MSc in Computational Mechanics
Universitat Politecnica de Catalunya
CIMNE-Kratos Multiphysics
Tutor: Riccardo Rossi
Marc Núñez

Enhancement of the mmg process in Kratos Multiphysics

Federico Parisi

20-09-2020

Contents

1. Abstract

In this report it is going to be explained the enhancement of the mmg remeshing process in the simulation software *Kratos Multiphysics* from CIMNE.

The enhancement explained here, will regard the remeshing process and the generation of a .mdpa file in which the mesh are stored. The required tools to implement the code and check its correctness will be briefly explained. Then the software will be explained and how it works, the remeshing process (mmg) and the implementation, together with the results. Finally it will be explained, in a detailed way, how to use the above-mentioned function in order to consequently save and store different refined mesh.

This method is based on the so-called Hessian remeshing.

2. Prerequisites

The software that is the topic of this report, is *Kratos Multiphysics*. It is filed on the online platform of *Github*. In order to work on it, it has to be downloaded from this archive. The platform and the community of *Github* will be better described in paragraph 2.2.

In order to see the results of *Kratos Multiphysics*, it is necessary a software that let the output files to be shown. This because the above-mentioned code has no interface, but it is a simulation code that will give as outputs some files. Those files will be plugged in another program in order to be seen. The software used for this purpose is *GiD* 15.0 and will be described in the following paragraph.

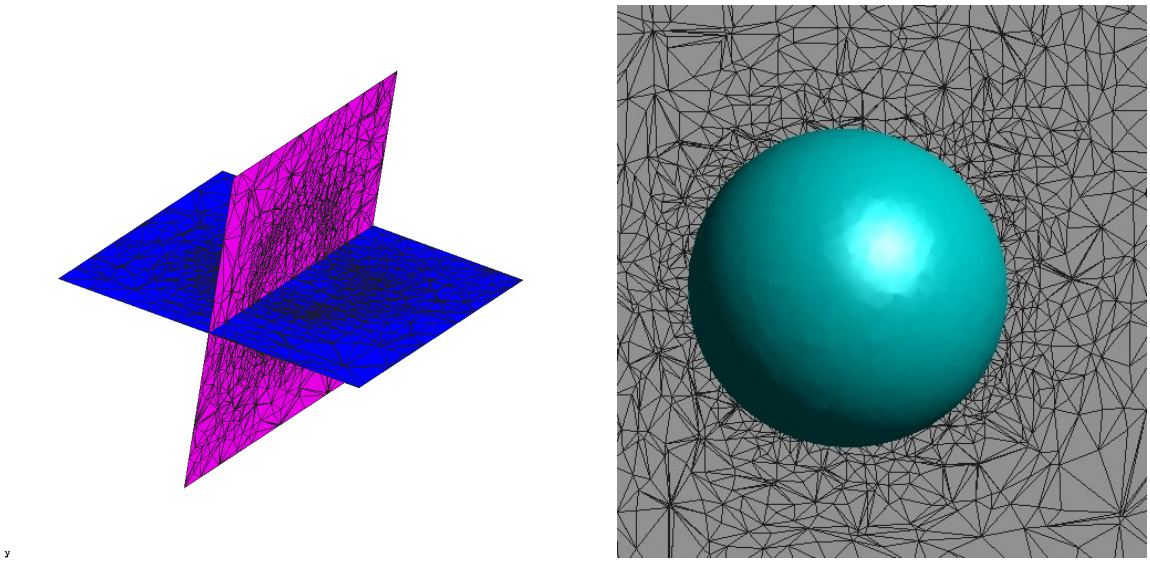
2.1 GiD 15.0

GiD 15.0 is a software for pre and post processing. In this report, it will be mainly considered the post processing part, as the output files will be analyzed in order to compare the different meshes generated.

The post process of GiD lets the user to study the results under different point of view. It permits to see the mesh generated, together with the simulation going on.

In the case of this implementation, GiD post processor has been used in order to check the difference in geometry and in meshes after the Hessian remeshing process. This process will be described in paragraph 3.1. Moreover, it has been used to check the correctness of the implementation of the code. In order to do so, after having stored the different meshes in different files, they have been loaded in the GiD post processor. These files allow to check if the right parameters have been caught.

An other useful option that has been learned during this study, is cutting the iso planes, cutting the objects in order to see the interior nodes. In figure 5.2 a) and b) are shown the iso surfaces of a sphere plotted in GiD, and the planes cut.



(a) Planes Cut (b) Sphere with Mesh

Figure 2.1: GiD Tools Examples

In figure 5.2 a) is shown the cut planes, over which are projected the interior nodes on the considered plane. In figure 5.2 b) is shown the whole sphere with the external meshes and a smooth render of the object, in chapter 3.1 will be appreciated the utility of this function.

2.2 Github

Github is an online community of developer, in which are archived various code. This website let the developers see, download, modify and upload the required scripts. In order to do so, a Git terminal will let the laptop communicate with the online archives of Github. From this terminal, the user can easily handle the files on the laptop and on the website. In this way, the scripts can be stored in the community.

When implementing a script, after cloning the code and writing the implementation, the modification done has to be approved by the community of that software. In the case of this report, the considered script is the *mmg_process.py*, which uses an external library for remeshing in *Kratos Multiphysics*. This library will be described in paragraph 3.1.

Github uses branches to define the path of the implementations in order to link the implementation to the topic and to the developer who did it. The branches also link the *Github* online platform to the laptop of the single developer, so to upload the changes done.

After the approval of the implementation, the change can be merged to the script.

3. The Software

Kratos Multiphyisics is a simulation software that uses python and C++ in order to work. This software calls different libraries depending on the problem, from the structural analysis, to the fluid-dynamics applications. The library that will be described in this report, and then implemented, is the *mmg_process.py*.

3.1 MMG library

This library executes a refinement of the mesh during the simulation run. This remeshing is based on a meshing technique called $metric - based\ refinement$ in which a metric value is imposed at every point in space.

To better understand this technique, the reader is invited to take a look to the article and the book in the References (6, 6).

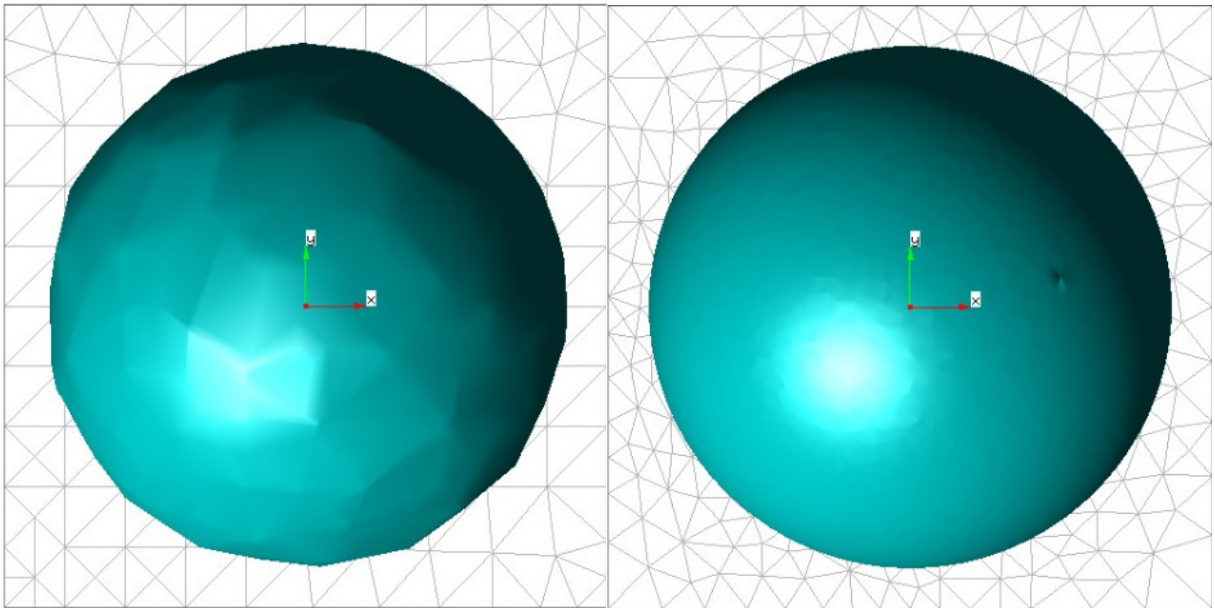
3.2 Kratos Multiphysics

What will be focused on this report, is the structure of this library, and how it works. This library is written in C++ that defines different classes. Each of these classes is called by a parameters that can be set to $true$ or $false$ depending on the needings of the users.

In this function, the main processes are 4. Two of them are launched before and after the whole simulation, while the other two are launched at the beginning and at the end of each time step.

In order to call this process, the .json file has to be modified, adding the required parameters in the "*auxiliar_process*" list. When this function is called, it will run the above-mentioned four C++ classes. Depending on the settings of the parameters, the functions will be launched or not. The main function called in this process, is the $ExecuteRefinement$ that uses the metric-based refinement with level set techniques in order to refine the meshes.

The efficiency of this method can be appreciated comparing the two Spheres in figures 3.1.



(a) Without refinement (b) With refinement

Figure 3.1: Hessian Remeshing

In figure 3.1 it is shown the comparison between a discretized sphere, before and after applying the metric-based refinement process.

As it can be seen, the coarse mesh does not get accurately the geometry of the object, but the mesh is easy to compute. On the other hand, while decreasing the interpolation error, the accuracy of the mesh increases and the object is much smoother. The mesh is going to be refined only in the region of interest, as noticing on the side of the domain, the mesh is still coarse, as the object is not touching it.

Refining the mesh means that the software can easily get the discontinuities in the flow (like vortices etc). But refining everywhere it is going to be computationally expensive, so this method is focused on refining where those discontinuities become a problem for the solver. That is why it is based on an average of certain parameters. In the case of a fluid problem, usually those parameters are the velocity and pressure, but it can be any quantity.

4. Implementation

Up to now, the *mmg_process* was taking part at each time step. That means that it was so computationally expensive, as it was computing each time step the average in every node and applying the remeshing process. This can be very useful in some cases, but will make the simulation very slow.

While this is not necessary for the considered case, it can be more useful to save some refined mesh at posteriori, when the simulation ends. This process has been implemented in the *ExecuteFinalize* class explained in chapter 3.1. This implementation can be activated or deactivated depending on the user needs, setting the related flag in the default parameters.

The useful part of this implementation is that, imposing some for loops in the main kratos file, the software will save different meshes that can be used later for the simulation to run.

This will save a lot of time as it will refine the mesh only at the end of the simulation, instead of doing it at each time step. The saved mesh files will be refined at each for loop, in order to have different level of refinements, depending on the user needs. The way to do it will be better explained in the following chapter.

In figures 4.2 and 4.1 are shown the implementation done.

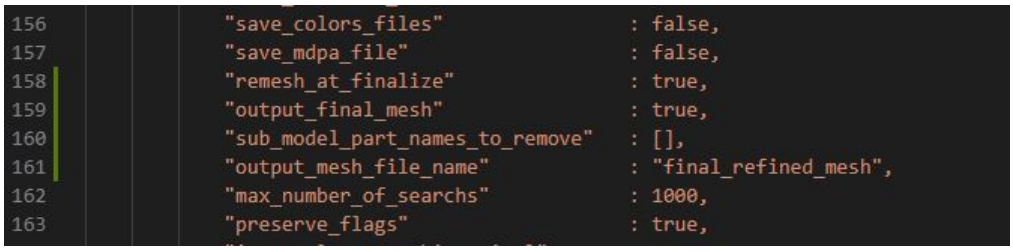


Figure 4.1: Default Parameters

In Figure 4.1 the flags that lead the process are imposed (lines 157 to 159). These flags are set to *true* if the process has to be launched while executing the script, or to $false$ if not. Those can be set even in the .json file for each case, as will be better explained in chapter 4.3.

Line 160 is needed in order not to have more sub model parts. This is because, if so, Kratos starts having problems in getting which sub model part to call and it will crash.

Line 161 is where it is imposed the name of the file that will contain the new mesh. In order

to save different files, as can be requested from the user, it will need to be called in a for loop. This process will be better described in the chapter 4.3.

```
424     def ExecuteFinalize(self):
425         """ This method is executed in order to finalize the simulation and save the refined mesh in a new .mdpa file
426
427         Keyword arguments:
428         self -- It signifies an instance of a class.
429         """
430         self.remesh_at_finalize = self.settings["remesh_at_finalize"].GetBool()
431         self.output_final_mesh = self.settings["output_final_mesh"].GetBool()
432         output_mesh_file_name = self.settings["output_mesh_file_name"].GetString()
433         sub_model_part_names_to_remove = self.settings["sub_model_part_names_to_remove"].GetStringArray()
434         if self.remesh_at_finalize:
435             for sub_model_part_name in sub_model_part_names_to_remove:
436                 if self.main_model_part.HasSubModelPart(sub_model_part_name):
437                     self.main_model_part.RemoveSubModelPart(sub_model_part_name)
438             self._ExecuteRefinement()
439         if self.output_final_mesh:
440             KratosMultiphysics.ModelPartIO(output_mesh_file_name, KratosMultiphysics.IO.WRITE | KratosMultiphysics.IO.MESH_ONLY).WriteModelPart(self.main_model_part)
441
```

Figure 4.2: Class Definition

In figure 4.2 it is shown the implementation of the class mentioned in chapter 3.1.
The $ExecuteRefinement$ is automatically launched at the end of each simulation. The features inside that class are launched only if they are activated in the default parameter in figure 4.1.

As it can be seen, the script first gets the boolean variable in order to check if they are set to true or false. Secondly, it will get the name of the mesh files that will be saved, finally it will get the name of the model part to remove in order not to over-read them.

Then they will start the if condition, accordingly to the boolean variables. If the remeshing parameter is set to true, it will first remove the part that has to be removed (if present), then it will execute the metric-based refinement, shown in the Apppendix 7.

Finally, if the user wants to save the mesh, so the boolean variable is set to true, the script will save it in a file with the name imposed in the default parameters.

4.1 Instructions

All the features described above, are executed at the end of the simulation, once. This means that if the user wants to save more .mdpa files storing different meshes, it will be necessary to impose a for loop in which the simulation runs inside it.

In this case, the user may want to have different characteristics for each file, in order to store different sets of meshes and re-use them later. To this purpose, in this section some instructions are written in order to well use this implementation.

```
67  if __name__ == "__main__":
68
69      for i in range(0, 5):
70
71          with open("ProjectParameters.json",'r') as parameter_file:
72              parameters = KratosMultiphysics.Parameters(parameter_file.read())
73
74          interpo_error = parameters["processes"]["auxiliar_process_list"][1]["Parameters"]["hessian_strategy_parameters"]["interpolation_error"].GetDouble()
75          interpo_error = interpo_error/2**i
76          parameters["processes"]["auxiliar_process_list"][1]["Parameters"]["hessian_strategy_parameters"]["interpolation_error"].SetDouble(interpo_error)
77
78          if i == 0:
79              new_mesh = parameters["solver_settings"]["model_import_settings"]["input_filename"].GetString()
80          else:
81              new_mesh = "remeshed_hessian_mesh_"+str(i)
82
83          model = KratosMultiphysics.Model()
84          simulation = FluidDynamicsAnalysisWithFlush(model,parameters)
85          simulation.Run()
86
87          main_model_part = model.GetModelPart('FluidModelPart')
88
89          parameters["solver_settings"]["model_import_settings"]["input_filename"].SetString(new_mesh)
90          parameters["processes"]["auxiliar_process_list"][1]["Parameters"]["output_mesh_file_name"].SetString(new_mesh)
91
92          CreateGidControlOutput("remeshed_hessian"+str(i), main_model_part)
```

Figure 4.3: 5 Loops

In figure 4.3 it is shown an example of a five times loop in order to save and store different meshes. In this example, what is differing between each mesh, is the interpolation error. Decreasing the interpolation error each loop, the generated mesh will be each time finer.

Looking at the script, it can be seen how this function works. First of all, the class previously defined will be executed in line 85, while the simulation is launched. In order to refine the mesh, the interpolation error is called from the settings in which it has been defined. In this case, it is defined in the .json file in the auxiliar processes in which the parameters are set to launch the mmg. After calling it and modifiyng according to the user's needs, it is set again in order to launch the simulation with the new parameter.

The same is done to save and store the new meshes in lines 89-90.

It has to be noticed that for the first loop, the script has to call the original .mdpa file, containing the geometry. This is necessary in order to make the code working as it needs a previous file to analyse and refine. Then the new .mdpa file can be set with the preferred name, in the same way as explained for the interpolation error.

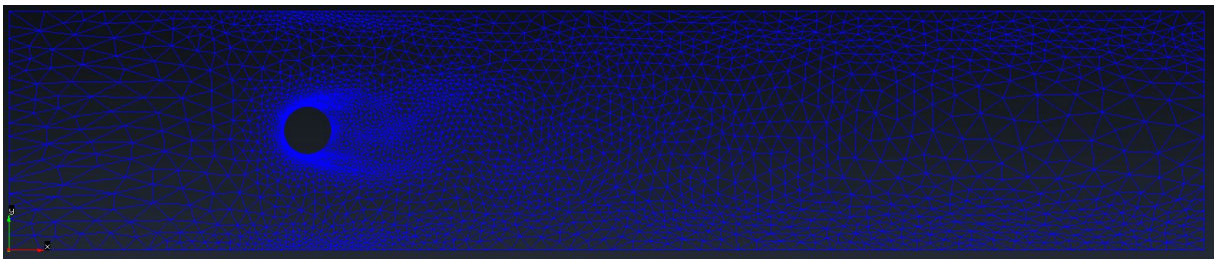
The last line is needed in order to create a file that can be visualized in GiD and it is not related to this implementation.

5. Results

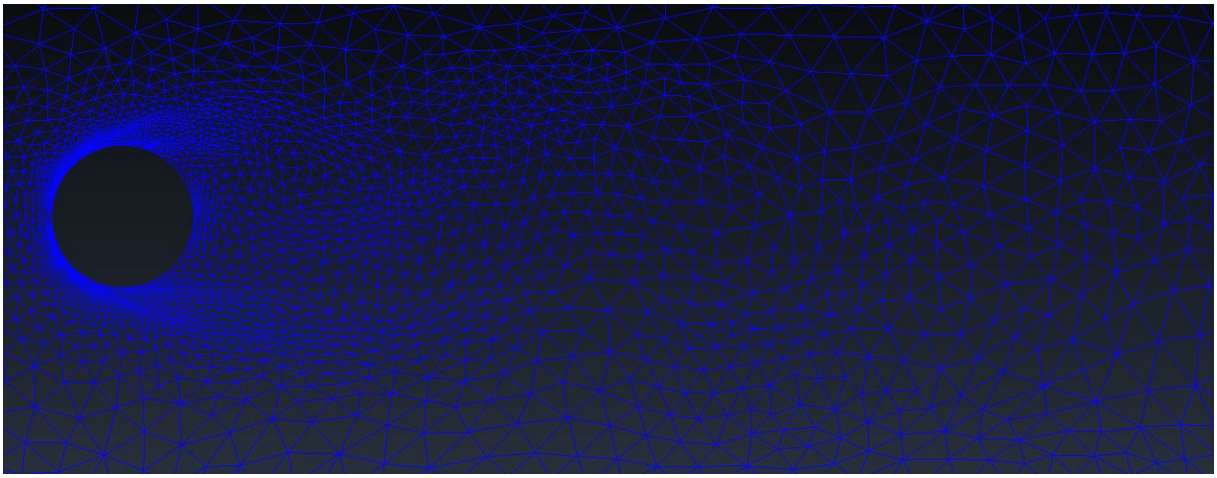
In conclusion, the code has been tested, and in this section are going to be shown the differences of the stored meshes each loop.

In order to check the correctness and efficiency of the code, a 40 seconds flow around a cylinder has been studied. The fluid has a dynamic viscosity of:

$$\nu = 1x10^{-3};$$



(a) Full System



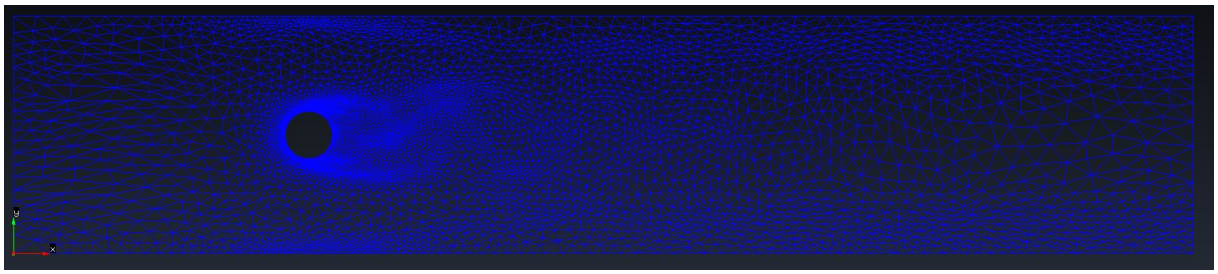
(b) Vortices Zoomed

Figure 5.1: Mesh 0

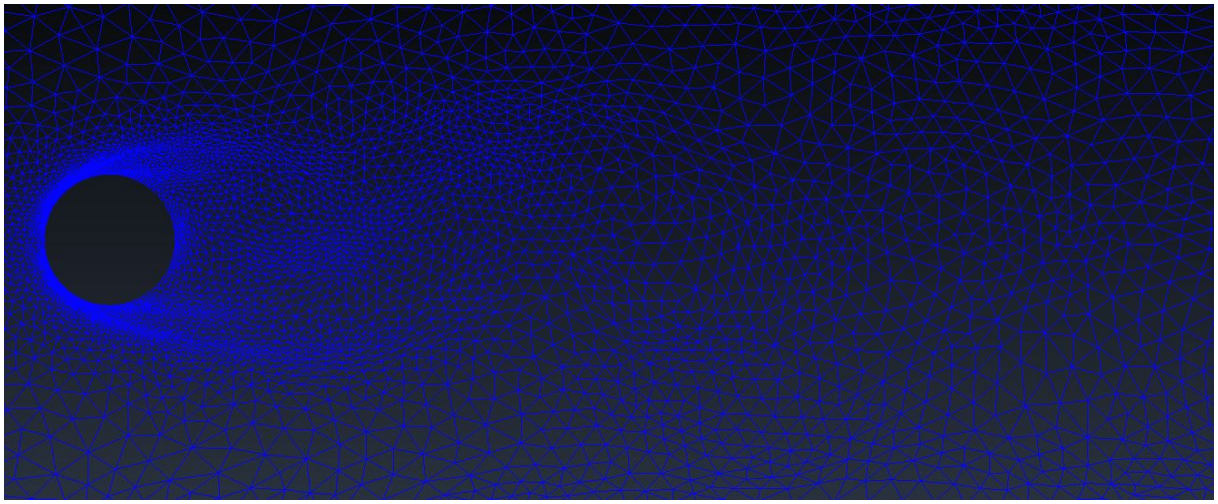
The meshes shown in figure 5.1 are the initial ones, with an interpolation error equal to:

$$interp\_error = 0.01;$$

As can be appreciated, the script is able to catch the vortices, but the mesh are not so refined.



(a) Full System



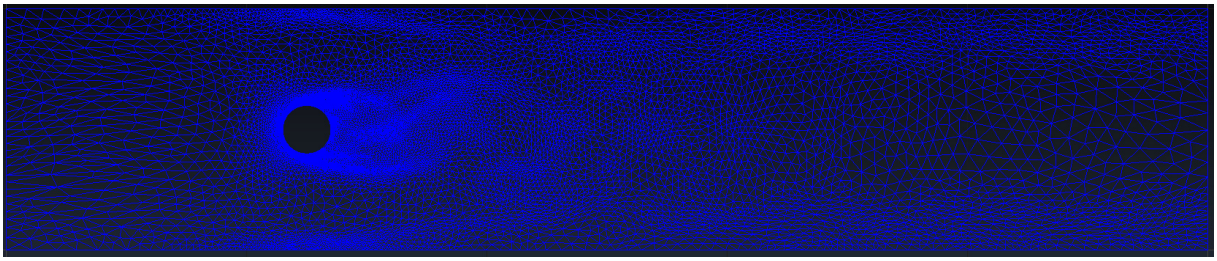
(b) Vortices Zoomed

Figure 5.2: Mesh 1

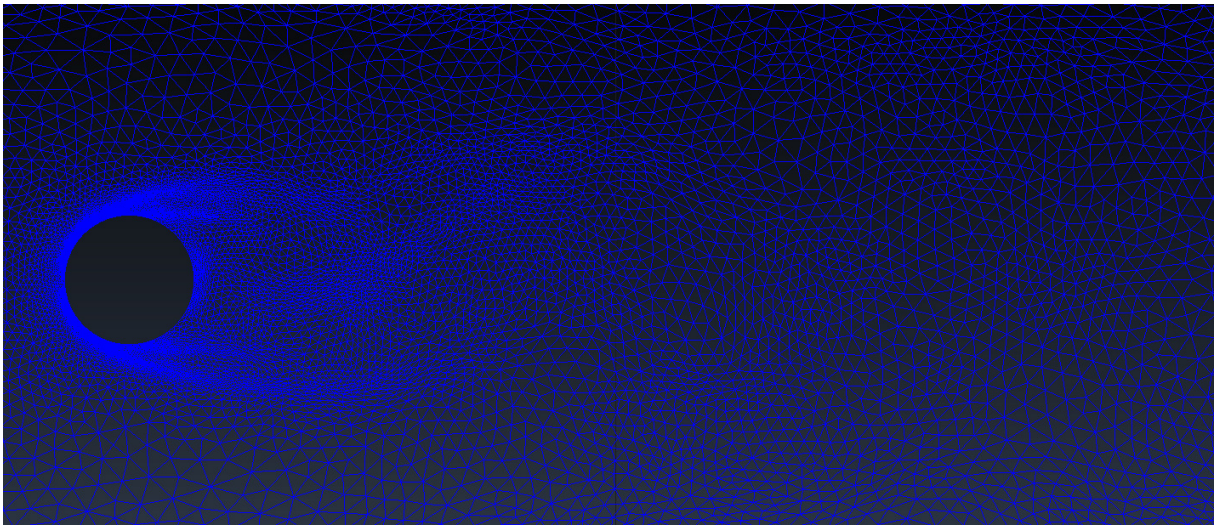
While increasing the interpolation error, in this case it is equal to

$$interp\_error = 0.01/2 = 0.005;$$

the script can better catch the parts of the system in which the vortices are taking place.



(a) Full System



(b) Vortices Zoomed

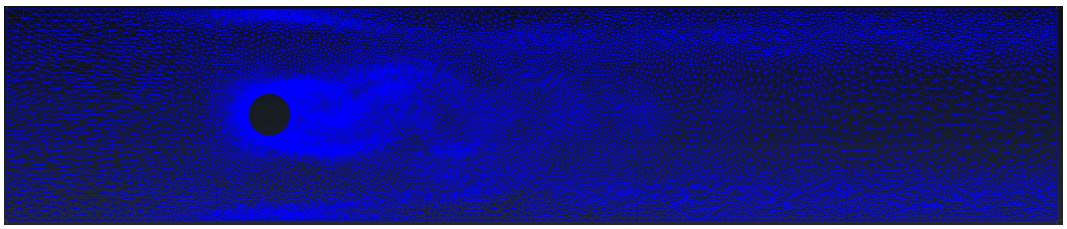
Figure 5.3: Mesh 2

From this mesh, the vortices are caught better and better. It is focusing on the vortices, as they are the parts of the system more difficult to catch. The stored mesh starts being bigger and bigger, as from this configuration, the number of nodes is over 10.000.

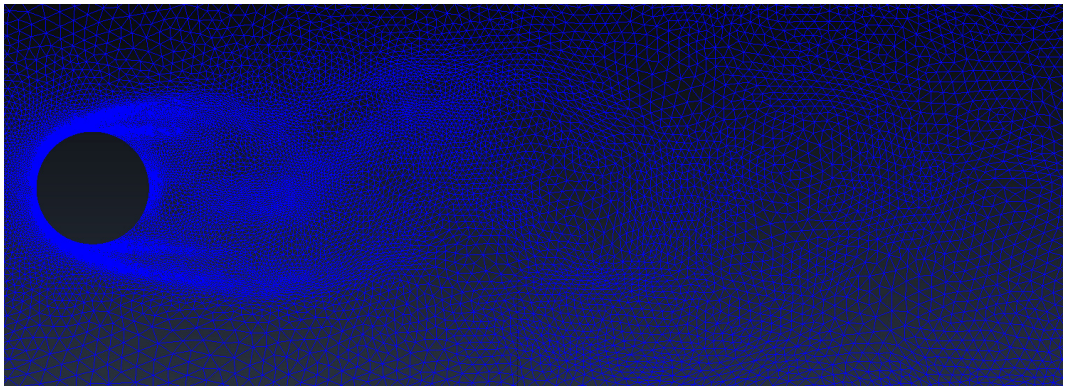
The interpolation error is set to

$$interp\_error = 0.01/2^2 = 0.0025;$$

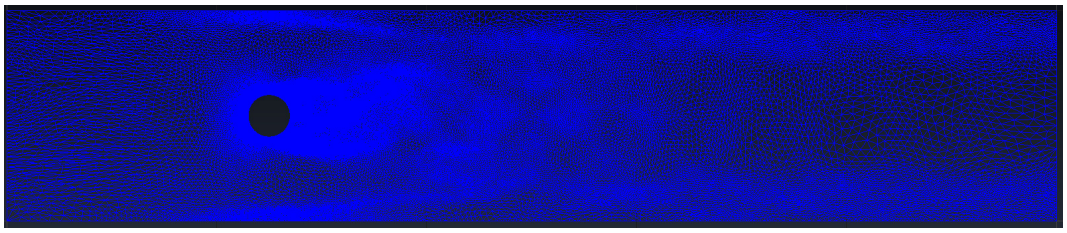
It means that the code is better catching the abrupts changings in the values of velocity and pressure, but it is slower to run.



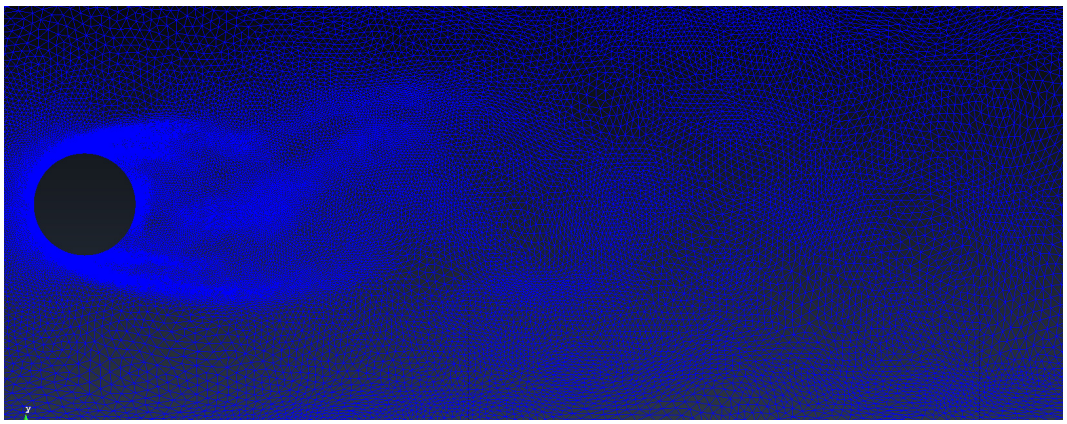
(a) Full System - Mesh 3



(b) Vortices Zoomed - Mesh 3



(c) Full System - Mesh 4



(d) Vortices Zoomed - Mesh 4

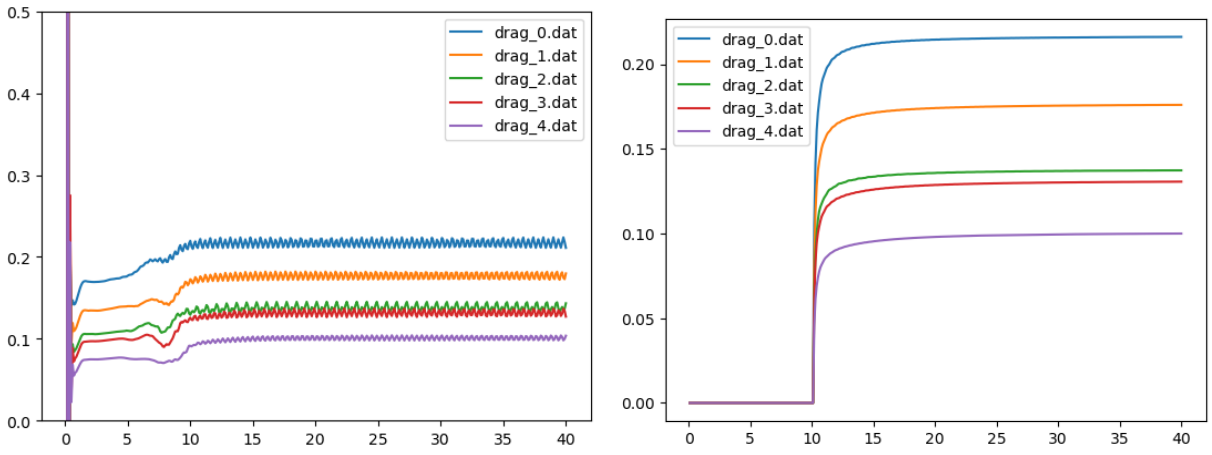
Figure 5.4: Mesh 3 and 4

In meshes 3 and 4 it can be appreciated the detailed caption of the vortices. These meshes have a lot of nodes, and they are very refined. The computation time is big, but the accuracy in the simulation is very detailed. The interpolation error is set respectively to:

$$interp\_error(3) = 0.01/2^3 = 0.00125;$$

$$interp\_error(4) = 0.01/2^4 = 0.000625;$$

As it can be seen from figures 5.5 a) and b), it is plotted the convergence of the drag forces in the different configurations.



(a) Raw Drag

(b) Time Averaged Drag

Figure 5.5: Drag Plots

It can be appreciated that in figure 5.5 a), the vortices are well caught. After the flow reaches the cylinder, in fact it starts having a periodic continuity in results, after almost ten seconds of simulation. This is why in the time averaged plot, it is not taken into account the first ten seconds of simulation.

The convergence is not perfectly reached, as the difference of subsequent levels is not going down. It can be seen from figure 5.5 that the lines don't get horizontal. This happens for high Reynolds number.

6. Conclusions

In conclusion, the enhancement has been implemented and tested. It has been approved from the community of GitHub and finally merged.

This enhancement is not a substitution of the previous one, but an added feature. This feature will let the user saving time in case of different simulations on the same system.

In order to have it completed, it will be needed to define a class in the mmg library, that will be the one related to the average quantity computation. This class will be added in the $ExecuteFinalize$ and will complete the mmg re-meshing process. In the tests reported here, the average has been computed in the main loop, outside the mmg library.

References

— GiD Reference Manual
— *An immersed boundary method using unstructured anisotropic mesh adaptation combined with level − sets and penalization techniques* - R.Abgrall, H.Beaugendre, C.Dobrzynski
— *Metric − Based Anisotropic Mesh Adaptation* - Frédéric Alauzet

7. Appendix

```python
def _ExecuteRefinement(self):
    """ This method is the one responsible to execute the remeshing

    Keyword arguments:
    self -- It signifies an instance of a class.
    """
    if self.strategy == "LevelSet":
        # Calculate the gradient
        self.local_gradient.Execute()

    # Recalculate NODAL_H
    self.find_nodal_h.Execute()

    # Initialize metric
    if self.strategy == "Hessian" or self.strategy == "LevelSet":
        self.initialize_metric.Execute()

    KratosMultiphysics.Logger.PrintInfo("MMG Remeshing Process", "Calculating the metrics")
    # Execute metric computation
    for metric_process in self.metric_processes:
        metric_process.Execute()

    # Debug before remesh
    if self.settings["debug_mode"].GetString() == "GiD": # GiD
        self._debug_output_gid(self.main_model_part.ProcessInfo[KratosMultiphysics.STEP], "", "BEFORE_")
    elif self.settings["debug_mode"].GetString() == "VTK": # VTK
        self._debug_output_vtk(self.main_model_part.ProcessInfo[KratosMultiphysics.STEP], "", "BEFORE_")

    # Execute before remesh
    self._AuxiliarCallsBeforeRemesh()

    # Actually remesh
    KratosMultiphysics.Logger.PrintInfo("MMG Remeshing Process", "Remeshing")
    self.mmg_process.Execute()

    # Execute after remesh
    self._AuxiliarCallsAfterRemesh()

    # Debug after remesh
    if self.settings["debug_mode"].GetString() == "GiD": # GiD
        self._debug_output_gid(self.main_model_part.ProcessInfo[KratosMultiphysics.STEP], "", "AFTER_")
    elif self.settings["debug_mode"].GetString() == "VTK": # VTK
        self._debug_output_vtk(self.main_model_part.ProcessInfo[KratosMultiphysics.STEP], "", "AFTER_")

    if self.strategy == "LevelSet":
        self.local_gradient.Execute() # Recalculate gradient after remeshing

```

```
605         # Recalculate NODAL_H
606         self.find_nodal_h.Execute()
607
608         # We need to set that the model part has been modified (later on we will act in consequence)
609         self.main_model_part.Set(KratosMultiphysics.MODIFIED, True)
610
611         # Deactivate to avoid remesh again
612         self.remesh_executed = True
613
614         KratosMultiphysics.Logger.PrintInfo("MMG Remeshing Process", "Remesh finished")
615
```