Ellipse` is `Ellipsis`.)

`Line (` *expression* `) = {` *expression*`,` *expression* `};`
: Creates a straight line segment. The *expression* inside the parentheses is the line segment's identification number; the two *expression*s inside the braces on the right hand side give identification numbers of the start and end points of the segment.

`Spline (` *expression* `) = {` *expression-list* `};`
: Creates a spline curve. The *expression* inside the parentheses is the spline's identification number; the *expression-list* on the right hand side should contain the identification numbers of all the spline's control points.

`Line Loop (` *expression* `) = {` *expression-list* `};`
: Creates an oriented line loop. The *expression* inside the parentheses is the line loop's identification number; the *expression-list* on the right hand side should

contain the identification numbers of all the elementary lines that constitute the line loop. A line loop must be a closed loop, and the elementary lines should be ordered and oriented (using negative identification numbers to specify reverse orientation). If the orientation is correct, but the ordering is wrong, Gmsh will actually reorder the list internally to create a consistent loop. Although Gmsh supports it, it is not recommended to specify multiple line loops (or subloops) in a single `Line Loop` command. (Line loops are used to create surfaces: see Section 3.1.3 [Surfaces], page 37.)

`Physical Line (` *expression* `) = {` *expression-list* `};`
Creates a physical line. The *expression* inside the parentheses is the physical line's identification number; the *expression-list* on the right hand side should contain the identification numbers of all the elementary lines that need to be grouped inside the physical line. Specifying negative identification numbers in the *expression-list* will reverse the orientation of the mesh elements belonging to the corresponding elementary lines in the saved mesh.

3.1.3 Surfaces

`Plane Surface (` *expression* `) = {` *expression-list* `};`
Creates a plane surface. The *expression* inside the parentheses is the plane surface's identification number; the *expression-list* on the right hand side should contain the identification numbers of all the line loops defining the surface. The first line loop defines the exterior boundary of the surface; all other line loops define holes in the surface. A line loop defining a hole should not have any lines in common with the exterior line loop (in which case it is not a hole, and the two surfaces should be defined separately). Likewise, a line loop defining a hole should not have any lines in common with another line loop defining a hole in the same surface (in which case the two line loops should be combined).

`Ruled Surface (` *expression* `) = {` *expression-list* `};`
Creates a ruled surface, i.e., a surface that can be interpolated using transfinite interpolation. The *expression* inside the parentheses is the ruled surface's identification number; the *expression-list* on the right hand side should the identification number of a single line loop, composed of either three or four elementary lines.

`Surface Loop (` *expression* `) = {` *expression-list* `};`
Creates a surface loop (a shell). The *expression* inside the parentheses is the surface loop's identification number; the *expression-list* on the right hand side should contain the identification numbers of all the elementary surfaces that constitute the surface loop. A surface loop must always represent a closed shell, and the elementary surfaces should be oriented consistently (using negative identification numbers to specify reverse orientation). (Surface loops are used to create volumes: see Section 3.1.4 [Volumes], page 38.)

`Physical Surface (` *expression* `) = {` *expression-list* `};`
Creates a physical surface. The *expression* inside the parentheses is the physical surface's identification number; the *expression-list* on the right hand side should

contain the identification numbers of all the elementary surfaces that need to be grouped inside the physical surface. Specifying negative identification numbers in the *expression-list* will reverse the orientation of the mesh elements belonging to the corresponding elementary surfaces in the saved mesh.

3.1.4 Volumes

`Volume ( ` *expression* ` ) = { ` *expression-list* ` };`

Creates a volume. The *expression* inside the parentheses is the volume's identification number; the *expression-list* on the right hand side should contain the identification numbers of all the surface loops defining the volume. The first surface loop defines the exterior boundary of the volume; all other surface loops define holes in the volume. A surface loop defining a hole should not have any surfaces in common with the exterior surface loop (in which case it is not a hole, and the two volumes should be defined separately). Likewise, a surface loop defining a hole should not have any surfaces in common with another surface loop defining a hole in the same volume (in which case the two surface loops should be combined).

(A deprecated synonym for `Volume` is `Complex Volume`.)

`Physical Volume ( ` *expression* ` ) = { ` *expression-list* ` };`

Creates a physical volume. The *expression* inside the parentheses is the physical volume's identification number; the *expression-list* on the right hand side should contain the identification numbers of all the elementary volumes that need to be grouped inside the physical volume.

3.1.5 Extrusions

Lines, surfaces and volumes can also be created through extrusion of points, lines and surfaces, respectively. Here is the syntax of the geometrical extrusion commands (go to Section 4.2.2 [Structured grids], page 47, to see how these commands can be extended in order to also extrude the mesh):

extrude:

`Extrude Point | Line | Surface { ` *expression* `, { ` *expression-list* ` } };`

Extrudes the *expression*-th point, line or surface using a translation. The *expression-list* should contain three *expression*s giving the X, Y and Z components of the translation vector.

`Extrude Point | Line | Surface { ` *expression* `, { ` *expression-list* ` }, { ` *expression-list* ` }, ` *expression* ` };`

Extrudes the *expression*-th point, line or surface using a rotation. The first *expression-list* should contain three *expression*s giving the X, Y and Z direction of the rotation axis; the second *expression-list* should contain three *expression*s giving the X, Y and Z components of any point on this axis; the last *expression* should contain the rotation angle (in radians).

`Extrude Point | Line | Surface {` *expression*`, {` *expression-list* `}, {` *expression-list* `}, {` *expression-list* `}, ` *expression* `};`

Extrudes the *expression*-th point, line or surface using a translation combined with a rotation. The first *expression-list* should contain three *expression*s giving the X, Y and Z components of the translation vector; the second *expression-list* should contain three *expression*s giving the X, Y and Z direction of the rotation axis; the third *expression-list* should contain three *expression*s giving the X, Y and Z components of any point on this axis; the last *expression* should contain the rotation angle (in radians).

3.1.6 Transformations

Geometrical transformations can be applied to elementary entities, or to copies of elementary entities (using the `Duplicata` command: see below). The syntax of the transformation commands is:

transform:

`Dilate { {` *expression-list* `}, ` *expression* `} {` *transform-list* `}`

Scales all elementary entities (points, lines or surfaces) in *transform-list* by a factor *expression*. The *expression-list* should contain three *expression*s giving the X, Y and Z direction of the homothetic transformation.

`Rotate { {` *expression-list* `}, {` *expression-list* `}, ` *expression* `} {` *transform-list* `}`

Rotates all elementary entities (points, lines or surfaces) in *transform-list* by an angle of *expression* radians. The first *expression-list* should contain three *expression*s giving the X, Y and Z direction of the rotation axis; the second *expression-list* should contain three *expression*s giving the X, Y and Z components of any point on this axis.

`Symmetry {` *expression-list* `} {` *transform-list* `}`

Transforms all elementary entities (points, lines or surfaces) symmetrically to a plane. The *expression-list* should contain four *expression*s giving the coefficients of the plane's equation.

`Translate {` *expression-list* `} {` *transform-list* `}`

Translates all elementary entities (points, lines or surfaces) in *transform-list*. The *expression-list* should contain three *expression*s giving the X, Y and Z components of the translation vector.

with

transform-list:
`Point | Line | Surface {` *expression-list* `}; ... |`
`Duplicata { Point | Line | Surface {` *expression-list* `}; ... } |`
transform

3.1.7 Miscellaneous

Here is a list of all other geometry commands currently available:

`Coherence;`
: Removes all duplicate elementary geometrical entities (e.g., points having identical coordinates). Note that Gmsh executes the `Coherence` command automatically after each geometrical transformation, unless `Geometry.AutoCoherence` is set to zero (see Section 3.2 [Geometry options], page 40).

`Delete { Point | Line | Surface | Volume {` *expression-list* `}; ... }`
: Deletes all elementary entities (points, lines, surfaces or volumes) whose identification numbers are given in *expression-list*.

`Hide { Point | Line | Surface | Volume {` *expression-list* `}; ... }`
: Hide the entities listed in *expression-list*, if `General.VisibilityMode` is set to `0` or `1`.

`Hide` *char-expression*`;`
: Hide the entity *char-expression*, if `General.VisibilityMode` is set to `0` or `1` (*char-expression* can for example be `"*"`).

`Show { Point | Line | Surface | Volume {` *expression-list* `}; ... }`
: Show the entities listed in *expression-list*, if `General.VisibilityMode` is set to `0` or `1`.

`Show` *char-expression*`;`
: Show the entity *char-expression*, if `General.VisibilityMode` is set to `0` or `1` (*char-expression* can for example be `"*"`).

3.2 Geometry options

Geometry options control the behavior of geometry commands, as well as the way geometrical entities are handled in the graphical user interface. For the signification of the 'Saved in:' field in the following list, see Section 2.7 [General options], page 18.

`Geometry.AutoCoherence`
: Should all duplicate entities be automatically removed?
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.CirclePoints`
: Number of points used to draw a circle/ellipse
Default value: `20`
Saved in: `General.OptionsFileName`

`Geometry.CircleWarning`
: Warn if circle arc is greater than Pi
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.ExtrudeSplinePoints`
: Number of control points for splines created during extrusion
Default value: `5`
Saved in: `General.OptionsFileName`

`Geometry.Light`
: Enable lighting for the geometry
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.Lines`
: Display geometry curves?
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.LineNumbers`
: Display curve numbers?
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.LineSelectWidth`
: Display width of selected lines (in pixels)
Default value: `2`
Saved in: `General.OptionsFileName`

`Geometry.LineType`
: Display lines as solid color segments (0) or 3D cylinders (1)
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.LineWidth`
: Display width of lines (in pixels)
Default value: `2`
Saved in: `General.OptionsFileName`

`Geometry.Normals`
: Display size of normal vectors (in pixels)
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.OldCircle`
: Use old circle description (compatibility option for old Gmsh geometries)
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.OldNewReg`
: Use old newreg definition for geometrical transformations (compatibility option for old Gmsh geometries)
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.Points`
: Display geometry points?
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.PointNumbers`
Display points numbers?
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.PointSelectSize`
Display size of selected points (in pixels)
Default value: `5`
Saved in: `General.OptionsFileName`

`Geometry.PointSize`
Display size of points (in pixels)
Default value: `4`
Saved in: `General.OptionsFileName`

`Geometry.PointType`
Display points as solid color dots (0) or 3D spheres (1)
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.ScalingFactor`
Global geometry scaling factor
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.StlCreateElementary`
Treat each STL input face as a new geometrical surface
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.StlCreatePhysical`
Automatically create physical entities when importing STL faces as geometrical surfaces
Default value: `1`
Saved in: `General.OptionsFileName`

`Geometry.Surfaces`
Display geometry surfaces?
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.SurfaceNumbers`
Display surface numbers?
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.Tangents`
Display size of tangent vectors (in pixels)
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.Volumes`
Display geometry volumes? (not implemented yet)
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.VolumeNumbers`
Display volume numbers? (not implemented yet)
Default value: `0`
Saved in: `General.OptionsFileName`

`Geometry.Color.Points`
Normal geometry point color
Default value: `{178,182,129}`
Saved in: `General.OptionsFileName`

`Geometry.Color.Lines`
Normal geometry curve color
Default value: `{0,0,255}`
Saved in: `General.OptionsFileName`

`Geometry.Color.Surfaces`
Normal geometry surface color
Default value: `{128,128,128}`
Saved in: `General.OptionsFileName`

`Geometry.Color.Volumes`
Normal geometry volume color
Default value: `{128,128,128}`
Saved in: `General.OptionsFileName`

`Geometry.Color.PointsSelect`
Selected geometry point color
Default value: `{255,0,0}`
Saved in: `General.OptionsFileName`

`Geometry.Color.LinesSelect`
Selected geometry curve color
Default value: `{255,0,0}`
Saved in: `General.OptionsFileName`

`Geometry.Color.SurfacesSelect`
Selected geometry surface color
Default value: `{255,0,0}`
Saved in: `General.OptionsFileName`

`Geometry.Color.VolumesSelect`
Selected geometry volume color
Default value: `{255,0,0}`
Saved in: `General.OptionsFileName`

`Geometry.Color.Tangents`
: Tangent geometry vectors color
Default value: `{255,255,0}`
Saved in: `General.OptionsFileName`

`Geometry.Color.Normals`
: Normal geometry vectors color
Default value: `{255,0,0}`
Saved in: `General.OptionsFileName`

4 Mesh module

Gmsh's mesh module regroups several 1D, 2D and 3D mesh algorithms, all producing grids conforming in the sense of finite elements (see Section 1.2 [Mesh], page 3).

The 2D unstructured algorithms generate triangles or both triangles and quadrangles (when `Recombine Surface` is used: see Section 4.2.3 [Miscellaneous mesh commands], page 48). The 3D unstructured algorithm only generates tetrahedra.

The 2D structured algorithms (transfinite, elliptic and extrusion) generate triangles by default, but quadrangles can be obtained by using the `Recombine` commands (see Section 4.2.2 [Structured grids], page 47, and Section 4.2.3 [Miscellaneous mesh commands], page 48). The 3D structured algorithms generate tetrahedra, hexahedra, prisms and pyramids, depending on the type of the surface meshes they are based on.

4.1 Elementary vs. physical entities

If only elementary geometrical entities are defined (or if the `Mesh.SaveAll` option is set; see Section 4.3 [Mesh options], page 49), the grid produced by the mesh module will be saved "as is". That is, all the elements in the grid will be saved to disk using the identification number of the elementary entities they discretize as their region number (see Section 9.1 [Gmsh mesh file formats], page 125). This can sometimes be inconvenient:

- mesh elements cannot be duplicated;
- the orientation of the mesh elements (the ordering of their nodes) is determined entirely by the orientation of their "parent" elementary entities, and cannot be modified;
- elements belonging to different elementary entities cannot be linked as being part of a larger group having a physical or mathematical meaning (like 'Left wing', 'Metallic part', 'Dirichlet boundary condition', ...).

To remedy these problems, the geometry module introduces the notion of "physical" entities (see Chapter 3 [Geometry module], page 35). The purpose of physical entities is to assemble elementary entities into larger, possibly overlapping groups, and to control the orientation of the elements in these groups. If physical entities are defined, the output mesh only contains those elements that belong to physical entities. The introduction of such physical entities in large models usually greatly facilitates the manipulation of the model (e.g., using 'Tools->Visibility' in the GUI) and the interfacing with external solvers.

4.2 Mesh commands

The mesh module commands mostly permit to modify the characteristic lengths and specify structured grid parameters. The actual mesh "actions" (i.e., "mesh the lines", "mesh the surfaces" and "mesh the volumes") cannot be specified in the input ASCII text input files. They have to be given either in the GUI or on the command line (see Chapter 8 [Running Gmsh], page 119, and Section 8.3 [Command-line options], page 120).

4.2.1 Characteristic lengths

The 'size' of a mesh element is defined as the length of the segment for a line segment, the radius of the circumscribed circle for a triangle and the radius of the circumscribed sphere for a tetrahedron. There are three main ways to specify the size of the mesh elements for a given geometry:

1. You can specify characteristic lengths at the points of the geometrical model (with the `Point` command: see Section 3.1.1 [Points], page 35). The actual size of the mesh elements will be computed by linearly interpolating these characteristic lengths on the initial mesh (see Section 1.2 [Mesh], page 3). This might sometimes lead to over-refinement in some areas, so that you may have to add "dummy" geometrical entities in the model in order to get the desired element sizes.

   This method works with all the algorithms implemented in the mesh module. The final element sizes are of course constrained by the structured algorithms for which the element sizes are explicitly specified (e.g., transfinite and extruded grids: see Section 4.2.2 [Structured grids], page 47).

2. You can use geometrical "attractors", an elaborate version of the method described in the preceding item: see the definition of the `Attractor` command below.

   Attractors only work with the 2D anisotropic algorithm (see the `Mesh.Algorithm` option in Section 4.3 [Mesh options], page 49).

3. You can give Gmsh an explicit background mesh in the form of a scalar post-processing view (see Section 6.1 [Post-processing commands], page 79, and Chapter 9 [File formats], page 125) in which the nodal values are the target element sizes. This method is very general but it requires a first (usually rough) mesh and a way to compute the target sizes on this mesh (usually through an error estimation procedure, in an iterative process of mesh adaptation). Note that the target element sizes can be constrained by the characteristic lengths defined in the geometrical model if the `Mesh.ConstrainedBackgroundMesh` option is set. To load a background mesh, use the `-bgm` command-line option (see Section 8.3 [Command-line options], page 120) or select 'Apply as background mesh' in the post-processing view option menu.

   Background meshes are supported by all algorithms except the algorithms based on Netgen.

Here are the mesh commands that are related to the specification of characteristic lengths:

`Attractor Point | Line { `*expression-list*` } = { `*expression*`, `*expression*`, `*expression*` };`

Specifies a characteristic length attractor. The *expression-list* should contain the identification numbers of the elementary points or lines to serve as attractors; the two first *expression*s prescribe refinement factors in a coordinate system local to the entities, and the last *expression* a decay factor. This feature is still experimental, and only works with the 2D anisotropic algorithm (see `Mesh.Algorithm` in Section 4.3 [Mesh options], page 49). An example of the use of attractors is given in Section 7.7 [t7.geo], page 113.

Please note that attractors are an *experimental* feature (to be considered *at most* alpha-quality...). Use at your own risk.

`Characteristic Length { `*expression-list*` } = `*expression*`;`
: Modifies the characteristic length of the points whose identification numbers are listed in *expression-list*. The new value is given by *expression*.

4.2.2 Structured grids

`Extrude Point | Line | Surface { `*expression*`, { `*expression-list*` } } { `*layers*`; ... };`
: Extrudes both the geometry and the mesh using a translation (see Section 3.1.5 [Extrusions], page 38). The *layers* option determines how the mesh is extruded and has the following syntax:

  layers:
  `Layers { { `*expression-list*` }, < { `*expression-list*` }, >`
  `{ `*expression-list*` } } | Recombine`

  The first *expression-list* defines how many elements should be created in each extruded layer. The (optional) second *expression-list* assigns a region number to each layer, which, if non-zero, overrides the elementary entity number of the extruded entity. This is useful when there is more than one layer, as the elements in each layer can then be identified in a unique way. If the region number is set to zero, or if the *expression-list* is omitted, the elements are associated with the automatically created elementary geometrical entity (line, surface or volume) created during the extrusion. The last *expression-list* gives the normalized height of each layer (the list should contain a sequence of n numbers $0 < h1 < h2 < \ldots < hn <= 1$). See Section 7.3 [t3.geo], page 99, for an example.

  For line extrusions, the `Recombine` option will recombine triangles into quadrangles when possible. For surface extrusions, the `Recombine` option will recombine tetrahedra into prisms, hexahedra or pyramids.

`Extrude Point | Line | Surface { `*expression*`, { `*expression-list*` }, { `*expression-list*` }, `*expression*` } { `*layers*`; ... };`
: Extrudes both the geometry and the mesh using a rotation (see Section 3.1.5 [Extrusions], page 38). The *layers* option is defined as above.

`Extrude Point | Line | Surface { `*expression*`, { `*expression-list*` }, { `*expression-list*` }, { `*expression-list*` }, `*expression*` } { `*layers*`; ... };`
: Extrudes both the geometry and the mesh using a combined translation and rotation (see Section 3.1.5 [Extrusions], page 38). The *layers* option is defined as above.

`Transfinite Line { `*expression-list*` } = `*expression*` < Using Progression | Bump `*expression*` >;`
: Selects the lines in *expression-list* to be meshed with the 1D transfinite algorithm. The *expression* on the right hand side gives the number of nodes that will be created on the line (this overrides any characteristic length prescription—see Section 4.2.1 [Characteristic lengths], page 46). The optional argument '`Using Progression` *expression*' instructs the transfinite algorithm to distribute the nodes following a geometric progression (`Progression 2` meaning for example that each line element in the series will be twice as long as the preceding

one). The optional argument '`Using Bump` *expression*' instructs the transfinite algorithm to distribute the nodes with a refinement at both ends of the line.

(A deprecated synonym for `Progression` is `Power`.)

`Transfinite Surface {` *expression* `} = {` *expression-list* `};`
Selects the surface *expression* to be meshed with the 2D transfinite algorithm (the surface can only have three or four sides). The *expression-list* should contain the identification numbers of the points on the boundary of the surface. The ordering of these point numbers defines the ordering and orientation of the mesh elements, and should thus follow the node ordering for triangles or quadrangles given in Section 9.3 [Gmsh node ordering], page 134.

`Transfinite Volume {` *expression* `} = {` *expression-list* `};`
Selects the volume *expression* to be meshed with the 3D transfinite algorithm (the volume can only have six or eight faces). The *expression-list* should contain the identification numbers of the points on the boundary of the volume. The ordering of these point numbers defines the ordering and orientation of the mesh elements, and should thus follow the node ordering for prisms or hexahedra given in Section 9.3 [Gmsh node ordering], page 134.

`Elliptic Surface {` *expression* `} = {` *expression-list* `};`
Selects the surface *expression* to be meshed with the 2D elliptic algorithm (the surface can only have four sides). The *expression-list* should contain the identification numbers of the points on the boundary of the surface. The ordering of these point numbers defines the ordering and orientation of the mesh elements, and should thus follow the node ordering for triangles or quadrangles given in Section 9.3 [Gmsh node ordering], page 134.

4.2.3 Miscellaneous

Here is a list of all other mesh commands currently available:

`Color` *color-expression* `{ Point | Line | Surface | Volume {` *expression-list* `}; ... }`
Sets the mesh color of the entities in *expression-list* to *color-expression*.

`Hide { Point | Line | Surface | Volume {` *expression-list* `}; ... }`
Hides the mesh of the entities in *expression-list*, if `General.VisibilityMode` is set to `0` or `2`.

`Hide` *char-expression*`;`
Hides the mesh of the entity *char-expression*, if `General.VisibilityMode` is set to `0` or `2` (*char-expression* can for example be `"*"`).

`Recombine Surface {` *expression-list* `} < =` *expression* `>;`
Recombines the triangular meshes of the surfaces listed in *expression-list* into mixed triangular/quadrangular meshes. The optional *expression* on the right hand side specifies the maximum recombination angle allowed (a large *expression* leading to more triangle recombinations).

`Save` *char-expression*`;`
: Saves the mesh in a file named *char-expression*, using the current `Mesh.Format` (see Section 4.3 [Mesh options], page 49). If the path in *char-expression* is not absolute, *char-expression* is appended to the path of the current file.

`Show { Point | Line | Surface | Volume {` *expression-list* `}; ... }`
: Shows the mesh of the entities in *expression-list*, if `General.VisibilityMode` is set to `0` or `2`.

`Show` *char-expression*`;`
: Shows the mesh of the entity *char-expression*, if `General.VisibilityMode` is set to `0` or `2` (*char-expression* can for example be `"*"`).

4.3 Mesh options

Mesh options control the behavior of mesh commands, as well as the way meshes are displayed in the graphical user interface. For the signification of the 'Saved in:' field in the following list, see Section 2.7 [General options], page 18.

`Mesh.TriangleOptions`
: Options for Jonathan Shewchuk's Triangle isotropic algorithm
Default value: `"praqzBPY"`
Saved in: `General.OptionsFileName`

`Mesh.Algorithm`
: 2D mesh algorithm (1=isotropic, 2=anisotropic, 3=triangle)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.Algorithm3D`
: 3D mesh algorithm (1=isotropic, 4=netgen)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.AllowDegeneratedExtrude`
: Allow the generation of degenerated hexahedra or prisms during extrusion
Default value: `0`
Saved in: `-`

`Mesh.AngleSmoothNormals`
: Threshold angle below which normals are not smoothed
Default value: `180`
Saved in: `General.OptionsFileName`

`Mesh.CharacteristicLengthFactor`
: Factor applied to all characteristic lengths (and background meshes)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.ColorCarousel`
: Mesh coloring (0=by element type, 1=by elementary entity, 2=by physical entity, 3=by partition)

Default value: 1
Saved in: General.OptionsFileName

Mesh.ConstrainedBackgroundMesh
Should the background mesh be constrained by the characteristic lengths associated with the geometry?
Default value: 0
Saved in: General.OptionsFileName

Mesh.CpuTime
CPU time for the generation of the current mesh (in seconds)
Default value: 0
Saved in: -

Mesh.CutPlane
Enable mesh cut plane
Default value: 0
Saved in: -

Mesh.CutPlaneAsSurface
Draw the intersection volume layer as a surface
Default value: 0
Saved in: -

Mesh.CutPlaneOnlyVolume
Cut only the volume elements
Default value: 0
Saved in: -

Mesh.CutPlaneA
First cut plane equation coefficient ('A' in 'AX+BY+CZ+D=0')
Default value: 1
Saved in: -

Mesh.CutPlaneB
Second cut plane equation coefficient ('B' in 'AX+BY+CZ+D=0')
Default value: 0
Saved in: -

Mesh.CutPlaneC
Third cut plane equation coefficient ('C' in 'AX+BY+CZ+D=0')
Default value: 0
Saved in: -

Mesh.CutPlaneD
Fourth cut plane equation coefficient ('D' in 'AX+BY+CZ+D=0')
Default value: 0
Saved in: -

Mesh.Dual
Display the dual mesh obtained by barycentric subdivision
Default value: 0
Saved in: General.OptionsFileName

`Mesh.ElementOrder`
: Element order (1=linear elements, 2=quadratic elements)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.Explode`
: Explode elements (between 0=point and 1=non-transformed)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.Format`
: Mesh output format (1=MSH, 2=UNV, 3=GREF, 19=VRML)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.GammaInf`
: Only display elements whose Gamma factor is greater than GammaInf
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.GammaSup`
: Only display elements whose Gamma factor is smaller than GammaSup
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.Interactive`
: Show the construction of the 2D mesh in real time (only with the 2D anisotropic algorithm)
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.Light`
: Enable lighting for the mesh
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.LightTwoSide`
: Light both sides of mesh elements (leads to slower rendering)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.Lines`
: Display mesh lines (1D elements)?
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.LineNumbers`
: Display mesh line numbers?
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.LineType`
Display mesh lines as solid color segments (0) or 3D cylinders (1)
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.LineWidth`
Display width of mesh lines (in pixels)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.MinimumCirclePoints`
Minimum number of points used to mesh a circle
Default value: `7`
Saved in: `General.OptionsFileName`

`Mesh.MshFileVersion`
MSH mesh file version to generate
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.NbHexahedra`
Number of hexahedra in the current mesh
Default value: `0`
Saved in: `-`

`Mesh.NbNodes`
Number of nodes in the current mesh
Default value: `0`
Saved in: `-`

`Mesh.NbPrisms`
Number of prisms in the current mesh
Default value: `0`
Saved in: `-`

`Mesh.NbPyramids`
Number of pyramids in the current mesh
Default value: `0`
Saved in: `-`

`Mesh.NbQuadrangles`
Number of quadrangles in the current mesh
Default value: `0`
Saved in: `-`

`Mesh.NbTetrahedra`
Number of tetrahedra in the current mesh
Default value: `0`
Saved in: `-`

`Mesh.NbTriangles`
: Number of triangles in the current mesh
Default value: `0`
Saved in: `-`

`Mesh.Normals`
: Display size of normal vectors (in pixels)
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.Optimize`
: Optimize the mesh using Netgen to improve the quality of tetrahedral elements
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.Points`
: Display mesh vertices (nodes)?
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.PointsPerElement`
: Display mesh nodes per element (slower, but permits to visualize only a subset of the nodes)
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.PointInsertion`
: Point insertion method for isotropic 2D algorithm (1=center of circumscribed circle, 2=cog)
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.PointNumbers`
: Display mesh node numbers?
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.PointSize`
: Display size of mesh vertices (in pixels)
Default value: `4`
Saved in: `General.OptionsFileName`

`Mesh.PointType`
: Display mesh vertices as solid color dots (0) or 3D spheres (1)
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.RadiusInf`
: Only display elements whose Radius is greater than RadiusInf
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.RadiusSup`
: Only display elements whose Radius is smaller than RadiusSup
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.RandomFactor`
: Random factor used in 2D and 3D meshing algorithm (test other values when the algorithm fails)
Default value: `0.0001`
Saved in: `General.OptionsFileName`

`Mesh.SaveAll`
: Ignore Physical definitions and save all elements
Default value: `0`
Saved in: `-`

`Mesh.ScalingFactor`
: Global scaling factor applied to the saved mesh
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.Smoothing`
: Number of smoothing steps applied to the final mesh
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.SmoothNormals`
: Smooth the mesh normals?
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.SpeedMax`
: Disable dubious point insertion tests
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.SurfaceEdges`
: Display edges of surface mesh?
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.SurfaceFaces`
: Display faces of surface mesh?
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.SurfaceNumbers`
: Display surface mesh element numbers?
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.Tangents`
: Display size of tangent vectors (in pixels)
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.VertexArrays`
: Use OpenGL vertex arrays to draw triangular meshes?
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.VolumeEdges`
: Display edges of volume mesh?
Default value: `1`
Saved in: `General.OptionsFileName`

`Mesh.VolumeFaces`
: Display faces of volume mesh?
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.VolumeNumbers`
: Display volume mesh element numbers?
Default value: `0`
Saved in: `General.OptionsFileName`

`Mesh.Color.Points`
: Mesh node color
Default value: `{0,255,0}`
Saved in: `General.OptionsFileName`

`Mesh.Color.PointsSup`
: Second order mesh node color
Default value: `{255,0,255}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Lines`
: Mesh line color
Default value: `{0,255,0}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Triangles`
: Mesh triangle color (if Mesh.ColorCarousel=0)
Default value: `{160,150,255}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Quadrangles`
: Mesh quadrangle color (if Mesh.ColorCarousel=0)
Default value: `{130,120,225}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Tetrahedra`
Mesh tetrahedron color (if Mesh.ColorCarousel=0)
Default value: `{160,150,255}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Hexahedra`
Mesh hexahedron color (if Mesh.ColorCarousel=0)
Default value: `{130,120,225}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Prisms`
Mesh prism color (if Mesh.ColorCarousel=0)
Default value: `{232,210,23}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Pyramids`
Mesh pyramid color (if Mesh.ColorCarousel=0)
Default value: `{217,113,38}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Tangents`
Tangent mesh vector color
Default value: `{255,255,0}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Normals`
Normal mesh vector color
Default value: `{255,0,0}`
Saved in: `General.OptionsFileName`

`Mesh.Color.One`
First color in color carousel
Default value: `{232,210,23}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Two`
Second color in color carousel
Default value: `{226,167,29}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Three`
Third color in color carousel
Default value: `{217,113,38}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Four`
Fourth color in color carousel
Default value: `{201,54,54}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Five`
: Fifth color in color carousel
Default value: `{169,12,86}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Six`
: Sixth color in color carousel
Default value: `{114,2,141}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Seven`
: Seventh color in color carousel
Default value: `{67,30,188}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Eight`
: Eighth color in color carousel
Default value: `{44,86,211}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Nine`
: Ninth color in color carousel
Default value: `{32,145,223}`
Saved in: `General.OptionsFileName`

`Mesh.Color.Ten`
: Tenth color in color carousel
Default value: `{25,193,230}`
Saved in: `General.OptionsFileName`

5 Solver module

Five external solvers can be interfaced simultaneously with Gmsh.

If you just want to start a solver from the solver module, with no further interactions between the solver and Gmsh, just edit the options relative to one of the five available solvers (e.g., `Solver.Name0`, `Solver.Executable0`, ...; see Section 5.1 [Solver options], page 59), and set the corresponding "client-server" option to zero (e.g., `Solver.ClientServer0 = 0`). This doesn't require any modification to be made to the solver.

If you want the solver to interact with Gmsh (for error messages, option definitions, post-processing, etc.), you need to link your solver with the '`GmshClient.c`' file and add the appropriate function calls inside your program. You can then proceed as in the previous case, but this time you should set the client-server option to 1 (e.g., `Solver.ClientServer0 = 1`), so that Gmsh and the solver can communicate through a Unix socket. See Section 5.2 [Solver example], page 73, for an example of how to interface a C solver. Bindings for solvers written in other languages (e.g., Perl) are available on http://www.geuz.org/gmsh/.

5.1 Solver options

`Solver.Name0`
: Name of solver 0
Default value: `"GetDP"`
Saved in: `General.OptionsFileName`

`Solver.Help0`
: Help string for solver 0
Default value: `"A General environment for the treatment of Discrete Problems. Copyright (C) 1997-2004 Patrick Dular and Christophe Geuzaine. Visit http://www.geuz.org/getdp/ for more info"`
Saved in: `General.OptionsFileName`

`Solver.Executable0`
: System command to launch solver 0 (should not contain the '&' character)
Default value: `"getdp"`
Saved in: `General.OptionsFileName`

`Solver.Extension0`
: Default file name extension for solver 0
Default value: `".pro"`
Saved in: `General.OptionsFileName`

`Solver.MeshName0`
: Default mesh file name for solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.MeshCommand0`
: Command used to specify the mesh file for solver 0
Default value: `"-msh %s"`
Saved in: `General.OptionsFileName`

`Solver.SocketCommand0`
Command to specify the socket to solver 0
Default value: `"-socket %s"`
Saved in: `General.OptionsFileName`

`Solver.NameCommand0`
Command to specify the problem name to solver 0
Default value: `"%s"`
Saved in: `General.OptionsFileName`

`Solver.OptionCommand0`
Command to get options from solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstOption0`
Label of first option for solver 0
Default value: `"Resolution"`
Saved in: `General.OptionsFileName`

`Solver.SecondOption0`
Label of second option for solver 0
Default value: `"PostOperation"`
Saved in: `General.OptionsFileName`

`Solver.ThirdOption0`
Label of third option for solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FourthOption0`
Label of fourth option for solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FifthOption0`
Label of fifth option for solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstButton0`
Label of first button for solver 0
Default value: `"Pre"`
Saved in: `General.OptionsFileName`

`Solver.FirstButtonCommand0`
Command associated with the first button for solver 0
Default value: `"-pre %s"`
Saved in: `General.OptionsFileName`

`Solver.SecondButton0`
: Label of second button for solver 0
Default value: `"Cal"`
Saved in: `General.OptionsFileName`

`Solver.SecondButtonCommand0`
: Command associated with the second button for solver 0
Default value: `"-cal"`
Saved in: `General.OptionsFileName`

`Solver.ThirdButton0`
: Label of third button for solver 0
Default value: `"Pos"`
Saved in: `General.OptionsFileName`

`Solver.ThirdButtonCommand0`
: Command associated with the third button for solver 0
Default value: `"-bin -pos %s"`
Saved in: `General.OptionsFileName`

`Solver.FourthButton0`
: Label of fourth button for solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FourthButtonCommand0`
: Command associated with the fourth button for solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FifthButton0`
: Label of fifth button for solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FifthButtonCommand0`
: Command associated with the fifth button for solver 0
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Name1`
: Name of solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Help1`
: Help string for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Executable1`
System command to launch solver 1 (should not contain the '&' character)
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Extension1`
Default file name extension for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.MeshName1`
Default mesh file name for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.MeshCommand1`
Command used to specify the mesh file for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SocketCommand1`
Command to specify the socket to solver 1
Default value: `"-socket %s"`
Saved in: `General.OptionsFileName`

`Solver.NameCommand1`
Command to specify the problem name to solver 1
Default value: `"%s"`
Saved in: `General.OptionsFileName`

`Solver.OptionCommand1`
Command to get options from solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstOption1`
Label of first option for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SecondOption1`
Label of second option for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.ThirdOption1`
Label of third option for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FourthOption1`
: Label of fourth option for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthOption1`
: Label of fifth option for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FirstButton1`
: Label of first button for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FirstButtonCommand1`
: Command associated with the first button for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.SecondButton1`
: Label of second button for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.SecondButtonCommand1`
: Command associated with the second button for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.ThirdButton1`
: Label of third button for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.ThirdButtonCommand1`
: Command associated with the third button for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthButton1`
: Label of fourth button for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthButtonCommand1`
: Command associated with the fourth button for solver 1
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthButton1`
Label of fifth button for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FifthButtonCommand1`
Command associated with the fifth button for solver 1
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Name2`
Name of solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Help2`
Help string for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Executable2`
System command to launch solver 2 (should not contain the '&' character)
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Extension2`
Default file name extension for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.MeshName2`
Default mesh file name for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.MeshCommand2`
Command used to specify the mesh file for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SocketCommand2`
Command to specify the socket to solver 2
Default value: `"-socket %s"`
Saved in: `General.OptionsFileName`

`Solver.NameCommand2`
Command to specify the problem name to solver 2
Default value: `"%s"`
Saved in: `General.OptionsFileName`

`Solver.OptionCommand2`
: Command to get options from solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstOption2`
: Label of first option for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SecondOption2`
: Label of second option for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.ThirdOption2`
: Label of third option for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FourthOption2`
: Label of fourth option for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FifthOption2`
: Label of fifth option for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstButton2`
: Label of first button for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstButtonCommand2`
: Command associated with the first button for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SecondButton2`
: Label of second button for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SecondButtonCommand2`
: Command associated with the second button for solver 2
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.ThirdButton2`
Label of third button for solver 2
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.ThirdButtonCommand2`
Command associated with the third button for solver 2
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthButton2`
Label of fourth button for solver 2
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthButtonCommand2`
Command associated with the fourth button for solver 2
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthButton2`
Label of fifth button for solver 2
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthButtonCommand2`
Command associated with the fifth button for solver 2
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.Name3`
Name of solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.Help3`
Help string for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.Executable3`
System command to launch solver 3 (should not contain the '&' character)
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.Extension3`
Default file name extension for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.MeshName3`
: Default mesh file name for solver 3
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.MeshCommand3`
: Command used to specify the mesh file for solver 3
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SocketCommand3`
: Command to specify the socket to solver 3
Default value: `"-socket %s"`
Saved in: `General.OptionsFileName`

`Solver.NameCommand3`
: Command to specify the problem name to solver 3
Default value: `"%s"`
Saved in: `General.OptionsFileName`

`Solver.OptionCommand3`
: Command to get options from solver 3
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstOption3`
: Label of first option for solver 3
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SecondOption3`
: Label of second option for solver 3
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.ThirdOption3`
: Label of third option for solver 3
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FourthOption3`
: Label of fourth option for solver 3
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FifthOption3`
: Label of fifth option for solver 3
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstButton3`
Label of first button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FirstButtonCommand3`
Command associated with the first button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.SecondButton3`
Label of second button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.SecondButtonCommand3`
Command associated with the second button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.ThirdButton3`
Label of third button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.ThirdButtonCommand3`
Command associated with the third button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthButton3`
Label of fourth button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthButtonCommand3`
Command associated with the fourth button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthButton3`
Label of fifth button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthButtonCommand3`
Command associated with the fifth button for solver 3
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.Name4`
: Name of solver 4
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Help4`
: Help string for solver 4
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Executable4`
: System command to launch solver 4 (should not contain the '&' character)
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.Extension4`
: Default file name extension for solver 4
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.MeshName4`
: Default mesh file name for solver 4
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.MeshCommand4`
: Command used to specify the mesh file for solver 4
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SocketCommand4`
: Command to specify the socket to solver 4
Default value: `"-socket %s"`
Saved in: `General.OptionsFileName`

`Solver.NameCommand4`
: Command to specify the problem name to solver 4
Default value: `"%s"`
Saved in: `General.OptionsFileName`

`Solver.OptionCommand4`
: Command to get options from solver 4
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.FirstOption4`
: Label of first option for solver 4
Default value: `""`
Saved in: `General.OptionsFileName`

`Solver.SecondOption4`
Label of second option for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.ThirdOption4`
Label of third option for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthOption4`
Label of fourth option for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthOption4`
Label of fifth option for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FirstButton4`
Label of first button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FirstButtonCommand4`
Command associated with the first button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.SecondButton4`
Label of second button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.SecondButtonCommand4`
Command associated with the second button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.ThirdButton4`
Label of third button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.ThirdButtonCommand4`
Command associated with the third button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthButton4`
Label of fourth button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FourthButtonCommand4`
Command associated with the fourth button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthButton4`
Label of fifth button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.FifthButtonCommand4`
Command associated with the fifth button for solver 4
Default value: ""
Saved in: `General.OptionsFileName`

`Solver.MaximumDelay`
Maximum delay allowed for solver response (in seconds)
Default value: `4`
Saved in: `General.OptionsFileName`

`Solver.Plugins`
Enable default solver plugins?
Default value: `0`
Saved in: `General.OptionsFileName`

`Solver.ClientServer0`
Connect solver 0 to the Gmsh server
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.MergeViews0`
Automatically merge any post-processing view created by solver 0
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.PopupMessages0`
Automatically display messages produced by solver 0
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.ClientServer1`
Connect solver 1 to the Gmsh server
Default value: `0`
Saved in: `General.OptionsFileName`

`Solver.MergeViews1`
Automatically merge any post-processing view created by solver 1
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.PopupMessages1`
Automatically display messages produced by solver 1
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.ClientServer2`
Connect solver 2 to the Gmsh server
Default value: `0`
Saved in: `General.OptionsFileName`

`Solver.MergeViews2`
Automatically merge any post-processing view created by solver 2
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.PopupMessages2`
Automatically display messages produced by solver 2
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.ClientServer3`
Connect solver 3 to the Gmsh server
Default value: `0`
Saved in: `General.OptionsFileName`

`Solver.MergeViews3`
Automatically merge any post-processing view created by solver 3
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.PopupMessages3`
Automatically display messages produced by solver 3
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.ClientServer4`
Connect solver 4 to the Gmsh server
Default value: `0`
Saved in: `General.OptionsFileName`

`Solver.MergeViews4`
Automatically merge any post-processing view created by solver 4
Default value: `1`
Saved in: `General.OptionsFileName`

`Solver.PopupMessages4`
: Automatically display messages produced by solver 4
Default value: `1`
Saved in: `General.OptionsFileName`

5.2 Solver example

Here is a small example of how to interface a C solver with Gmsh. The following listing reproduces the '`utils/solvers/mysolver.c`' file from the Gmsh source distribution.

```
/* $Id: mysolver.c,v 1.5 2004/05/22 01:24:19 geuzaine Exp $ */
/*
 * Copyright (C) 1997-2004 C. Geuzaine, J.-F. Remacle
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without
 * restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, and/or sell copies of the
 * Software, and to permit persons to whom the Software is furnished
 * to do so, provided that the above copyright notice(s) and this
 * permission notice appear in all copies of the Software and that
 * both the above copyright notice(s) and this permission notice
 * appear in supporting documentation.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR
 * ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
 * WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
 * ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
 * OF THIS SOFTWARE.
 *
 *  Please report all bugs and problems to <gmsh@geuz.org>.
 */

/* This file contains a dummy client solver for Gmsh. It does not
   solve anything, but shows how to program your own solver to interact
   with the Gmsh solver module.

   To compile this solver, type something like:

   gcc -o mysolver.exe mysolver.c GmshClient.c
```

```
   To run it, merge the contents of the file mysolver.opt into your
   default Gmsh option file, or lauch Gmsh with the command:

   gmsh -option mysolver.opt

   You will then see a new button labeled "My C solver" in Gmsh's
   solver menu.
*/

/* We start by including some standard headers. Under Windows, you
   will need to install the cygwin tools (http://www.cygwin.com) to
   compile this example (as well as your own solver), since the Gmsh
   solver interface uses Unix sockets. */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>

/* Now we include the Gmsh client interface definitions. At the time
   of this writing, the client interface contains only three
   functions: Gmsh_Connect, Gmsh_SendString and Gmsh_Disconnect. This
   example shows how to use these functions in order to program some
   simple interactions between a solver and Gmsh. */

#include "GmshClient.h"

/* The following typedef defines the two actions of our dummy solver:
   either output some valid option strings, or run a dummy computation
   and output a post-processing map. */

typedef enum { options, run } action;

/* Let's now define some fake CPU intensive functions: */

long worktime()
{
  struct timeval tp;
  gettimeofday(&tp, (struct timezone *)0);
  return (long)tp.tv_sec * 1000000 + (long)tp.tv_usec;
}

void work()
{
```

```
  long t1 = worktime();
  while(1) {
    if(worktime() - t1 > 1.e5)
      break;
  }
}

/* And here we go with the main routine of the solver: */

int main(int argc, char *argv[])
{
  action what = run;
  int i = 0, s;
  char *name = NULL, *option = NULL, *socket = NULL, tmp[256];
  FILE *file;

  /* 1. Loop through the command line arguments, remember the action
     the solver has to perform (what) and store the socket name, the
     option name and the problem name. */

  while(i < argc) {
    if(argv[i][0] == '-') {
      if(!strcmp(argv[i] + 1, "socket")) {
        i++;
        if(argv[i])
          socket = argv[i++];
      }
      else if(!strcmp(argv[i] + 1, "options")) {
        i++;
        what = options;
      }
      else if(!strcmp(argv[i] + 1, "run")) {
        i++;
        what = run;
        if(argv[i])
          option = argv[i++];
      }
    }
    else
      name = argv[i++];
  }

  /* 2. If the '-socket' option was not given, we cannot connect to
        Gmsh... */

  if(!socket) {
    printf("No socket specified: running non-interactively...\n");
```

```
  exit(1);
}

/* 3. Try to connect to the socket given by the '-socket' command
   line option: */

s = Gmsh_Connect(socket);
switch (s) {

  /* 3.1. If the socket is <0, issue an error... */

case -1:
  printf("Couldn't create socket %s\n", socket);
  break;
case -2:
  printf("Couldn't connect to socket %s\n", socket);
  break;
default:

  /* 3.2. ...otherwise, send the GMSH_CLIENT_START command (together
     with the process ID of the solver), check if a problem name was
     specified, and decide what to do according to the 'what'
     variable: */

  sprintf(tmp, "%d", getpid());
  Gmsh_SendString(s, GMSH_CLIENT_START, tmp);
  if(!name) {
    Gmsh_SendString(s, GMSH_CLIENT_ERROR, "Missing file name");
    Gmsh_Disconnect(s);
    exit(1);
  }
  switch (what) {

    /* 3.2.1. If what==options, the solver sends the valid options
       (here for the first option): */

  case options:
    Gmsh_SendString(s, GMSH_CLIENT_OPTION_1, "Val1");
    Gmsh_SendString(s, GMSH_CLIENT_OPTION_1, "Val2");
    Gmsh_SendString(s, GMSH_CLIENT_OPTION_1, "Val3");
    break;

    /* 3.2.2. If what==run, the solver runs the chosen option,
       updates the progress message, issues some information data,
       produces a post-processing map and asks Gmsh to merge it: */

  case run:
```

```
      sprintf(tmp, "Running %s with option %s...", name, option);
      Gmsh_SendString(s, GMSH_CLIENT_INFO, tmp);
      for(i = 0; i < 10; i++) {
        sprintf(tmp, "%d %% complete", 10*i);
        Gmsh_SendString(s, GMSH_CLIENT_PROGRESS, tmp);
        work();
      }
      sprintf(tmp, "Done with %s!", name);
      Gmsh_SendString(s, GMSH_CLIENT_INFO, tmp);
      file = fopen("mysolver.pos", "wb");
      if(!file) {
        Gmsh_SendString(s, GMSH_CLIENT_ERROR, "Unable to open output file");
      }
      else {
        fprintf(file, "View \"%s\"{\n", option);
        fprintf(file, "ST(0,0,0,1,0,0,0,1,0){0,1,2};\n");
        fprintf(file, "};\n");
        fclose(file);
        Gmsh_SendString(s, GMSH_CLIENT_VIEW, "mysolver.pos");
      }
      break;
    }

    /* 3.3. We can now disconnect the solver from Gmsh: */

    Gmsh_SendString(s, GMSH_CLIENT_STOP, "Goodbye!");
    Gmsh_Disconnect(s);
    break;
  }

  /* 4. That's it! */

}
```

To define the above solver as the second external solver in Gmsh, you should define the following solver options (either merge them in your Gmsh option file, or use the `-option` command-line option—see Section 8.3 [Command-line options], page 120):

```
// These options define 'mysolver.exe' as the second solver in Gmsh's
// solver module, under the name 'My C Solver'.
Solver.Name1 = "My C solver";
Solver.Help1 = "A simple example of the client/server
solver implementation in Gmsh...";
Solver.Executable1 = "./mysolver.exe";
Solver.Extension1 = "";
Solver.MeshName1 = "";
```

```
Solver.MeshCommand1 = "";
Solver.SocketCommand1 = "-socket %s";
Solver.NameCommand1 = "%s";
Solver.OptionCommand1 = "-options";
Solver.FirstOption1 = "Option";
Solver.SecondOption1 = "";
Solver.ThirdOption1 = "";
Solver.FourthOption1 = "";
Solver.FifthOption1 = "";
Solver.FirstButton1 = "Run";
Solver.FirstButtonCommand1 = "-run %s";
Solver.SecondButton1 = "";
Solver.SecondButtonCommand1 = "";
Solver.ThirdButton1 = "";
Solver.ThirdButtonCommand1 = "";
Solver.FourthButton1 = "";
Solver.FourthButtonCommand1 = "";
Solver.FifthButton1 = "";
Solver.FifthButtonCommand1 = "";
Solver.ClientServer1 = 1;
Solver.MergeViews1 = 1;
Solver.PopupMessages1 = 1;
```

6 Post-processing module

Gmsh's post-processing module can handle multiple scalar, vector or tensor data sets along with the geometry and the mesh. The data sets should be given in one of Gmsh's post-processing file formats described in Chapter 9 [File formats], page 125. Once loaded into Gmsh, scalar fields can be displayed as iso-value lines and surfaces or color maps, whereas vector fields can be represented either by three-dimensional arrows or by displacement maps. (Tensor fields are currently displayed as Von-Mises effective stresses. To display other (combinations of) components, use `Plugin(Extract)`: see Section 6.2 [Post-processing plugins], page 80.)

In Gmsh's jargon, each data set is called a "view", and can arbitrarily mix all types of elements and fields. Each view is given a name, and can be manipulated either individually (each view has its own button in the GUI and can be referred to by its index in a script) or globally (see the `PostProcessing.Link` option in Section 6.3 [Post-processing options], page 86).

By default, Gmsh treats all post-processing views as three-dimensional plots, i.e., draws the scalar, vector and tensor primitives (points, lines, triangles, tetrahedra, etc.) in 3D space. But Gmsh can also represent each post-processing view containing *scalar points* as two-dimensional ("X-Y") plots, either space- or time-oriented:

- in a '2D space table', the scalar points are taken in the same order as they are defined in the post-processing view: the abscissa of the 2D graph is the curvilinear abscissa of the curve defined by the point series, and only one curve is drawn using the values associated with the points. If several time steps are available, each time step generates a new curve;
- in a '2D time table', one curve is drawn for each scalar point in the view and the abscissa is the time step.

Although visualization is usually mostly an interactive task, Gmsh exposes all the post-processing commands and options to the user in its scripting language to permit a complete automation of the post-processing process (see e.g., Section 7.8 [t8.geo], page 114, and Section 7.9 [t9.geo], page 117).

The two following sections summarize all available post-processing commands and options. Most options apply to both 2D and 3D plots (colormaps, point/line sizes, interval types, time step selection, etc.), but some are peculiar to 3D (lightning, element selection, etc.) or 2D plots (graph style, labels, etc.). Note that 2D plots can be positioned explicitly inside the graphical window, or be automatically positioned in order to avoid overlaps.

Sample post-processing files in human-readable "parsed" format (see Section 9.2.1 [Parsed post-processing file format], page 129) are available in the '`tutorial`' directory of Gmsh's distribution ('`.pos`' files).

6.1 Post-processing commands

`Combine TimeSteps;`
: Combines the data from all the post-processing views having the same name into new multi-time-step views.

`Combine Views;`
: Combines all post-processing views in a single new view.

`Delete View[`*expression*`];`
: Deletes (removes) the *expression*-th post-processing view. Note that post-processing view numbers start at 0.

`Duplicata View[`*expression*`];`
: Duplicates the *expression*-th post-processing view.

  Note that `Duplicata` creates a logical duplicate of the view without actually duplicating the data in memory. This is very useful when you want multiple simultaneous renderings of the same large dataset (usually with different display options), but you cannot afford to store all copies in memory. If what you really want is multiple physical copies of the data, just merge the file containing the post-processing view multiple times.

`Plugin (`*string*`) . Run;`
: Executes the plugin *string*. The list of default plugins is given in Section 6.2 [Post-processing plugins], page 80.

`Plugin (`*string*`) .` *string* `=` *expression* `|` *char-expression*`;`
: Sets an option for a given plugin. See Section 6.2 [Post-processing plugins], page 80, for a list of default plugins and Section 7.9 [t9.geo], page 117, for some examples.

`Save View[`*expression*`]` *char-expression*`;`
: Saves the the *expression*-th post-processing view in a file named *char-expression*. If the path in *char-expression* is not absolute, *char-expression* is appended to the path of the current file.

`View "`*string*`" {` *string* `(` *expression-list* `) {` *expression-list* `}; ... };`
: Creates a new post-processing view, named "*string*". This is the easiest way to create a post-processing view, but also the least efficient (the view is read through Gmsh's script parser, which can become a bit slow if the view is very large—e.g., with one million elements). Nevertheless, this "parsed" post-processing format (explained in detail in Section 9.2.1 [Parsed post-processing file format], page 129) is very powerful, since all the values are *expressions*. Two other formats, better suited for very large data sets, are described in Section 9.2.2 [ASCII post-processing file format], page 130, and Section 9.2.3 [Binary post-processing file format], page 133.

6.2 Post-processing plugins

`Plugin(CutGrid)`
: Plugin(CutGrid) cuts a tetrahedron view with a rectangular grid defined by the 3 points ('X0','Y0','Z0') (origin), ('X1','Y1','Z1') (axis of U) and ('X2','Y2','Z2') (axis of V). The number of points along U and V is set with the options 'nPointsU' and 'nPointsV'. If 'iView' < 0, the plugin is run on the current view.

Plugin(CutGrid) creates one new view.

Numeric options:

`X0` Default value: `-1`

`Y0` Default value: `-1`

`Z0` Default value: `0`

`X1` Default value: `-1`

`Y1` Default value: `0`

`Z1` Default value: `0`

`X2` Default value: `0`

`Y2` Default value: `-1`

`Z2` Default value: `0`

`nPointsU` Default value: `20`

`nPointsV` Default value: `20`

`iView` Default value: `-1`

`Plugin(CutMap)`

Plugin(CutMap) extracts the isovalue surface of value 'A' from the view 'iView' and draws the 'dTimeStep'-th value of the view 'dView' on this isovalue surface. If 'iView' < 0, the plugin is run on the current view. If 'dTimeStep' < 0, the plugin uses, for each time step in 'iView', the corresponding time step in 'dView'. If 'dView' < 0, the plugin uses 'iView' as the value source.

Plugin(CutMap) creates as many views as there are time steps in 'iView'.

Numeric options:

`A` Default value: `1`

`dTimeStep`
Default value: `-1`

`dView` Default value: `-1`

`iView` Default value: `-1`

`recurLevel`
Default value: `4`

`Plugin(CutParametric)`

Plugin(CutParametric) cuts a triangle/tetrahedron scalar view 'iView' with the parametric function ('X'(u), 'Y'(u), 'Z'(u)), using 'nPointsU' values of the parameter u in ['minU', 'maxU']. If 'iView' < 0, the plugin is run on the current view.

Plugin(CutParametric) creates one new view.

String options:

`X` Default value: `"0 + 1 * Cos(u)"`

`Y` Default value: `"0 + 1 * Sin(u)"`

`Z` Default value: `"0"`

Numeric options:

`minU` Default value: `0`

`maxU` Default value: `6.28319`

`nPointsU` Default value: `360`

`iView` Default value: `-1`

`Plugin(CutPlane)`

Plugin(CutPlane) cuts the view 'iView' with the plane 'A'*X + 'B'*Y + 'C'*Z + 'D' = 0. If 'iView' < 0, the plugin is run on the current view.

Plugin(CutPlane) creates one new view.

Numeric options:

`A` Default value: `1`

`B` Default value: `0`

`C` Default value: `0`

`D` Default value: `-0.01`

`iView` Default value: `-1`

`recurLevel`
Default value: `4`

`Plugin(CutSphere)`

Plugin(CutSphere) cuts the view 'iView' with the sphere (X-'Xc')^2 + (Y-'Yc')^2 + (Z-'Zc')^2 = 'R'^2. If 'iView' < 0, the plugin is run on the current view.

Plugin(CutSphere) creates one new view.

Numeric options:

`Xc` Default value: `0`

`Yc` Default value: `0`

`Zc` Default value: `0`

`R` Default value: `0.25`

`iView` Default value: `-1`

`recurLevel`
Default value: `4`

`Plugin(DecomposeInSimplex)`

Plugin(DecomposeInSimplex) decomposes all non-simplectic elements (quadrangles, prisms, hexahedra, pyramids) in the view 'iView' into simplices (triangles, tetrahedra). If 'iView' < 0, the plugin is run on the current view.

Plugin(DecomposeInSimplex) is executed in-place.

Numeric options:

`iView` Default value: `-1`

`Plugin(DisplacementRaise)`

Plugin(DisplacementRaise) transforms the coordinates of the elements in the view 'iView' using the vectorial values (the displacements) stored in the 'dTimeStep'-th time step of the view 'dView'. If 'iView' < 0, the plugin is run on the current view. If 'dView' < 0, the plugin looks for the displacements in the view located just after 'iView' in the view list.

Plugin(DisplacementRaise) is executed in-place.

Numeric options:

`Factor` Default value: `1`

`dTimeStep`
Default value: `0`

`dView` Default value: `-1`

`iView` Default value: `-1`

`Plugin(Evaluate)`

Plugin(Evaluate) sets the 'Component'-th component of the 'TimeStep'-th time step in the view 'iView' to the expression 'Expression'. In addition to the usual mathematical functions (Exp, Log, Sqrt, Sin, Cos, Fabs, etc.) and operators (+, -, *, /, ^), 'Expression' can contain the symbols x, y, z and v, which represent the three spatial coordinates and the value of the field, respectively. If 'iView' < 0, the plugin is run on the current view.

Plugin(Evaluate) is executed in-place.

String options:

`Expression`
Default value: `"0.01*(Fabs(Sin(30*y)*Fabs(Cos(30*x)))+0.3)"`

Numeric options:

`TimeStep` Default value: `0`

`Component`
Default value: `0`

`iView` Default value: `-1`

`Plugin(Extract)`

Plugin(Extract) extracts a combination of components from the view 'iView'. If 'Expression1' or 'Expression2' is empty, the plugin creates a scalar view using 'Expression0'; otherwise the plugin creates a vector view. In addition to the usual mathematical functions (Exp, Log, Sqrt, Sin, Cos, Fabs, etc.) and operators (+, -, *, /, ^), the expressions can contain the symbols v0, v1, v2, ..., vn, which represent the n components of the field, and the symbols x, y and z, which represent the three spatial coordinates. If 'iView' < 0, the plugin is run on the current view.

Plugin(Extract) creates one new view.

String options:

`Expression0`
Default value: `"v0"`

`Expression1`
Default value: `""`

`Expression2`
Default value: `""`

Numeric options:

`iView` Default value: `-1`

`Plugin(HarmonicToTime)`
Plugin(HarmonicToTime) takes the values in the time steps 'realPart' and 'imaginaryPart' of the view 'iView', and creates a new view containing ('iView'['realPart'] * cos(p) - 'iView'['imaginaryPart'] * sin(p)), with p = 2*Pi*k/'nSteps', k = 0, ..., 'nSteps'-1. If 'iView' < 0, the plugin is run on the current view.

Plugin(HarmonicToTime) creates one new view.

Numeric options:

`realPart` Default value: `0`

`imaginaryPart`
Default value: `1`

`nSteps` Default value: `20`

`iView` Default value: `-1`

`Plugin(Skin)`
Plugin(Skin) extracts the skin (the boundary) of the view 'iView'. If 'iView' < 0, the plugin is run on the current view.

Plugin(Skin) creates one new view.

Numeric options:

`iView` Default value: `-1`

`Plugin(Smooth)`
Plugin(Smooth) averages the values at the nodes of the scalar view 'iView'. If 'iView' < 0, the plugin is run on the current view.

Plugin(Smooth) is executed in-place.

Numeric options:

`iView` Default value: `-1`

`Plugin(SphericalRaise)`
Plugin(SphericalRaise) transforms the coordinates of the elements in the view 'iView' using the values associated with the 'TimeStep'-th time step. Instead of elevating the nodes along the X, Y and Z axes as in View['iView'].RaiseX, View['iView'].RaiseY and View['iView'].RaiseZ, the raise is applied along the radius of a sphere centered at ('Xc', 'Yc', 'Zc'). If 'iView' < 0, the plugin is run on the current view.

Plugin(SphericalRaise) is executed in-place.

Numeric options:

`Xc` Default value: `0`

`Yc` Default value: `0`

`Zc` Default value: `0`

`Raise` Default value: `1`

`TimeStep` Default value: `0`

`iView` Default value: `-1`

`Plugin(StreamLines)`

Plugin(StreamLines) computes stream lines from a triangle/tetrahedron vector view 'iView' and optionally interpolates the scalar view 'dView' on the resulting stream lines. It takes as input a grid defined by the 3 points ('X0','Y0','Z0') (origin), ('X1','Y1','Z1') (axis of U) and ('X2','Y2','Z2') (axis of V). The number of points that are going to be transported along U and V is set with the options 'nPointsU' and 'nPointsV'. Then, we solve the equation DX(t)/dt = V(x,y,z) with X(t=0) chosen as the grid and V(x,y,z) interpolated on the vector view. The timestep and the maximum number of iterations are set with the options 'MaxIter' and 'DT'. The time stepping scheme is a RK44. If 'iView' < 0, the plugin is run on the current view.

Plugin(StreamLines) creates one new view. This view contains multi-step vector points if 'dView' < 0, or single-step scalar lines if 'dView' >= 0.

Numeric options:

`X0` Default value: `2.39`

`Y0` Default value: `0.445`

`Z0` Default value: `0`

`X1` Default value: `2.39`

`Y1` Default value: `0.94`

`Z1` Default value: `0`

`X2` Default value: `2.39`

`Y2` Default value: `0.445`

`Z2` Default value: `1`

`nPointsU` Default value: `20`

`nPointsV` Default value: `1`

`MaxIter` Default value: `100`

`DT` Default value: `0.1`

`dView` Default value: `-1`

`iView` Default value: `-1`

`Plugin(Transform)`

Plugin(Transform) transforms the coordinates of the nodes of the view 'iView' by the matrix ['A11' 'A12' 'A13'] ['A21' 'A22' 'A23'] ['A31' 'A32' 'A33']. If 'iView' < 0, the plugin is run on the current view.

Plugin(Transform) is executed in-place.

Numeric options:

`A11` Default value: `1`

`A12` Default value: `0`

`A13` Default value: `0`

`A21` Default value: `0`

`A22` Default value: `1`

`A23` Default value: `0`

`A31` Default value: `0`

`A32` Default value: `0`

`A33` Default value: `1`

`iView` Default value: `-1`

`Plugin(Triangulate)`

Plugin(Triangulate) triangulates the points in the view 'iView', assuming that all the points belong to a surface that can be univoquely projected into a plane. If 'iView' < 0, the plugin is run on the current view.

Plugin(Triangulate) creates one new view.

Numeric options:

`iView` Default value: `-1`

6.3 Post-processing options

General post-processing option names have the form '`PostProcessing.`*string*'. Options peculiar to post-processing views take two forms:

1. options that should apply to all views can be set through '`View.`*string*', *before any view is loaded*;
2. options that should apply only to the n-th view take the form '`View[`n`].`*string*' (n = 0, 1, 2, ...), *after the n-th view is loaded*.

See Section 7.8 [t8.geo], page 114, and Section 7.9 [t9.geo], page 117, for some examples.

`PostProcessing.AnimationDelay`

Delay (in seconds) between frames in automatic animation mode
Default value: `0.25`
Saved in: `General.OptionsFileName`

`PostProcessing.AnimationCycle`
Cycle through views instead of time steps in automatic animation mode
Default value: `0`
Saved in: `General.OptionsFileName`

`PostProcessing.CombineRemoveOriginal`
Remove original views after a Combine operation
Default value: `1`
Saved in: `General.OptionsFileName`

`PostProcessing.Format`
Default file format for post-processing views
Default value: `0`
Saved in: `-`

`PostProcessing.HorizontalScales`
Display value scales horizontally
Default value: `0`
Saved in: `General.OptionsFileName`

`PostProcessing.Link`
Link post-processing views (0=none, 1/2=changes in visible/all, 3/4=everything in visible/all)
Default value: `0`
Saved in: `General.OptionsFileName`

`PostProcessing.NbViews`
Current number of views merged
Default value: `0`
Saved in: `-`

`PostProcessing.Plugins`
Enable default post-processing plugins?
Default value: `1`
Saved in: `General.OptionsFileName`

`PostProcessing.Scales`
Show value scales
Default value: `1`
Saved in: `General.OptionsFileName`

`PostProcessing.Smoothing`
Apply (non-reversible) smoothing to post-processing view when merged
Default value: `0`
Saved in: `General.OptionsFileName`

`PostProcessing.VertexArrays`
Use OpenGL vertex arrays to draw triangles in post-processing views?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.AbscissaName`
Abscissa name for 2D graphs
Default value: `""`
Saved in: `General.OptionsFileName`

`View.AbscissaFormat`
Abscissa number format for 2D graphs (in standard C form)
Default value: `"%.3e"`
Saved in: `General.OptionsFileName`

`View.FileName`
Default post-processing view file name
Default value: `""`
Saved in: `-`

`View.Format`
Number format (in standard C form)
Default value: `"%.3e"`
Saved in: `General.OptionsFileName`

`View.Name`
Default post-processing view name
Default value: `""`
Saved in: `-`

`View.AlphaChannel`
Global alpha channel value (used only if != 1)
Default value: `1`
Saved in: `General.OptionsFileName`

`View.AngleSmoothNormals`
Threshold angle below which normals are not smoothed
Default value: `180`
Saved in: `General.OptionsFileName`

`View.ArrowHeadRadius`
Relative radius of arrow head
Default value: `0.12`
Saved in: `General.OptionsFileName`

`View.ArrowLocation`
Arrow location (1=cog, 2=node)
Default value: `1`
Saved in: `General.OptionsFileName`

`View.ArrowSize`
Display size of arrows (in pixels)
Default value: `60`
Saved in: `General.OptionsFileName`

`View.ArrowSizeProportional`
: Scale the arrows according to the norm of the vector
Default value: `1`
Saved in: `General.OptionsFileName`

`View.ArrowStemLength`
: Relative length of arrow stem
Default value: `0.56`
Saved in: `General.OptionsFileName`

`View.ArrowStemRadius`
: Relative radius of arrow stem
Default value: `0.02`
Saved in: `General.OptionsFileName`

`View.AutoPosition`
: Position the scale or the 2D graph automatically to avoid overlaps
Default value: `1`
Saved in: `General.OptionsFileName`

`View.Boundary`
: Draw the 'N minus b'-dimensional boundary of the simplex (N=simplex dimension, b=option value)
Default value: `0`
Saved in: `General.OptionsFileName`

`View.CustomMax`
: User-defined maximum value to be displayed
Default value: `0`
Saved in: `-`

`View.CustomMin`
: User-defined minimum value to be displayed
Default value: `0`
Saved in: `-`

`View.DisplacementFactor`
: Displacement amplification
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawHexahedra`
: Display post-processing hexahedra?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawLines`
: Display post-processing lines?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawPoints`
Display post-processing points?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawPrisms`
Display post-processing prisms?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawPyramids`
Display post-processing pyramids?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawQuadrangles`
Display post-processing quadrangles?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawScalars`
Display scalar values?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawStrings`
Display post-processing annotation strings?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawTensors`
Display tensor values?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawTetrahedra`
Display post-processing tetrahedra?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawTriangles`
Display post-processing triangles?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.DrawVectors`
Display vector values?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.Explode`
Explode elements (between 0=point and 1=non-transformed)
Default value: `1`
Saved in: `General.OptionsFileName`

`View.ExternalView`
Index of the view used to color vector fields (-1=self)
Default value: `-1`
Saved in: `General.OptionsFileName`

`View.Grid`
Grid mode for 2D graphs (0=none, 1=simple, 2=frame, 3=grid)
Default value: `2`
Saved in: `General.OptionsFileName`

`View.Height`
Height (in pixels) of the scale or 2D graph
Default value: `200`
Saved in: `General.OptionsFileName`

`View.IntervalsType`
Type of interval display (1=iso, 2=continuous, 3=discrete, 4=numeric)
Default value: `2`
Saved in: `General.OptionsFileName`

`View.Light`
Enable lighting for the view
Default value: `1`
Saved in: `General.OptionsFileName`

`View.LightTwoSide`
Light both sides of view elements (leads to slower rendering)
Default value: `1`
Saved in: `General.OptionsFileName`

`View.LineType`
Display lines as solid color segments (0) or 3D cylinders (1)
Default value: `0`
Saved in: `General.OptionsFileName`

`View.LineWidth`
Display width of lines (in pixels)
Default value: `1`
Saved in: `General.OptionsFileName`

`View.Max` Maximum value in the view (do not change this!)
Default value: `-1e+200`
Saved in: `-`

`View.Min` Minimum value in the view (do not change this!)
Default value: `1e+200`
Saved in: `-`

`View.NbAbscissa`
Number of abscissa intervals for 2D graphs
Default value: `5`
Saved in: `General.OptionsFileName`

`View.NbIso`
Number of intervals
Default value: `15`
Saved in: `General.OptionsFileName`

`View.NbTimeStep`
Number of time steps in the view (do not change this!)
Default value: `1`
Saved in: `-`

`View.OffsetX`
Translation of the view along X-axis (in model coordinates)
Default value: `0`
Saved in: `-`

`View.OffsetY`
Translation of the view along Y-axis (in model coordinates)
Default value: `0`
Saved in: `-`

`View.OffsetZ`
Translation of the view along Z-axis (in model coordinates)
Default value: `0`
Saved in: `-`

`View.PointSize`
Display size of points (in pixels)
Default value: `3`
Saved in: `General.OptionsFileName`

`View.PointType`
Display points as solid color dots (0) or 3D spheres (1)
Default value: `0`
Saved in: `General.OptionsFileName`

`View.PositionX`
Horizontal position (in pixels) of the upper left corner of the scale or 2D graph
Default value: `100`
Saved in: `General.OptionsFileName`

`View.PositionY`
Vertical position (in pixels) of the upper left corner of the scale or 2D graph
Default value: `50`
Saved in: `General.OptionsFileName`

`View.RaiseX`
Elevation of the view along X-axis (in model coordinates)
Default value: `0`
Saved in: `-`

`View.RaiseY`
Elevation of the view along Y-axis (in model coordinates)
Default value: `0`
Saved in: `-`

`View.RaiseZ`
Elevation of the view along Z-axis (in model coordinates)
Default value: `0`
Saved in: `-`

`View.RangeType`
Value scale range type (1=default, 2=custom, 3=per time step)
Default value: `1`
Saved in: `General.OptionsFileName`

`View.SaturateValues`
Saturate the view values to custom min and max (1=true, 0=false)
Default value: `0`
Saved in: `General.OptionsFileName`

`View.ScaleType`
Value scale type (1=linear, 2=logarithmic, 3=double logarithmic)
Default value: `1`
Saved in: `General.OptionsFileName`

`View.ShowElement`
Show element boundaries?
Default value: `0`
Saved in: `General.OptionsFileName`

`View.ShowScale`
Show value scale?
Default value: `1`
Saved in: `General.OptionsFileName`

`View.ShowTime`
Show time value in the scale? (1=only if NbTimeStep>1, 2=always)
Default value: `1`
Saved in: `General.OptionsFileName`

`View.SmoothNormals`
Smooth the normals?
Default value: `0`
Saved in: `General.OptionsFileName`

`View.TensorType`
: Tensor Visualization Type
Default value: `0`
Saved in: `General.OptionsFileName`

`View.TimeStep`
: Current time step displayed
Default value: `0`
Saved in: `-`

`View.Type`
: Type of graph (1=3D, 2=2D-space, 3=2D-time)
Default value: `1`
Saved in: `-`

`View.VectorType`
: Vector display type (1=segment, 2=arrow, 3=pyramid, 4=3D arrow, 5=displacement)
Default value: `4`
Saved in: `General.OptionsFileName`

`View.Visible`
: Is the view visible?
Default value: `1`
Saved in: `-`

`View.Width`
: Width (in pixels) of the scale or 2D graph
Default value: `300`
Saved in: `General.OptionsFileName`

`View.ColorTable`
: Color table used to draw the view
Saved in: `General.OptionsFileName`

7 Tutorial

The nine following examples introduce new features gradually, starting with '`t1.geo`'. The files corresponding to these examples are available in the '`tutorial`' directory of the Gmsh distribution.

This tutorial does not explain the mesh and post-processing file formats: see Chapter 9 [File formats], page 125, for this.

To learn how to run Gmsh on your computer, see Chapter 8 [Running Gmsh], page 119.

7.1 't1.geo'

```
/*********************************************************************
 *
 *  Gmsh tutorial 1
 *
 *  Variables, elementary entities (points, lines, surfaces), physical
 *  entities (points, lines, surfaces), background mesh
 *
 *********************************************************************/

// The simplest construction of Gmsh's scripting language is the
// 'affectation'. The following command defines a new variable 'lc':

lc = 0.009;

// This variable can then for example be used in the definition of
// Gmsh's simplest 'elementary entity', a 'Point'. A Point is defined
// by a list of four numbers: its three coordinates (X, Y and Z), and
// a characteristic length which sets the target element size at the
// point:

Point(1) = {0, 0, 0, 9.e-1 * lc};

// The actual distribution of the mesh element sizes is then obtained
// by interpolation of these characteristic lengths throughout the
// geometry. There are also other possibilities to specify
// characteristic lengths: attractors (see 't7.geo') and background
// meshes (see 'bgmesh.pos').

// As can be seen in the previous definition, more complex expressions
// can be constructed from variables and floating point
// constants. Here, the product of the variable 'lc' by the constant
// 9.e-1 is given as the fourth argument of the list defining the
// point.

// We can then define some additional points as well as our first
```

```
// curve.  Curves are Gmsh's second type of elementery entities, and,
// amongst curves, straight lines are the simplest. A straight line is
// defined by a list of point numbers. In the commands below, for
// example, the line 1 starts at point 1 and ends at point 2:

Point(2) = {.1, 0,  0, lc} ;
Point(3) = {.1, .3, 0, lc} ;
Point(4) = {0,  .3, 0, lc} ;

Line(1) = {1,2} ;
Line(2) = {3,2} ;
Line(3) = {3,4} ;
Line(4) = {4,1} ;

// The third elementary entity is the surface. In order to define a
// simple rectangular surface from the four lines defined above, a
// line loop has first to be defined. A line loop is a list of
// connected lines, a sign being associated with each line (depending
// on the orientation of the line):

Line Loop(5) = {4,1,-2,3} ;

// We can then define the surface as a list of line loops (only one
// here, since there are no holes--see 't4.geo'):

Plane Surface(6) = {5} ;

// At this level, Gmsh knows everything to display the rectangular
// surface 6 and to mesh it. An optional step is needed if we want to
// associate specific region numbers to the various elements in the
// mesh (e.g. to the line segments discretizing lines 1 to 4 or to the
// triangles discretizing surface 6). This is achieved by the
// definition of 'physical entities'. Physical entities will group
// elements belonging to several elementary entities by giving them a
// common number (a region number), and specifying their orientation.

// We can for example group the points 1 and 2 into the physical
// entity 1:

Physical Point(1) = {1,2} ;

// Consequently, two punctual elements will be saved in the output
// files, both with the region number 1. The mechanism is identical
// for line or surface elements:

Physical Line(10) = {1,2,4} ;
```

```
MySurface = 100;
Physical Surface(MySurface) = {6} ;

// All the line elements created during the meshing of lines 1, 2 and
// 4 will be saved in the output file with the region number 10; and
// all the triangular elements resulting from the discretization of
// surface 6 will be given the region number 100.

// Note that, if no physical entities are defined, all the elements in
// the mesh will be directly saved with their default orientation and
// with a region number equal to the number of the elementary entity
// they discretize.
```

7.2 't2.geo'

```
/*********************************************************************
 *
 *  Gmsh tutorial 2
 *
 *  Includes, geometrical transformations, extruded geometries,
 *  elementary entities (volumes), physical entities (volumes)
 *
 *********************************************************************/

// We first include the previous tutorial file, in order to use it as
// a basis for this one:

Include "t1.geo";

// We can then add new points and lines in the same way as we did in
// 't1.geo':

Point(5) = {0, .4, 0, lc};
Line(5) = {4, 5};

// But Gmsh also provides tools to tranform (translate, rotate, etc.)
// elementary entities or copies of elementary entities. For example,
// the point 3 can be moved by 0.05 units to the left with:

Translate {-0.05, 0, 0} { Point{3}; }

// The resulting point can also be duplicated and translated by 0.1
// along the y axis:

tmp[] = Translate {0, 0.1, 0} { Duplicata{ Point{3}; } } ;
```

```
// In this case, we assigned the result of the Translate command to a
// list, so that we can retrieve the number of the newly created point
// and use it to create new lines and a new surface:

Line(7) = {3,tmp[0]};
Line(8) = {tmp[0],5};
Line Loop(10) = {5,-8,-7,3};
Plane Surface(11) = {10};

// Of course, these transformation commands not only apply to points,
// but also to lines and surfaces. We can for example translate a copy
// of surface 6 by 0.12 units along the z axis and define some
// additional lines and surfaces with:

h = 0.12;
Translate {0, 0, h} { Duplicata{ Surface{6}; } }

Line(106) = {1,8};
Line(107) = {2,12};
Line(108) = {3,16};
Line(109) = {4,7};

Line Loop(110) = {1,107,-103,-106}; Plane Surface(111) = {110};
Line Loop(112) = {2,107,104,-108};  Plane Surface(113) = {112};
Line Loop(114) = {3,109,-105,-108}; Plane Surface(115) = {114};
Line Loop(116) = {4,106,-102,-109}; Plane Surface(117) = {116};

// Volumes are the fourth type of elementary entities in Gmsh. In the
// same way one defines line loops to build surfaces, one has to
// define surface loops (i.e. 'shells') to build volumes. The
// following volume does not have holes and thus consists of a single
// surface loop:

Surface Loop(118) = {117,-6,111,-113,101,115};
Volume(119) = {118};

// Another way to define a volume is by extruding a surface. The
// following command extrudes the surface 11 along the z axis and
// automatically creates a new volume:

Extrude Surface { 11, {0, 0, h} };

// All these geometrical transformations automatically generate new
// elementary entities. The following command permits to manually
// specify a characteristic length for some of the new points:

Characteristic Length {tmp[0], 2, 12, 3, 16, 6, 22} = lc * 4;
```

```
// Note that, if the transformation tools are handy to create complex
// geometries, it is also sometimes useful to generate the 'flat'
// geometry, with an explicit list of all elementary entities. This
// can be achieved by selecting the 'File->Save as->Gmsh unrolled
// geometry' menu or by typing
//
// > gmsh t2.geo -0
//
// on the command line.

// To save all the tetrahedra discretizing the volumes 119 and 120
// with a common region number, we finally define a physical
// volume:

Physical Volume (1) = {119,120};
```

7.3 't3.geo'

```
/*********************************************************************
 *
 *  Gmsh tutorial 3
 *
 *  Extruded meshes, options
 *
 *********************************************************************/

// Again, we start by including the first tutorial:

Include "t1.geo";

// As in 't2.geo', we plan to perform an extrusion along the z axis.
// But here, instead of only extruding the geometry, we also want to
// extrude the 2D mesh. This is done with the same 'Extrude' command,
// but by specifying the number of layers (4 in this case, with 8, 4,
// 2 and 1 subdivisions, respectively), with volume numbers 9000 to
// 9003 and respective heights equal to h/4:

h = 0.1;

Extrude Surface { 6, {0,0,h} } {
  Layers { {8,4,2,1}, {9000:9003}, {0.25,0.5,0.75,1} };
};

// The extrusion can also be performed with a rotation instead of a
// translation, and the resulting mesh can be recombined into prisms
```

```
// (wedges). All rotations are specified by an axis direction
// ({0,1,0}), an axis point ({-0.1,0,0.1}) and a rotation angle
// (-Pi/2):

Extrude Surface { 122, {0,1,0} , {-0.1,0,0.1} , -Pi/2 } {
  Recombine; Layers { 7, 9004, 1 };
};

// Note that a translation ({-2*h,0,0}) and a rotation ({1,0,0},
// {0,0.15,0.25}, Pi/2) can also be combined:

aa[] = Extrude Surface {news-1, {-2*h,0,0}, {1,0,0} , {0,0.15,0.25} , Pi/2}{
  Layers { 10, 1 }; Recombine;
}; ;

// In this last extrusion command we didn't specify an explicit
// volume number (which is equivalent to setting it to "0"),
// which means that the elements will simply belong the automatically
// created volume (whose number we get from the aa[] list).

// We finally define a new physical volume to save all the tetrahedra
// with a common region number (101):

Physical Volume(101) = {9000:9004, aa[1]};

// Let us now change some options... Since all interactive options are
// accessible in Gmsh's scripting language, we can for example define
// a global characteristic length factor, redefine some background
// colors, disable the display of the axes, and select an initial
// viewpoint in XYZ mode (disabling the interactive trackball-like
// rotation mode) directly in the input file:

Mesh.CharacteristicLengthFactor = 4;
General.Color.Background = {120,120,120};
General.Color.Foreground = {255,255,255};
General.Color.Text = White;
Geometry.Color.Points = Orange;
General.Axes = 0;
General.Trackball = 0;
General.RotationCenterGravity = 0;
General.RotationCenterX = 0;
General.RotationCenterY = 0;
General.RotationCenterZ = 0;
General.RotationX = 10;
General.RotationY = 70;
General.TranslationX = -0.2;
```

```
// Note that all colors can be defined literally or numerically, i.e.
// 'General.Color.Background = Red' is equivalent to
// 'General.Color.Background = {255,0,0}'; and also note that, as with
// user-defined variables, the options can be used either as right or
// left hand sides, so that the following command will set the surface
// color to the same color as the points:

Geometry.Color.Surfaces = Geometry.Color.Points;

// You can click on the '?'  button in the status bar of the graphic
// window to see the current values of all options. To save all the
// options in a file, you can use the 'File->Save as->Gmsh options'
// menu. To save the current options as the default options for all
// future Gmsh sessions, you should use the 'Tools->Options->Save'
// button.
```

7.4 't4.geo'

```
/*********************************************************************
 *
 *  Gmsh tutorial 4
 *
 *  Built-in functions, holes, strings, mesh color
 *
 *********************************************************************/

// As usual, we start by defining some variables, some points and some
// lines:

cm = 1e-02;

e1 = 4.5*cm; e2 = 6*cm / 2; e3 =  5*cm / 2;

h1 = 5*cm; h2 = 10*cm; h3 = 5*cm; h4 = 2*cm; h5 = 4.5*cm;

R1 = 1*cm; R2 = 1.5*cm; r = 1*cm;

ccos = ( -h5*R1 + e2 * Hypot(h5,Hypot(e2,R1)) ) / (h5^2 + e2^2);
ssin = Sqrt(1-ccos^2);

Lc1 = 0.01;
Lc2 = 0.003;

Point(1) = { -e1-e2, 0.0  , 0.0 , Lc1};
Point(2) = { -e1-e2, h1   , 0.0 , Lc1};
Point(3) = { -e3-r , h1   , 0.0 , Lc2};
```

```
Point(4) = { -e3-r , h1+r , 0.0 , Lc2};
Point(5) = { -e3   , h1+r , 0.0 , Lc2};
Point(6) = { -e3   , h1+h2, 0.0 , Lc1};
Point(7) = {  e3   , h1+h2, 0.0 , Lc1};
Point(8) = {  e3   , h1+r , 0.0 , Lc2};
Point(9) = {  e3+r , h1+r , 0.0 , Lc2};
Point(10)= {  e3+r , h1   , 0.0 , Lc2};
Point(11)= {  e1+e2, h1   , 0.0 , Lc1};
Point(12)= {  e1+e2, 0.0  , 0.0 , Lc1};
Point(13)= {  e2   , 0.0  , 0.0 , Lc1};

Point(14)= {  R1 / ssin , h5+R1*ccos, 0.0 , Lc2};
Point(15)= {  0.0       , h5        , 0.0 , Lc2};
Point(16)= { -R1 / ssin , h5+R1*ccos, 0.0 , Lc2};
Point(17)= { -e2        , 0.0       , 0.0 , Lc1};

Point(18)= { -R2  , h1+h3   , 0.0 , Lc2};
Point(19)= { -R2  , h1+h3+h4, 0.0 , Lc2};
Point(20)= {  0.0 , h1+h3+h4, 0.0 , Lc2};
Point(21)= {  R2  , h1+h3+h4, 0.0 , Lc2};
Point(22)= {  R2  , h1+h3   , 0.0 , Lc2};
Point(23)= {  0.0 , h1+h3   , 0.0 , Lc2};

Point(24)= {  0 , h1+h3+h4+R2, 0.0 , Lc2};
Point(25)= {  0 , h1+h3-R2,    0.0 , Lc2};

Line(1)  = {1 ,17};
Line(2)  = {17,16};

// Since not all curves are straight lines, Gmsh provides many other
// curve primitives: splines, B-splines, circle arcs, ellipse arcs,
// etc. Here we define a new circle arc, starting at point 14 and
// ending at point 16, with the circle's center being the point 15:

Circle(3) = {14,15,16};

// Note that, in Gmsh, circle arcs should always be stricly smaller
// than Pi. We can then define additional lines and circles, as well
// as a new surface:

Line(4)  = {14,13};
Line(5)  = {13,12};
Line(6)  = {12,11};
Line(7)  = {11,10};
Circle(8) = { 8, 9,10};
Line(9)  = { 8, 7};
Line(10) = { 7, 6};
```

```
Line(11) = { 6, 5};
Circle(12) = { 3, 4, 5};
Line(13) = { 3, 2};
Line(14) = { 2, 1};
Line(15) = {18,19};
Circle(16) = {21,20,24};
Circle(17) = {24,20,19};
Circle(18) = {18,23,25};
Circle(19) = {25,23,22};
Line(20) = {21,22};

Line Loop(21) = {17,-15,18,19,-20,16};
Plane Surface(22) = {21};

// But we still need to define the exterior surface. Since it has a
// hole, its definition now requires two lines loops:

Line Loop(23) = {11,-12,13,14,1,2,-3,4,5,6,7,-8,9,10};
Plane Surface(24) = {23,21};

// Finally, we can add some comments by embedding a post-processing
// view containing some strings, and change the color of some mesh
// entities:

View "comments" {
  // 10 pixels from the left and 15 pixels from the top of the graphic
  // window:
  T2(10,15,0){"File created on Fri Oct 18 23:50:20 2002"};

  // 10 pixels from the left and 10 pixels from the bottom of the
  // graphic window:
  T2(10,-10,0){"Copyright (C) My Company"};

  // in the model, at (X,Y,Z) = (0.0,0.11,0.0):
  T3(0,0.11,0,0){"Hole"};
};

Color Grey70{ Surface{ 22 }; }
Color Purple{ Surface{ 24 }; }
Color Red{ Line{ 1:14 }; }
Color Yellow{ Line{ 15:20 }; }
```

7.5 ‘t5.geo’

```
/*********************************************************************
 *
```

```
 *  Gmsh tutorial 5
 *
 *  Characteristic lengths, arrays of variables, functions, loops
 *
 ********************************************************************/

// Again, we start be defining some characteristic lengths:

lcar1 = .1;
lcar2 = .0005;
lcar3 = .055;

// If we wanted to change these lengths globally (without changing the
// above definitions), we could give a global scaling factor for all
// characteristic lengths on the command line with the '-clscale'
// option (or with 'Mesh.CharacteristicLengthFactor' in an option
// file). For example, with:
//
// > gmsh t5 -clscale 1
//
// this input file produces a mesh of approximately 2,500 nodes and
// 13,000 tetrahedra (in 4 seconds on a 1.2GHz PC). With
//
// > gmsh t5 -clscale 0.2
//
// (i.e. with all characteristic lengths divided by 5), the mesh
// counts approximately 260,000 nodes and 1.6 million tetrahedra (and
// the computation takes 16 minutes on the same machine).

// Let us proceed by defining some elementary entities describing a
// truncated cube:

Point(1) = {0.5,0.5,0.5,lcar2}; Point(2) = {0.5,0.5,0,lcar1};
Point(3) = {0,0.5,0.5,lcar1};   Point(4) = {0,0,0.5,lcar1};
Point(5) = {0.5,0,0.5,lcar1};   Point(6) = {0.5,0,0,lcar1};
Point(7) = {0,0.5,0,lcar1};     Point(8) = {0,1,0,lcar1};
Point(9) = {1,1,0,lcar1};       Point(10) = {0,0,1,lcar1};
Point(11) = {0,1,1,lcar1};      Point(12) = {1,1,1,lcar1};
Point(13) = {1,0,1,lcar1};      Point(14) = {1,0,0,lcar1};

Line(1) = {8,9};    Line(2) = {9,12};  Line(3) = {12,11};
Line(4) = {11,8};   Line(5) = {9,14};  Line(6) = {14,13};
Line(7) = {13,12};  Line(8) = {11,10}; Line(9) = {10,13};
Line(10) = {10,4};  Line(11) = {4,5};  Line(12) = {5,6};
Line(13) = {6,2};   Line(14) = {2,1};  Line(15) = {1,3};
Line(16) = {3,7};   Line(17) = {7,2};  Line(18) = {3,4};
Line(19) = {5,1};   Line(20) = {7,8};  Line(21) = {6,14};
```

```
Line Loop(22) = {-11,-19,-15,-18};   Plane Surface(23) = {22};
Line Loop(24) = {16,17,14,15};       Plane Surface(25) = {24};
Line Loop(26) = {-17,20,1,5,-21,13}; Plane Surface(27) = {26};
Line Loop(28) = {-4,-1,-2,-3};       Plane Surface(29) = {28};
Line Loop(30) = {-7,2,-5,-6};        Plane Surface(31) = {30};
Line Loop(32) = {6,-9,10,11,12,21};  Plane Surface(33) = {32};
Line Loop(34) = {7,3,8,9};           Plane Surface(35) = {34};
Line Loop(36) = {-10,18,-16,-20,4,-8}; Plane Surface(37) = {36};
Line Loop(38) = {-14,-13,-12,19};    Plane Surface(39) = {38};

// Instead of using included files, let us now use a user-defined
// function in order to carve some holes in the cube:

Function CheeseHole

  // In the following commands we use the reserved variable name
  // 'newp', which automatically selects a new point number. This
  // number is chosen as the highest current point number, plus
  // one. (Note that, analogously to 'newp', the variables 'newc',
  // 'news', 'newv' and 'newreg' select the highest number amongst
  // currently defined curves, surfaces, volumes and 'any entities
  // other than points', respectively.)

  p1 = newp; Point(p1) = {x,  y,  z,  lcar3} ;
  p2 = newp; Point(p2) = {x+r,y,  z,  lcar3} ;
  p3 = newp; Point(p3) = {x,  y+r,z,  lcar3} ;
  p4 = newp; Point(p4) = {x,  y,  z+r,lcar3} ;
  p5 = newp; Point(p5) = {x-r,y,  z,  lcar3} ;
  p6 = newp; Point(p6) = {x,  y-r,z,  lcar3} ;
  p7 = newp; Point(p7) = {x,  y,  z-r,lcar3} ;

  c1 = newreg; Circle(c1) = {p2,p1,p7};
  c2 = newreg; Circle(c2) = {p7,p1,p5};
  c3 = newreg; Circle(c3) = {p5,p1,p4};
  c4 = newreg; Circle(c4) = {p4,p1,p2};
  c5 = newreg; Circle(c5) = {p2,p1,p3};
  c6 = newreg; Circle(c6) = {p3,p1,p5};
  c7 = newreg; Circle(c7) = {p5,p1,p6};
  c8 = newreg; Circle(c8) = {p6,p1,p2};
  c9 = newreg; Circle(c9) = {p7,p1,p3};
  c10 = newreg; Circle(c10) = {p3,p1,p4};
  c11 = newreg; Circle(c11) = {p4,p1,p6};
  c12 = newreg; Circle(c12) = {p6,p1,p7};

  // We need non-plane surfaces to define the spherical cheese
  // holes. Here we use ruled surfaces, which can have 3 or 4
```

```
  // sides:

  l1 = newreg; Line Loop(l1) = {c5,c10,c4};   Ruled Surface(newreg) = {l1};
  l2 = newreg; Line Loop(l2) = {c9,-c5,c1};   Ruled Surface(newreg) = {l2};
  l3 = newreg; Line Loop(l3) = {c12,-c8,-c1}; Ruled Surface(newreg) = {l3};
  l4 = newreg; Line Loop(l4) = {c8,-c4,c11};  Ruled Surface(newreg) = {l4};
  l5 = newreg; Line Loop(l5) = {-c10,c6,c3};  Ruled Surface(newreg) = {l5};
  l6 = newreg; Line Loop(l6) = {-c11,-c3,c7}; Ruled Surface(newreg) = {l6};
  l7 = newreg; Line Loop(l7) = {-c2,-c7,-c12};Ruled Surface(newreg) = {l7};
  l8 = newreg; Line Loop(l8) = {-c6,-c9,c2};  Ruled Surface(newreg) = {l8};

  // Please note that all surface meshes are generated by projecting a
  // 2D planar mesh onto the surface, and that this method gives nice
  // results only if the surface's curvature is small enough. If not,
  // you will have to cut the surface in pieces.

  // We then use an array of variables to store the surface loops
  // identification numbers for later reference (we will need these to
  // define the final volume):

  theloops[t] = newreg ;

  Surface Loop(theloops[t]) = {l8+1,l5+1,l1+1,l2+1,l3+1,l7+1,l6+1,l4+1};

  thehole = newreg ;
  Volume(thehole) = theloops[t] ;

Return

// We can use a 'For' loop to generate five holes in the cube:

x = 0 ; y = 0.75 ; z = 0 ; r = 0.09 ;

For t In {1:5}

  x += 0.166 ;
  z += 0.166 ;

  Call CheeseHole ;

  // We define a physical volume for each hole:

  Physical Volume (t) = thehole ;

  // We also print some variables on the terminal (note that, since
  // all variables are treated internally as floating point numbers,
  // the format string should only contain valid floating point format
```

```
  // specifiers):

  Printf("Hole %g (center = {%g,%g,%g}, radius = %g) has number %g!",
         t, x, y, z, r, thehole) ;

EndFor

// We can then define the surface loop for the exterior surface of the
// cube:

theloops[0] = newreg ;

Surface Loop(theloops[0]) = {35,31,29,37,33,23,39,25,27} ;

// The volume of the cube, without the 5 cheese holes, is now defined
// by 6 surface loops (the exterior surface and the five interior
// loops).  To reference an array of variables, its identifier is
// followed by '[]':

Volume(186) = {theloops[]} ;

// We finally define a physical volume for the elements discretizing
// the cube, without the holes (whose elements were already tagged
// with numbers 1 to 5 in the `For' loop):

Physical Volume (10) = 186 ;
```

7.6 `t6.geo`

```
/*********************************************************************
 *
 *  Gmsh tutorial 6
 *
 *  Transfinite meshes
 *
 *********************************************************************/

// We start by defining a more complex geometry, using the same
// commands as in the previous examples:

r_int  = 0.05 ;
r_ext  = 0.051 ;
r_far  = 0.125 ;
r_inf  = 0.4 ;
phi1   = 30. * (Pi/180.) ;
angl   = 45. * (Pi/180.) ;
```

```
nbpt_phi   = 5 ; nbpt_int   = 20 ;
nbpt_arc1  = 10 ; nbpt_arc2  = 10 ;
nbpt_shell = 10 ; nbpt_far   = 25 ; nbpt_inf = 15 ;

lc0 = 0.1 ; lc1 = 0.1 ; lc2 = 0.3 ;

Point(1) = {0,     0, 0, lc0} ;
Point(2) = {r_int, 0, 0, lc0} ;
Point(3) = {r_ext, 0, 0, lc1} ;
Point(4) = {r_far, 0, 0, lc2} ;
Point(5) = {r_inf, 0, 0, lc2} ;
Point(6) = {0, 0,  r_int, lc0} ;
Point(7) = {0, 0,  r_ext, lc1} ;
Point(8) = {0, 0,  r_far, lc2} ;
Point(9) = {0, 0,  r_inf, lc2} ;

Point(10) = {r_int*Cos(phi1), r_int*Sin(phi1), 0, lc0} ;
Point(11) = {r_ext*Cos(phi1), r_ext*Sin(phi1), 0, lc1} ;
Point(12) = {r_far*Cos(phi1), r_far*Sin(phi1), 0, lc2} ;
Point(13) = {r_inf*Cos(phi1), r_inf*Sin(phi1), 0, lc2} ;

Point(14) = {r_int/2, 0, 0, lc2} ;
Point(15) = {r_int/2*Cos(phi1), r_int/2*Sin(phi1), 0, lc2} ;
Point(16) = {r_int/2, 0, r_int/2, lc2} ;
Point(17) = {r_int/2*Cos(phi1), r_int/2*Sin(phi1), r_int/2, lc2} ;
Point(18) = {0, 0, r_int/2, lc2} ;
Point(19) = {r_int*Cos(angl), 0, r_int*Sin(angl), lc2} ;
Point(20) = {r_int*Cos(angl)*Cos(phi1), r_int*Cos(angl)*Sin(phi1),
             r_int*Sin(angl), lc2} ;
Point(21) = {r_ext*Cos(angl), 0, r_ext*Sin(angl), lc2} ;
Point(22) = {r_ext*Cos(angl)*Cos(phi1), r_ext*Cos(angl)*Sin(phi1),
             r_ext*Sin(angl), lc2} ;
Point(23) = {r_far*Cos(angl), 0, r_far*Sin(angl), lc2} ;
Point(24) = {r_far*Cos(angl)*Cos(phi1), r_far*Cos(angl)*Sin(phi1),
             r_far*Sin(angl), lc2} ;
Point(25) = {r_inf, 0, r_inf, lc2} ;
Point(26) = {r_inf*Cos(phi1), r_inf*Sin(phi1),  r_inf, lc2} ;

Circle(1) = {2,1,19};  Circle(2) = {19,1,6};  Circle(3) = {3,1,21};
Circle(4) = {21,1,7};  Circle(5) = {4,1,23};  Circle(6) = {23,1,8};
Line(7)   = {5,25};    Line(8)   = {25,9};
Circle(9) = {10,1,20}; Circle(10)= {20,1,6};  Circle(11) = {11,1,22};
Circle(12)= {22,1,7};  Circle(13)= {12,1,24}; Circle(14) = {24,1,8};
Line(15)  = {13,26};   Line(16)  = {26,9};
Circle(17)= {19,1,20}; Circle(18)= {21,1,22}; Circle(19) = {23,1,24};
Circle(20)= {25,1,26}; Circle(21)= {2,1,10};  Circle(22) = {3,1,11};
```

```
Circle(23)= {4,1,12};  Circle(24)= {5,1,13};

Line(25) = {1,14};  Line(26) = {14,2};  Line(27) = {2,3};
Line(28) = {3,4};   Line(29) = {4,5};   Line(30) = {1,15};
Line(31) = {15,10}; Line(32) = {10,11}; Line(33) = {11,12};
Line(34) = {12,13}; Line(35) = {14,15}; Line(36) = {14,16};
Line(37) = {15,17}; Line(38) = {16,17}; Line(39) = {18,16};
Line(40) = {18,17}; Line(41) = {1,18};  Line(42) = {18,6};
Line(43) = {6,7};   Line(44) = {16,19}; Line(45) = {19,21};
Line(46) = {21,23}; Line(47) = {23,25}; Line(48) = {17,20};
Line(49) = {20,22}; Line(50) = {22,24}; Line(51) = {24,26};
Line(52) = {7,8};   Line(53) = {8,9};

Line Loop(54) = {39,-36,-25,41};  Ruled Surface(55) = {54};
Line Loop(56) = {44,-1,-26,36};   Ruled Surface(57) = {56};
Line Loop(58) = {3,-45,-1,27};    Ruled Surface(59) = {58};
Line Loop(60) = {5,-46,-3,28};    Ruled Surface(61) = {60};
Line Loop(62) = {7,-47,-5,29};    Ruled Surface(63) = {62};
Line Loop(64) = {-2,-44,-39,42};  Ruled Surface(65) = {64};
Line Loop(66) = {-4,-45,2,43};    Ruled Surface(67) = {66};
Line Loop(68) = {-6,-46,4,52};    Ruled Surface(69) = {68};
Line Loop(70) = {-8,-47,6,53};    Ruled Surface(71) = {70};
Line Loop(72) = {-40,-41,30,37};  Ruled Surface(73) = {72};
Line Loop(74) = {48,-9,-31,37};   Ruled Surface(75) = {74};
Line Loop(76) = {49,-11,-32,9};   Ruled Surface(77) = {76};
Line Loop(78) = {-50,-11,33,13};  Ruled Surface(79) = {78};
Line Loop(80) = {-51,-13,34,15};  Ruled Surface(81) = {80};
Line Loop(82) = {10,-42,40,48};   Ruled Surface(83) = {82};
Line Loop(84) = {12,-43,-10,49};  Ruled Surface(85) = {84};
Line Loop(86) = {14,-52,-12,50};  Ruled Surface(87) = {86};
Line Loop(88) = {16,-53,-14,51};  Ruled Surface(89) = {88};
Line Loop(90) = {-30,25,35};      Ruled Surface(91) = {90};
Line Loop(92) = {-40,39,38};      Ruled Surface(93) = {92};
Line Loop(94) = {37,-38,-36,35};  Ruled Surface(95) = {94};
Line Loop(96) = {-48,-38,44,17};  Ruled Surface(97) = {96};
Line Loop(98) = {18,-49,-17,45};  Ruled Surface(99) = {98};
Line Loop(100) = {19,-50,-18,46}; Ruled Surface(101) = {100};
Line Loop(102) = {20,-51,-19,47}; Ruled Surface(103) = {102};
Line Loop(104) = {-2,17,10};      Ruled Surface(105) = {104};
Line Loop(106) = {-9,-21,1,17};   Ruled Surface(107) = {106};
Line Loop(108) = {-4,18,12};      Ruled Surface(109) = {108};
Line Loop(110) = {-11,-22,3,18};  Ruled Surface(111) = {110};
Line Loop(112) = {-13,-23,5,19};  Ruled Surface(113) = {112};
Line Loop(114) = {-6,19,14};      Ruled Surface(115) = {114};
Line Loop(116) = {-15,-24,7,20};  Ruled Surface(117) = {116};
Line Loop(118) = {-8,20,16};      Ruled Surface(119) = {118};
Line Loop(120) = {-31,-35,26,21}; Ruled Surface(121) = {120};
```

```
Line Loop(122) = {32,-22,-27,21}; Ruled Surface(123) = {122};
Line Loop(124) = {33,-23,-28,22}; Ruled Surface(125) = {124};
Line Loop(126) = {34,-24,-29,23}; Ruled Surface(127) = {126};

Surface Loop(128) = {93,-73,-55,95,-91};
Volume(129) = {128}; // int
Surface Loop(130) = {107,-75,-97,95,57,121};
Volume(131) = {130}; // int b
Surface Loop(132) = {105,-65,-97,-83,-93};
Volume(133) = {132}; // int h
Surface Loop(134) = {99,-111,77,123,59,107};
Volume(135) = {134}; // shell b
Surface Loop(136) = {99,-109,67,105,85};
Volume(137) = {136}; // shell h
Surface Loop(138) = {113,79,-101,-111,-125,-61};
Volume(139) = {138}; // ext b
Surface Loop(140) = {115,-69,-101,-87,-109};
Volume(141) = {140}; // ext h
Surface Loop(142) = {103,-117,-81,113,127,63};
Volume(143) = {142}; // inf b
Surface Loop(144) = {89,-119,71,103,115};
Volume(145) = {144}; // inf h

// Once the geometry is defined, we then add transfinite mesh commands
// in order to explicitly define a structured mesh.

// 1. Transfinite line commands specify the number of points on the
// curves and their distribution (‘Progression 2’ means that each line
// element in the series will be twice as long as the preceding one):

Transfinite Line{35,21,22,23,24,38,17,18,19,20} = nbpt_phi ;
Transfinite Line{31,26,48,44,42} = nbpt_int Using Progression 0.88;
Transfinite Line{41,37,36,9,11,1,3,13,5,15,7} = nbpt_arc1 ;
Transfinite Line{30,25,40,39,10,2,12,4,14,6,16,8} = nbpt_arc2 ;
Transfinite Line{32,27,49,45,43} = nbpt_shell ;
Transfinite Line{33,28,46,50,52} = nbpt_far Using Progression 1.2 ;
Transfinite Line{34,29,51,47,53} = nbpt_inf Using Progression 1.05;

// 2. Transfinite surfaces are defined by an ordered list of the
// points on their boundary (the ordering of these points defines the
// ordering of the mesh elements). Note that a transfinite surface can
// only have 3 or 4 sides:

Transfinite Surface{55} = {1,14,16,18};
Transfinite Surface{57} = {14,2,19,16};
Transfinite Surface{59} = {2,3,21,19};
Transfinite Surface{61} = {3,4,23,21};
```

```
Transfinite Surface{63} = {4,5,25,23};
Transfinite Surface{73} = {1,15,17,18};
Transfinite Surface{75} = {15,10,20,17};
Transfinite Surface{77} = {10,11,22,20};
Transfinite Surface{79} = {11,12,24,22};
Transfinite Surface{81} = {12,13,26,24};
Transfinite Surface{65} = {18,16,19,6};
Transfinite Surface{67} = {6,19,21,7};
Transfinite Surface{69} = {7,21,23,8};
Transfinite Surface{71} = {8,23,25,9};
Transfinite Surface{83} = {17,18,6,20};
Transfinite Surface{85} = {20,6,7,22};
Transfinite Surface{87} = {22,7,8,24};
Transfinite Surface{89} = {24,8,9,26};
Transfinite Surface{91} = {1,14,15};
Transfinite Surface{95} = {15,14,16,17};
Transfinite Surface{93} = {18,16,17};
Transfinite Surface{121} = {15,14,2,10};
Transfinite Surface{97} = {17,16,19,20};
Transfinite Surface{123} = {10,2,3,11};
Transfinite Surface{99} = {20,19,21,22};
Transfinite Surface{107} = {10,2,19,20};
Transfinite Surface{105} = {6,20,19};
Transfinite Surface{109} = {7,22,21};
Transfinite Surface{111} = {11,3,21,22};
Transfinite Surface{101} = {22,21,23,24};
Transfinite Surface{125} = {11,3,4,12};
Transfinite Surface{115} = {8,24,23};
Transfinite Surface{113} = {24,12,4,23};
Transfinite Surface{127} = {12,13,5,4};
Transfinite Surface{103} = {24,23,25,26};
Transfinite Surface{119} = {9,26,25};
Transfinite Surface{117} = {13,5,25,26};

// 3. Transfinite volumes are also defined by an ordered list of the
// points on their boundary (the ordering defines the ordering of the
// mesh elements).  A transfinite volume can only have 6 or 8 faces:

Transfinite Volume{129} = {1,14,15,18,16,17};
Transfinite Volume{131} = {17,16,14,15,20,19,2,10};
Transfinite Volume{133} = {18,17,16,6,20,19};
Transfinite Volume{135} = {10,2,19,20,11,3,21,22};
Transfinite Volume{137} = {6,20,19,7,22,21};
Transfinite Volume{139} = {11,3,4,12,22,21,23,24};
Transfinite Volume{141} = {7,22,21,8,24,23};
Transfinite Volume{143} = {12,4,5,13,24,23,25,26};
Transfinite Volume{145} = {8,24,23,9,26,25};
```

```
// As with Extruded meshes, the 'Recombine' command tells Gmsh to
// recombine the simplices into quadrangles, prisms or hexahedra when
// possible:

Recombine Surface {55:127};

// We finish by defing some physical entities:

VolInt           = 1000 ;
SurfIntPhi0      = 1001 ;  SurfIntPhi1      = 1002 ;
SurfIntZ0        = 1003 ;

VolShell         = 2000 ;
SurfShellInt     = 2001 ;  SurfShellExt     = 2002 ;
SurfShellPhi0    = 2003 ;  SurfShellPhi1    = 2004 ;
SurfShellZ0      = 2005 ;
LineShellIntPhi0 = 2006 ;
LineShellIntPhi1 = 2007 ;  LineShellIntZ0   = 2008 ;
PointShellInt    = 2009 ;

VolExt           = 3000 ;
VolInf           = 3001 ;
SurfInf          = 3002 ;
SurfExtInfPhi0   = 3003 ;  SurfExtInfPhi1   = 3004 ;
SurfExtInfZ0     = 3005 ;
SurfInfRight     = 3006 ;
SurfInfTop       = 3007 ;

Physical Volume  (VolInt)           = {129,131,133} ;
Physical Surface (SurfIntPhi0)      = {55,57,65} ;
Physical Surface (SurfIntPhi1)      = {73,75,83} ;
Physical Surface (SurfIntZ0)        = {91,121} ;

Physical Volume  (VolShell)         = {135,137} ;
Physical Surface (SurfShellInt)     = {105,107} ;
Physical Surface (SurfShellExt)     = {109,111} ;
Physical Surface (SurfShellPhi0)    = {59,67} ;
Physical Surface (SurfShellPhi1)    = {77,85} ;
Physical Surface (SurfShellZ0)      = {123} ;
Physical Line    (LineShellIntPhi0) = {1,2} ;
Physical Line    (LineShellIntPhi1) = {9,10} ;
Physical Line    (LineShellIntZ0)   = 21 ;
//Physical Point   (PointShellInt)    = 6 ;

Physical Volume  (VolExt)           = {139,141} ;
Physical Volume  (VolInf)           = {143,145} ;
```

```
Physical Surface (SurfExtInfPhi0)   = {61,63,69,71} ;
Physical Surface (SurfExtInfPhi1)   = {79,87,81,89} ;
Physical Surface (SurfExtInfZ0)     = {125,127} ;
Physical Surface (SurfInfRight)     = {117} ;
Physical Surface (SurfInfTop)       = {119} ;
```

7.7 't7.geo'

```
/*********************************************************************
 *
 *  Gmsh tutorial 7
 *
 *  Anisotropic meshes, attractors
 *
 *********************************************************************/

// The anisotropic 2D mesh generator can be selected with:

Mesh.Algorithm = 2 ;

// One can force a 4 step Laplacian smoothing of the mesh with:

Mesh.Smoothing = 4 ;

lc = .1;

Point(1) = {0.0,0.0,0,lc};
Point(2) = {1.2,-0.2,0,lc};
Point(3) = {1,1,0,lc};
Point(4) = {0,1,0,lc};

Line(1) = {3,2};
Line(2) = {2,1};
Line(3) = {1,4};
Line(4) = {4,3};

Line Loop(5) = {1,2,3,4};
Plane Surface(6) = {5};

Point(5) = {0.1,0.2,0,lc};
Point(11) = {0.4,0.7,-1,lc};
Point(12) = {0.5,0.5,0,lc};
Point(22) = {0.9,0.9,1,lc};

Line(5) = {11,22};
```

```
Spline(7) = {4,5,12,2};

// Isotropic and anisotropic attractors can be defined on points and
// lines (this is still experimental and known to be unstable: use at
// your own risk!):

Attractor Point{1} = {0.01, 0.01, 2};

Attractor Line{5} = {0.3, 0.01, 2};

Attractor Line{7} = {0.1, 0.02, 8};
```

7.8 ‘t8.geo’

```
/*********************************************************************
 *
 *  Gmsh tutorial 8
 *
 *  Post-processing, scripting, animations, options
 *
 *********************************************************************/

// We first include ‘t1.geo’ as well as some post-processing views:

Include "t1.geo" ;
Include "view1.pos" ;
Include "view1.pos" ;
Include "view4.pos" ;

// We then set some general options:

General.Trackball = 0 ;
General.RotationX = 0 ;
General.RotationY = 0 ;
General.RotationZ = 0 ;
General.Color.Background = White ;
General.Color.Foreground = Black ;
General.Color.Text = Black ;
General.Orthographic = 0 ;
General.Axes = 0 ;
General.SmallAxes = 0 ;

// We also set some options for each post-processing view:

v0 = PostProcessing.NbViews-4;
```

```
v1 = v0+1;
v2 = v0+2;
v3 = v0+3;

View[v0].IntervalsType = 2 ;
View[v0].OffsetZ = 0.05 ;
View[v0].RaiseZ = 0 ;
View[v0].Light = 1 ;
View[v0].ShowScale = 0;
View[v0].SmoothNormals = 1;

View[v1].IntervalsType = 1 ;
View[v1].ColorTable = { Green, Blue } ;
View[v1].NbIso = 10 ;
View[v1].ShowScale = 0;

View[v2].Name = "Test..." ;
View[v2].IntervalsType = 2 ;
View[v2].Type = 2;
View[v2].IntervalsType = 2 ;
View[v2].AutoPosition = 0;
View[v2].PositionX = 85;
View[v2].PositionY = 50;
View[v2].Width = 200;
View[v2].Height = 130;

View[v3].Type = 3;
View[v3].RangeType = 2;
View[v3].IntervalsType = 4 ;
View[v3].ShowScale = 0;
View[v3].Grid = 0;
View[v3].CustomMin = View[v2].CustomMin;
View[v3].CustomMax = View[v2].CustomMax;
View[v3].AutoPosition = 0;
View[v3].PositionX = View[v2].PositionX;
View[v3].PositionY = View[v2].PositionY;
View[v3].Width = View[v2].Width;
View[v3].Height = View[v2].Height;

// We then loop from 1 to 255 with a step of 1. (To use a different
// step, just add a third argument in the list. For example, 'For num
// In {0.5:1.5:0.1}' would increment num from 0.5 to 1.5 with a step
// of 0.1.)

t = 0 ;

For num In {1:255}
```

```
View[v0].TimeStep = t ;
View[v1].TimeStep = t ;
View[v2].TimeStep = t ;
View[v3].TimeStep = t ;

t = (View[v0].TimeStep < View[v0].NbTimeStep-1) ? t+1 : 0 ;

View[v0].RaiseZ += 0.01/View[v0].Max * t ;

If (num == 3)
  // We want to create 320x240 frames when num == 3:
  General.GraphicsWidth = 320 ;
  General.GraphicsHeight = 240 ;
EndIf

// It is possible to nest loops:
For num2 In {1:50}

  General.RotationX += 10 ;
  General.RotationY = General.RotationX / 3 ;
  General.RotationZ += 0.1 ;

  Sleep 0.01; // sleep for 0.01 second
  Draw; // draw the scene

  If (num == 3)
    // The 'Print' command saves the graphical window; the 'Sprintf'
    // function permits to create the file names on the fly:
    Print Sprintf("t8-%02g.gif", num2);
    Print Sprintf("t8-%02g.jpg", num2);
  EndIf

EndFor

If(num == 3)
  // Here we could make a system call to generate a movie. For example,

  // with whirlgif:
  //
  // System "whirlgif -minimize -loop -o t8.gif t8-*.gif";

  // with mpeg_encode:
  //
  // System "mpeg_encode t8.par";

  // with mencoder:
```

```
    //
    // System "mencoder 'mf://*.jpg' -mf fps=5 -o t8.mpg -ovc lavc
    //         -lavcopts vcodec=mpeg1video:vhq";
    // System "mencoder 'mf://*.jpg' -mf fps=5 -o t8.mpg -ovc lavc
    //         -lavcopts vcodec=mpeg4:vhq";

    // with ffmpeg:
    //
    // System "ffmpeg -hq -r 5 -b 800 -vcodec mpeg1video
    //         -i t8-%02d.jpg t8.mpg"
    // System "ffmpeg -hq -r 5 -b 800 -i t8-%02d.jpg t8.asf"
  EndIf

EndFor
```

7.9 't9.geo'

```
/*********************************************************************
 *
 *  Gmsh tutorial 9
 *
 *  Post-processing, plugins
 *
 *********************************************************************/

// Plugins can be added to Gmsh in order to extend its
// capabilities. For example, post-processing plugins can modify a
// view, or create a new view based on previously loaded
// views. Several default plugins are statically linked with Gmsh,
// e.g. CutMap, CutPlane, CutSphere, Skin, Transform or Smooth.
// Plugins can be controlled in the same way as other options: either
// from the graphical interface (right click on the view button, then
// 'Plugins'), or from the command file.

// Let us for example include a three-dimensional scalar view:

Include "view3.pos" ;

// We then set some options for the 'CutMap' plugin (which extracts an
// isovalue surface from a 3D scalar view), and run it:

Plugin(CutMap).A = 0.67 ; // iso-value level
Plugin(CutMap).iView = 0 ; // source view is View[0]
Plugin(CutMap).Run ;

// We also set some options for the 'CutPlane' plugin (which computes
```

```
// a section of a 3D view), and then run it:

Plugin(CutPlane).A = 0 ;
Plugin(CutPlane).B = 0.2 ;
Plugin(CutPlane).C = 1 ;
Plugin(CutPlane).D = 0 ;
Plugin(CutPlane).Run ;

// We finish by setting some view options and redrawing the scene:

View[0].Light = 1;
View[0].IntervalsType = 2;
View[0].NbIso = 6;
View[0].SmoothNormals = 1;

View[1].IntervalsType = 2;

View[2].IntervalsType = 2;

Draw;
```

8 Running Gmsh

8.1 Interactive mode

Gmsh's first operating mode is the 'interactive graphical mode'. To launch Gmsh in interactive mode, just click or double-click on the Gmsh icon (Windows and Mac), or type

```
> gmsh
```

at your shell prompt in a terminal (Unix). This will open two windows: the graphic window (with a status bar at the bottom) and the menu window (with a menu bar and some context-dependent buttons).

To open the first tutorial file (see Chapter 7 [Tutorial], page 95), select the 'File->Open' menu, and choose '`t1.geo`' in the input field. When using a terminal, you can also specify the file name directly on the command line, i.e.:

```
> gmsh t1.geo
```

To perform the mesh generation, go to the mesh module (by selecting 'Mesh' in the module menu) and choose the required dimension in the context-dependent buttons ('1D' will mesh all the lines; '2D' will mesh all the surfaces—as well as all the lines if '1D' was not called before; '3D' will mesh all the volumes—and all the surfaces if '2D' was not called before). To save the resulting mesh in the current mesh format, choose 'Save' in the context-dependent buttons, or select the appropriate format with the 'File->Save as' menu. The default mesh file name is based on the name of the first input file on the command line (or '`untitled`' if there wasn't any input file given), with an appended extension depending on the mesh format[1].

To create a new geometry or to modify an existing geometry, select 'Geometry' in the module menu, and follow the context-dependent buttons. For example, to create a spline, select 'Elementary', 'Add', 'New' and 'Spline'. You will then be asked to select a list of points, and to type *e* to finish the selection (or *q* to abort it). Once the interactive command is completed, a text string is automatically added at the end of the current project file. You can edit this project file by hand at any time by pressing the 'Edit' button in the 'Geometry' menu and then reloading the project by pressing 'Reload'. For example, it is often faster to define variables and points directly in the project file, and then use the graphical user interface to define the lines, the surfaces and the volumes interactively.

Several files can be loaded simultaneously in Gmsh. The first one defines the project, while the others are appended ('merged') to this project. You can merge such files with the 'File->Merge' menu, or by directly specifying the names of the files on the command line. For example, to merge the post-processing views contained in the files '`view1.pos`' and '`view2.pos`' together with the geometry of the first tutorial '`t1.geo`', you can type the following command:

```
> gmsh t1.geo view1.pos view2.pos
```

In the Post-Processing module (select 'Post-Processing' in the module menu), two view buttons will appear, respectively labeled 'a scalar map' and 'a vector map'. A mouse click

[1] Nearly all the interactive commands have shortcuts: see Section 8.5 [Keyboard shortcuts], page 123, or select 'Help->Keyboard shortcuts' in the menu.

on the name will toggle the visibility of the selected view, while a click on the arrow button on the right will provide access to the view's options. If you want the modifications made to one view to affect also all the other views, select the 'Apply next changes to all views' or 'Force same options for all views' option in the 'Tools->Options->Post-processing' menu.

Note that all the options specified interactively can also be directly specified in the ASCII input files. All available options, with their current values, can be saved into a file by selecting 'File->Save as->Gmsh options', or simply viewed by pressing the '?' button in the status bar. To save the current options as your default preferences for all future Gmsh sessions, use the 'Tools->Options->Save' button.

8.2 Non-interactive mode

Gmsh's second operating mode is the non-interactive (or 'batch') mode. In this mode, there is no graphical user interface, and all operations are performed without any user interaction [2]. For example, to mesh the first tutorial in non-interactive mode, just type:

```
> gmsh t1.geo -2
```

To mesh the same example, but with the background mesh available in the file '`bgmesh.pos`', type:

```
> gmsh t1.geo -2 -bgm bgmesh.pos
```

For the list of all command-line options, see Section 8.3 [Command-line options], page 120.

8.3 Command-line options

Geometry options:

`-0` parse input files, output unrolled geometry, and exit

Mesh options:

`-1, -2, -3`
perform batch 1D, 2D and 3D mesh generation

`-saveall` save all elements (discard physical group definitions)

`-o file` specify mesh output file name

`-format string`
set output mesh format (msh, unv, gref)

`-algo string`
select 2D mesh algorithm (iso, tri, aniso, netgen)

`-smooth int`
set mesh smoothing

`-optimize`
optimize quality of tetrahedral elements

[2] If you compile Gmsh without the graphical user interface, i.e., with `./configure --disable-gui`, this is the only mode you have access to.

`-order int`
: set the order of the generated elements (1, 2)

`-scale float`
: set global scaling factor

`-meshscale float`
: set mesh scaling factor

`-clscale float`
: set characteristic length scaling factor

`-rand float`
: set random perturbation factor

`-bgm file` load backround mesh from file

`-constrain`
: constrain background mesh with characteristic lengths

`-histogram`
: print mesh quality histogram

`-extrude` use old extrusion mesh generator

`-recombine`
: recombine meshes from old extrusion mesh generator

`-interactive`
: display 2D mesh construction interactively

Post-processing options:

`-noview` hide all views on startup

`-link int` select link mode between views (0, 1, 2, 3, 4)

`-smoothview`
: smooth views

`-combine` combine input views into multi-time-step ones

Display options:

`-nodb` disable double buffering

`-fontsize int`
: specify the font size for the GUI

`-scheme string`
: specify FLTK GUI scheme

`-display string`
: specify display

Other options:

`-a, -g, -m, -s, -p`
: start in automatic, geometry, mesh, solver or post-processing mode

`-pid`
: print pid on stdout

`-v int`
: set verbosity level

`-string "string"`
: parse string before project file

`-option file`
: parse option file before GUI creation

`-convert file file`
: perform batch conversion of views and meshes into latest file formats

`-version`
: show version number

`-info`
: show detailed version information

`-help`
: show this message

8.4 Mouse actions

In the following, for a 2 button mouse, *Middle button* = *Shift*+*Left button*. For a 1 button mouse, *Middle button* = *Shift*+*Left button* and *Right button* = *Alt*+*Left button*.

Move the mouse:

- highlight the elementary geometrical entity currently under the mouse pointer and display its properties in the status bar
- size a rubber zoom started with *Ctrl*+*Left button*

Left button:

- rotate
- accept a rubber zoom started with *Ctrl*+*Left button*

Ctrl+*Left button*: start (anisotropic) rubber zoom

Middle button:

- zoom (isotropic)
- cancel a rubber zoom

Ctrl+*Middle button*: orthogonalize display

Right button:

- pan
- cancel a rubber zoom
- pop up menu on post-processing view button

Ctrl+*Right button*: reset to default viewpoint

8.5 Keyboard shortcuts

Keyboard shortcuts:

Left arrow go to previous time step

Right arrow
go to next time step

Up arrow make previous view visible

Down arrow
make next view visible

< go back to previous context

> go forward to next context

0

Esc reload geometry input file

1

F1 mesh curves

2

F2 mesh surfaces

3

F3 mesh volumes

Alt+a hide/show small axes

Shift+a raise (show) all open windows

Alt+Shift+a
hide/show big moving axes

Alt+b hide/show all bounding boxes

Alt+c loop through predefined color schemes

Alt+d change mesh display mode (solid/wireframe)

Shift+d decrease animation delay

Ctrl+Shift+d
increase animation delay

Alt+f change redraw mode (fast/full)

g go to geometry module

Shift+g show geometry options

Alt+h hide/show all post-processing views

Alt+i hide/show all post-processing scales

Shift+i	show statistics window
Alt+l	hide/show geometry lines
Alt+Shift+l	hide/show surface mesh edges
m	go to mesh module
Alt+m	change visibility of all mesh entities
Shift+m	show mesh options
Ctrl+m	merge file
Shift+n	show general options
Ctrl+n	new file
Alt+o	change projection mode (ortho/perspective)
Shift+o	show option window
Ctrl+o	open file
p	go to post-processor module
Alt+p	hide/show geometry points
Shift+p	show general post-processing options
Alt+Shift+p	hide/show mesh points
Ctrl+q	quit
Alt+s	hide/show geometry surfaces
Alt+Shift+s	hide/show mesh surfaces
Ctrl+s	save mesh in default format
Ctrl+Shift+s	save file as
Alt+t	loop through interval modes for all post-processing views
Alt+v	hide/show geometry volumes
Alt+Shift+v	hide/show mesh volumes
Alt+w	enable/disable all lighting
Shift+w	show current post-processing view options
Alt+x	set X view
Alt+y	set Y view
Alt+z	set Z view

9 File formats

This chapter describes the file formats that cannot be modified by the user. (These formats have version numbers that are independent of the Gmsh version number.)

All non-parsed file formats have sections enclosed between `$Key` and `$EndKey` tags.

9.1 Gmsh mesh file formats

Please note that the list of nodes and elements in Gmsh's mesh files do not have to be dense or ordered (i.e., the node and element numbers do not have to be given in a consecutive or even an ordered way). A sample C++ program to transform the formats so that all lists are dense and ordered is available in the source distribution ('`utils/misc/mshsort.cpp`'). This program is also a good example on how to read and write files in the '`.msh`' format.

9.1.1 Version 1.0

The '`.msh`' file format, version 1.0, is Gmsh's old native mesh file format, now superseeded by the format described in Section 9.1.2 [Version 2.0], page 127.

In the '`.msh`' file format, version 1.0, the file is divided in two sections, defining the nodes (`$NOD`-`$ENDNOD`) and the elements (`$ELM`-`$ENDELM`) in the mesh:

```
$NOD
number-of-nodes
node-number x-coord y-coord z-coord
...
$ENDNOD
$ELM
number-of-elements
elm-number elm-type reg-phys reg-elem number-of-nodes node-number-list
...
$ENDELM
```

where

number-of-nodes
: is the number of nodes in the mesh.

node-number
: is the number (index) of the *n*-th node in the mesh. Note that the *node-number*s do not have to be given in a consecutive (or even an ordered) way.

x-coord y-coord z-coord
: are the floating point values giving the X, Y and Z coordinates of the *n*-th node.

number-of-elements
: is the number of elements in the mesh.

elm-number
: is the number (index) of the *n*-th element in the mesh. Note that the *elm-number*s do not have to be given in a consecutive (or even an ordered) way.

elm-type defines the geometrical type of the *n*-th element:

`1` Line (2 nodes).

`2` Triangle (3 nodes).

`3` Quadrangle (4 nodes).

`4` Tetrahedron (4 nodes).

`5` Hexahedron (8 nodes).

`6` Prism (6 nodes).

`7` Pyramid (5 nodes).

`8` Second order line (3 nodes: 2 associated with the vertices and 1 with the edge).

`9` Second order triangle (6 nodes: 3 associated with the vertices and 3 with the edges).

`10` Second order quadrangle (9 nodes: 4 associated with the vertices, 4 with the edges and 1 with the face).

`11` Second order tetrahedron (10 nodes: 4 associated with the vertices and 6 with the edges).

`12` Second order hexahedron (27 nodes: 8 associated with the vertices, 12 with the edges, 6 with the faces and 1 with the volume).

`13` Second order prism (18 nodes: 6 associated with the vertices, 9 with the edges and 3 with the quadrangular faces).

`14` Second order pyramid (14 nodes: 5 associated with the vertices, 8 with the edges and 1 with the quadrangular face).

`15` Point (1 node).

See below for the ordering of the nodes.

reg-phys is the number of the physical entity to which the element belongs.

reg-elem is the number of the elementary entity to which the element belongs.

number-of-nodes
is the number of nodes for the *n*-th element. This is redundant, but kept for backward compatibility.

node-number-list
is the list of the *number-of-nodes* node numbers of the *n*-th element. The ordering of the nodes is given in Section 9.3 [Gmsh node ordering], page 134; for second order elements, the first order nodes are given first, followed by the nodes associated with the edges, followed by the nodes associated with the quadrangular faces (if any), followed by the nodes associated with the volume (if any). The ordering of these additional nodes follows the ordering of the edges and quadrangular faces given in Section 9.3 [Gmsh node ordering], page 134.

9.1.2 Version 2.0

Version 2.0 of the '`.msh`' file format is Gmsh's new native mesh file format. It is very similar to the old one (see Section 9.1.1 [Version 1.0], page 125), but is more general: it contains information about itself and allows to associate an arbitrary number of integer tags with each element.

The '`.msh`' file format, version 2.0, is divided in three sections, defining the file format (`$MeshFormat-$EndMeshFormat`), the nodes (`$Nodes-$EndNodes`) and the elements (`$Elements-$EndElements`) in the mesh:

```
$MeshFormat
2.0 file-type data-size
$EndMeshFormat
$Nodes
number-of-nodes
node-number x-coord y-coord z-coord
...
$EndNodes
$Elements
number-of-elements
elm-number elm-type number-of-tags < tag > ... node-number-list
...
$EndElements
```

where

file-type is an integer equal to 0 in the ASCII file format.

data-size is an integer equal to the size of the floating point numbers used in the file (usually, *data-size* = sizeof(double)).

number-of-nodes
is the number of nodes in the mesh.

node-number
is the number (index) of the *n*-th node in the mesh. Note that the *node-number*s do not have to be given in a consecutive (or even an ordered) way.

x-coord y-coord z-coord
are the floating point values giving the X, Y and Z coordinates of the *n*-th node.

number-of-elements
is the number of elements in the mesh.

elm-number
is the number (index) of the *n*-th element in the mesh. Note that the *elm-number*s do not have to be given in a consecutive (or even an ordered) way.

elm-type defines the geometrical type of the *n*-th element:

`1` Line (2 nodes).

`2` Triangle (3 nodes).

`3` Quadrangle (4 nodes).

`4` Tetrahedron (4 nodes).

`5` Hexahedron (8 nodes).

`6` Prism (6 nodes).

`7` Pyramid (5 nodes).

`8` Second order line (3 nodes: 2 associated with the vertices and 1 with the edge).

`9` Second order triangle (6 nodes: 3 associated with the vertices and 3 with the edges).

`10` Second order quadrangle (9 nodes: 4 associated with the vertices, 4 with the edges and 1 with the face).

`11` Second order tetrahedron (10 nodes: 4 associated with the vertices and 6 with the edges).

`12` Second order hexahedron (27 nodes: 8 associated with the vertices, 12 with the edges, 6 with the faces and 1 with the volume).

`13` Second order prism (18 nodes: 6 associated with the vertices, 9 with the edges and 3 with the quadrangular faces).

`14` Second order pyramid (14 nodes: 5 associated with the vertices, 8 with the edges and 1 with the quadrangular face).

`15` Point (1 node).

See below for the ordering of the nodes.

number-of-tags
: gives the number of tags for the n-th element. By default, Gmsh generates meshes with two tags and reads files with an arbitrary number of tags: see below.

tag
: is an integer tag associated with the n-th element. By default, the first tag is the number of the physical entity to which the element belongs; the second is the number of the elementary geometrical entity to which the element belongs; the third is the number of a mesh partition to which the element belongs.

node-number-list
: is the list of the node numbers of the n-th element. The ordering of the nodes is given in Section 9.3 [Gmsh node ordering], page 134; for second order elements, the first order nodes are given first, followed by the nodes associated with the edges, followed by the nodes associated with the quadrangular faces (if any), followed by the nodes associated with the volume (if any). The ordering of these additional nodes follows the ordering of the edges and quadrangular faces given in Section 9.3 [Gmsh node ordering], page 134.

9.2 Gmsh post-processing file formats

Gmsh can read and write data sets in three different file formats: the "parsed", "ASCII" and "binary" file formats. The parsed format is the oldest and most flexible, but also the slowest to read/write. The ASCII and binary formats are less flexible but allow for faster read/write operations, which is useful for (very) large data sets.

Gmsh can convert any format to any other, either in a script (cf. `Save View` and `PostProcessing.Format` in Section 6.1 [Post-processing commands], page 79, and Section 6.3 [Post-processing options], page 86, respectively) or in the graphical user interface (using the 'View->Save as' menu).

9.2.1 Parsed post-processing file format

Gmsh's oldest post-processing format ("parsed") is read by Gmsh's script parser (see also Section 6.1 [Post-processing commands], page 79). You can thus, for example, embed parsed post-processing views directly into your geometrical descriptions (see, e.g., Section 7.4 [t4.geo], page 101).

The parsed format is very powerful, since all the values are *expressions*, and it can be easily generated "on-the-fly", as there is no header containing *a priori* information on the size of the data set. Its syntax is also very permissive, which makes it ideal for testing purposes. Its main disadvantage resides in the overhead introduced by the parser, which makes loading a view in parsed format slower than loading a view in ASCII or binary format. This is only a disadvantage for very large data sets, tough.

A post-processing view in parsed format is defined as follows (there can be one or more views in the same file):

```
View "string" {
  type ( list-of-coords ) { list-of-values };
  ...
};
```

The 26 objects that can be displayed are:

	type	#*list-of-coords*	#*list-of-values*
scalar point	SP	3	1 * *nb-time-steps*
vector point	VP	3	3 * *nb-time-steps*
tensor point	TP	3	9 * *nb-time-steps*
scalar line	SL	6	2 * *nb-time-steps*
vector line	VL	6	6 * *nb-time-steps*
tensor line	TL	6	18 * *nb-time-steps*
scalar triangle	ST	9	3 * *nb-time-steps*
vector triangle	VT	9	9 * *nb-time-steps*
tensor triangle	TT	9	27 * *nb-time-steps*
scalar quadrangle	SQ	12	4 * *nb-time-steps*
vector quadrangle	VQ	12	12 * *nb-time-steps*
tensor quadrangle	TQ	12	36 * *nb-time-steps*

scalar tetrahedron	SS	12	4 * *nb-time-steps*
vector tetrahedron	VS	12	12 * *nb-time-steps*
tensor tetrahedron	TS	12	36 * *nb-time-steps*
scalar hexahedron	SH	24	8 * *nb-time-steps*
vector hexahedron	VH	24	24 * *nb-time-steps*
tensor hexahedron	TH	24	72 * *nb-time-steps*
scalar prism	SI	18	6 * *nb-time-steps*
vector prism	VI	18	18 * *nb-time-steps*
tensor prism	TI	18	54 * *nb-time-steps*
scalar pyramid	SY	15	5 * *nb-time-steps*
vector pyramid	VY	15	15 * *nb-time-steps*
tensor pyramid	TY	15	45 * *nb-time-steps*
text 2d	T2	4	arbitrary
text 3d	T3	5	arbitrary

The coordinates are given 'by node'[1], i.e.:

- (*coord1*, *coord2*, *coord3*) for a point,
- (*coord1-node1*, *coord2-node1*, *coord3-node1*, *coord1-node2*, *coord2-node2*, *coord3-node2*) for a line,
- (*coord1-node1*, *coord2-node1*, *coord3-node1*, *coord1-node2*, *coord2-node2*, *coord3-node2*, *coord1-node3*, *coord2-node3*, *coord3-node3*) for a triangle,
- etc.

The values are given by time step, by node and by component, i.e.:

comp1-node1-time1, *comp2-node1-time1*, *comp3-node1-time1*,
comp1-node2-time1, *comp2-node2-time1*, *comp3-node2-time1*,
comp1-node3-time1, *comp2-node3-time1*, *comp3-node3-time1*,
comp1-node1-time2, *comp2-node1-time2*, *comp3-node1-time2*,
comp1-node2-time2, *comp2-node2-time2*, *comp3-node2-time2*,
comp1-node3-time2, *comp2-node3-time2*, *comp3-node3-time2*,
...

9.2.2 ASCII post-processing file format

The ASCII post-processing file is divided in several sections: one format section, enclosed between `$PostFormat-$EndPostFormat` tags, and one or more post-processing views, enclosed between `$View-$EndView` tags:

`$PostFormat`
`1.2` *file-type data-size*
`$EndPostFormat`
`$View`
view-name nb-time-steps
nb-scalar-points nb-vector-points nb-tensor-points

[1] Beware that this is different from the ordering of the node coordinates in the ASCII and binary post-processing file formats.

nb-scalar-lines nb-vector-lines nb-tensor-lines
nb-scalar-triangles nb-vector-triangles nb-tensor-triangles
nb-scalar-quadrangles nb-vector-quadrangles nb-tensor-quadrangles
nb-scalar-tetrahedra nb-vector-tetrahedra nb-tensor-tetrahedra
nb-scalar-hexahedra nb-vector-hexahedra nb-tensor-hexahedra
nb-scalar-prisms nb-vector-prisms nb-tensor-prisms
nb-scalar-pyramids nb-vector-pyramids nb-tensor-pyramids
nb-text2d nb-text2d-chars nb-text3d nb-text3d-chars
time-step-values
< *scalar-point-value* > ...
< *vector-point-value* > ...
< *tensor-point-value* > ...
< *scalar-line-value* > ...
< *vector-line-value* > ...
< *tensor-line-value* > ...
< *scalar-triangle-value* > ...
< *vector-triangle-value* > ...
< *tensor-triangle-value* > ...
< *scalar-quadrangle-value* > ...
< *vector-quadrangle-value* > ...
< *tensor-quadrangle-value* > ...
< *scalar-tetrahedron-value* > ...
< *vector-tetrahedron-value* > ...
< *tensor-tetrahedron-value* > ...
< *scalar-hexahedron-value* > ...
< *vector-hexahedron-value* > ...
< *tensor-hexahedron-value* > ...
< *scalar-prism-value* > ...
< *vector-prism-value* > ...
< *tensor-prism-value* > ...
< *scalar-pyramid-value* > ...
< *vector-pyramid-value* > ...
< *tensor-pyramid-value* > ...
< *text2d* > ... < *text2d-chars* > ...
< *text3d* > ... < *text3d-chars* > ...
`$EndView`

where

file-type is an integer equal to 0 in the ASCII file format.

data-size is an integer equal to the size of the floating point numbers used in the file (usually, *data-size* = sizeof(double)).

view-name is a string containing the name of the view (max. 256 characters).

nb-time-steps
is an integer giving the number of time steps in the view.

nb-scalar-points
nb-vector-points
. . . are integers giving the number of scalar points, vector points, . . . in the view.

nb-text2d

nb-text3d are integers giving the number of 2D and 3D text strings in the view.

nb-text2d-chars
nb-text3d-chars
are integers giving the total number of characters in the 2D and 3D strings.

time-step-values
is a list of *nb-time-steps* double precision numbers giving the value of the time (or any other variable) for which an evolution was saved.

scalar-point-value
vector-point-value
. . . are lists of double precision numbers giving the node coordinates and the values associated with the nodes of the *nb-scalar-points* scalar points, *nb-vector-points* vector points, . . ., for each of the *time-step-values*.

For example, *vector-triangle-value* is defined as:

coord1-node1 coord1-node2 coord1-node3
coord2-node1 coord2-node2 coord2-node3
coord3-node1 coord3-node2 coord3-node3
comp1-node1-time1 comp2-node1-time1 comp3-node1-time1
comp1-node2-time1 comp2-node2-time1 comp3-node2-time1
comp1-node3-time1 comp2-node3-time1 comp3-node3-time1
comp1-node1-time2 comp2-node1-time2 comp3-node1-time2
comp1-node2-time2 comp2-node2-time2 comp3-node2-time2
comp1-node3-time2 comp2-node3-time2 comp3-node3-time2
. . .

text2d is a list of 4 double precision numbers:

coord1 coord2 style index

where *coord1* and *coord2* give the coordinates of the leftmost element of the 2D string in screen coordinates, *index* gives the starting index of the string in *text2d-chars* and *style* is currently unused.

text2d-chars
is a list of *nb-text2d-chars* characters. Substrings are separated with the '^' character (which is a forbidden character in regular strings).

text3d is a list of 5 double precision numbers

coord1 coord2 coord3 style index

where *coord1*, *coord2* and *coord3* give the coordinates of the leftmost element of the 3D string in model (real world) coordinates, *index* gives the starting index of the string in *text3d-chars* and *style* is currently unused.

text3d-chars
is a list of *nb-text3d-chars* chars. Substrings are separated with the '^' character.

9.2.3 Binary post-processing file format

The binary post-processing file format is the same as the ASCII file format described in Section 9.2.2 [ASCII post-processing file format], page 130, except that:

1. *file-type* equals 1.
2. all lists of floating point numbers and characters are written in binary format
3. there is an additional integer, of value 1, written before *time-step-values*. This integer is used for detecting if the computer on which the binary file was written and the computer on which the file is read are of the same type (little or big endian).

Here is a pseudo C code to write a post-processing file in binary format:

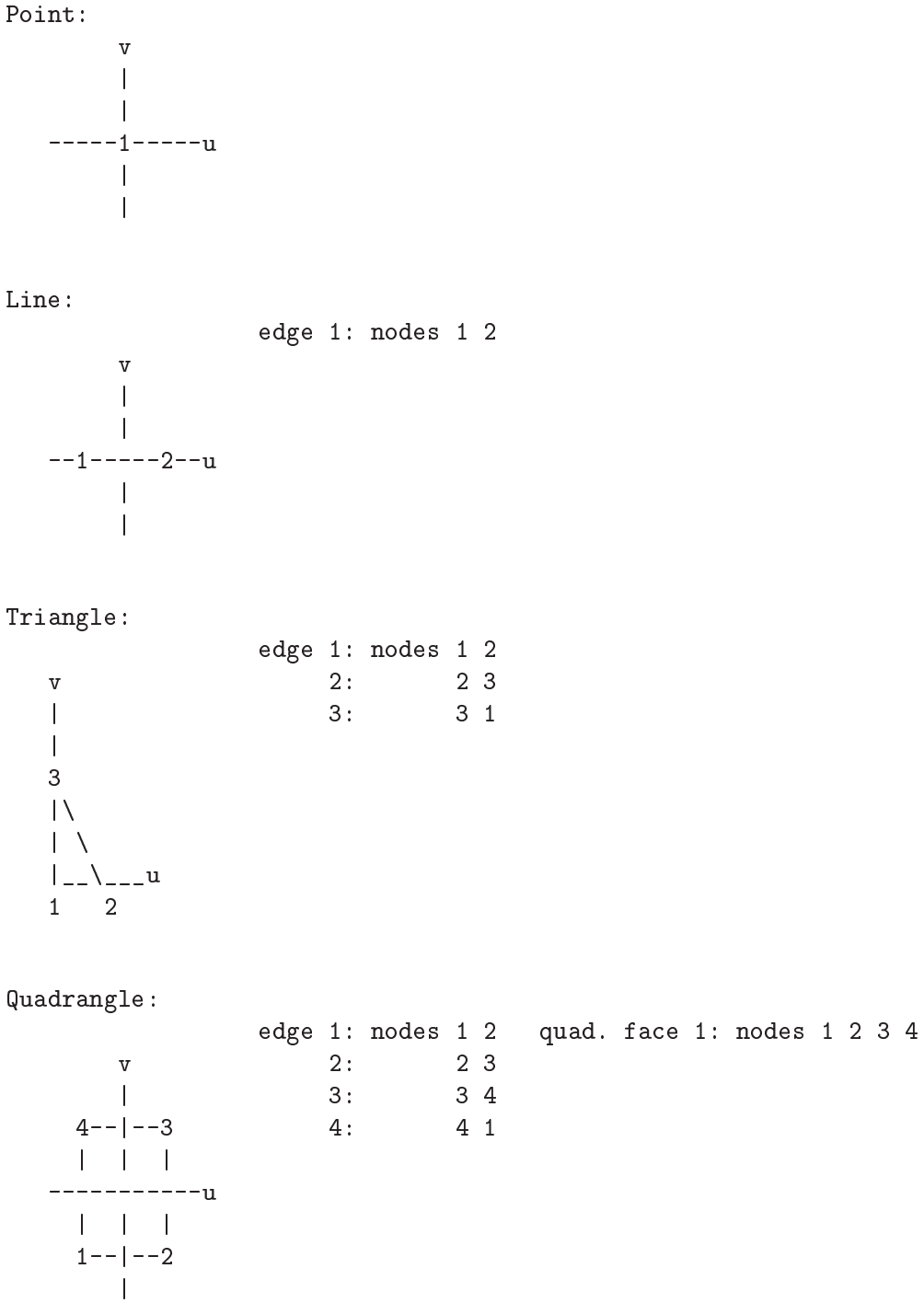
```
int one = 1;

fprintf(file, "$PostFormat\n");
fprintf(file, "%g %d %d\n", 1.2, 1, sizeof(double));
fprintf(file, "$EndPostFormat\n");
fprintf(file, "$View\n");
fprintf(file, "%s %d "
  "%d %d %d "
  "%d %d %d "
  "%d %d %d "
  "%d %d %d "
  "%d %d %d "
  "%d %d %d "
  "%d %d %d "
  "%d %d %d "
  "%d %d %d %d\n",
  view-name, nb-time-steps,
  nb-scalar-points, nb-vector-points, nb-tensor-points,
  nb-scalar-lines, nb-vector-lines, nb-tensor-lines,
  nb-scalar-triangles, nb-vector-triangles, nb-tensor-triangles,
  nb-scalar-quadrangles, nb-vector-quadrangles, nb-tensor-quadrangles,
  nb-scalar-tetrahedra, nb-vector-tetrahedra, nb-tensor-tetrahedra,
  nb-scalar-hexahedra, nb-vector-hexahedra, nb-tensor-hexahedra,
  nb-scalar-prisms, nb-vector-prisms, nb-tensor-prisms,
  nb-scalar-pyramids, nb-vector-pyramids, nb-tensor-pyramids,
  nb-text2d, nb-text2d-chars, nb-text3d, nb-text3d-chars);
fwrite(&one, sizeof(int), 1, file);
fwrite(time-step-values, sizeof(double), nb-time-steps, file);
fwrite(all-scalar-point-values, sizeof(double), ..., file);
...
fprintf(file, "\n$EndView\n");
```

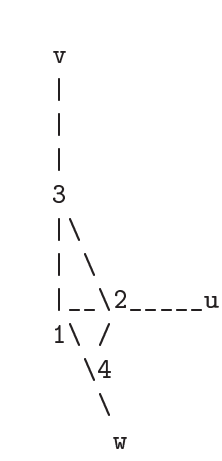
In this pseudo-code, *all-scalar-point-values* is the array of double precision numbers containing all the *scalar-point-value* lists, put one after each other in order to form a long array of doubles. The principle is the same for all other kinds of values.

9.3 Gmsh node ordering

For all mesh and post-processing file formats, the reference elements are defined as follows:

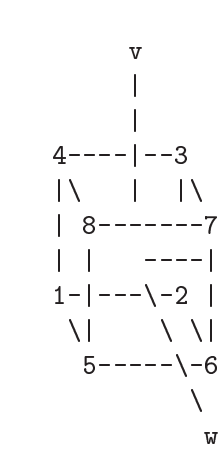


Tetrahedron:



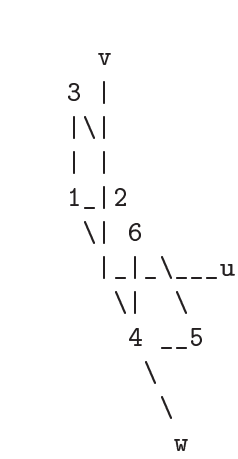
edge 1:	nodes	1 2
2:		2 3
3:		3 1
4:		4 1
5:		4 3
6:		4 2

Hexahedron:



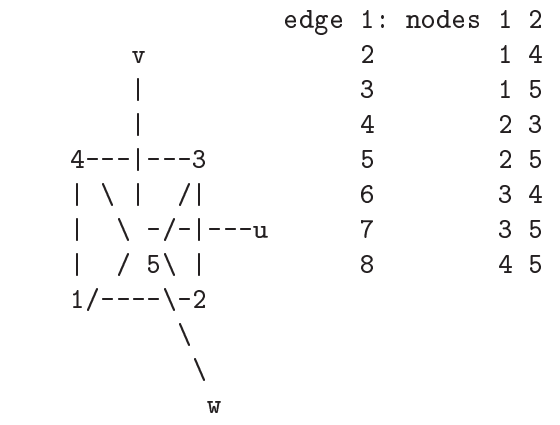
edge 1:	nodes	1 2	quad. face 1:	nodes	1 2 3 4
2:		1 4	2:		1 2 5 6
3:		1 5	3:		1 4 5 8
4:		2 3	4:		2 3 6 7
5:		2 6	5:		3 4 7 8
6:		3 4	6:		5 6 7 8
7:		3 7			
8:		4 8			
9:		5 6			
10:		5 8			
11:		6 7			
12:		7 8			

Prism:



edge 1:	nodes	1 2	quad. face 1:	nodes	1 2 4 5
2:		1 3	2:		1 3 4 6
3:		1 4	3:		2 3 5 6
4:		2 3			
5:		2 5			
6:		3 6			
7:		4 5			
8:		4 6			
9:		5 6			

Pyramid:



edge	nodes	quad. face	nodes
1	1 2	1	1 2 3 4
2	1 4		
3	1 5		
4	2 3		
5	2 5		
6	3 4		
7	3 5		
8	4 5		

10 Programming notes

Gmsh was originally written in C, and later enhanced with various C++ additions. The resulting code is a hybrid C/C++ beast, hopefully not too badly structured... The scripting language is parsed using Lex and Yacc (actually, Flex and Bison), while the GUI relies on OpenGL for the 3D graphics and FLTK for the widget set. See http://www.opengl.org, http://www.mesa3d.org and http://www.fltk.org for more information.

Gmsh's build system is based on autoconf, and should work on most modern platforms providing standard compliant C and C++ compilers. Practical notes on how to compile Gmsh's source code are included in the distribution. Note that compiling the Windows version requires the Cygwin tools (freely available from http://www.cygwin.com). See Appendix B [Frequently asked questions], page 153, for more information.

10.1 Coding style

If you plan to contribute code to the Gmsh project, here are some easy rules to make the code easy to read/debug/maintain:

1. please enable full warnings for your compiler (e.g., add `-Wall` to `FLAGS` in the '`variables`' file);
2. always use the `Msg()` function to print information, errors, ...;
3. indent your files using '`utils/misc/indent.sh`';

10.2 Option handling

To add a new option in Gmsh:

1. create the option in the `Context_T` class ('`Common/Context.h`') if it's a classical option, or in the `Post_View` class ('`Common/View.h`') if it's a post-processing view-dependent option;
2. in '`Common/DefaultOptions.h`', give a name (for the parser to be able to access it), a reference to a handling routine (i.e. `opt_XXX`) and a default value for this option;
3. create the handling routine `opt_XXX` in '`Common/Options.cpp`' (and add the prototype in '`Common/Options.h`');
4. optional: create the associated widget in '`Fltk/GUI.cpp`';
5. optional: if no special callback is to be associated to the widget, add the handling routine `opt_XXX` to the OK callback for the corresponding option panel (in '`Fltk/Callbacks.cpp`').

11 Bugs, versions and credits

11.1 Bugs

If you think you have found a bug in Gmsh, you can report it by electronic mail to the Gmsh mailing list at gmsh@geuz.org. Please send as precise a description of the problem as you can, including sample input files that produce the bug. Don't forget to mention both the version of Gmsh and the version of your operation system (see Section 8.3 [Command-line options], page 120 to see how to get this information).

See Appendix B [Frequently asked questions], page 153, and the 'TODO' file in the distribution to see which problems we already know about.

11.2 Versions

```
$Id: VERSIONS,v 1.264 2004/10/30 15:23:45 geuzaine Exp $

New since 1.56: generalized displacement maps to display arbitrary
view types; the arrows representing a vector field can now also be
colored by the values from other scalar, vector or tensor fields; new
adaptive high order visualization mode; new options for solvers
(SocketCommand and NameCommand) and views (ArrowSizeProportional);
fixed display of undesired solver plugin popups; enhanced interactive
plugin behaviour; new Plugin(HarmonicToTime); various small bug fixes
and enhancements;

New in 1.56: new post-processing option to draw a scalar view raised
by a displacement view without using Plugin(DisplacementRaise) (makes
drawing arbitrary scalar fields on deformed meshes much easier);
better post-processing menu (arbitrary number of views+scrollable+show
view number); improved view->combine; new horizontal post-processing
scales; new option to draw the mesh nodes per element; views can now
also be saved in "parsed" format; fixed various path problems on
Windows; small bug fixes.

New in 1.55: added background mesh support for Triangle; meshes can
now be displayed using "smoothed" normals (like post-processing
views); added GUI for clipping planes; new interactive
clipping/cutting plane definition; reorganized the Options GUI;
enhanced 3D iso computation; enhanced lighting; many small bug fixes.

New in 1.54: integrated Netgen (3D mesh quality optimization +
alternative 3D algorithm); Extrude Surface now always automatically
creates a new volume (in the same way Extrude Point or Extrude Line
create new lines and surfaces, respectively); fixed UNV output; made
the "Layers" region numbering consistent between lines, surfaces and
```

volumes; fixed home directory problem on Win98; new Plugin(CutParametric); the default project file is now created in the home directory if no current directory is defined (e.g., when double-clicking on the icon on Windows/MacOS); fixed the discrepancy between the orientation of geometrical surfaces and the associated surface meshes; added automatic orientation of surfaces in surface loops; generalized Plugin(Triangulate) to handle vector and tensor views; much nicer display of discrete iso-surfaces and custom ranges using smooth normals; small bug fixes and cleanups.

New in 1.53: completed support for second order elements in the mesh module (lines, triangles, quadrangles, tetrahedra, hexahedra, prisms and pyramids); various background mesh fixes and enhancements; major performance improvements in mesh and post-processing drawing routines (OpenGL vertex arrays for tri/quads); new Plugin(Evaluate) to evaluate arbitrary expressions on post-processing views; generalized Plugin(Extract) to handle any combination of components; generalized "Coherence" to handle transfinite surface/volume attributes; plugin options can now be set in the option file (like all other options); added "undo" capability during geometry creation; rewrote the contour guessing routines so that entities can be selected in an arbitrary order; Mac users can now double click on geo/msh/pos files in the Finder to launch Gmsh; removed support for FLTK 1.0; rewrote most of the code related to quadrangles; fixed 2d elliptic algorithm; removed all OpenGL display list code and options; fixed light positioning; new BoundingBox command to set the bounding box explicitly; added support for inexpensive "fake" transparency mode; many code cleanups.

New in 1.52: new raster ("bitmap") PostScript/EPS/PDF output formats; new Plugin(Extract) to extract a given component from a post-processing view; new Plugin(CutGrid) and Plugin(StreamLines); improved mesh projection on non-planar surfaces; added support for second order tetrahedral elements; added interactive control of element order; refined mesh entity drawing selection (and renamed most of the corresponding options); enhanced log scale in post-processing; better font selection; simplified View.Raise{X,Y,Z} by removing the scaling; various bug fixes (default postscript printing mode, drawing of 3D arrows/cylinders on Linux, default home directory on Windows, default initial file browser directory, extrusion of points with non-normalized axes of rotation, computation of the scene bounding box in scripts, + the usual documentation updates).

New in 1.51: initial support for visualizing mesh partitions; integrated version 2.0 of the MSH mesh file format; new option to compute post-processing ranges (min/max) per time step; Multiple views can now be combined into multi time step ones (e.g. for programs that generate data one time step at a time); new syntax: #var[] returns the

size of the list var[]; enhanced "gmsh -convert"; temporary and error files are now created in the home directory to avoid file permission issues; new 3D arrows; better lighting support; STL facets can now be converted into individual geometrical surfaces; many other small improvements and bug fixes (multi timestep tensors, color by physical entity, parser cleanup, etc.).

New in 1.50: small changes to the visibility browser + made visibility scriptable (new Show/Hide commands); fixed (rare) crash when deleting views; split File->Open into File->Open and File->New to behave like most other programs; Mac versions now use the system menu bar by default (if possible); fixed bug leading to degenerate and/or duplicate tetrahedra in extruded meshes; fixed crash when reloading sms meshes.

New in 1.49: made Merge, Save and Print behave like Include (i.e., open files in the same directory as the main project file if the path is relative); new Plugin(DecomposeInSimplex); new option View.AlphaChannel to set the transparency factor globally for a post-processing view; new "Combine Views" command; various bug fixes and cleanups.

New in 1.48: new DisplacementRaise plugin to plot arbitrary fields on deformed meshes; generalized CutMap, CutPlane, CutSphere and Skin plugins to handle all kinds of elements and fields; new "Save View[n]" command to save views from a script; many small bug fixes (configure tests for libpng, handling of erroneous options, multi time step scalar prism drawings, copy of surface mesh attributes, etc.).

New in 1.47: fixed extrusion of surfaces defined by only two curves; new syntax to retrieve point coordinates and indices of entities created through geometrical transformations; new PDF and compressed PostScript output formats; fixed numbering of elements created with "Extrude Point/Line"; use $GMSH_HOME as home directory if defined.

New in 1.46: fixed crash for very long command lines; new options for setting the displacement factor and Triangle's parameters + renamed a couple of options to more sensible names (View.VectorType, View.ArrowSize); various small bug fixes; documentation update.

New in 1.45: small bug fixes (min/max computation for tensor views, missing physical points in read mesh, "jumping" geometry during interactive manipulation of large models, etc.); variable definition speedup; restored support for second order elements in one- and two-dimensional meshes; documentation updates.

New in 1.44: new reference manual; added support for PNG output; fixed

```
small configure script bugs.

New in 1.43: fixed solver interface problem on Mac OS X; new option to
specify the interactive rotation center (default is now the pseudo
"center of gravity" of the object, instead of (0,0,0)).

New in 1.42: suppressed the automatic addition of a ".geo" extension
if the file given on the command line is not recognized; added missing
Layer option for Extrude Point; fixed various small bugs.

New in 1.41: Gmsh is now licensed under the GNU General Public
License; general code cleanup (indent).

New in 1.40: various small bug fixes (mainly GSL-related).

New in 1.39: removed all non-free routines; more build system work;
implemented Von-Mises tensor display for all element types; fixed
small GUI bugs.

New in 1.38: fixed custom range selection for 3D iso graphs; new build
system based on autoconf; new image reading code to import bitmaps as
post-processing views.

New in 1.37: generalized smoothing and cuts of post-processing views;
better Windows integration (solvers, external editors, etc.); small
bug fixes.

New in 1.36: enhanced view duplication (one can now use "Duplicata
View[num]" in the input file); merged all option dialogs in a new
general option window; enhanced discoverability of the view option
menus; new 3D point and line display; many small bug fixes and
enhancements ("Print" format in parser, post-processing statistics,
smooth normals, save window positions, restore default options, etc.).

New in 1.35: graphical user interface upgraded to FLTK 1.1 (tooltips,
new file chooser with multiple selection, full keyboard navigation,
cut/paste of messages, etc.); colors can be now be directly assigned
to mesh entities; initial tensor visualization; new keyboard animation
(right/left arrow for time steps; up/down arrow for view cycling); new
VRML output format for surface meshes; new plugin for spherical
elevation plots; new post-processing file format (version 1.2)
supporting quadrangles, hexahedra, prisms and pyramids; transparency
is now enabled by default for post-processing plots; many small bug
fixes (read mesh, ...).

New in 1.34: improved surface mesh of non-plane surfaces; fixed
orientation of elements in 2D anisotropic algorithm; minor user
```

```
interface polish and additions (mostly in post-processing options);
various small bug fixes.

New in 1.33: new parameterizable solver interface (allowing up to 5
user-defined solvers); enhanced 2D aniso algorithm; 3D initial mesh
speedup.

New in 1.32: new visibility browser; better floating point exception
checks; fixed infinite looping when merging meshes in project files;
various small clean ups (degenerate 2D extrusion, view->reload, ...).

New in 1.31: corrected ellipses; PostScript output update (better
shading, new combined PS/LaTeX output format); more interface polish;
fixed extra memory allocation in 2D meshes; Physical Volume handling
in unv format; various small fixes.

New in 1.30: interface polish; fix crash when extruding quadrangles.

New in 1.29: translations and rotations can now be combined in
extrusions; fixed coherence bug in Extrude Line; various small
bug fixes and additions.

New in 1.28: corrected the 'Using Progression' attribute for
tranfinite meshes to actually match a real geometric progression; new
Triangulate plugin; new 2D graphs (space+time charts); better
performance of geometrical transformations (warning: the numbering of
some automatically created entities has changed); new text primitives
in post-processing views (file format updated to version 1.1); more
robust mean plane computation and error checks; various other small
additions and clean-ups.

New in 1.27: added ability to extrude curves with Layers/Recombine
attributes; new PointSize/LineWidth options; fixed For/EndFor loops in
included files; fixed error messages (line numbers+file names) in
loops and functions; made the automatic removal of duplicate
geometrical entities optional (Geometry.AutoCoherence=0); various
other small bug fixes and clean-ups.

New in 1.26: enhanced 2D anisotropic mesh generator (metric
intersections); fixed small bug in 3D initial mesh; added alternative
syntax for built-in functions (for GetDP compatibility); added line
element display; Gmsh now saves all the elements in the mesh if no
physical groups are defined (or if Mesh.SaveAll=1).

New in 1.25: fixed bug with mixed recombined/non-recombined extruded
meshes; Linux versions are now build with no optimization, due to bugs
in gcc 2.95.X.
```

New in 1.24: fixed characteristic length interpolation for Splines; fixed edge swapping bug in 3D initial mesh; fixed degenerated case in geometrical extrusion (ruled surface with 3 borders); fixed generation of degenerated hexahedra and prisms for recombined+extruded meshes; added BSplines creation in the GUI; integrated Jonathan Shewchuk's Triangle as an alternative isotropic 2D mesh generator; added AngleSmoothNormals to control sharp edge display with smoothed normals; fixed random crash for lighted 3D iso surfaces.

New in 1.23: fixed duplicate elements generation + non-matching tetrahedra faces in 3D extruded meshes; better display of displacement maps; fixed interactive ellipsis construction; generalized boundary operator; added new explode option for post-processing views; enhanced link view behavior (to update only the changed items); added new default plugins: Skin, Transform, Smooth; fixed various other small bugs (mostly in the post-processing module and for extruded meshes).

New in 1.22: fixed (yet another) bug for 2D mesh in the mean plane; fixed surface coherence bug in extruded meshes; new double logarithmic scale, saturate value and smoothed normals option for post-processing views; plugins are now enabled by default; three new experimental statically linked plugins: CutMap (extracts a given iso surface from a 3D scalar map), CutPlane (cuts a 3D scalar map with a plane section), CutSphere (cuts a 3D scalar map with a sphere); various other bug fixes, additions and clean-ups.

New in 1.21: fixed more memory leaks; added -opt command line option to parse definitions directly from the command line; fixed missing screen refreshes during contour/surface/volume selection; enhanced string manipulation functions (Sprintf, StrCat, StrPrefix); many other small fixes and clean-ups.

New in 1.20: fixed various bugs (memory leaks, functions in included files, solver command selection, ColorTable option, duplicate nodes in extruded meshes (not finished yet), infinite loop on empty views, orientation of recombined quadrangles...); reorganized the interface menus; added constrained background mesh and mesh visibility options; added mesh quality histograms; changed default mesh colors; reintegrated the old command-line extrusion mesh generator.

New in 1.19: fixed seg. fault for scalar simplex post-processing; new Solver menu; interface for GetDP solver through sockets; fixed multiple scale alignment; added some options + full option descriptions.

New in 1.18: fixed many small bugs and incoherences in

post-processing; fixed broken background mesh in 1D mesh generation.

New in 1.17: corrected physical points saving; fixed parsing of DOS files (carriage return problems); easier geometrical selections (cursor change); plugin manager; enhanced variable arrays (sublist selection and affectation); line loop check; New arrow display; reduced number of 'fatal' errors + better handling in interactive mode; fixed bug when opening meshes; enhanced File->Open behavior for meshes and post-processing views.

New in 1.16: added single/double buffer selection (only useful for Unix versions of Gmsh run from remote hosts without GLX); fixed a bug for recent versions of the opengl32.dll on Windows, which caused OpenGL fonts not to show up.

New in 1.15: added automatic visibility setting during entity selection; corrected geometrical extrusion bug.

New in 1.14: corrected a few bugs in the GUI (most of them were introduced in 1.13); added interactive color selection; made the option database bidirectional (i.e. scripts now correctly update the GUI); default options can now be saved and automatically reloaded at startup; made some changes to the scripting syntax (PostProcessing.View[n] becomes View[n]; Offset0 becomes OffsetX, etc.); corrected the handling of simple triangular surfaces with large characteristic lengths in the 2D isotropic algorithm; added an ASCII to binary post-processing view converter.

New in 1.13: added support for JPEG output on Windows.

New in 1.12: corrected vector lines in the post-processing parsed format; corrected animation on Windows; corrected file creation in scripts on Windows; direct affectation of variable arrays.

New in 1.11: corrected included file loading problem.

New in 1.10: switched from Motif to FLTK for the GUI. Many small tweaks.

New in 1.00: added PPM and YUV output; corrected nested If/Endif; Corrected several bugs for pixel output and enhanced GIF output (dithering, transparency); slightly changed the post-processing file format to allow both single and double precision numbers.

New in 0.999: added JPEG output and easy MPEG generation (see t8.geo in the tutorial); clean up of export functions; small fixes; Linux versions are now compiled with gcc 2.95.2, which should fix the

```
problems encountered with Mandrake 7.2.

New in 0.998: corrected bug introduced in 0.997 in the generation of
the initial 3D mesh.

New in 0.997: corrected bug in interactive surface/volume selection;
Added interactive symmetry; corrected geometrical extrusion with
rotation in degenerated or partially degenerated cases; corrected bug
in 2D mesh when meshing in the mean plane.

New in 0.996: arrays of variables; enhanced Printf and Sprintf;
Simplified options (suppression of option arrays).

New in 0.995: totally rewritten geometrical database (performance has
been drastically improved for all geometrical transformations, and
most notably for extrusion). As a consequence, the internal numbering
of geometrical entities has changed: this will cause incompatibilities
with old .geo files, and will require a partial rewrite of your old
.geo files if these files made use of geometrical transformations. The
syntax of the .geo file has also been clarified. Many additions for
scripting purposes. New extrusion mesh generator. Preliminary version
of the coupling between extruded and Delaunay meshes. New option and
procedural database. All interactive operations can be scripted in the
input files. See the last example in the tutorial for an example. Many
stability enhancements in the 2D and 3D mesh algorithms. Performance
boost of the 3D algorithm. Gmsh is still slow, but the performance
becomes acceptable. An average 1000 tetrahedra/second is obtained on a
600Mhz computer for a mesh of one million tetrahedra. New anisotropic
2D mesh algorithm. New (ASCII and binary) post-processing file format
and clarified mesh file format. New handling for interactive rotations
(trackball mode). New didactic interactive mesh construction (watch
the Delaunay algorithm in real time on complex geometries: that's
exciting ;-). And many, many bug fixes and cleanings...

New in 0.992: corrected recombined extrusion; corrected ellipses; added
simple automatic animation of post-processing maps; fixed various bugs.

New in 0.991: fixed a serious allocation bug in 2D algorithm, which
caused random crashes. All users should upgrade to 0.991.

New in 0.990: bug fix in non-recombined 3D transfinite meshes.

New in 0.989: added ability to reload previously saved meshes; some
new command line options; reorganization of the scale menu; GIF
output.

New in 0.987: fixed bug with smoothing (leading to the possible
```

```
generation of erroneous 3d meshes); corrected bug for mixed 3D meshes;
moved the 'toggle view link' option to Opt->Postprocessing_Options.

New in 0.986: fixed overlay problems; SGI version should now also run
on 32 bits machines; fixed small 3d mesh bug.

New in 0.985: corrected colormap bug on HP, SUN, SGI and IBM versions;
corrected small initialization bug in postscript output.

New in 0.984: corrected bug in display lists; added some options in
Opt->General.

New in 0.983: corrected some seg. faults in interactive mode; corrected
bug in rotations; changed default window sizes for better match with
1024x768 screens (default X resources can be changed: see ex03.geo).

New in 0.982: lighting for mesh and post-processing; corrected 2nd
order mesh on non plane surfaces; added example 13.
```

11.3 Credits

```
$Id: CREDITS,v 1.19 2004/11/02 17:55:15 geuzaine Exp $

             Gmsh is copyright (C) 1997-2004

                  Christophe Geuzaine
                <geuzaine at acm.caltech.edu>

                             and

                 Jean-Francois Remacle
                <remacle at gce.ucl.ac.be>

Major code contributions to Gmsh have been provided by Nicolas Tardieu
<ntardieu at giref.ulaval.ca> (help with the GSL integration, new "STL
to elementary geometry" interface, Netgen integration).

Other code contributors include: David Colignon <David.Colignon at
univ.u-3mrs.fr> for new colormaps; Patrick Dular <patrick.dular at
ulg.ac.be> for transfinite mesh bug fixes; Laurent Stainier
<l.stainier at ulg.ac.be> for help with the MacOS port and the tensor
display code; Pierre Badel <badel at freesurf.fr> for help with the
GSL integration; and Marc Ume <Marc.Ume at digitalgraphics.be> for the
original list code.
```

Special thanks to Bill Spitzak <spitzak at users.sourceforge.net>, Michael Sweet <easysw at users.sourceforge.net>, Matthias Melcher <mm at matthiasm.com> and others for the Fast Light Tool Kit on which Gmsh's GUI is based. See http://www.fltk.org for more info on this excellent object-oriented, cross-platform toolkit.

Thanks to the following folks who have contributed by providing fresh ideas on theoretical or programming topics, who have sent patches, requests for changes or improvements, or who gave us access to exotic machines for testing Gmsh: Juan Abanto <juanabanto at yahoo.com>, Olivier Adam <o.adam at ulg.ac.be>, Guillaume Alleon <guillaume.alleon at airbus.aeromatra.com>, Eric Bechet <eric.bechet at epost.de>, Laurent Champaney <laurent.champaney at meca.uvsq.fr>, Pascal Dupuis <Pascal.Dupuis at esat.kuleuven.ac.be>, Philippe Geuzaine <geuzaine at gnat.colorado.edu>, Johan Gyselinck <johan.gyselinck at ulg.ac.be>, Francois Henrotte <fhenrott at esat.kuleuven.ac.be>, Benoit Meys <bmeys at techspace-aero.be>, Nicolas Moes <moes at tam9.mech.nwu.edu>, Osamu Nakamura <naka at hasaki.sumitomometals.co.jp>, Chad Schmutzer <schmutze at acm.caltech.edu>, Jean-Luc Fl'ejou <jean-luc.flejou at edf.fr>, Xavier Dardenne <dardenne at tele.ucl.ac.be>, Christophe Prud'homme <prudhomm at debian.org>, Sebastien.Clerc <Sebastien.Clerc at space.alcatel.fr>, Jose Miguel Pasini <jmp84 at cornell.edu>, Philippe Lussou <plussou at necs.fr>, Jacques Kools <JKools at veeco.com>, Bayram Yenikaya <yenikaya at math.umn.edu>, Peter Hornby <p.hornby at arrc.csiro.au>, Krishna Mohan Gundu <gkmohan at gmail.com>, Chris Stott <C.Stott@surrey.ac.uk>, Timmy Schumacher <Tim.Schumacher@colorado.edu>.

Appendix A Tips and tricks

- Install the 'info' version of this user's guide! On your (Unix) system, this can be achieved by
  1. copying all '`gmsh.info*`' files to the place where your info files live (usually '`/usr/info`'), and
  2. issuing the command `install-info /usr/info/gmsh.info /usr/info/dir`.

  You will then be able to access the documentation with the command `info gmsh`. Note that particular sections ('nodes') can be accessed directly. For example, `info gmsh surfaces` or `info gmsh surf` will take you directly to Section 3.1.3 [Surfaces], page 37.
- Use emacs to edit your files, and load the C++ mode! This permits automatic syntax highlighting and easy indentation. Automatic loading of the C++ mode for '`.geo`' files can be achieved by adding the following command in your `.emacs` file: `(setq auto-mode-alist (append '(("\\.geo$" . c++-mode)) auto-mode-alist))`.
- Define common geometrical objects and options in separate files, reusable in all your problem definition structures.
- Save your preferred options with 'Tools->Options->Save'. To reset default options, erase the `General.OptionsFileName` (usually '`.gmsh-options`' in your home directory) or use the 'Restore default options' button in 'Tools->Options->General->Output'.
- In the graphical user interface:
  - dragging the mouse in a numeric input field slides the value. The left button moves one step per pixel, the middle by 10 * step, and the right button by 100 * step;
  - selecting the content of an input field, or lines in the message console (Tools->Message console), copies the selected text to the clipboard;
- Read Appendix B [Frequently asked questions], page 153...

Appendix B Frequently asked questions

```
$Id: FAQ,v 1.45 2004/10/27 20:53:04 geuzaine Exp $

This is the Gmsh FAQ

*****************************************************************

Section 1: The basics

* 1.1 What is Gmsh?

Gmsh is an automatic three-dimensional finite element mesh generator,
primarily Delaunay, with built-in pre- and post-processing
facilities. Its primal design goal is to provide a simple meshing tool
for academic problems with parametric input and up to date
visualization capabilities.  One of the strengths of Gmsh is its
ability to respect a characteristic length field for the generation of
adapted meshes on lines, surfaces and volumes.

* 1.2 What are the terms and conditions of use?

Gmsh is distributed under the terms of the GNU General Public
License. See the file doc/LICENSE for more information, or go to the
GNU foundation's web site at http://www.gnu.org.

* 1.3 What does 'Gmsh' mean?

Nothing ;-)

(Note that in the US, people tend to pronounce 'Gmsh' as
'Gee-mesh'. Yeah.)

* 1.4 Where can I find more information?

<http://www.geuz.org/gmsh/> is the primary location to obtain
information about Gmsh. You will for example find a complete reference
manual as well as a searchable archive of the Gmsh mailing list
(<gmsh@geuz.org>) on this webpage.

*****************************************************************

Section 2: Installation

* 2.1 Which OSes does Gmsh run on?

Gmsh is known to run on Windows 95/98/NT/2000/XP, Linux, Mac OS X,
```

```
Compaq Tru64 Unix (aka OSF1, aka Digital Unix), Sun OS, IBM AIX, SGI
IRIX, FreeBSD and HP-UX. It should compile on any Unix-like operating
system, provided that you have access to a recent C and C++ compiler.

* 2.2 Are there additional requirements to run Gmsh?

You should have the OpenGL libraries installed on your system, and in
the path of the library loader. A free replacement for OpenGL can be
found at <http://www.mesa3d.org>.

* 2.3 What do I need to compile Gmsh from the sources?

You need a C and a C++ compiler (e.g. the GNU compilers gcc and g++)
as well as the GSL (version 1.2 or higher; freely available from
http://sources.redhat.com/gsl/) and FLTK (version 1.1.x, configured
with OpenGL support; freely available from http://www.fltk.org).
You'll also need the jpeg library if you want to save jpeg images, and
the libpng and zlib libraries if you want to save png images.

Under Windows, you will need the Cygwin tools and compilers (freely
available from http://www.cygwin.com).

* 2.4 How to I compile Gmsh?

Just type

./configure; make; make install

If you change some configuration options (type ./configure --help to
get the list of all available choices), don't forget to do 'make
clean' before rebuilding Gmsh.

*****************************************************************

Section 3: General problems

* 3.1 Gmsh [from a binary distribution] complains about missing
libraries.

Try 'ldd gmsh' (or 'otool -L gmsh' on Mac OS X) to check if all the
required shared libraries are installed on your system. If not,
install them. If it still doesn't work, recompile Gmsh from the
sources.

* 3.2 Gmsh keeps re-displaying its graphics when other windows
partially hide the graphical window.
```

```
Disable opaque move in your window manager.

* 3.3 The graphics display very slowly.

Are you are executing Gmsh from a remote host (via the network)
without GLX?  You should turn double buffering off (with the -nodb
command line option).

*********************************************************************

Section 4: Geometry module

* 4.1 Does Gmsh support NURBS curves/surfaces?

Not yet.

* 4.2 Gmsh is very slow when I use many transformations (Translate,
Rotate, Symmetry, Extrude, etc. ). What's wrong?

The default behavior of Gmsh is to check and suppress all duplicate
entities (points, lines and surfaces) each time a transformation
command is issued. This can slow down things a lot if many
transformations are performed. There are two solutions to this
problem:
- you may save the unrolled geometry in another file (e.g. with
  gmsh file.geo -0 > flat.geo), and use this new file for subsequent
  computations;
- you may set the 'Geometry.AutoCoherence' option to 0. This will
  prevent any automatic duplicate check/replacement. If you still
  need to remove the duplicates entities, simply add 'Coherence;' at
  strategic locations in your geo files (e.g. before the creation of
  line loops, etc.).

*********************************************************************

Section 5: Mesh module

* 5.1 What should I do when the 2D unstructured algorithm fails?

Try one of the other 2D algorithms, e.g.:
- on the command line: gmsh -algo tri
- in the interface: Tools->Options->Mesh->2D->Isotropic algorithm (Triangle)
- in input files: Mesh.Algorithm = 3
The old 2D algorithm may disappear once all its features are
integrated in the new ones, so please don't send bug reports on the
old algorithm anymore.
```

* 5.2 The new 2D unstructured algorithms also fail! Then what?

Send us your geometry, and we will investigate. Please keep the following in mind though: 2D (surface) meshes are generated by projecting a 2D mesh in the mean plane of the surface. This gives nice results only if the surface curvature is small enough. Otherwise you _have_ to cut the surface in pieces. For example, using half circles to define a cylinder will fail with the unstructured algorithm (you should define arcs with angles smaller than Pi, and thus define the cylinder with at least three patch surfaces).

* 5.3 What should I do when the 3D unstructured algorithm fails?

The 3D algorithm is still very experimental. Try to change some characteristic lengths in your input file to generate meshes that better suit the geometrical details of your structure.

* 5.4 I changed the characteristic lengths, but the 3D algorithm still does not work. What should I do?

Buy a professional mesh generator ;-) You can also try to use Netgen instead of the default algorithm for the 3D mesh. Note that all surface meshes have to be oriented with exterior normals in this case.

* 5.5 The 3D algorithm is reaaaaally slow. Can you improve it?

We are working on it. But since we have a (very) limited amount of time to spend on the development of Gmsh, this may take a while. For very big meshes, see the answer to the previous question...

* 5.6 The quality of the elements generated by the 3D algorithm is very bad.

Upgrade to Gmsh >= 1.54 and use "Optimize quality". If badly shaped elements still exist due to the surface recovery step, you can try to use Netgen instead of the default algorithm for the 3D mesh. Note that all surface meshes have to be oriented with exterior normals in this case.

* 5.7 Non-recombined 3D extruded meshes sometimes fail.

The swapping algorithm is not very clever at the moment. Try to change the surface mesh a bit, or recombine your mesh to generate prisms or hexahedra.

* 5.8 Tools->Visibility does not seem to work with extruded meshes.

```
This is partially fixed in Gmsh >= 1.54. Since numbers are explicitly
assigned to mesh entities in the extrude commands (which partially
destroys the geometry/mesh relationship), the Visibility tool will
only work as expected when displaying Elementary entities. To
visualize extruded Physical entities, the only solution is to save the
mesh, and to read it again.

* 5.9 Does Gmsh support curved elements?

Yes, Gmsh can generate both 1st order and 2nd order elements. To
generate second order elements, click on 'Second order elements' in
the mesh menu after the mesh is completed. To always generate 2nd
order elements, select 'Generate second order elements' in the mesh
option panel. From the command line, you can also use '-order 2'.

*****************************************************************

Section 6: Solver module

* 6.1 How do I integrate my own solver with Gmsh?

If you want to simply launch a program from within Gmsh, just edit the
options to define your solver commands (e.g. Solver.Name0,
Solver.Executable0, etc.), and set the ClientServer option to zero
(e.g. Solver.ClientServer0 = 0).

If you want your solver to interact with Gmsh (for error messages,
option definitions, post-processing, etc.), you will need to link your
solver with the 'GmshClient.c' file and add the appropriate function
calls inside your program. You will of course also need to define your
solver commands in an option file, but this time you should set the
ClientServer variable to 1 (e.g. Solver.ClientServer = 1). A complete
example on how to build a solver that interacts with Gmsh is available
at http://www.geuz.org/gmsh/doc/mysolver.tgz.

* 6.2 On Windows, Gmsh does not seem to find the solver
executable. What's wrong?

The solver executable (e.g. 'getdp.exe') has to be in your path. If
not, simply go to the solver options
(e.g. Solver->GetDP->Options->Executable) to specify its location.

On recent versions of Microsoft Windows (XP SP2), it seems that you
need to execute the solver (e.g. launch 'getdp.exe') at least once
independently of Gmsh in order to authorize future executions.

* 6.3 Can I launch Gmsh from my solver (instead of launching my solver
```

from Gmsh) in order to monitor a solution?

Sure. A simple C program showing how to do this is given in
'utils/misc/callgmsh.c'.

****

Section 7: Post-processing module

* 7.1 How do I compute a section of a plot?

Use 'View->Plugins->Cut plane'.

* 7.2 How do I animate my plots?

If the views contain multiple time steps, you can press the 'play'
button under the graphic window, or change the time step by hand in
the view option panel. You can also use the left and right arrow keys
to change the time step in all visible views in real time.

If you want to loop through different views instead of time steps, you
can use the 'Loop through views instead of time steps' option in the
view option panel, or use the up and down arrow keys.

* 7.3 How do I visualize a deformed mesh?

Load a vector view containing the displacement field, and select
'Vector type->Displacement' in the view options. If the displacement
is too small (or too large), you can scale it with the 'Displacement
factor' option. (Remember that you can drag the mouse in all numeric
input fields to slide the value!)

* 7.4 Can I visualize a field on a deformed mesh?

Yes, there are several ways to do that.

The easiest is to load two views: the first one containing a
displacement field (a vector view that will be used to deform the
mesh), and the second one containing the field you want to display
(this view has to contain the same number of elements as the
displacement view). You should then set 'Vector type' to
'Displacement' in the first view, as well as set 'Data source' to
point to the second view. (You might want to make the second view
invisible, too. If you want to amplify or decrease the amount of
deformation, just modify the 'Displacement factor' option.)

Another solution is to use the DiplacementRaise plugin.

* 7.5 Can I color the arrows representing a vector field with data from a scalar field?

Yes: load both the vector and the scalar fields (the two views must have the same number of elements) and, in the vector field options, select the scalar view in 'Data source'.

* 7.6 Can I color iso-value surfaces with data from another scalar view?

Yes, using the CutMap plugin.

* 7.7 Is there a way to save animations?

Yes. For example, have a look at tutorial/t8.geo or demos/anim-seq.script.

* 7.8 Is there a way to visualize only certain components of vector/tensor fields?

Yes: use 'View->Plugin->Extract'.

Appendix C License

GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, 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 show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

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

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions

for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you

indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING 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.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE 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.

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 public, 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 convey 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)` *yyyy 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 2 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, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston,
MA  02111-1307, USA.
```

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

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

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision 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, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This 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 Library General Public License instead of this License.

Concept index

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Syntax index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V