D2.1 Meshing ”stub” implementation of the capabilities to be delivered

Document information table

<table>
<thead>
<tr>
<th>Contract number:</th>
<th>800898</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project acronym:</td>
<td>ExaQUte</td>
</tr>
<tr>
<td>Project Coordinator:</td>
<td>CIMNE</td>
</tr>
<tr>
<td>Document Responsible Partner:</td>
<td>INRIA</td>
</tr>
<tr>
<td>Deliverable Type:</td>
<td>Report, Other</td>
</tr>
<tr>
<td>Dissemination Level:</td>
<td>Public</td>
</tr>
<tr>
<td>Related WP &amp; Task:</td>
<td>WP 2 Task 2.1</td>
</tr>
<tr>
<td>Status:</td>
<td>Final Version</td>
</tr>
</tbody>
</table>

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800898
Authoring

<table>
<thead>
<tr>
<th>Authors</th>
<th>Partner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algiane Froehly</td>
<td>INRIA</td>
</tr>
<tr>
<td>Vicente Mataix</td>
<td>CIMNE</td>
</tr>
<tr>
<td>Carlos Roig</td>
<td>CIMNE</td>
</tr>
<tr>
<td>Contributors</td>
<td></td>
</tr>
</tbody>
</table>

Change Log

<table>
<thead>
<tr>
<th>Versions</th>
<th>Modified Page/Sections</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0</td>
<td>Created document, filling basic information</td>
<td></td>
</tr>
<tr>
<td>V1.1</td>
<td>Added serial mesher API</td>
<td></td>
</tr>
<tr>
<td>V1.2</td>
<td>Added parallel mesher API</td>
<td></td>
</tr>
<tr>
<td>V1.3</td>
<td>Corrected WP and related Task numbers</td>
<td></td>
</tr>
<tr>
<td>V1.4</td>
<td>Added Introduction</td>
<td></td>
</tr>
</tbody>
</table>

Approval

<table>
<thead>
<tr>
<th>Approved by:</th>
<th>Name</th>
<th>Partner</th>
<th>Date</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task leader</td>
<td>Algiane Froehly</td>
<td>INRIA</td>
<td>30.7.18</td>
<td>OK</td>
</tr>
<tr>
<td>WP leader</td>
<td>Algiane Froehly</td>
<td>INRIA</td>
<td>30.7.18</td>
<td>OK</td>
</tr>
<tr>
<td>Coordinator</td>
<td>Riccardo Rossi</td>
<td>CIMNE</td>
<td>30.7.18</td>
<td>OK</td>
</tr>
</tbody>
</table>
Executive summary

This document presents a description of the MMG interface adapted to Kratos as well as its parallel counterpart with ParMMG which will be implemented in task 2.1. In the description are included the following items:

- Description of the requirements of the interfaces;
- Description of the data structures used both in Kratos and MMG;
- Proposal of an initial interface for ParMMG.
- The input needed for the model;
- The definition of the model system matrices;
- The solver and time marching schemes;
- The results obtained, and the link to the 3D detailed model.
Table of contents

1 Introduction 8

2 Interface Description: existing MMG interface 8
   2.1 InitMesh .......................................................... 8
   2.2 InitializeMeshData .......................................... 8
   2.3 InitializeSolData ............................................ 10
   2.4 CheckMeshData ................................................ 10
   2.5 ExecuteRemeshing ........................................... 10

3 Interface Proposal: ParMMG interface 12
   3.1 InitMesh .......................................................... 12
   3.2 InitializeMeshData .......................................... 12
   3.3 InitializeSolData ............................................ 13
   3.4 ExecuteRemeshing ........................................... 14
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MMGProcess Initialization</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Submodelpart used in order to set BC</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Colour submodelpart creation</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>MMGProcess Initialize Mesh Data</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>MMGProcess Initialize Solution Data</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>MMGProcess Check</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>MMGProcess Remesh</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>MMGProcess Finalize</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>ParMMGProcess Initialization</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>ParMMGProcess Initialize Mesh Data</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>ParMMGProcess Initialize Metric Data</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>ParMMGProcess Remesh</td>
<td>22</td>
</tr>
<tr>
<td>13</td>
<td>ParMMGProcess Finalize</td>
<td>23</td>
</tr>
</tbody>
</table>
List of Tables

1 Nomenclature / Acronym list ........................................... 7
### Nomenclature / Acronym list

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>BC</td>
<td>Boundary Condition</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface</td>
</tr>
<tr>
<td>GP</td>
<td>Gauss Point</td>
</tr>
</tbody>
</table>

Table 1: Nomenclature / Acronym list


1 Introduction

This deliverable describes the status of the current implementation of Kratos Multiphysics [1] and MMG [2] [3] interface to provide an API which will allow to use its meshing capabilities as described in the WP2. Also, we define the first iteration of the future interface for parallel meshing based on ParMMG.

2 Interface Description: existing MMG interface

The way the interface will be exposed to Kratos Multiphysics is via a remeshing process that will be placed inside Kratos own meshing application (MeshingApplication) called MMGProcess. This process will prepare the input modelpart data and will convert the model into MMG format. This conversion will be done using several call to MMG exposed methods which are directly wrapped inside Kratos.

This Kratos process considers as input the modelpart of the problem we are interested in remesh and the configuration parameters that has been read from a .json file. This gives us a lot of flexibilty, because the parameters can be changed dinamically without refactoring, so adding functionalities to the process is easier than in a coventional input design.

It is important to notice at this point that MMG public API makes a distinction between 2D and 3D and offers two sets of different functions which are available to use. From the point of view of Kratos such distinction does not exist and the interface will be made in such a way that it remains transparent.

The proposed workflow of the process can be seen in the following workflow where the specific calls to MMG are displayed.

\[\text{Note that we abstract MMG2D and MMG3D into MMGX}\_ in order to simplify the diagram.}\]

\[\text{Note also that MMGX}\_\_Set\_iparameter and MMGX\_\_Set\_dparameter (for integer and doubles) are also abstracted to MMGX\_\_Set\_Xparameter for similar reasons.}\]

2.1 InitMesh

As its name indicates it will serve as the init function and directly call the init function of MMG. This will create and initialize the pointers of the mesh and the solution used by MMG. This calls MMGX\_Init\_mesh from the MMG API (see Figure [1]).

2.2 InitializeMeshData

Auxiliar sub model part creation

We will start with the creation of auxiliar sub model parts for flags. These model parts are created in order to preserve the flags of the nodes, elements and conditions after remesh. These auxiliar submodelparts will be removed after remesh.

Colour mapping

After that, we will follow with the creation of a colour map, or model part list. Here we will compute a map that will serve us in order to preserve Kratos sub-modelpart structure. In our implementations we use a process to set the BC (both Neumann or
Dirichlet). We identify the parts where to set this BC with the use of submodelparts (see Figure 2). The same component (node, condition, element) can belong to different submodelparts, so if we want to have an unique ids for each component we need to compute the common belonging to each component. We call this colour see Figure 3.

**Node preservation**

At this stage, if we want to preseve a part of the mesh we have the option to block individual nodes using the `MMGXD_Set_requiredVertex` function of MMG.

**Transfer mesh data to MMG**

In order to be able of doing that we will compute first the number of total nodes, elements and conditions in order to allocate the memory necessary to remesh. Also, we will convert Kratos nodes to MMG vertices and finally geometries in Kratos to geometries in MMG which will be mapped to edges, triangles, prism and tetrahedras depending on the source geometry (see Figure 4).

- **Kratos.Line2D2** → MMG Edge (MMG2D_Set_edge);
- **Kratos.Line3D2** → MMG Edge (MMG3D_Set_edge);
2.3 InitializeSolData

Here we allocate the metric. We always use tensor metrics by default due to its generality although scalar wrapper also exists.

This is done in two different steps, first we allocate the size using MMGX::Set::solSize and then setting its values using MMGX::Set::tensorSol, MMG::Set::vectorSol and MMGX::Set::solSize (see Figure 5).

2.4 CheckMeshData

Before proceed with the remeshing we call the routines from MMG designed in order to check that the information transferred to the API of MMG is consistent, this is the method MMGX::Chk::meshData(). See Figure 6.

2.5 ExecuteRemeshing

We call the remeshing process as well as setting the general configuration values in mmg. At detail, the list of parameters set in this step are the following:

- MMGX::DPARAM::hausd: The Hausdorff controls the smoothness of the boundary;
- MMGX::IPARAM::nomove: This parameters sets if we allow to avoid or allow the point relocation;
- MMGX::IPARAM::nosurf: This parameters sets if we don’t allow the surfaces modification;
- MMGX::IPARAM::noinsert: This parameters sets if we don’t insert nodes on mesh;

These parameters are registered as enums.
• **MMGX.DIPARAM_noswap**: This parameter sets if we don’t swap the mesh;

• **MMGX.DIPARAM_angle**: This parameter is used in order to deactivate/activate the angle detection;

• **MMGX.DPARAM_hgrad**: This parameter is used in order to set the gradation;

• **MMGX.DPARAM_hmin**: This parameter sets the minimum size of the mesh;

• **MMGX.DPARAM_hmax**: This parameter sets the maximum size of the mesh.

And finally we call the remeshing method **MMGX.mmgXdlib**() as seen in Figure [7].

After executing the remesh, the new nodes, conditions and elements are created and transferred to the corresponding submodelparts using the respective colours previously computed, which correspond with the **MMG** reference.

**Save output mesh and solution**

In case we are interested in getting the mesh and solution obtained in the .mesh and .sol file of **MMG** it exists the possibility to export the new mesh and solution to external files with the use of **SaveSolutionToFile()**. It calls the **MMGX.Set_outputMeshName** and **MMGX.Set_outputSolName** methods from the **MMG** API to set the filenames and the **MMGX.saveMesh** and **MMGX.saveSol** methods to write the output files.

**FreeMemory**

After we have computed all the relative to **MMG** we can now release the memory relative to it. We use the **MMGX.Free_all**() method from the API for that purpose. See Figure [8] for more details.

**ReorderAllIds**

During this step the ids of the element, conditions and nodes are reordered. So the ids will vary from 1 to n, being n the number of each component. This will not affect to the problem itself, due to the **OOP** nature of **Kratos**, if the ids are reordered, it will be reorder in all the objects depending of it.

This step must be done before and after remesh in order to avoid problematics with the remeshing process. Some steps of the process, including call to **MMG**, could do some assumptions about ordered ids of the different containers of the problem. Then the best way to avoid these problematics will be to reorder the ids, following the previously exposed methodology.

**Interpolation of mesh information**

This step is necessary in order to preserve the information of the old mesh in the new mesh. We use different processes and utilities in **Kratos** to interpolate both nodal values and integration values from the old mesh to the new mesh. In order to interpolate internal variables the following procedures are available:

• **Closest Point Transfer**: It transfer the values from the closest GP;

• **Least-Square projection Transfer**: It transfers from the closest GP from the old mesh;
• **Shape Function Transfer**: It transfers GP values to the nodes in the old mesh and then interpolates to the new mesh using the shape functions all the time.

**InitializeElementsAndConditions**

Finally, the created elements and conditions are initialized in order to be usable in *Kratos*.

### 3 Interface Proposal: *ParMMG* interface

We propose to update the exiting *MMGProcess* process of *Kratos* and to couple *Kratos-Multiphysics* and *ParMMG* in the same way as the *KratosMultiphysics-MMG* coupling:

- **MeshingApplication**(*Kratos*) ← **MMGProcess**(*KratosMultiphys.*) ← **MMG** API.
- The parameters and geometry of the problem are given as input for *Kratos* in a `.json` file.

This section describes the workflow for the *Kratos-ParMMG* coupling.

#### 3.1 InitMesh

It is the method used to initialize the **MMGProcess** process. It calls the initialization function of *ParMMG* (**PMMG_Init_mesh**) that creates the pointer toward the main structure of *ParMMG* (a **ParMesh**) and initializes its fields:

- the mesh dimension (unlike *MMG*, *ParMMG* doesn’t make distinction between 2D and 3D, thus it is needed to know the mesh dimension in order to call the suitable *MMG* functions for the *MMG* structures initialization);
- a pointer toward a *MMG* mesh;
- a pointer toward a *MMG* metric;
- a pointer toward an array of *MMG* solutions (if needed);
- the MPI communicator in which we will work.

See Figure 9 for the diagramm of the **MMGProcess** initialization.

#### 3.2 InitializeMeshData

**Auxiliar sub model part creation**

This part is similar to the creation of the sub model part with *MMG*.

**Node preservation**

The **PMMG_Set_requiredVertex** function of *ParMMG* allows to block individual nodes and to preserve a part of the mesh.
Transfer mesh data to ParMMG

On each processor, we will need to transfer the local mesh to ParMMG. Again, we will need to compute first the number of nodes, elements and conditions to be able to allocate the memory for the remesher. We set the mesh size calling the \texttt{PMMG\_Set\_meshSize} function. Then, the Kratos mesh is converted into a ParMMG mesh using the following map:

- \texttt{Kratos\_Line2D2} or \texttt{Kratos\_Line3D2} → PMMG Edge (PMMG\_Set\_edge);
- \texttt{Kratos\_Triangle3D3} → PMMG Triangle (PMMG\_Set\_triangle);
- \texttt{Kratos\_Quadrialeteral3D4} → PMMG Quadrilateral (PMMG\_Set\_quadrilateral);
- \texttt{Kratos\_Tetrahedra3D4} → PMMG Tetrahedron (PMMG\_Set\_tetrahedron);
- \texttt{Kratos\_Prism3D6} → PMMG prism (PMMG\_Set\_prism);

Additionally, must provide the communication data to ParMMG. Note that ParMMG uses the partition of the dual graph of the mesh (partition of the mesh elements) while Kratos uses a partition of the mesh (partition of both the mesh elements and nodes). It leads to have more communication informations in Kratos than in ParMMG (ghost nodes). For this preliminary version of the interface specification, we will only deal with the element partition (the element partition is given to ParMMG that gives back the new element partition, leaving to Kratos the responsibility of the nodes partition).

To set the communication data from Kratos to ParMMG we need:

- the local to global mapping of the nodes and elements using the \texttt{PMMG\_Set\_loc2GlobVertices} and \texttt{PMMG\_Set\_loc2GlobTetrahedra};
- for each couple of adjacent processors, the processors IDs and the list of the nodes at their interface (PMMG\_Set\_interfaceNodes).

3.3 InitializeSolData

As ParMMG modifies both the mesh and its distribution among the processors, it must performs the solutions interpolation over the new mesh. It implies to give the solutions to ParMMG.

Thus, the InitializeSolData process allocates now the metric and the solutions structure (Figure 11).

Metric allocation

We allocate the metric structure given its size (PMMG\_Set\_metSize), then we set the metric values using PMMG\_Set\_tensorMet.

Solutions array allocation

We allocate the array of solution structure given the number of solutions per entity and the type of each solution (scalar, vectorial or tensorial) using the PMMG\_Set\_allSolsSizes function. Then we set each solution value using the PMMG\_Set\_ithSol\_inAllSols function.
3.4 ExecuteRemeshing

The ExecuteRemeshing process sets the ParMMG parameters and run the ParMMG library. The list of parameters setted in this step are the following:

- **PMMG_DPARAM_hausd**: The Hausdorff controls the smoothness of the boundary;
- **PMMG_IPARAM_nomove**: This parameters sets if we allow to avoid or allow the point recolocation;
- **PMMG_IPARAM_nosurf**: This parameters sets if we don’t allow the surfaces modification;
- **PMMG_IPARAM_noinsert**: This parameters sets if we don’t insert nodes on mesh;
- **PMMG_IPARAM_noswap**: This parameters sets if we don’t swap the mesh;
- **PMMG_IPARAM_angle**: This parameters is used in order to deactivate/activate the angle detection;
- **PMMG_DPARAM_hgrad**: This parameter is used in order to set the gradation;
- **PMMG_DPARAM_hmin**: This parameters sets the minimum size of the mesh;
- **PMMG_DPARAM_hmax**: This parameters sets the maximum size of the mesh.

Because of the mesh repartitionning, the interpolation of the solutions from the old mesh into the new one must be done in ParMMG.

The interpolation method is chosen depending on the **PMMG_IPARAM_interp** argument and the wanted method:

- **PMMG_INTERP_linear**: linear interpolation of the solution from the old mesh in the new one;
- **PMMG_INTERP_closestPoint**: transfer of the solution from the closest GP.

Last we call the remeshing method **PMMG_parmmglib_distributed()** as seen in Figure 12.

After the remeshing step, the new nodes, conditions and elements are created and transferred to the corresponding submodelparts using the respective colours previously computed, which correspond with the ParMMG reference.

**Save output mesh, metric and solutions**

We can save the mesh, metric and solutions at .mesh and .sol format (ParMMG format) using the **SaveSolutionToFile()** process. It calls the **PMMG_Set_outputMeshName**, **PMMG_Set_outputMetName** and **PMMG_Set_outputSolsName** methods from the *PMMG API* to set the filenames and the **PMMG_saveMesh**, **PMMG_saveMet** and **PMMG_saveAllSols** methods to write the output files.
**FreeMemory**

We release the *ParMMG* memory calling the `PMMG.Free_all()` method. See Figure 13 for more details.

**ReorderAllIds**  The only reordering needed from *Kratos* is before remeshing in order to avoid potential problematics. After remesh *Kratos* will take the ids provided by *ParMMG* which already computes an internal reordering.

**InitializeElementsAndConditions**

Finally, the created elements and conditions are initialised in order to be usable in *Kratos*.

\[^2\]These parameters are registered as enums
References


Figure 4: MMGProcess Initialize Mesh Data
Figure 5: MMGProcess Initialize Solution Data

Figure 6: MMGProcess Check
Figure 7: MMGProcess Remesh

Figure 8: MMGProcess Finalize
Figure 9: ParMMGProcess Initialization
Figure 10: ParMMGProcess Initialize Mesh Data
Figure 11: ParMMGProcess Initialize Metric Data

Figure 12: ParMMGProcess Remesh
Figure 13: ParMMGProcess Finalize