Centre of Excellence

MICROCARD-2

A user guide for mmg3d and ParMmg

March 18, 2026

Work on this documentation was funded by the MICROCARD-2 project (project ID 101172576). The project is supported by the EuroHPC Joint Undertaking and its members (including top-up funding by ANR, BMBF, and Ministero dello sviluppo economico). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or EuroHPC. Neither the European Union nor EuroHPC can be held responsible for them.

Co-funded by the European Union

Contents

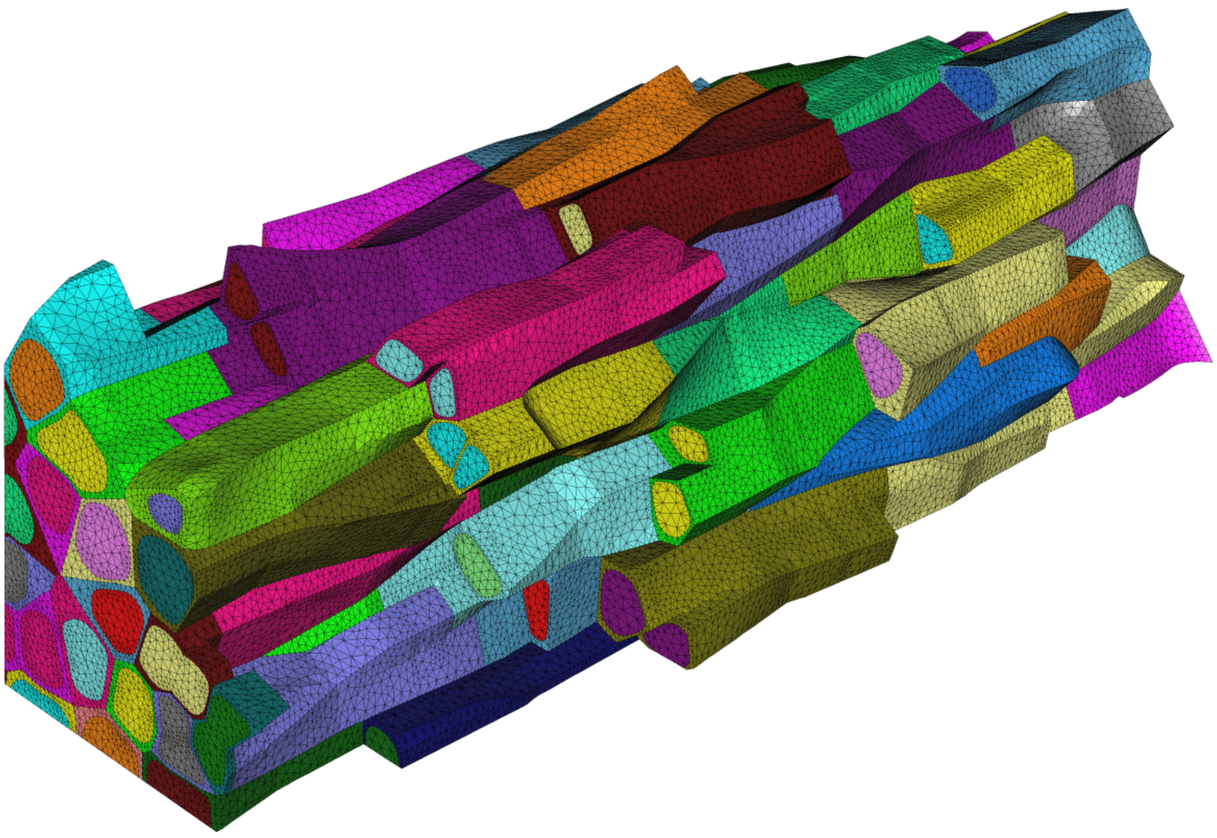

Figure 1: Example of a multimaterial mesh, remeshed with mmg3d. The colors distinguish the materials. In this case a mesh of cardiac tissue is shown, in which each cell has its own material and the space between the cells is partitioned, each part also having its own material.

1 Introduction

mmg3d [1, 2] is a mesh adaptation code for unstructured meshes with a particularly developed ability to handle multimaterial meshes, that is, meshes in which the volume elements are labeled with "references" and boundaries between elements with different references are preserved as internal surfaces. Mmg3d is available as a library and as a standalone program. The library is used in numerous applications and is widely used in academia. Its capabilities include

- improvement of element quality,
- adaptation of edge lengths to a possibly anisotropic and/or inhomogeneous metric,
- lagrangian movement of boundaries, and
- level-set discretization.

ParMmg is a process-parallel version of mmg3d [3, 4], initiated by Cécile Dobrzynski. In contrast to mmg3d it is not maintained by the Inria-led Mmg Consortium, but it was developed at Inria, with support from the European Union and national funding organizations. The code has a smaller developer base, and less developed documentation than the other members of the Mmg family.

This report provides a user manual for ParMmg, and covers mmg3d as well, but not its siblings mmg2d and mmgs, which are surface remeshing codes. The mmg3d and ParMmg programs share many concepts and options. In the following description we will write Mmg when we refer to either program.

2 Concepts

2.1 mesh entities

Boundaries are the same as *surfaces*.

Corners are vertices that are treated specially by the remesher. Continuity of surfaces and *ridges* will not be preserved at *corners*. The surface regularization (`-xreg` option) will leave *corners* in place, just like *required vertices*. (In fact, in mmg3d-5.7.0, the surface regularization did not respect *required vertices* but it did respect *corners*. In version 5.8 *required vertices* are respected too.) Mmg creates a corner when

1. three or more *edges* meet at a *vertex*,
2. one *edge* that is marked as *ridge*, *required* or *non-manifold* ends at a vertex, or
3. two edges with different *references* meet at a vertex.

Corners are specified in the .mesh format using the Corners keyword, using vertex indices.

Edges are straight line segments that connect a pair of *vertices*. Edges can be said to exist implicitly as parts of *triangles* or *tetrahedra*, but here *edges* refers to those edges that are explicitly declared in the input, inner workings, or output of Mmg. In fact, *edges* can only be created if they already exist implicitly as part of a tetrahedron. Edges specified explicitly in the input are *reference edges*, They are meaningful to Mmg if they are turned into *feature edges* by marking them as *ridges* or by making them *required*. In the .mesh format (section 9), edges are specified by the Edges keyword, with two vertex numbers and a reference each.

Domains are sets of elements (usually tetrahedra) with the same *reference*.

Feature edges are *reference edges* and edges that are listed as *ridges*. When one, three, or more than three *feature edges* meet in a *vertex*, Mmg will make this vertex a *corner*.

Non-manifold edges are (explicit or implicitly-defined) edges that are surrounded by tetrahedra with 3 or more references. The non-manifoldness refers to the surfaces in this case; the volume remains manifold.

Parts or **Partitions** are subsets of meshes or graphs. The proper term is "part" but the word "partition" is often used, also by the developers of Mmg, and therefore occurs in data keywords for example. For Mmg, parts are (usually compact) sets of volumic elements such as *tetrahedra*. Lower-dimensional entities such as *vertices* can be shared by parts.

References are integer tags that can be given to all mesh entities. They can influence the detection of *reference edges* and *surfaces*. References must be nonnegative; negative values are reserved for internal purposes.

Reference edges are *edges* between two triangles with different references [1] or edges explicitly provided in Mmg input (in this case they can be between two triangles with the same reference). The treatment will be the same in both cases.

Required entities are mesh entities that must remain as they are. That is, they cannot be deleted and their properties (coordinates and reference) and sub-entities must remain the same. The entity's index in the mesh can change though. Mmg recognizes required vertices, edges, triangles, quadrilaterals, and tetrahedra.

When Mmg reads required entities, it will make their sub-entities required in the output.

Required vertices are *vertices* that must remain where they are: they cannot be deleted or moved. In the .mesh format, required vertices are specified with the RequiredVertices keyword. Sometimes Mmg adds required vertices when it encounters a situation that it cannot handle.

Required edges are edges that must remain where they are. This means that the vertices at their ends are treated as *required vertices* and the edge cannot disconnect from either of them (it cannot be split, swapped or collapsed, for example). Sometimes Mmg adds required edges when it encounters a situation that it cannot handle. Required edges can be specified in the .mesh format with the RequiredEdges keyword.

Currently (Mmg version 5.8.0/ParMmg version 1.5.0) Mmg does not always preserve the *required* status of edges. It does not do so, for example, for edges that do not lie on any tetrahedron with face tags or references. Loosely said, *required edges* are lost when they do not lie on *surfaces*, *boundaries*, or on *triangles* with a nonzero reference. (Technically: the status can be lost if none of the tetrahedra neighbouring the edge has a face tag or a face reference. If they don't have these, the data structures that would store the required status can be removed.)

Required triangles are triangles that must remain where they are. This means that all three of their edges are treated as *required edges* and their vertices as *required vertices*. Moreover, no new vertices can be inserted in the triangle (this is an operation that Mmg will currently not perform anyway, but it may do so in the future). Required triangles can be specified in the .mesh format with the RequiredTriangles keyword.

Normally Mmg respects *triangles* and therefore *required triangles* only if they occur at the interface between two different domains or on the outer mesh boundary. When the `-opnbdy` option is given, other *required triangles* are also preserved.

Required tetrahedra are tetrahedra that must remain where they are. This means that all of their triangles, edges, and vertices are treated as *required*. Moreover, no vertices can be inserted inside.

Ridges are edges with a ridge flag. This flag can be set when the edge is marked as a ridge in the input, or when ridge detection is activated and the angle between the normals of the (two) triangles of a surface edge exceeds the ridge detection threshold. Internally, non-manifold edges (junctions between more than 3 domains) are also treated as ridges, but since version 5.7 these will generally not be mentioned as such in the output. When an edge is flagged as a ridge, surface smoothing will not be applied to its adjacent triangles. Ridge flags are always exported.

Solutions are numerical fields defined on mesh entities (usually vertices). They can be scalar, vector, or tensor, and can serve as metric, level-set function, or size map. They can be stored in the .sol data format described above.

Surfaces are sets of triangles whose two tetrahedra have different references, or that have only one tetrahedron (i.e. they form the outer surface of the mesh). Surfaces can be remeshed but their geometry will be preserved within the limits imposed by the `-hausd` parameter.

Tetrahedra are three-dimensional simplices. They are defined by four *vertex* numbers and a *reference*. Tetrahedron references define *domains* for Mmg, and are preserved, even when the tetrahedra themselves are not.

Triangles are two-dimensional simplices defined by three *vertex* numbers and a *reference*. In Mmg input they are usually only needed when mesh triangles need to be made *required*. Mmg will add *triangles* on all *boundaries*. It expects either all boundary triangles to be provided, or none. If boundaries are partially provided, Mmg will complete them and give a warning. Moreover, if the `-opnbdy` option is not given, Mmg will remove (non-*required*) *triangles* whose two tetrahedra have the same reference. Triangles and their *references* in Mmg output can be important for simulation codes as they are often used to specify boundary conditions.

Vertices are the most basic mesh entity. They are points in 3D space, defined by three cartesian coordinates and a *reference*.

2.2 mesh properties

2.2.1 valid and conforming meshes

Meshes are usually required to be valid and conforming. A tetrahedral mesh is *valid* if the interiors of any pair of tetrahedra are mutually disjoint, i.e. their intersection is empty. A tetrahedral mesh is *conforming* if it is valid and the intersection of the two tetrahedra is limited to a vertex, an edge, or a face of the mesh.

2.2.2 non-manifolds

Meshes are usually expected to be manifold: surfaces have two sides that cannot be reached from the other side and the boundaries of volume domains are surfaces. But Mmg can handle non-manifoldness to some extent.

A volumetric mesh can have multiple domains. The boundaries between these domains together define a non-manifold surface, as different boundaries touch or cross each other. Mmg tolerates this, and can even regularize the boundaries while leaving their intersections intact and in place. For this purpose it treats them internally as *ridges*.

Level set discretization can be performed on multiple volumetric domains, and it can work correctly even if the volumes are not strictly manifold (e.g. two tetrahedra of the same domain touching only on a vertex or an edge, with the surrounding space belonging to another domain). However, the resulting mesh will be difficult for the remesher and probably also for simulation software that uses it.

Remeshing of a non-manifold multimaterial mesh can result in errors, as Mmg performs tests on the weak spots to see if edges can be swapped, for example.

3 Overview

Mmg performs operations on tetrahedral meshes. By defaults it performs mesh modifications in order to bring the edge lengths in the mesh into the specified range, while striving to observe also the specified *gradation*, the maximum ratio of adjacent edge lengths. These operations generally result in improved element quality, even if Mmg does not target element quality explicitly.

The desired edge lengths can be provided in several ways. The `-hmin` and `-hmax` arguments specify bounds, and are intended to avoid numerical issues. Target edge lengths can be provided

- using `-hsiz` (for constant and isotropic edge length),
- using `-optim` to improve the mesh quality using the input edge lengths as a guide, or

- using a metric file (for local and/or anisotropic edge lengths).

Mmg has two different mesh improvement algorithms, named Delaunay and Pattern. The Delaunay algorithm is used in optimization mode and the Pattern algorithm in level-set mode. The Pattern algorithm is more robust with respect to ambiguities in the connectivity of domains in multidomain cases, but in other situations the reports vary.

When not explicitly disabled, this mesh improvement is always performed. It can be preceded by two other operations: level-set discretization and lagrangian displacement [5].

level-set discretization When a file containing a level-set function is provided, the `-ls` option allows to explicitly mesh an implicit domain and the `-lssurf` option splits only mesh boundaries on a provided isovalue. The file should be in Mmg's `.sol` format (see section 9) and should specify the function values on the vertices of the mesh.

lagrangian displacement When a displacement file is provided, the mmg3d `-lag` option applies a lagrangian displacement to boundary mesh vertices. This option is not yet available in ParMmg, and only available in mmg3d if it is compiled with the `USE_ELAS` directive.

return value Mmg returns

0 if successful,
1 if the process failed but could save a conforming mesh, and
2 if the process failed and couldn't save a conforming mesh.

3.1 ParMmg Algorithm

ParMmg works by partitioning a mesh (or reading an already partitioned mesh) and performing serial remeshing on the parts, with the part boundaries fixed. After remeshing, the mesh is repartitioned with different boundaries and the parts are serially remeshed again. This process is iterated several times. The default number of iterations is 3, but up to 20 iterations may be needed for hard meshes.

Optionally, ParMmg can further divide the parts into mesh groups, which are remeshed individually, also with their boundaries fixed [4]. Instead of remeshing one part, each ParMmg process now remeshes several groups one by one. The underlying idea is that remeshing time is more than proportionally smaller for small meshes. However, experience showed that the increased number of iterations needed because of the extra fixed boundaries can offset this advantage.

3.2 Running ParMmg

In order to enable parallel operation using the message passing interface (MPI), ParMmg is typically run with the `mpirun` program or an equivalent. For example:

```
mpirun -np 8 parmmg_O3 -hmin 1 -hmax 3 input.mesh
```

Because ParMmg must perform multiple iterations and needs to repartition inbetween, it can be expected to need about 4 processes to achieve the same runtime as the serial mmg3d. The speedup with more processes greatly depends on the size of the mesh parts.

4 Command-line arguments

4.1 Common options for mmg3d and ParMmg

There can be two non-option arguments. The first is taken to be the input mesh, and the second as the output mesh. If the first is ⟨basename⟩`.mesh(b)`, then Mmg will attempt to read a file ⟨basename⟩`.sol` which is assumed to be a metric file, except when the `-ls` option is given: in that case it is assumed to contain the level set function. The files that Mmg reads and writes can also be specified with options, as detailed in section 4.1.2. The extension of the output filename determines the output file format, as detailed in the description of the `-out` option.

In addition to the output mesh, Mmg will always write a file ⟨basename⟩`.sol`, which contains a metric.

4.1.1 General options

`-h` : Print help.

`-v` ⟨integer⟩ Tune the level of verbosity. This option takes values between -1 and 10.

- 0 is the lowest verbosity. The libraries don't print anything and the applications print minimal information (release, copyright, building date, input and output filenames);
- 1 is the default verbosity. It adds the input and output mesh qualities (minimal, maximal and mean element quality + index of the lowest quality element), a summary of the remeshing waves and information about the output mesh (number of each entity);
- 2 adds informations about the edge lengths (smallest, average and largest edge length + indices of the smallest and largest edges);
- 3 adds statistics about the edge length;
- 4 adds histograms of mesh quality and edge length and some analysis information;
- 5 adds more detailed remeshing waves and more analysis information;
- 6 prints every wave of remeshing.

`-m` ⟨integer⟩ allows to set the maximal amount of memory that Mmg may use (in MB). By default, mmg3d allows itself to take half the physical memory of the machine. ParMmg takes all the memory available on the node and distributes it among the MPI processes running on the same node. While mmg3d checks this limit strictly and terminates if it detects that it will need more, ParMmg is more permissive: it tries to respect the limit, but as in several part of the software the distributed mesh cannot be saved while maintaining its conformity through the parts, it tries to pursue until a step where the mesh will be valid.

`-d` Turn on debug mode. This will cause extra tests to be performed.

`-default` Write a local parameter file (with extension `.mmg3d`) with the values that were set. Details about parameter files can be found in section 5.

`-val` Print the default parameter values.

4.1.2 File specification

`-in` ⟨filename⟩ Input mesh. From ⟨filename⟩ Mmg will extract ⟨basename⟩, consisting of everything before the extension. Recognized extensions are listed under the `-out` option. If the extension is `.mesh` or `.meshb` and the file does not exist, ParMmg will use the filename to extract ⟨basename⟩ and attempt to read the files ⟨basename⟩`.`⟨index⟩`.mesh(b)` where ⟨index⟩ is an integer ranging from 0 to the number of processes with which ParMmg is running. This set of files must represent a distributed mesh with as many parts as there are processes. If the files specify a different number of parts then ParMmg will emit an error message and stop.

`-out` ⟨filename⟩ Output mesh file. If this option is not given, the second non-option argument is taken as the output filename. If that is not present either, the output filename will be ⟨basename⟩`.o.mesh`.
Similar to input files, ParMmg can translate this name into a series of filenames, one for each process.
The filename extension determines in which format the mesh is written. Mmg3d and ParMmg recognize `.mesh` and `.meshb` (section 9). Mmg3d in addition recognizes `.vtu` (VTK format), while ParMmg recognizes also `.h5` and `.xdmf` for HDF5 format (section 9.4), and `.pvtu` for the parallel version of the VTK format. HDF and VTK output are only available if ParMmg was compiled with the necessary libraries, which is optional.
When ParMmg writes parallel output in `.mesh` or `.meshb` format it writes one file per process. In all other cases the output is written in a single file.

`-sol` ⟨filename⟩ specifies that the metric, displacement or level-set function should be read from this file rather than from the default ⟨basename⟩`.sol`. The metric can also be specified with the `-met` option. The displacement is only used in lagrangian mode, the level set only in level-set mode, and the metric can be used in all modes. In mesh adaptation mode, either `-sol` or `-met` can be given, but not both, because there are then no other inputs than the metric. For the definition of the metric see the description of the `-met` option below.

`-met` ⟨filename⟩ Specifies the metric file [6]. This can be used instead of `-sol` in adaptation mode, and in addition to `-sol` in level-set mode. It has no effect in lagrangian mode. When a metric is given, Mmg strives to create edges that have unit length under the given metric. In other words, for an edge specified by a vector e and a metric specified by a tensor M, Mmg tries to approach $e'Me = 1$. In its most general form, M is a symmetric positive-definite tensor of rank 3 and order 2. To specify such a tensor, 6 elements must be specified: m_{11}, m_{12}, m_{13}, m_{22}, m_{23}, m_{33}. **Attention:** the order in a `.sol` file is very unusual: $m_{11}, m_{12}, m_{22}, m_{13}, m_{23}, m_{33}$. In the mmg3d API the order is normal. If the tensor is diagonal, only the 3 diagonal elements need to be specified. An isotropic tensor, equivalent to a scalar metric, can be specified with a single number.

`-f` ⟨filename⟩ Parameter file to load (see section 5)

4.1.3 Mode specification

The default mode is mesh adaptation. The following options change the mode.

`-lag` ⟨integer⟩ (only in mmg3d)
Lagrangian mesh displacement according to mode 0, 1 or 2. This option is only available when Mmg was compiled with the `USE_ELAS` compilation flag. The LinearElasticity library must be found to activate this flag. If this option is specified, the `-sol` option is supposed to specify a file containing a displacement vector for each vertex in the mesh.
Mmg provides 3 different Lagrangian displacement modes:

0 the object is moved inside the mesh using point relocation only (constant connectivity);
1 edge swapping is allowed but there is no point insertion/collapse;
2 all operators of the remsher are allowed (point insertion/collapse, edge swapping, point relocation).

In the first mode, only relatively small displacements can be achieved.

`-ls` ⟨float⟩ Perform level-set discretization, i.e. cut mesh edges where the given function (see above) has the given value, or is zero, when no value is provided. The mesh elements on both sides of the level set will be given different *references*. The values given can be controlled with a parameter file (section 5). If the input mesh already has different *domains*, the two references for each of them can be controlled individually.

`-lssurf` ⟨float⟩ Split only the boundary, not the volume, on the level set.

`-isoref` ⟨integer⟩ Choose the *reference* of the inserted iso-surface in level-set discretization mode

4.1.4 Parameters

`-A` Enable anisotropy (without metric file).

`-ar` ⟨float⟩ Threshold for angle detection. Boundary edges whose two face outward normals have an angle larger than this threshold will be treated internally as *ridges*. This has various consequences; among others such edges cannot be swapped. Thus, *ridges* create constraints for the remesher, and it can be desirable to limit their detection. With the `-nr` option the detection can be disabled entirely. If necessary, *ridges* that should be recognized can also be specified in the input mesh.
The angle is specified in degrees, the default value is 45. Internally, Mmg uses the cosine of this angle to take decisions. Setting the angle to 0 or 180 degrees disables angle detection like `-nr` does.

`-nr` Disable angle detection. Boundary edges whose two face normals have an angle larger than this threshold will be treated internally as *ridges*. This has various consequences; among others such edges cannot be swapped. Thus, *ridges* create constraints for the remesher, and it can be desirable to disable their detection. With the `-ar` option the detection can also be restricted above a given threshold. If necessary, *ridges* that should be recognized can also be specified in the input mesh.

`-hmin` ⟨float⟩

`-hmax` ⟨float⟩ These options allow to truncate the edge lengths to be greater than the hmin parameter and lower than hmax. The default values for these parameters are computed from the mesh bounding box or, if provided, from the given metric:

- without metric, hmin is set to 0.01 times the bounding box size and hmax to two times the bounding box size.
- with metric, hmin is set to 0.1 times the smallest prescribed size and hmax to 10 times the maximal prescribed size.

`-hsiz` ⟨float⟩ Strive for a constant edge length of this value.

`-hausd` ⟨float⟩ Control the allowed Hausdorff distance between the surface mesh and the underlying surface geometry. The Hausdorff distance is the maximum of the maxima of the distances from any point on one mesh to the other mesh. The distance is specified in the same units as the mesh. The default value is 0.01.

`-hgrad` ⟨float⟩ Limit the gradation (maximum ratio of edge lengths for edges that touch the same vertex). The default value is 1.3. A value of -1 disables gradation completely.

`-hgradreq` ⟨float⟩ Control the gradation from required entities toward others. The default value is 2.3.

`-rmc [`⟨float⟩`]` In level-set mode, remove components whose volume fraction is less than this value (10^{-5} if no value is provided) of the whole mesh volume. This option is not available in ParMmg.

`-opnbdy` Preserve input triangles at the interface of two domains of the same reference. By default, Mmg considers only boundaries that are delimiting the mesh (external boundaries) or that are at the interface between two different domains (internal boundaries). Thus, if the input contains a portion of surface (defined by Triangles in the mesh) that is not at the interface between two domains (defined by tetrahera with different references), this surface is deleted. With the -opnbdy option such surfaces will be retained. This option is not available in ParMmg.

`-nofem` Do not force Mmg to create a "finite element mesh" (FEM). A FEM will not have tetrahedra with multiple boundary faces. A FEM requirement puts extra constraints on the remesher, so it can be useful to relax this demand.

This option may be replaced by a positive `-fem` option in future versions of Mmg.

`-noinsert` Forbid insertion or deletion of points. The sequence `-noinsert -nomove -noswap` forbids all changes to the mesh. This can be useful to separate level-set discretization from mesh improvement.

`-nomove` forbids point relocation operators.

`-noswap` forbids edge-flipping and face-flipping operators.

`-nosurf` This option ensures that surfaces will not be modified, by making Mmg consider all surface edges as required edges. A side effect of this is that, because the lengths of required edges dictates the length of adjacent edges, all edge lengths near surfaces will adapt to the lengths of all surface edges and not just the required ones. This effect can be disabled with the `-nosizreq` option. The `-nosurf` option is to some extent redundant when the `-noinsert -nomove -noswap` options together: then all mesh modifications are disabled and nothing can happen to the surfaces. However, `-nosurf` also avoids a possibly costly analysis phase. For fastest operation, therefore, all four options should be given if no mesh improvement is required.

`-nosizreq` No imposition of the size of required edges over required vertices. Normally the average length of required edges at a vertex determines the length of all edges at that vertex. This option disables this requirement and allows other criteria to determine the edge lengths. Note that this applies to all edges that Mmg considers as required, not only those that are marked as such in the input mesh.

`-nreg` Normal regularization using laplacian-antilaplacian smoothing. This normalizes the length and orientation of the normals at vertices (which have been computed from the normals at triangles surrounding each vertex). Normals at vertices are used to compute the Bézier patch along which new surface points will be inserted and are included in Mmg output under the Normals keyword.

`-xreg [`⟨float⟩`]` Surface regularization by adaptation of vertex coordinates using laplacian-antilaplacian smoothing. This affects only Surface vertices, which includes those that lie on domain boundaries. The modifications are done near the end of the analysis stage, before any other mesh modifications are made. This option has an optional parameter: the regularization factor, with a default value of 0.4. The value must be between 0 and 1. Higher values yield stronger regularization with a higher risk of instability resulting in mesh problems.

`-nsd` ⟨integer⟩ Save only the subdomain with this index. The default value is 0, which saves all subdomains.

`-octree` ⟨integer⟩ Specify the maximal number of points per octree cell (only available without the PATTERN compilation flag). The octree is used by Mmg to partition the mesh vertices and thus, to speed up the insertion of vertices. During the insertion step, we don't want to insert a vertex too close to another one. Thus, before inserting a point, we search the octree cell to which the new point will belong and we check if it is not too close from another point of this cell or of one of the neighbouring cells. By default, an octree cell may contain at most 32 vertices, then it is split into 8 cells. You can use the `-octree` option to modify the maximal number of vertices per octree cell or to disable the octree (`-octree -1`).

`-optim` This allows to preserve the initial sizes of the mesh edges in order to improve the mesh quality without modifying the edge lengths. The prescribed size at a mesh vertex is computed (by the `MMG3D_DoSol` API function) as the mean of the lengths of the edges passing through this vertex. Mmg preserves the mean of the edge lengths at vertices. Thus, if the edges passing through a vertex have very different sizes, the resulting mesh may be far from the initial one. This option has no effect if a size map is provided.

`-optimLES` Strong isotropic mesh optimization for large-eddy simulations [3].

`-rn` ⟨integer⟩ Turn on (1) or off (0) vertex and element renumbering. Renumbering is done to ensure that neighbouring vertices/elements are near each other in memory, as much as possible. This can greatly improve the performance of codes that use the output mesh, and of Mmg itself. This option is only available if Mmg was compiled with the `USE_SCOTCH` compilation flag and if the SCOTCH library (`https://gitlab.inria.fr/scotch/scotch`) was found. The default value is 1.
The renumbering has a random component. Therefore, when renumbering is activated, the order in which Mmg sees the elements may vary from one run to another, and, with identical parameters, the resulting mesh may vary slightly in terms of number of vertices and elements, and quality.

4.2 Specific ParMmg options

`-niter` ⟨integer⟩ controls the number of parallel remeshing-repartitioning iterations. The default value is 3, which is quite low for most applications. More iterations can result in a better overall (mean or median) element quality, but there have been cases where the incidence of very bad elements was seen to increase with the number of iterations. This may be a particularity of complex multimaterial meshes.

`-nlayers` ⟨integer⟩ controls by how many mesh edges a parallel interface is displaced to repartition the mesh. A value of 2 or 3 is recommended. The default value is 2. High values can worsen load balancing.

`-mesh-size` ⟨integer⟩ controls the size (in elements) of the mesh groups that are sequentially remeshed on each process. The default value is negative, meaning that each part will be one mesh group. The argument must be a pure decimal number; exponents are silently ignored.

`-v` controls the verbosity of the ParMmg code itself

`-mmg-v` controls the verbosity of the internal (serial) mmg3d functions.

`-d` enables debug mode for ParMmg

`-mmg-d` enables debug mode for the internal mmg3d functions

`-field` ⟨filename⟩ loads a data field to interpolate from the input to the output mesh

`-noout` disables mesh output

`-centralized-output` instructs ParMmg to write output in a single file, even if the input was read from multiple files. By default ParMmg will write distributed output when the input was distributed. This option only has effect when `.mesh` or `.meshb` output is chosen because all other formats are intrinsically parallel (HDF5, VTU) or serial (pVTU).

`-distributed-output` instructs ParMmg to write output in multiple files, even if the input was read from a single file. By default ParMmg will write centralized output when the input was centralized. This option only has effect when `.mesh` or `.meshb` output is chosen because all other formats are intrinsically parallel or serial.

`-metis-ratio` ⟨integer⟩ specifies the number of metis supernodes per mesh

`-groups-ratio` ⟨float⟩ specifies the allowed imbalance between current and desired group size. It must be larger than 1 and the default value is 2. This option is obsolete and may be removed from future versions.

`-nobalance` disables load balancing of the output mesh

`-pure-partitioning` instructs ParMmg to perform only mesh partitioning; no level-set discretization or remeshing

5 Parameter file

A parameter file can be used to enable some specific features of Mmg. By default, Mmg reads a `DEFAULT.mmg3d` or a ⟨basename⟩`.mmg3d` parameter file.

The parameter file allows for example to set edge length requirements separately per element reference, and to specify a material mapping in level-set mode.

For more information see https://www.mmgtools.org/tutorials/tutorials_parameter_file.html.

6 Environment variables

The operation of Mmg can be influenced with environment variables. This is primarily meant for debugging purposes.

`MMG_SAVE_ANATET1`	save and quit after geometric remeshing (PHASE 2)
`MMG_SAVE_DEFSIZ`	save and quit just before gradation
`MMG_SAVE_GRADSIZ`	save and quit just after printing gradation info
`MMG_SAVE_ANATET2`	save and quit just after gradation and quality analysis (before optional renumbering and FEM topology testing)

7 Error and warning messages

This is a non-exhaustive overview of error and warning messages that may appear.

MMG5_coquilFaceErrorMessage and WARNING MMG5_coquilface An edge that is supposed to be manifold (part of exactly one surface) was found to have more than two boundary faces attached to it. This means that the surface involved is not in fact a manifold surface, and Mmg is trying to do something that requires it to be manifold. Mmg can deal with non-manifold edges but it will not perform the test on them that leads to this error message, so this really means that a surface you provided was not manifold, or Mmg mishandled it. *ridges* and required edges are also excluded from these tests.

Unable to collapse Mmg could not complete the operations that are supposed to reduce the number of mesh entities and increase the element quality. A mesh will be output but it won't be very good.

MMG5_minQualCheck: too bad quality for the worst element A tetrahedron has a volume of less than 10^{-15} in Mmg's internal normalized coordinates (in which the longest dimension of the mesh is 1).

8 Common problems

semantics of the input files Mmg uses `.sol` files for different purposes, and automatically reads the .sol file with the same base name as the mesh file it reads. Stale .sol files can cause error messages or unexpected behaviour when they were created for a different purpose. Note also that, for each mesh it saves, Mmg saves also a .sol file containing a size map. Without options telling it otherwise, Mmg will read this file when it is run on its own output.

ridges *ridges* in a mesh can make remeshing harder, and Mmg detects ridges automatically unless told not to do so with the `-nr` or `-ar` options. In some cases it can be better to let your own software mark the desired ridges and disable Mmg's ridge detection.

nonmanifold situations on the outer mesh boundary When the outer mesh boundary is flat, Mmg will normally leave it so. However, if a *domain* touches the outer mesh boundary with a single vertex, an ambiguity arises in the orientation of the surfaces that meet in this vertex and Mmg may move it slightly outward or inward. If this is undesirable it can be prevented by marking the vertex as a *required vertex*.

precision Since Mmg normalizes all coordinates before performing operations, and translates them back afterwards, it is possible that output coordinates are not exactly the same as input coordinates, even if a vertex did not move.

distributed input ParMmg does not check the data that describe the relations between mesh parts in distributed .mesh format (section 9.3). Errors and unexpected features may well cause it to crash.

9 File formats

9.1 native mesh format

Mmg's native mesh format is called .mesh format (after the extension), Medit format, MMG format (after the software) or Gamma Mesh Format (GMF, after the project team Gamma at Inria). It is a flexible and compact format for surface and volumetric meshes in two or three dimensions. It exists in both ascii and binary versions. Files in this format have the extensions `.mesh` and `.meshb`. The extensions are meaningful to the software: they distinguish the ascii and binary versions of the format.

Mmg mesh format comes in versions 1 and 2, version 2 having double-precision (64-bit) real numbers. GMF, which developed alongside Mmg and is not fully compatible, also has versions 3 and 4 with 64-bit integers. These differences are only meaningful in the binary version; the ascii version has in principle unlimited number ranges (but Mmg does have limits).

Since the binary version stores internal pointers in the file it also has limitations in file size. Versions 1 and 2 binary files are limited to 2GB.

The documentation on this format is a bit dispersed. Currently the most complete and up-to-date description is included with the source code of libMeshb (`https://github.com/LoicMarechal/libMeshb`) by Loïc Maréchal. LibMeshb is a very generic library for reading and writing files in .mesh(b) and .sol(b) formats.

There are older and incomplete descriptions in the book on mesh generation by Frey and George [7] and in the online documentation of the medit program (in French) (`https://www.ljll.fr/~frey/logiciels/Docmedit.dir/Docmedit.html`).

GMF continues to evolve: version 8 of libMeshb, currently under development, will recognize more keywords than the 207 that version 7 recognizes.

9.2 data or "solution" files

Files containing numerical data defined on mesh entities have extensions `.sol` and `.solb`, for "solution," and also come in ascii and binary versions. In some cases the data a user wishes to provide is indeed a solution to some equation, which governs for example the mesh density needed for its next approximation. But these formats are used for many different purposes. The files can provide a metric or "size map," a level set, etc. The data can be scalar or vector. The `.sol` format is a companion to the `.mesh` format and is similarly organized.

9.3 distributed .mesh format

In the ExaQUte project[1] a distributed version of the .mesh format has been developed. This means that the mesh is spread over multiple files, and the files contain information about how their mesh part relates to the other parts. The boundaries between the parts are described in

[1]EXAscale Quantification of Uncertainties for Technology and Science Simulation, EU project ID 800898, 2018–2021. The project does not have a website anymore but does have a Github group: `https://github.com/ExaQUte-project`.

terms of vertices or triangles (or both, but that is redundant). The distributed version can in principle use any version of the .mesh format, ascii as well as binary.

The format works with local numberings for all entities. To link with the neighbours, either the boundary vertices or boundary triangles are given a global numbering which is stored in the files, together with the numbers of the parts they are shared with. Normally, one would use one of the following combinations of keywords:

- `NumberOfPartitions` + `ParallelTriangleCommunicators` + `ParallelCommunicatorTriangles` or
- `NumberOfPartitions` + `ParallelVertexCommunicators` + `ParallelCommunicatorVertices`

In addition, `ParallelTriangles` and `ParallelVertices` can be given.

The format defines the following keywords:

NumberOfPartitions Total number of parts.

```
NumberOfPartitions
<Nparts>
```

ParallelTriangles This identifies the parallel triangles, i.e. the triangles that are shared with other parts. They are identified by their local triangle indices. This supposes that these triangles have been declared in the Triangles field. This field is redundant but some Mmg users rely on it. It is useful for applications that do not need to know with which other part a triangle is shared, and can be used for debugging. For debugging purposes, mmg3d can read this keyword too.

```
ParallelTriangles
N_parallel_triangles   # number of parallel triangles
id_t1       # local index (place in the Triangles field)
id_t2
...
id_tN
```

ParallelTriangleCommunicators This identifies the parts with which triangles are shared, and the number of triangles that are shared with each of them. The order in which they are stated attributes a sequential index to them, starting at zero. This index is used in ParallelCommunicatorTriangles. The partition identifiers are also numbers starting at zero.

```
ParallelTriangleCommunicators
Ncomm                  # number of parts we share triangles with
ID_part Nr_shared      # part ID and nr of triangles shared with it
ID_part Nr_shared
...
```

ParallelCommunicatorTriangles This connects the local triangle numbers with the global ones, and with the communicators listed in ParallelTriangleCommunicators. NOTE indeed it uses global indices; this was thought to be more practical for user input. The last number on each line is the sequential communicator index, not the part number itself. There is no count after the keyword; the number of lines that follows is the sum of all `Nr_shared` in `ParallelTriangleCommunicators`.

```
ParallelCommunicatorTriangles
ilocal iglobal icom
ilocal iglobal icom
...
```

ParallelVertices This works like ParallelTriangles. This field is redundant but some Mmg users rely on it. It is useful for applications that do not need to know with which other part or parts a vertex is shared, and for debugging. For debugging purposes, mmg3d can read this keyword too.

ParallelVertexCommunicators This works like ParallelTriangleCommunicators

ParallelCommunicatorVertices This works like ParallelCommunicatorTriangles

9.3.1 binary keywords

In the binary version of the .mesh format the field names are replaced by integers. Mmg uses the following values for the distributed mesh keywords:

```
70 ParallelTriangleCommunicators
71 ParallelVertexCommunicators
72 ParallelCommunicatorTriangles
73 ParallelCommunicatorVertices
74 NumberOfPartitions
```

These values are incompatible with Gamma Mesh Format/libMeshb, which has

```
70 ISolAtVertices,          "i", "i"
71 ISolAtEdges,             "i", "ii"
72 ISolAtTriangles,         "i", "iii"
73 ISolAtQuadrilaterals,    "i", "iiii"
74 ISolAtTetrahedra,        "i", "iiii"
```

where the last two columns indicate the numbers that are expected to follow.

No binary keywords have been defined for the (redundant) parallel entities `ParallelVertices`, `ParallelTetrahedra`, `ParallelTriangles`, `ParallelQuadrilaterals`, and `ParallelEdges`.

9.3.2 implementation features

ParMmg ignores `ParallelTriangles` and `ParallelVertices`.

Mmg does not save `ParallelVertices`, `ParallelTetrahedra`, `ParallelTriangles`, `ParallelQuadrilaterals`, and `ParallelEdges`.

If an input file contains both `ParallelTriangleCommunicators` and `ParallelVertexCommunicators`, ParMmg uses only the former. Consequently, it ignores `ParallelCommunicatorVertices` when `ParallelCommunicatorTriangles` is present.

ParMmg expects to read files named ⟨basename⟩`.`⟨index⟩`.mesh` where ⟨basename⟩ is the basename given as the input argument `-in` ⟨basename⟩`.mesh` and ⟨index⟩ is an integer number starting at zero and ranging to the number of parts. It will also output files this way, using the basename from the `-out` argument, or appending `.o` to the basename from the `-in` argument. When reading, ParMmg infers from the file ⟨basename⟩`.0.mesh` how many parts and files there are. Their number must match the number of processes with which ParMmg is running.

ParMmg will write the `ParallelCommunicatorTriangles` ordered by communicator: the triangles shared with the first communicator are listed first, followed by those shared with the second communicator, etc. This is a consequence of internal data structures and methods. There is no guarantee that it will remain this way.

ParMmg does not perform very strong checks on the mesh distribution data. Errors and unexpected features may well cause it to crash.

ParMmg expects the global indices to be larger than 0 and smaller than the total number of the respective entities. There are no checks but a larger number can cause a segmentation fault.

9.3.3 future plans

Because of the incompatibility with libMeshb and other difficulties the MICROCARD-2 consortium is considering a replacement of the distributed mesh keywords, to be discussed with the maintainers of Mmg and libMeshb and with other users.

9.4 other file formats

Mmg3d can read and write Parallel vtkUnstructuredGrid format (VTU) (`https://docs.vtk.org/en/latest/vtk_file_formats/vtkxml_file_format.html#punstructuredgrid`), while ParMmg supports the parallel version of this format (pVTU), but not the centralized version. These formats are only available if Mmg was compiled with VTK support.

Optionally, ParMmg can be compiled to also read and write Hierarchical Data Format version 5 (HDF5) (`https://support.hdfgroup.org/documentation`). The data structure is a straightforward translation of the .mesh format. Some of the keywords of the .mesh format become attributes in HDF5, e.g. `Dimension` and `NumberOfPartitions`. Data fields such as `Vertices` are split and renamed to store for example the integer *references* and the *corner* and *required* flags separately from the floating-point coordinates. Thus, vertex data are stored in the fields `ppoint`, `pref`, and `preq`.

Both meshes and data can be read and written in HDF5 format. Metadata accompanying the HDF5 files is written in eXtensible Data Model and Format (XDMF).

10 Other sources of information

The ParMmg wiki (https://github.com/MmgTools/ParMmg/wiki) covers command-line usage (tersely), library usage (which we do not cover in this report), and information about the algorithms of ParMmg. It also includes several tutorials.

The mmg3d, mmg2d and mmgs code and API documentation can be found on https://github.com/MmgTools/mmg.

The website of the Mmg consortium https://mmgtools.org/ has installation instructions, tutorials, descriptions of the algorithms and data structures, a list of publications, an image and movie gallery, and information for new contributors.

A technical report authored in 2019 by Cirrottola and Froehly explains the design decisions in the ParMmg code and compares it to alternatives [4].

LibMeshb (https://github.com/LoicMarechal/libMeshb) by Loïc Maréchal is a very generic library for reading and writing files in .mesh(b) and .sol(b) formats. The code repository includes a comprehensive English-language documentation of the formats.

11 About the MICROCARD project

Cardiac function is coordinated by an electric system whose disorders are among the most frequent causes of death and disease. Numerical models of this complex system are mature and widely used, but to match observations in aging and diseased hearts they need to move from a continuum approach to a representation of individual cells and their interconnections. This makes the problem more complex, harder to solve, and four orders of magnitude larger, necessitating exascale computers.

The EuroHPC-2019 MICROCARD project developed a simulation platform that can meet this challenge, by a joint effort of HPC experts, numerical scientists, biomedical engineers, and biomedical scientists, from academia and industry. The Centre of Excellence MICROCARD-2 will consolidate and scale up the MICROCARD results enabling digital twins of cardiac tissue.

We will further develop MICROCARD's numerical schemes, moving to second-order spatial discretization. We will develop mixed-precision preconditioners and data compression to reduce communication bandwidth. The highly successful efforts made in MICROCARD towards automated compilation of high-level model descriptions into optimized, energy-efficient system code for different CPUs and GPUs will be extended to upcoming architectures. We will continue efforts to robustify parallel remeshing software and add necessary functionality for parallel mesh partitioning and production of realistic synthetic tissue meshes needed for simulations.

The platform will be benchmarked with realistic test cases and be made accessible for a wide range of users with tailored workflows. It will be adaptable to similar biological systems such as nerves, and several of our products such as improved solvers, preconditioners, remeshers, and partitioners will be reusable in a wide range of applications.

The remeshing software mmg3d and its parallel version, ParMmg, are crucial tools for the MICROCARD projects. We have previously improved the robustness of these codes, in particular with respect to their ability to handle multimaterial meshes. However, since ParMmg is still an experimental code a user guide has been lacking so far. This report is a newly written user guide for ParMmg and it also covers important aspects of mmg3d which the existing documentation did not cover.

12 Acknowledgements

The initial development of ParMmg was performed within the framework of the ExaQUte project, funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 800898. It has received funding from the FUI22 ICARUS project, funded by Région Nouvelle-Aquitaine (RNA) under grant agreement 17001260-031 and the Fonds Unique intérministeriel (FUI) under grant agreement No DOS0050687100.

Improvements in the mmg3d and ParMmg codes were funded by the MICROCARD project (https://microcard.eu). This project has received funding from the European High-Performance Computing Joint Undertaking EuroHPC (JU) under grant agreement No 955495. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Italy, Germany, Austria, Norway, and Switzerland.

This user guide was initially written as Deliverable 5.2 of the MICROCARD-2 project (EU project ID 101172576). The project is supported by the EuroHPC Joint Undertaking and its members (including top-up funding by ANR, BMBF, and Ministero dello sviluppo economico). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or EuroHPC. Neither the European Union nor EuroHPC can be held responsible for them.

References

[1] C. Dapogny, C. Dobrzynski, P. Frey. Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. J. Comp. Phys., 2014;262:358–378. doi:10.1016/j.jcp.2014.01.005.

[2] G. Balarac, F. Basile, P. Bénard, F. Bordeu, J.-B. Chapelier, L. Cirrottola, G. Caumon, C. Dapogny, P. Frey, A. Froehly, G. Ghigliotti, R. Laraufie, G. Lartigue, C. Legentil, R. Mercier, V. Moureau, C. Nardoni, S. Pertant, M. Zakari. Tetrahedral remeshing in the context of large-scale numerical simulation and high performance computing. MathematicS In Action, 2022;11:129–164. doi:10.5802/msia.22.

[3] Pierre Benard, Guillaume Balarac, Vincent Moureau, Cecile Dobrzynski, Ghislain Lartigue, Yves D'Angelo. Mesh adaptation for large-eddy simulations in complex geometries. Int. J. Numer. Meth. Fluids, 2016;81:719–740. doi:10.1002/fld.4204.

[4] Luca Cirrottola, Algiane Froehly. Parallel unstructured mesh adaptation using iterative remeshing and repartitioning. Research report 9307, Inria, Nov. 2019. https://inria.hal.science/hal-02386837v1.

[5] Charles Dapogny, Florian Feppon. Shape optimization using a level set based mesh evolution method: an overview and tutorial. Comptes Rendus Mathématique, 2023;361:1267–1332. doi:10.5802/crmath.498.

[6] P. J. Frey, F. Alauzet. Anisotropic mesh adaptation for CFD computations. Comput. Methods Appl. MEch Engrg., 2005;194:5068–5082. doi:10.1016/j.cma.2004.11.025.

[7] Pascal Jean Frey, Paul-Louis George. Mesh Generation; Application to Finite Elements. Wiley, second edition, 2008.