ExaQUte

Exascale Quantification of Uncertainties for Technology and Science Simulation

D2.4 First Release of the mesh generation/adaptation capabilities

Document information table

Contract number:	800898
Project acronym:	ExaQUte
Project Coordinator:	CIMNE
Document Responsible Partner:	INRIA
Deliverable Type:	Report
Dissemination Level:	Public
Related WP & Task:	WP 2 Tasks 2.1, 2.2, 2.4
Status:	Final Version

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

Authoring

Prepared by:				
Authors	Partner	Modified Page/Sections	Version	Comments
Luca Cirrottola	INRIA			
Algiane Froehly	INRIA			
Brendan Keith	TUM			
Suneth Warnakulasuriya	TUM			

Change Log

Versions	Modified Page/Sections	Comments
V1.0	Created document, filling basic information	

Approval

Aproved by:				
	Name	Partner	Date	OK
WP leader	Algiane Froehly	INRIA	30.11.19	OK
Coordinator	Riccardo Rossi	CIMNE	30.11.19	OK

Executive summary

This document presents a description of the parallel mesh adaptation library for the first actual software release. Regarding the octree mesh-generation capabilities, the reader can refers to Deliverable 2.2.
As it is discussed in Section 1.3.2 of part B of the project proposal there are two parallel research lines aimed at developing scalable adaptive mesh refinement (AMR) algorithms and implementations. The first one is based on using octree-based mesh generation and adaptation for the whole simulation in combination with unfitted finite element methods (FEMs) and the use of algebraic constraints to deal with non-conformity of spaces. On the other hand the second strategy is based on the use of an initial octree mesh that, after making it conformal through the addition of template-based tetrahedral refinements, is adapted anisotropically during the calculation.

The core capabilities and kernel algorithms for both strategies were described in Deliverable 2.2. This reports concerns:

- the improvements from month 12 to 18 and the actual software release (v1.2.0) of the parallel mesh adaptation library (available on GitHub: `https://github.com/MmgTools/ParMmg/releases/tag/v1.2.0`);
- the advancement of adjoint-based adaptive mesh refinement for fluids flows at high Reynolds numbers.

Regarding the parallel mesh adaptation library, the following items are included:

- An outline of the anisotropic mesh adaptation algorithm.
- A description of the newly implemented algorithms from month 12 to 18.
- A presentation of current performances.
- An outline of the interfacing with the partner multiphysics solver Kratos.

Then, this deliverable describes the interfacing of CFD solvers with the parallel mesh adaptation library.

The last section presents experiments demonstrating the limitations of the adjoint-based mesh refinement in Navier-Stokes simulations at high Reynolds number.

Table of contents

List of Figures

List of Tables

Nomenclature / Acronym list

Acronym	Meaning
AMR	Adaptive mesh refinement and coarsening
SFC	Space-Filling curve
API	Application Programming Interface
OOP	Object Oriented Programming
BC	Boundary Condition
MPI	Message Passing Interface

Table 1: Nomenclature / Acronym list

1 Introduction

This report describes the advancement at month 18 of the workpackage 2.

We recall that this WP aims to develop parallel mesh generation and mesh adaptation tools to be used by solvers in the context of exascale computations. For this, two complementary strategies are followed: an octree-based strategy and an anisotropic mesh adaptation strategy.

The octree-based strategy

An octree mesh (so a non-conforming grid) is generated during a preprocessing step. This octree mesh is also adapted to limit errors during the solution computation.

- Advantages of octree meshes are obvious, a given cell may be subdivided into 8 children following a unique pattern and without impacting its neighbours so the refinement/unrefinement stencil is minimal and the methods can be efficiently parallelized.
- The price to pay is that the complexity is deported onto the solver and the way it manages the hanging nodes and edges. A second drawback is that elements can't be streched (squared cells) and that the number of cells is increased by the need to control the octree balancing (one need a maximal gap between the level of subdivision of two adjacent cells which is equivalent to control the maximum number of hanging nodes per cell face).

The anisotropic mesh adaptation strategy

A tetrahedral conforming mesh (previously generated using an external mesh tool) is adapted based on an a posteriori error estimator during the solution computation.

- Advantages of tetrahedral meshes is that they are very flexible and impose a unique constraint, the mesh conformity. Thus, we can produce anisotropic meshes (meshes streched in a given direction) with very high ratio of anisotropy as well as "optimal" meshes in term of the number of elements (the size of a tetrahedron doesn't impact the size of its neighbour).
- The main drawback of this method is due to the preservation of the mesh conformity: the modification of a given edge of a tetrahedra may impact all the tetrahedron that contains one of the extremity of the edge. It introduces lot of dependencies between the remeshing operators and make it very hard to parallelize.

Note that one can refer to the **Deliverable 2.2** for a more detailed presentation of both strategies, but as conclusion of this recall, both approaches are worth to explore because they place the difficulties on different parts of the software stack and because solving efficiently this difficulties remains a challenge.

2 Parallel unstructured mesh adaptation

The aim of the `ParMmg` software package is to build parallel mesh adaptation capabilities on top of the sequential open-source remesher `Mmg` [8] [3], while preserving the open-source spirit and the support for general-purpose applications through the library application-programming interface (API). The core of the algorithm has been detailed in the report for **Deliverable 2.2**, and it will be only briefly summarized here before focusing on the main developments achieved between month 12 and month 18. This last developpements are available on the master branch of the `https://github.com/MmgTools/ParMmg` repository.

The core idea of parallel iterative mesh adaptation over constrained interfaces is to employ sequential remeshing techniques in the domains interior, while parallel interfaces are constrained (i.e. adaptation is forbidden on the interface), then the interface is moved by a repartitioning method, mesh migration is performed and the process is iterated until an overall good mesh quality is reached.

Following previous works on the parallel usage of `Mmg` [16] [6], we focus on a two-levels partitioning scheme where the mesh is first partitioned on the parallel computer. Then, local data are further partitioned into mesh groups, which can be remeshed sequentially by the local process one after the other. The size of the mesh group is targeted to an optimal size for the `Mmg` remeshing process. Both interfaces between partitions and mesh groups are kept unchanged during the remeshing step.

After the remeshing step, the mesh is repartitioned in order to change the parallel interface and to allow a new remeshing step. This can be accomplished through several strategies. In [16] [6], a load balancing step is applied to the partitioning of the parallel mesh group graph rather, than of the parallel mesh element graph, in order to ease the partitioning task for the `Metis` library. To do that, the size of the mesh groups is recalibrated for the repartitiong task. A high weight is placed on graph edges proportionally to old parallel interfaces, so to penalize the occurrence of these edges in new partition interiors. `ParMmg` supports the same algorithm.

A certain compromise between an optimal load balancing and the number or remeshing-repartitioning iterations needs to be sought. We have found that load balancing algorithms can be difficult to tune for an effective interface displacement that limits the number of total iterations performed, as their primary aim is that of minimizing a communication cost function, without directly targeting mesh displacement. For this reason, we have implemented the support of direct interface displacement methods for mesh repartitioning. This constitute the main new development for month 18, and it will be detailed in the following.

2.1 Interface displacement by an advancing-front method

We follow the same idea employed in [10] [7] to move the interface by an advancing-front algorithm. We adopt an element-based partitioning scheme, thus we aim at propagating the front of parallel nodes by walking on edge connectivity, in order to mark the visited tetrahedra with the color brought by the front.

To this purpose, a front direction needs to be chosen. This operation allows to give each front node a unique color, based on the colors of the tetrahedra in its ball, and a logical operator to decide whether this color should be passed to the tetrahedra visited by the front, or not. This operation also allows to handle the case of colliding fronts.

Once a direction is chosen for the propagation of the interface, the front nodes visit all the tetrahedra in their ball and mark them for repartitioning, if it is compatible with the front direction. Then, the color is passed to the outer points of the ball, and a new front is created. By performing this process N times, an $N-$layer mesh subgroup is built just beside the old parallel interfaces. This process is reminiscent of a parallel version of the greedy algorithm for sequential mesh partitioning [11], where the seeds are not chosen independently but only the current node interfaces are allowed to grow their ball by N new layers. At the end of each step, a parallel update is performed for the color of the nodes on current parallel interfaces.

This algorithm effectively produces a new mesh partitioning that can be directly used for any already available mesh migration routines. An example of front advancement on 4 partitions in a sphere is given in figure 1. From the practical point of view, this algorithm needs to be complemented with: 1) the choice of the front advancement direction, and 2) handling of disconnected partitions, i.e. partitions where all elements cannot be reached by face adjacency, which are very likely to be produced by the advancing front [11][17].

2.2 Groups sorting for interface advancement direction

For every interface, a propagation direction needs to be chosen. In [14] and [9], processors are paired in order to balance the computational load. Since optimal load balancing is not the main aim of the remeshing phase, we have preferred to sort the current groups by the number of elements, and move the interface in the direction of the bigger mesh group. This naïve global sorting allows to effectively handling the case of colliding fronts, as each of them continues to propagate into the bigger partition found, without the need to form new pairings.

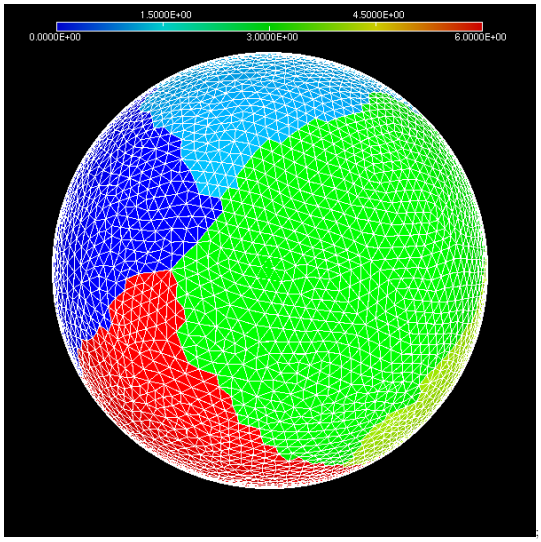
2.3 Correcting disconnected partitions

As already hinted in [11, 14], experience shows that any advancing-front mesh partitioning algorithm is likely to produce disconnected partitions in given conditions. Although `Mmg` is capable of handling disconnected partitions, and it could be argued that a solver should be capable of handling generic mesh partitions, this is often not an optimal situation to perform computations. For example, in mesh adaptation over constrained interfaces one should aim at keeping the ratio between surface and volume elements as high as possible, which is not the case when disconnected mesh parts appear in a partition. Disconnected partitions also require special treatments for the localization step for metrics interpolation from the old to the new mesh (like tree-searches), which are not needed for connected partitions.

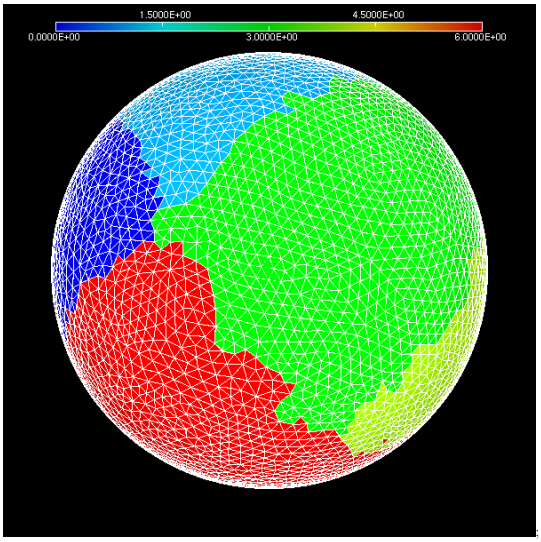
In the following, some examples of common problems are presented, together with the a posteriori correction implemented in `ParMmg`.

2.3.1 Examples of configurations leading to disconnected partitions

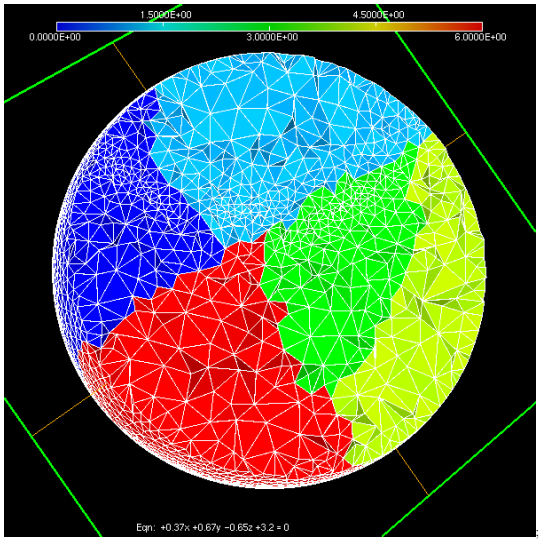
The next paragraph focuses on the presentation of the most common geometrical and partitioning configurations leading to disconnected meshes. For sake of simplicity, we use two-dimensional examples to illustrate this configurations.



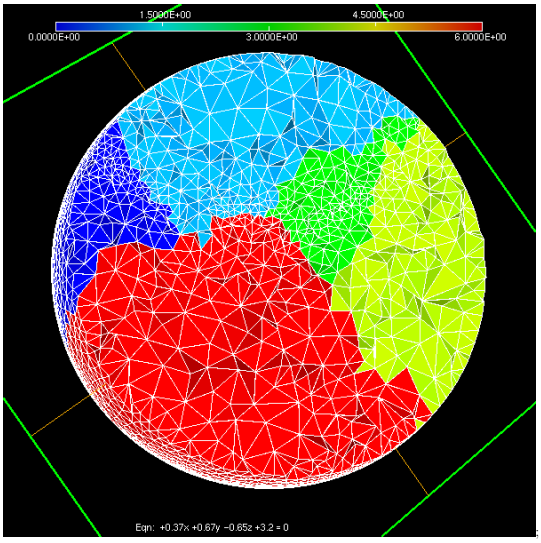
(a) Initial groups.



(b) Groups after front advancement.

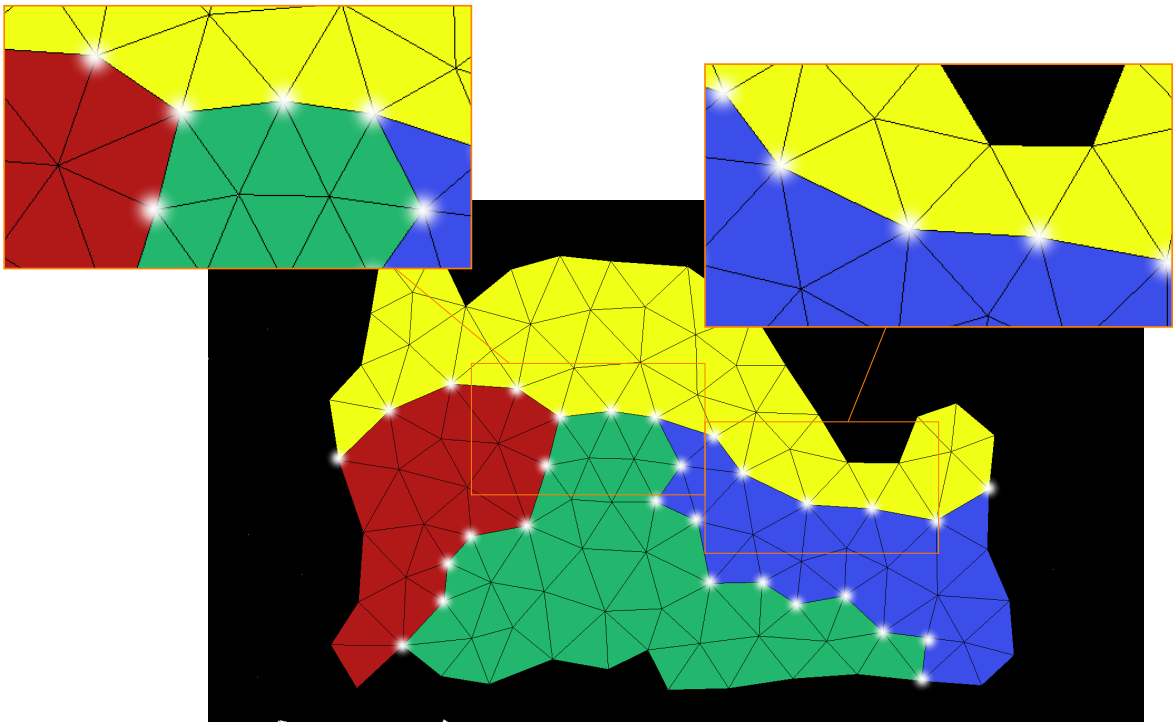


(c) Initial groups (volumic cut).

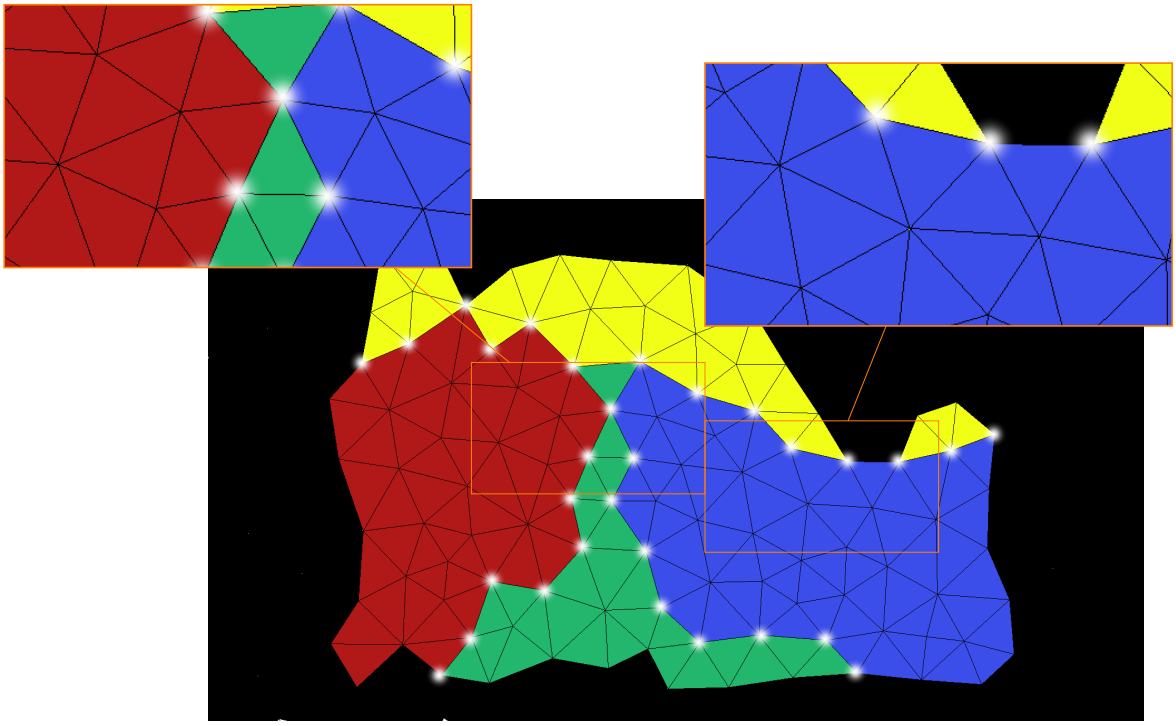


(d) Groups after front advancement (volumic cut).

Figure 1: Example of mesh repartitioning by interface displacement in a sphere. Left: Initial partitioning. Right: Final partitioning.



(a) Initial groups and interface front.



(b) Groups after one level of front advancement.

Figure 2: Two-dimensional illustration of some configurations leading to disconnected partitions. Right zoom: The extremity of a non-convex partition is cut away from its main domain by front advancement. Left zoom: unordered front advancement leading to an isolated bite (priority is red-blue-green-yellow).

Eaten non-convex partitions If the original partition is not convex, the interfaces could move inward so to cut away some partition extremities when the interface hits a boundary or when several interface fronts collide. This issue can be easily solved by checking and fixing the contiguity of the interior partition at the end of the interface displacement step. This configuration is illustrated in the right zoom in figure 2.

Isolated bites If care is not given to the order by which each node grows and marks its ball, it can happen that the color of a partition is propagated when it should not have been in the first place, and the following propagation of a higher-priority front takes some *isolated bites*[11] that disconnect the previously grown ball from its main partition. This configuration is illustrated in the left zoom in figure 2.

Parallel star configurations In the same way, if the front reaches an interface on a local partition and this information is not communicated to the remote processor, at the next propagation step the layer that propagated on the remote partition could not be reachable anymore by its original partition. This behavior is easily corrected by always performing a parallel update after one level of front advancement.

2.3.2 A posteriori corrections

From the practical point of view, it is difficult to predict all the situations that could lead to a disconnected partition in three dimensions. A more robust approach is that of a posteriori correcting partition connection after the front advancement step. This can be achieved by means of two kinds of checks, one on the contiguity of the local partition, one on the reachability of all layers that have propagated in the local partition by means of their remote counterpart. An a posteriori correction also allows for a generic choice of the interface propagation algorithm.

Contiguity fix for the local partition

1. Similarly to greedy algorithms, we choose a color and start from an element of this color: we attempt at touching all elements of this color by adjacency (through the element faces). Elements that we are able to reach by adjacency are stored in a list (*main_list*).

2. While we are able to find an element of the same color outside of *main_list*:

   - we build by adjacency a second list of elements of this color (*next_list*);
   - If *next_list* is not empty, we compare the lengths of *next_list* and *main_list* and we merge the smaller list of elements into an adjacent color. The biggest list is stored as *main_list* and we go back to step 2.

3. Go back to step 1 (choice of a color and construction of *main_list*) until all the colors have been treated.

Reachability fix for the inward-propagating remote partitions As hinted in [17], the front advancement can create subgroups cut away from any interface, or subgroups connected only to an interface of different color than their remote counterpart. These subgroups are not reachable by their remote partition anymore. To fix this problem, after a parallel exchange of remote colors on the current interface, for each interface face we attempt at building a list of elements of the same remote color which can be touched by adjacency, and we mark them as reachable. At the end of this process, we iteratively look for a list of adjacent elements which have not been reached by remote colors, and merge them into an adjacent color (reachable or not).

The current implementation is conceived in order to leave the interface displacement and the contiguity/reachability correction independent. This allows for a certain freedom in the experimentation of different advancing-front methods (for example, advancing on element adjacency instead of node adjacency) and interface sorting methods. Although a partition could be a priori split into an arbitrary number of disconnected parts, the contiguity correction algorithm only uses two lists to immediately compare two parts and merge the smallest into another color, thus limiting the memory usage. Similarly, the reachability correction algorithm only uses one list to store the current subgroup of unreachable elements. At the end of the two correction steps, all mesh elements are guaranteed to having been touched by a list only once, if the counter is decremented every time an element is merged in a different color.

2.4 Summary of the algorithm

The key features of the parallel remeshing algorithm can be summarized as follows:

- Two-level parallelization scheme (*distributed* mesh partitions and *local* mesh groups);
- Sequential remeshing of mesh groups over constrained parallel interfaces.;
- Mesh repartitioning by interface displacement.

A pseudocode for the `ParMmg` workflow is presented in algorithm 1. Mesh migration directly exploits the two-level parallelization schemes, as mesh parts to be sent to other processors are assembled into mesh groups to be used for parallel communication, while mesh repartitioning acts as an independent step which can be further tailored to ensure a more robust load balancing. A pseudocode for mesh repartitioning is presented in algorithm 2

2.5 Open-source implementation

The presented algorithm is implemented into the `ParMmg` software package for parallel unstructured mesh adaptation, released under the GNU Lesser General Public License (LGPL). The implementation is targeted to provide:

- Reusage of existing sequential remeshing libraries;
- Non-intrusive linkage with third-party solvers;
- Improvable parallel performances by means of dynamic load balancing.

Algorithm 1 `ParMmg` algorithm pseudocode.

```
1:  Input(mesh,metrics);                                  /* Initialization (sequential or parallel) */
2:  Group split;                                          /* Split partitions into groups */
3:  for i = 0, ..., i_max − 1 do                          /* Iterative remeshing-repartitioning */
4:      Update old groups;          /* Set background mesh for metrics interpolation */
5:      for igrp = 0, ..., ngrp − 1 do                    /* Loop on mesh groups */
6:          Mmg call;                                     /* Remeshing */
7:      end for
8:      Interpolate metrics;                              /* Recompute metrics */
9:      Mesh repartitioning;        /* Interface displacement and mesh migration */
10: end for
11: Output(mesh,metrics);                                 /* Return the adapted mesh */
```

Algorithm 2 Mesh repartitioning pseudocode.

```
1:  Input(mesh groups);
2:  Sort interfaces;                                      /* Choose interface propagation direction */
3:  Parallel update of interface colors;
4:  for i_layer = 0, ..., n_layer − 1 do                  /* Loop on mesh groups */
5:      Propagate node front;
6:  end for
7:  Correct contiguity;
8:  Correct reachability;
9:  Mesh migration;
10: Output(mesh groups);                                  /* Return repartitioned groups */
```

Open-source software packages are used for every step in the computing chain, from mesh partitioning, remeshing, node renumbering, mesh visualization. The remeshing kernel is the sequential `Mmg` library [8] [3]. Parallelization is performed through Message Passing Interface (MPI) libraries. Partitioning of a centralized input mesh is performed by means of the `Metis` library [15] [2], and the `Scotch` library [5] is employed for nodes renumbering to reduce cache misses. Finally, mesh files can be saved for visualization in the Medit format (readable by `Medit` [12] [1] and `Gmsh` [13]) and in VTK format [18][4]. Finally, version control is performed with `Git` and continuous integration testing with `Jenkins`.

All the needed modules have been implemented in C99 and all the parallel mesh adaptation kernel functions benefits from the updates available in the last release of the `Mmg` remesher.

The implemented API functions aim to provide the project partner with the tools to couple their computational mechanics solvers with the parallel mesh adaptation library. The requirements for these API functions can be found in the Deliverable 2.1 As such, the API functions fall into the three main categories described in the following paragraphs. Basic tutorials are available with the source code, and a complete documentation is underway on the project webpage.

Functions to initialize and recover a sequential or distributed mesh

These functions closely match the analogous API functions available in the Mmg remeshing library to initialize pointers to Mmg data structures and to set/get mesh entities (nodes, elements, boundary triangles) one-by-one or by arrays. The user is required to set mesh elements (tetrahedra) and nodes, together with boundary triangles on each local mesh.

Functions to set and get the interface entities of a distributed mesh

In order to be able to work both with node-centered and element-centered computational mechanics solvers, two separate sets of API functions are available to initialize either interface faces or nodes (if both are provided, nodes information are discarded). The user is asked to provide an array with the indices of interface faces/nodes in the local input mesh, together with an array with the global indices of the same entities (provided in the same order). This information is internally used to reconstruct the communication graph among processes.

Functions to check the correctness of the set interface entities against input data are also provided.

Main parallel mesh adaptation function

Two library functions are provided in order to run the parallel mesh algorithm starting from a sequential or a distributed mesh.

2.6 Parallel weak-scaling of a uniform refinement test case

For a preliminary evaluation of the parallel performances of the algorithm, we test its weak scalability for an uniform refinement case. The aim is that of distributing the work load on each mesh group as uniformly as possible.

The geometry is a sphere of radius 10. Input meshes have been generated with `Gmsh`, and their size grows with the number of processors (the input edge size is shown in table 2).

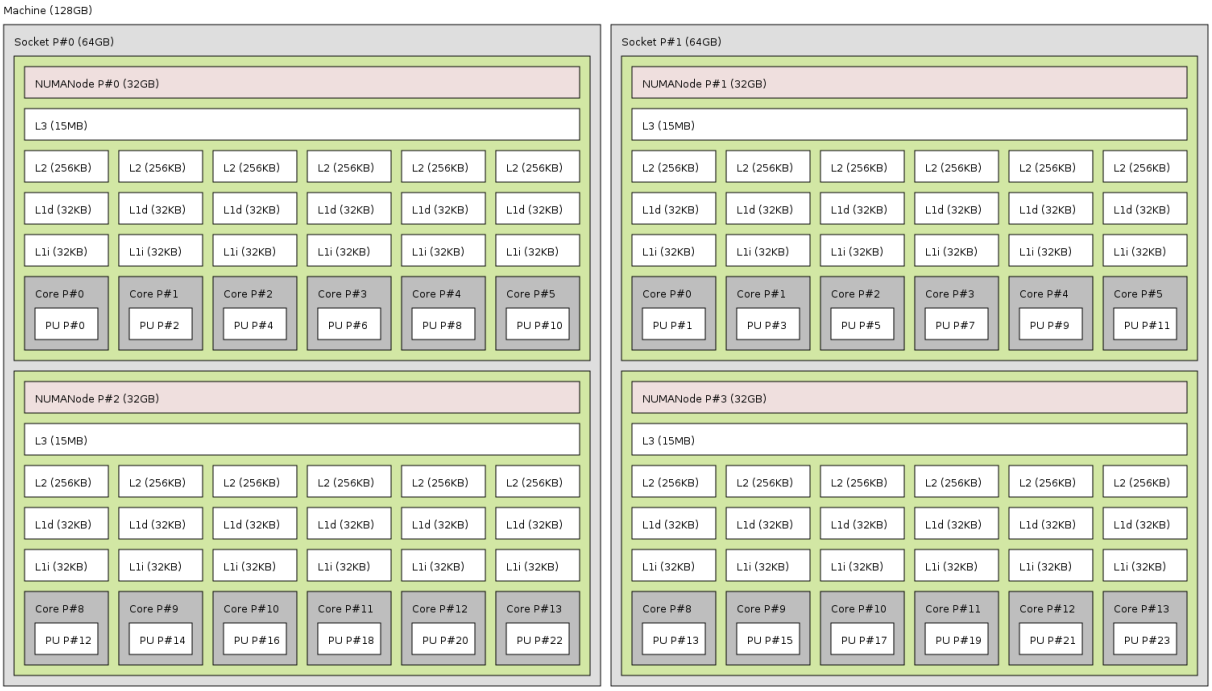


Figure 3: lstopo view of a miriel node

The assigned target mesh size is about 1/6 the original edge size. The size of the output adapted meshes are shown in table , compared with the sizes of the corresponding input meshes. The same results are visualized in figure 4, in order to check that the load is maintained approximately constant in the weak scaling test.

Tests have been performed on the Miriel nodes of PlaFRIM cluster[1], equipped with 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 (2.5 GHz) and 128 GB of RAM (see figure 3 for the lstopo view), connected through Infiniband QDR TrueScale (40 Gb/s) and Omnipath (100 Gb/s).

The computational time in the weak scaling test is shown in figure 5. The total time is split into the time spent in the remesher and the time spent in mesh repartitioning. In this phase, the time spent in migrating the mesh (MPI communication) is explicitly highlighted. It can be seen that the remeshing time is kept approximately constant with the number of cores, while the time for mesh migration constitute a bottleneck on more than 64 cores. The reasons for this behavior are currently under investigation. Even if performances can clearly still be optimized, parallel remeshing is able to generate billion-element meshes over 128 cores (table 3).

[1]Supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr/)

Cores	$h^{(\text{in})}$
1	0.16667
2	0.13228
4	0.10499
8	0.08333
16	0.066142
32	0.052497
64	0.041667
128	0.033071
256	0.026248

Table 2: Input mesh edge size with different number of cores.

Cores	Nodes (in)	Nodes (out)	Elements (in)	Elements (out)
1	3842	1591657	18990	9418375
2	7251	2645804	37752	15893278
4	13871	5439170	75195	32688539
8	26772	11147795	149329	67019257
16	52235	22703159	297586	136515675
32	102101	46023487	593845	276849115
64	205737	77677606	1192034	552962469
128	411553	164118099	2416482	1115114432
256	856547	328668480	5044573	2260289909

Table 3: Number of nodes and elements in the input and output meshes, with different number of cores.

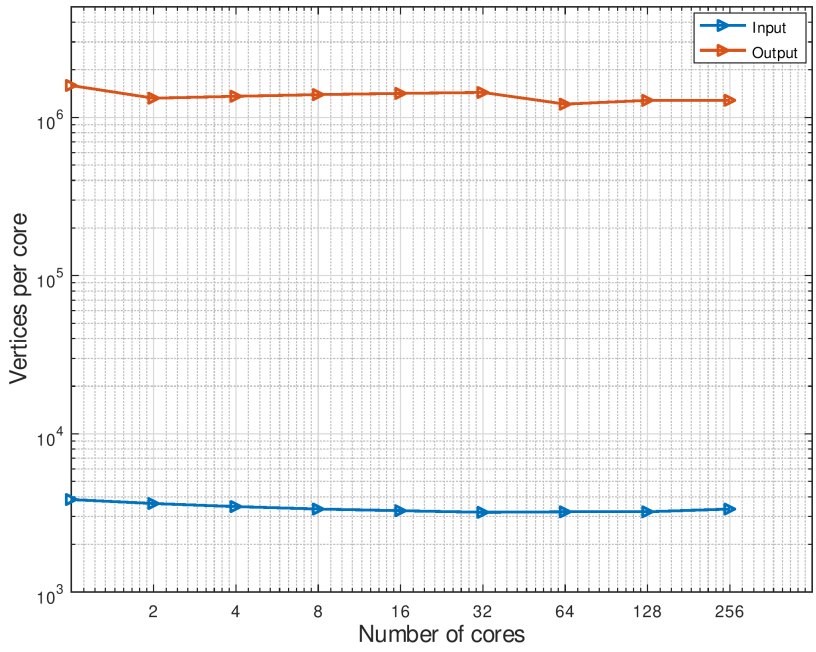


Figure 4: Number of nodes in the input and output meshes for the weak scaling test.

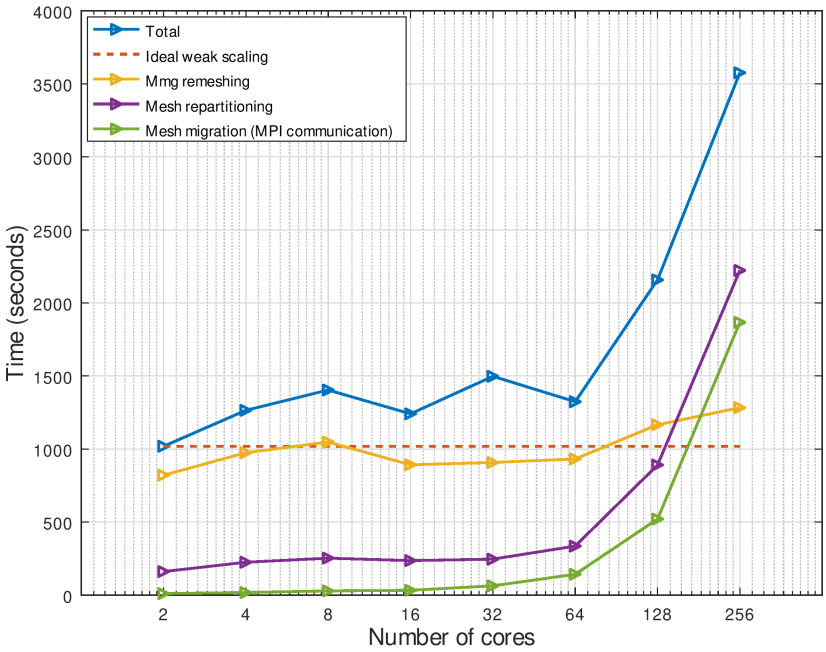


Figure 5: Computational time in the weak scaling test.

3 Interfacing of CFD solvers to newly developed capabilities

This release of `ParMmg` has been interfaced with the open-source `Kratos` multiphysics solver in collaboration with the Kratos team at CIMNE. `Kratos` was already interfacing the sequential `Mmg` library. Thus, the new software coupling only needs minor modifications in order to handle the parallel version.

From the `ParMmg` side, the main difference with the `Mmg` API functions is the presence of additional functions to *set* and *get* the nodes or faces on the parallel interfaces. The following code snippet illustrates an example interface to `ParMmg` in C. Parameters and functions related to parallel interfaces are highlighted in blue (in the example, we are passing parallel nodes).

```
/* Initialize ParMmg */
PMMG_Init_parMesh ( PMMG_ARG_start, /* ... */, PMMG_ARG_end );

PMMG_Set_meshSize ( parmesh,nVerts,nTets,nPrisms,nTris,nQuads,nEdges );
PMMG_Set_vertices ( parmesh,verts_tab,verts_refs );

/* Set mesh entities: */
/* Nodes...           */
/* Tetrahedra...      */

/* Communicators : construction through interface nodes or faces */
PMMG_Set_iparameter( parmesh, PMMG_IPARAM_APImode,PMMG_APIDISTRIB_nodes);
PMMG_Set_numberOfNodeCommunicators(parmesh, nNodeComm);
for( icomm = 0; icomm < nNodeComm; icomm++ ) {
   /* Set nb. of entities on interface and rank of the outward proc */
   PMMG_Set_ithNodeCommunicatorSize(parmesh,icomm,color_node[icomm],
                                    nitem_node_comm[icomm]);
   /* Set local and global index for each entity on the interface */
   PMMG_Set_ithNodeCommunicator_nodes(parmesh,icomm,
                                      idx_loc[icomm],idx_glo[icomm],
                                      nitems);
}

/* Main function : Parallel remeshing (distributed memory) */
PMMG_parmmglib_distributed ( parmesh );

/* Get mesh entities: */
/* Nodes...           */
/* Tetrahedra...      */

/* Free structures */
PMMG_Free_all(PMMG_ARG_start,/*...*/,PMMG_ARG_end);
```

From the `Kratos` side, the interfacing with `Mmg` is handled by a class `MmgProcess`

which had to be generalized to a new `ParMmgProcess` class, capable of setting and getting parallel entities in the `Kratos` parallel data structures.

The following code snippets illustrates an example of Python script for a simulation setup. The same parameters controlling the `Mmg` library can be used. `ParMmg` specific parameters and `Kratos` functions for parallel communicators and the `ParMmgProcess` object are highlighted in blue.

```
# Call the parallel fill communicator to assign the nodes to the right
# mesh containers
pfc=KratosMultiphysics.mpi.ParallelFillCommunicator(mainModelPart.GetRootModelPart())
pfc.Execute()

# ... Compute adaptation metrics ...

# We create the remeshing process
remesh_param = KratosMultiphysics.Parameters("""
                  {{
                       "filename"                          : "{filename}",
                       "save_external_files"               : true
                       "advanced_parameters"                  :
                       {{
                           "force_hausdorff_value"              : false,
                           "hausdorff_value"                    : 0.0001,
                           "no_move_mesh"                       : true,
                           "no_surf_mesh"                       : true,
                           "no_insert_mesh"                     : true,
                           "no_swap_mesh"                       : true,
                           "deactivate_detect_angle"            : false,
                           "force_gradation_value"              : false,
                           "gradation_value"                    : 1.3
                           "niter"                              : 4
                           "meshSize"                           : 30000
                           "metisRatio"                         : 82
                           "hgradreq"                           : 5.0
                           "APImode"                            : 0
                       }}
                   }}
                   """.format(filename="parts_"+ str(mpi.size)+"/in_"+str(mpi.rank)))


# Call a ParMmg process: ParMmg Set_* functions + parallel communicators
ParMmgProcess = MeshingApplication.ParMmgProcess3D(mainModelPart, remesh_param)
ParMmgProcess.Execute()
```

4 Adjoint-based adaptive mesh refinement

In this section, we report on the advancement of adjoint-based adaptive mesh refinement for fluids flows at high Reynolds numbers. As remarked previously, in Deliverable 2.3, adjoint-based (i.e., goal-oriented) adaptive mesh refinement in Navier–Stokes simulations has only been successfully demonstrated in a very limited number of studies with low Reynolds numbers. The main purpose of this section is to present a number of experiments performed by Suneth Warnakulasuriya at the chair of Structural Analysis at TUM which demonstrate some of the practical limitations of such applying such capabilities at high Reynolds numbers.

The upshot is that accurate computation of time-averaged sensitives becomes extremely difficult and expensive as the Reynolds number grows, even with a finite difference strategy. Compounded with the established fact that the adjoint problem is ill-posed over long time intervals, for high Reynolds numbers [**?**], this has greatly inhibited the incorporation of adjoint-based adaptive mesh refinement in the ExaQUte project.

4.1 Set-up

A comparison of drag sensitivities on mesh nodes for a flow over a cylinder is studied under finite difference method and adjoint method in this section. Figure 6 illustrates simulation setup. Figure 7 illustrates positions of the nodes which is considered in this comparison process. The properties of the comparison case is given in the Table 4.

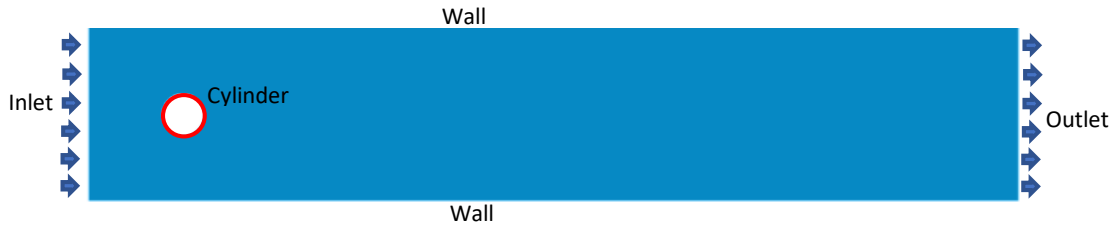


Figure 6: Comparison case setup with boundary conditions

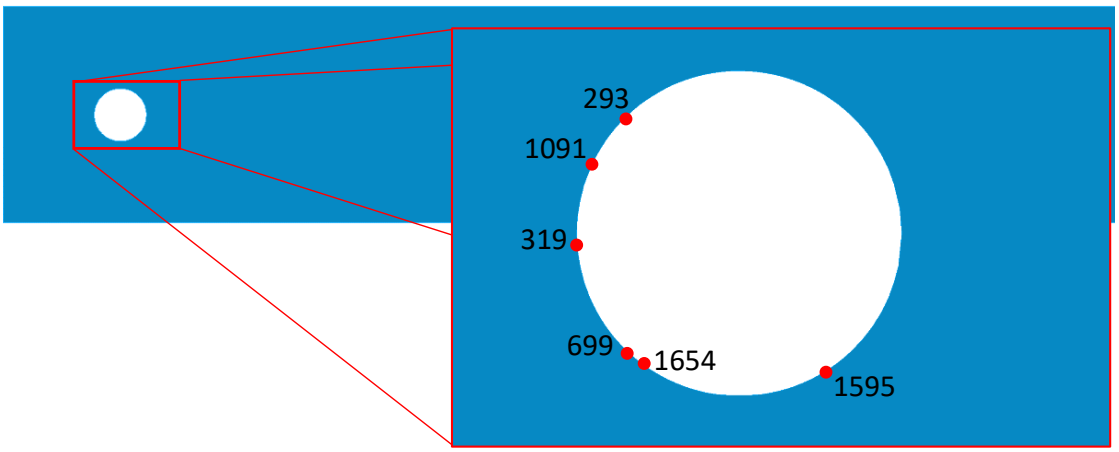


Figure 7: Node positions in the comparison case

Table 4: Properties of the validation case

Property	Symbol	Value
Cylinder diameter		$0.1\,m$
Density	ρ	$1.0\,kgm^{-3}$
Dynamic viscosity	μ	$1 \times 10^{-5}\,Pas$
Time step	Δt	$0.01\,s$

Different Reynold's numbers are simulated using this 2D comparison case by varying the dynamic viscosity. Inlet is assigned with a parabolic constant inlet velocity.

4.2 Finite Difference Sensitivities

Finite difference sensitivities are calculated by perturbing nodal positions of the nodes shown in Figure 7. Table 5 depicts value ranges used in this section for comparison. Filter radius (i.e. r_{filter}) with $1 \times 10^{-4}\,m$ is used to illustrate the finite difference sensitivities without vertex morphing, since there is only one node within that radius for all chosen nodes in the mesh used for simulations.

Figure 10 depicts the effect on neighboring nodes of node 293 when filter radius is varying. Vertex morphing is applied on the cylinder surface for control point (i.e. design space) perturbations, ensuring maximum perturbation in the particular mesh node (i.e. geometry space) is the perturbation which is expected. Therefore, the maximum perturbation illustrated in Figure 10 has maximum perturbation of $1 \times 10^{-8}\,m$. Therefore, perturbations in the design space may differ.

This finite difference sensitivity study is carried out for different Reynolds numbers, the Reynolds numbers are varied by varying the dynamic viscosity of the fluid while keeping a constant parabolic inlet velocity field which is having maximum velocity of $1.0\,ms^{-1}$.

Table 5: Perturbation values used in finite difference validation

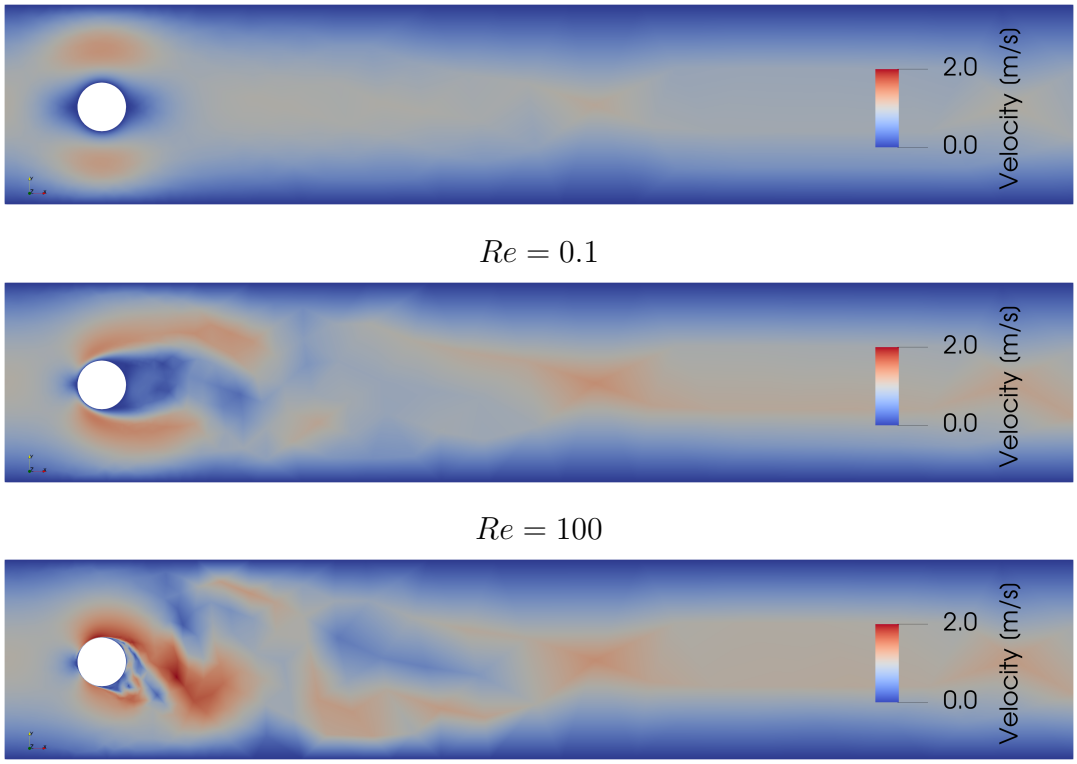
Property	Units	Values
Reynolds number		$0.1, 10^2, 10^4$
Perturbations	m	$1 \times 10^{-4}, 1 \times 10^{-5}, 3 \times 10^{-6}, 2 \times 10^{-6},$ $1 \times 10^{-6}, 3 \times 10^{-7}, 1 \times 10^{-7}, 1 \times 10^{-8}$
Filter radius	m	$8 \times 10^{-2}, 4 \times 10^{-2}, 2 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-4}$

4.2.1 Overall results

Figure 8 and Figure 9 depicts the velocity and pressure variations of the primal solutions for different Reynold's numbers. The Reynold's numbers are varied by varying the dynamic viscosity, while keeping the velocity inlet profile a constant.

4.2.2 Results of $Re = 0.1$

The raw drag forces for different perturbation step sizes with $r_{filter} = 1 \times 10^{-4}\,m$ is illustrated in Figure 11. The drag force illustrates a deterministic behaviour.



$Re = 0.1$

$Re = 100$

$Re = 10000$

Figure 8: Velocity contours of primal solutions for different Reynold's numbers at $t = 100.0\,s$

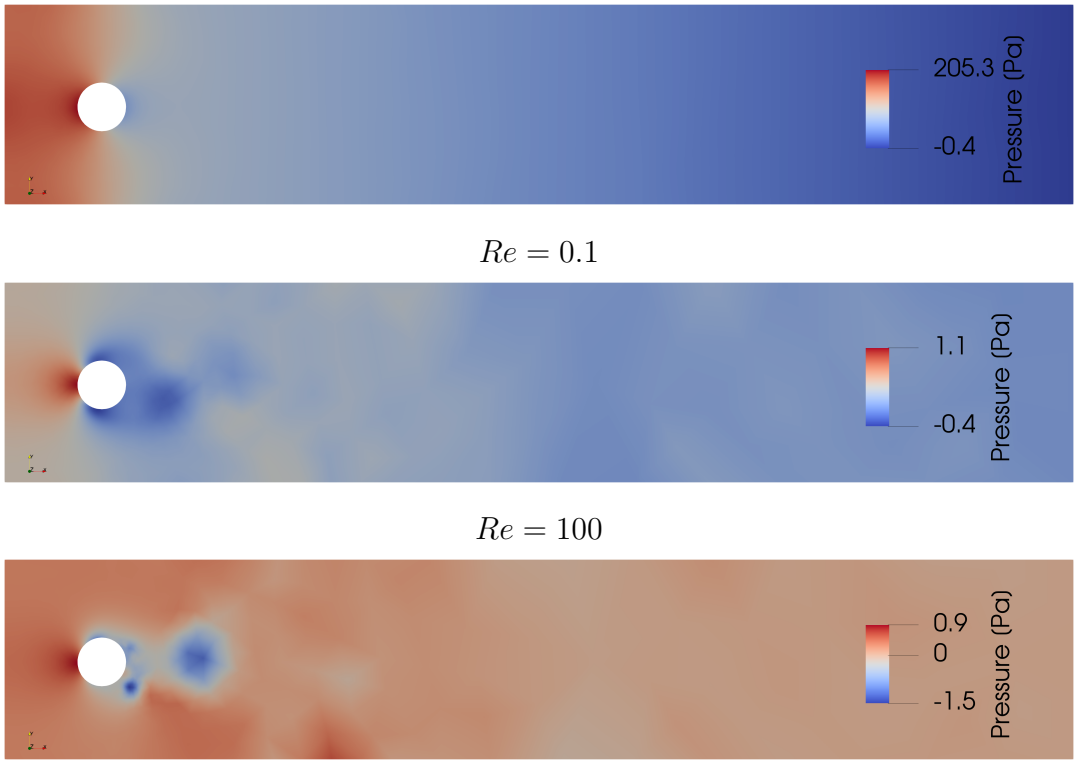
Then, finite difference sensitivities with $r_{filter} = 1 \times 10^{-4}\,m$ is calculated for different perturbations since this only involves nodal perturbations (no vertex morphing since filter radius is too smaller than the minimum distance between adjacent nodes in the mesh). Figure 12 depicts variations in the drag force with respect to variations in the perturbation. It is evident that, perturbations in the range of $[1 \times 10^{-8}\,m, 3 \times 10^{-7}\,m]$ doesn't change the calculated time averaged drag sensitivity, hence the convergence is achieved.

Then simulations with perturbations of $\Delta x = 1 \times 10^{-8}\,m$ is chosen for the filter radius study as depicted in Figure 13 for node 293 in "x" direction. It is evident that, for $Re = 0.1$, finite difference sensitivity for different filter radius agrees with their corresponding adjoint sensitivity calculations.

4.2.3 Results of $Re = 100$

The raw drag forces for different perturbation step sizes with $r_{filter} = 1 \times 10^{-4}\,m$ is illustrated in Figure 14. The drag force illustrates a deterministic behaviour.

Then, finite difference sensitivities with $r_{filter} = 1 \times 10^{-4}\,m$ is calculated for different perturbations since this only involves nodal perturbations (no vertex morphing since filter radius is too smaller than the minimum distance between adjacent nodes in the mesh). Figure 15 and Figure 16 depict variations in the drag force with respect to variations in the perturbation. It is evident that, perturbations in the range of $[1 \times 10^{-8}\,m, 3 \times 10^{-7}\,m]$ doesn't change the calculated time averaged drag sensitivity, hence the convergence is achieved.



$Re = 0.1$

$Re = 100$

$Re = 10000$

Figure 9: Pressure contours of primal solutions for different Reynold's numbers at $t = 100.0\,s$

Then simulations with perturbations of $\Delta x = 1 \times 10^{-8}\,m$ and $\Delta r = 1 \times 10^{-8}\,m$ are chosen for the filter radii study as depicted in Figure 17 for node 293 in "x" direction and Figure 18 for the same node in radial direction. It is evident that, for $Re = 100$, finite difference sensitivity for different filter radius agrees with their corresponding adjoint sensitivity calculations.

4.2.4 Results of $Re = 10000$

The raw drag forces for different perturbation step sizes with $r_{filter} = 1 \times 10^{-4}\,m$ is illustrated in Figure 19. The drag force illustrates a chaotic behaviour.

Due to the chaotic behaviour at $Re = 10000$, filter radius study was carried out for the chosen perturbation of $1 \times 10^{-8}\,m$ in both x, and y directions. The Figure 20 depicts the results of this study. It is evident from this, the chaotic behaviour at $Re = 10000$ makes finite difference sensitivity calculation to have an exponential growth due to "butter fly" effect. But, it shows less exponential growth with the increase of the r_{filter} radius. The node 293 is located in the attached region of the cylinder for this case. Further, a node at detached region of the cylinder (i.e. node 1595) is also analysed for time averaged drag force sensitivity as depicted in Figure 21. This node also describes less exponential growth when the r_{filter} radius is increased.

Results of the perturbation step size convergence study for $r_{filter} = 8 \times 10^{-2}\,m$ is illustrated in Figure 22. This

Figure 22 depicts that, if the perturbation step size is too small, then the noise

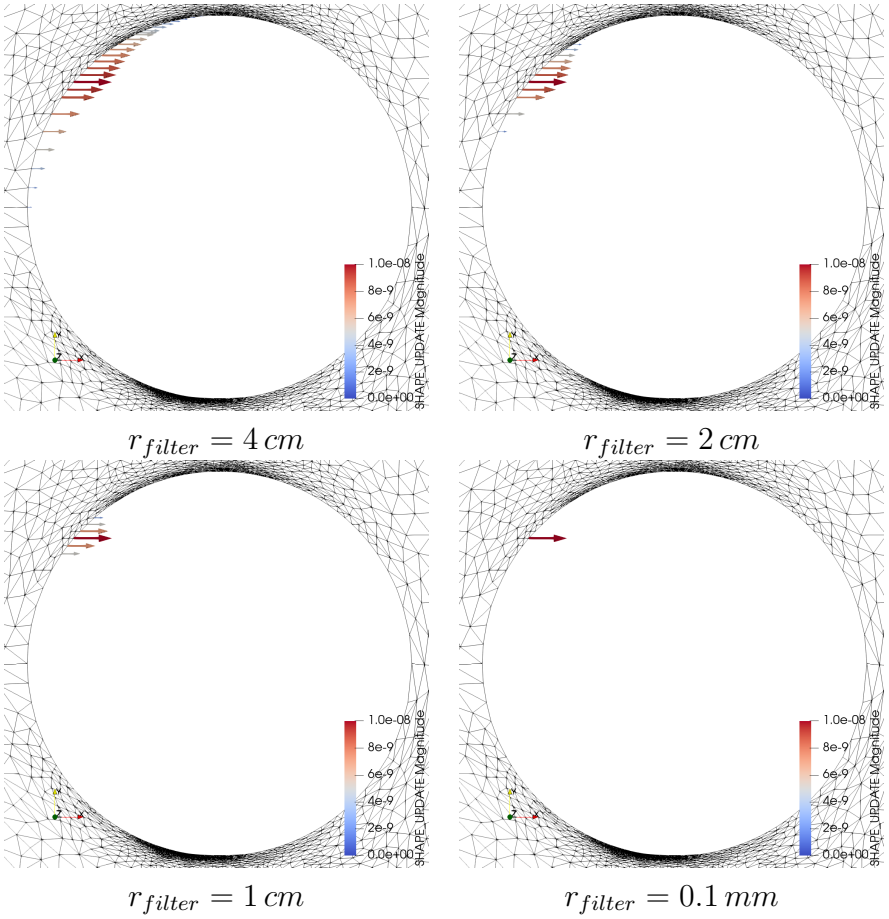


Figure 10: Effect on nodal perturbations with varying filter radius

dominates the finite difference solution for the case of the highest $r_{filter} = 8\,cm$ done in this numerical experiment. This is further elaborated in Figure 23. The range from $1\,mm$ to $5\,mm$ illustrate a constant gradient in the Figure 23, indicating a constant time averaged drag force sensitivity. These are large perturbations (i.e. in the range of $[1\%, 5\%]$ of the cylinder diameter), therefore mesh deformations may make some elements to have low quality. Therefore, in order to minimize the low quality elements in the mesh, radial perturbations are studied for the case of $Re = 10000$ and depicted in Figure 24 (Nodal perturbations are done in the inverse radial direction).

Linear approximation shown in Figure 24 illustrates a good fit for the data set in the range where max perturbation lies in between $[3\,mm, 6.5\,mm]$ for time averaged drag sensitivity in the radial direction at node 293. The negative value of the gradient of the linear approximation is taken as the reference time averaged drag sensitivity since perturbations are done in the inverse radial direction.

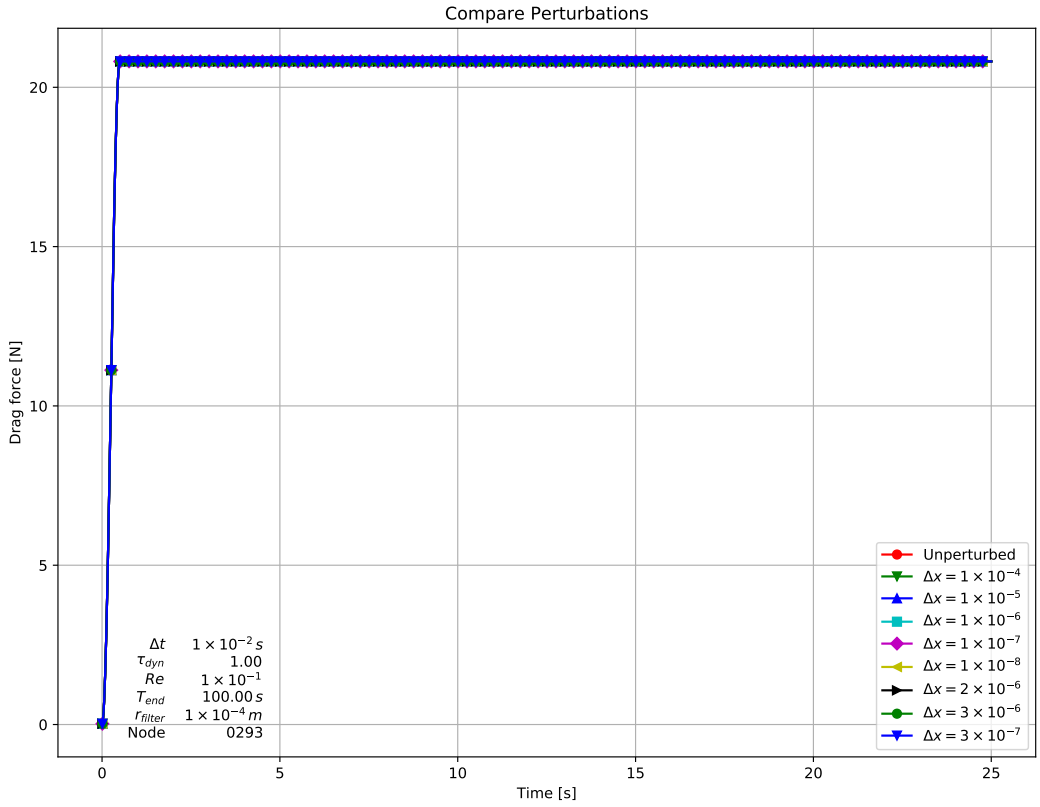


Figure 11: Drag force variation w.r.t. perturbation step sizes in "x" direction for node 293 with $Re = 0.1$

Table 6: Nodal time averaged drag sensitivity reference values

Node id	Direction	Time averaged drag sensitivity $[Nm^{-1}]$	R^2 error	N
293	Radial	-5.279×10^{-2}	0.9800	8
1595	Radial	$+1.963 \times 10^{-1}$	0.9796	12
1091	Radial	-1.629×10^{-1}	0.9953	5
319	Radial	-6.719×10^{-2}	0.9314	6
699	Radial	-6.444×10^{-2}	0.9304	6
1654	Radial	-7.698×10^{-2}	0.9578	5
855	Radial	$+1.283 \times 10^{-1}$	0.9601	15
2029	Radial	$+1.677 \times 10^{-1}$	0.9576	15

References

[1] `Medit` software package for scientific visualization on unstructured meshes. URL `https://github.com/ISCDtoolbox/Medit`.

[2] `Metis` software package for graph partitioning. URL `https://glaros.dtc.umn.edu/gkhome/metis/metis/overview`.

[3] `Mmg` software package for simplicial remeshing. URL `https://www.mmgtools.org/`.

[4] `VTK` software package for scientific visualization. URL `https://vtk.org/`.

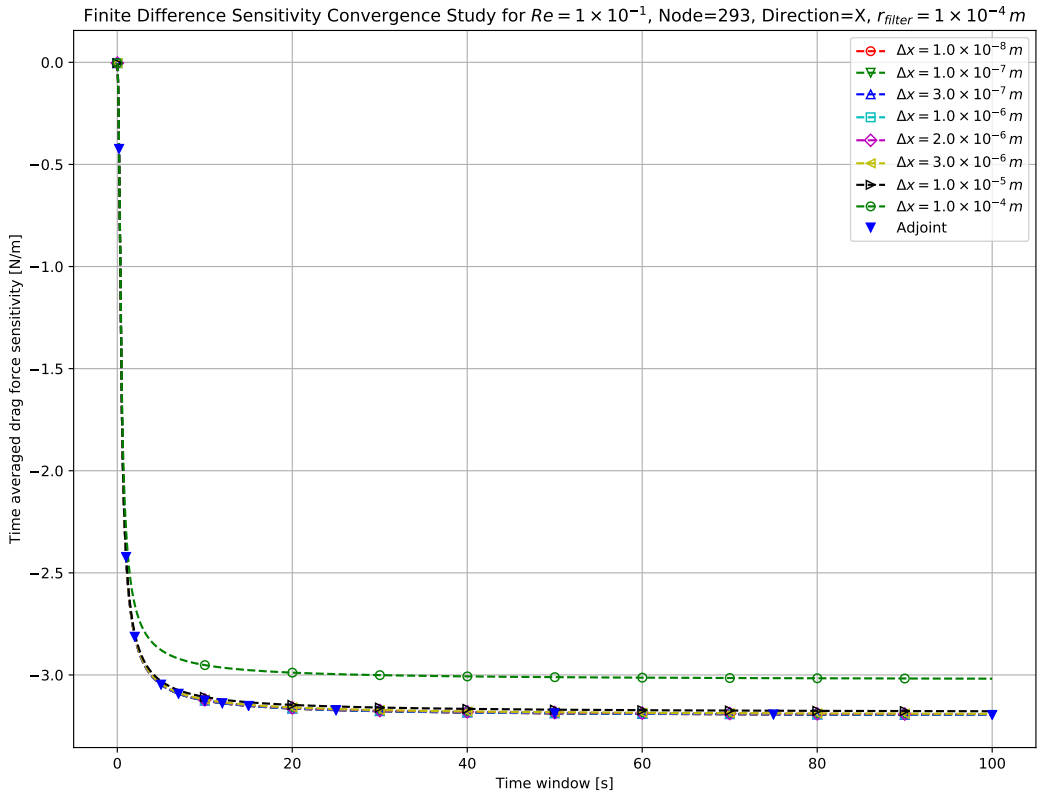


Figure 12: Time averaged drag sensitivity convergence for $Re = 0.1$ at Node=293 with $r_{filter} = 0.1\,mm$

[5] `Scotch` software package for graph partitioning. URL `https://gitlab.inria.fr/scotch/scotch`.

[6] P. Benard, G. Balarac, V. Moureau, C. Dobrzynski, G. Lartigue, and Y. D'Angelo. Mesh adaptation for large-eddy simulations in complex geometries. *International Journal for Numerical Methods in Fluids*, 81(12):719–740, 2016. doi:10.1002/fld.4204. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.4204`.

[7] P. A. Cavallo, N. Sinha, and G. M. Feldman. Parallel unstructured mesh adaptation method for moving body applications. *AIAA Journal*, 43(9):1937–1945, 2005. doi:10.2514/1.7818. URL `https://doi.org/10.2514/1.7818`.

[8] C. Dapogny, C. Dobrzynski, and P. Frey. Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. *Journal of Computational Physics*, 262:358 – 378, 2014. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2014.01.005. URL `http://www.sciencedirect.com/science/article/pii/S0021999114000266`.

[9] H. Digonnet, T. Coupez, P. Laure, and L. Silva. Massively parallel anisotropic mesh adaptation. *International Journal of High Performance Computing Applications*, Mar. 2017. doi:10.1177/1094342017693906. URL `https://hal.archives-ouvertes.fr/hal-01487424`.

[10] C. Dobrzynski and J.-F. Remacle. Parallel mesh adaptation. Poster,

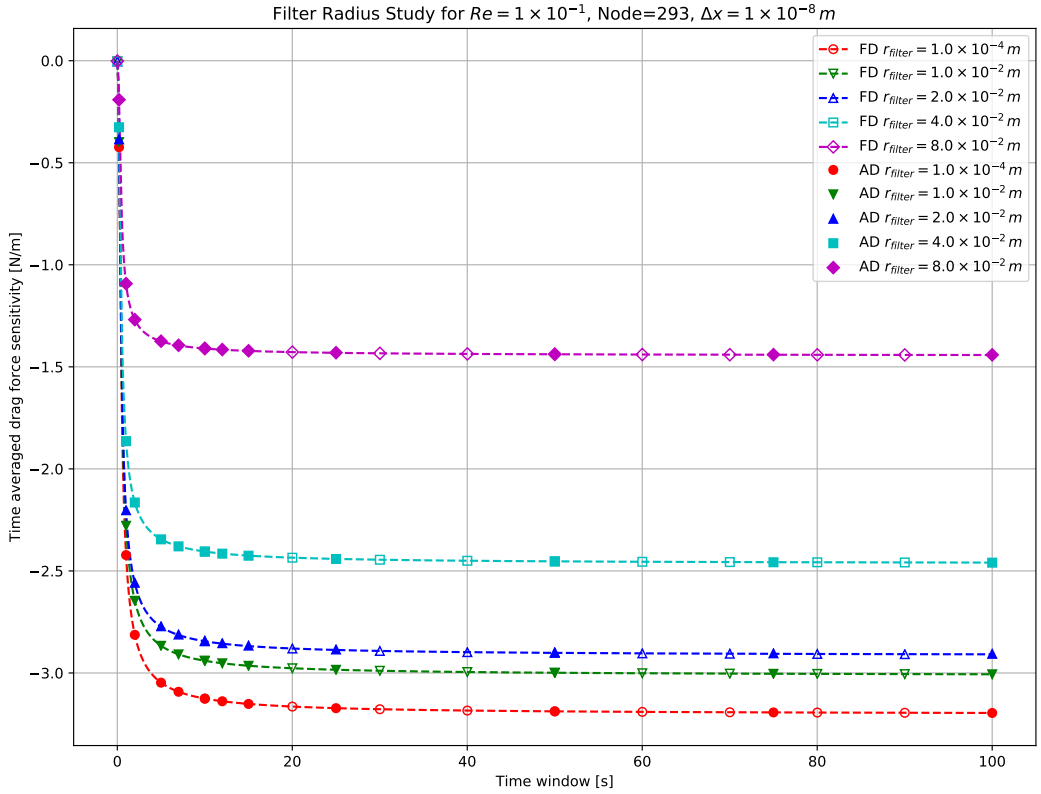


Figure 13: Time averaged drag sensitivity filter radius varitaions for $Re = 0.1$ at Node=293 with $\Delta x = 1 \times 10^{-8}\,m$

2007. URL https://www.math.u-bordeaux.fr/~dobrzyns/telechargement/parallmeshposter.pdf.

[11] C. Farhat. A simple and efficient automatic fem domain decomposer. *Computers & Structures*, 28(5):579 – 602, 1988. ISSN 0045-7949. doi:https://doi.org/10.1016/0045-7949(88)90004-1. URL http://www.sciencedirect.com/science/article/pii/0045794988900041.

[12] P. Frey. MEDIT : An interactive Mesh visualization Software. Technical Report RT-0253, INRIA, Dec. 2001. URL https://hal.inria.fr/inria-00069921.

[13] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. doi:10.1002/nme.2579. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2579.

[14] S. W. Hammond. *Mapping Unstructured Grid Computations to Massively Parallel Computers*. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, USA, 1992. UMI Order No. GAX93-02685.

[15] G. Karypis. *METIS 5.1.x Manual*. URL https://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf.

[16] V. Moureau, P. Domingo, and L. Vervisch. Design of a massively parallel cfd code for complex geometries. *Comptes Rendus Mécanique*, 339(2):141 – 148, 2011.

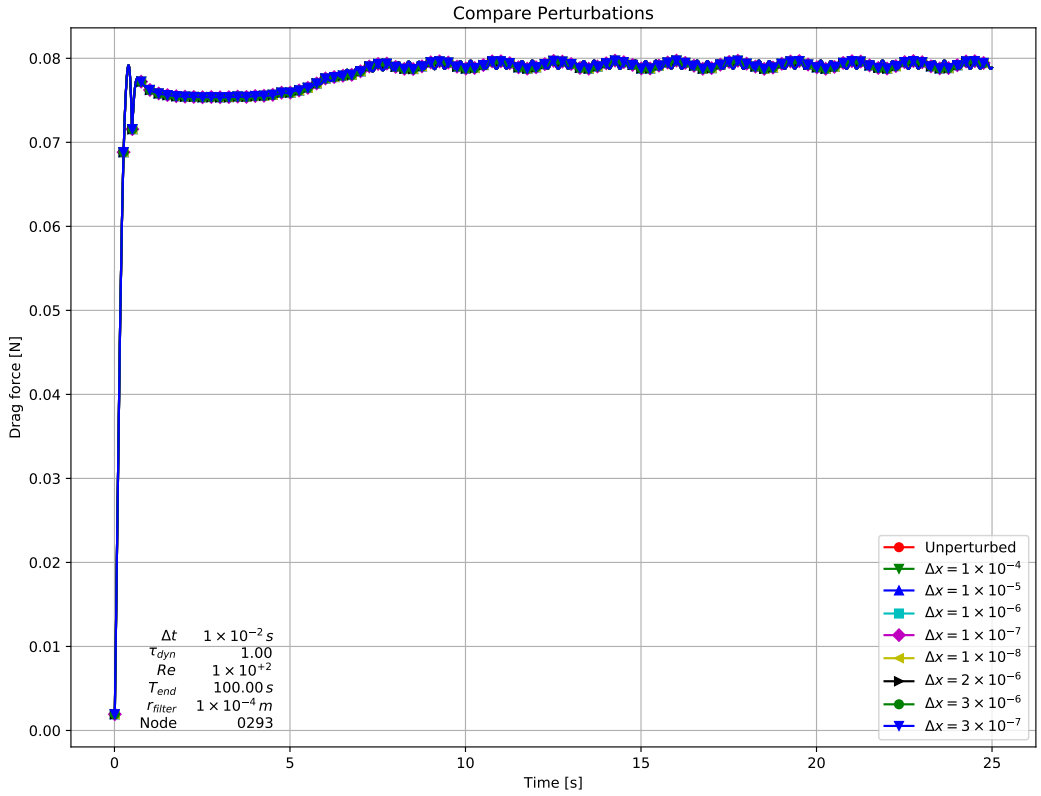


Figure 14: Drag force variation w.r.t. perturbation step sizes in "x" direction for node 293 with $Re = 100$

ISSN 1631-0721. doi:https://doi.org/10.1016/j.crme.2010.12.001. URL `http://www.sciencedirect.com/science/article/pii/S1631072110002111`. High Performance Computing.

[17] C. Ozturan. *Distributed Environment and Load Balancing for Adaptive Unstructured Meshes*. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, USA, 1995. UMI Order No. GAX96-22925.

[18] W. Schroeder, K. Martin, B. Lorensen, and I. Kitware. *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. Kitware, 2006. ISBN 9781930934191.

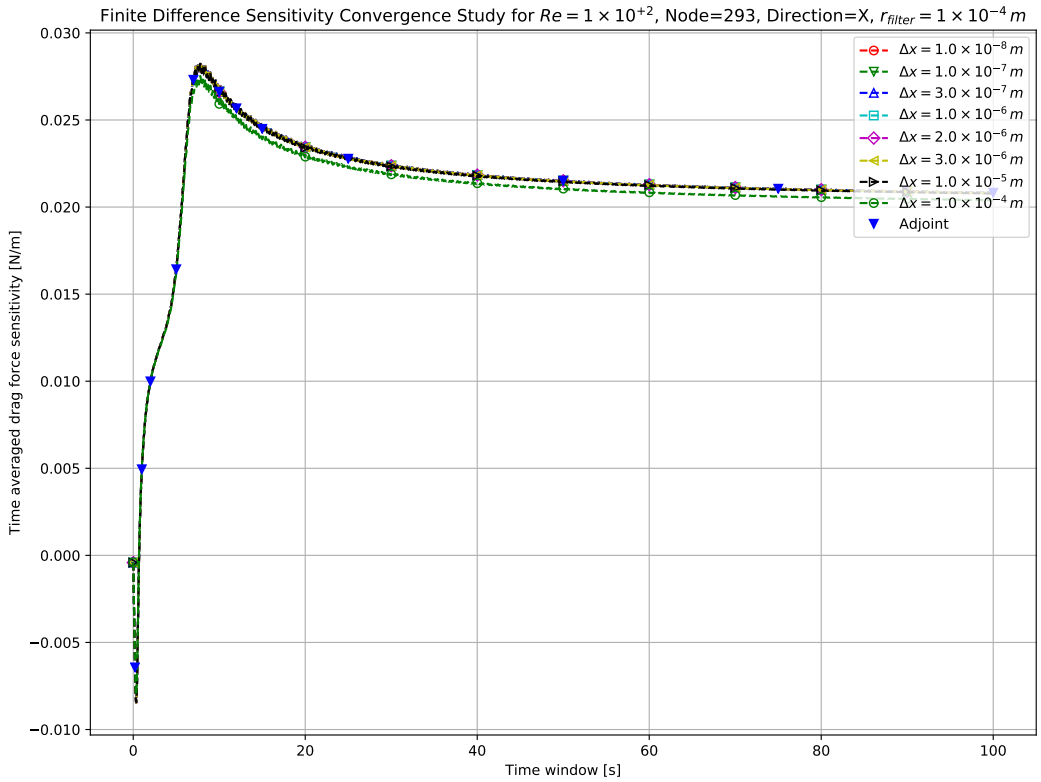


Figure 15: Time averaged drag sensitivity convergence for $Re = 100$ at Node=293 with $r_{filter} = 0.1\,mm$

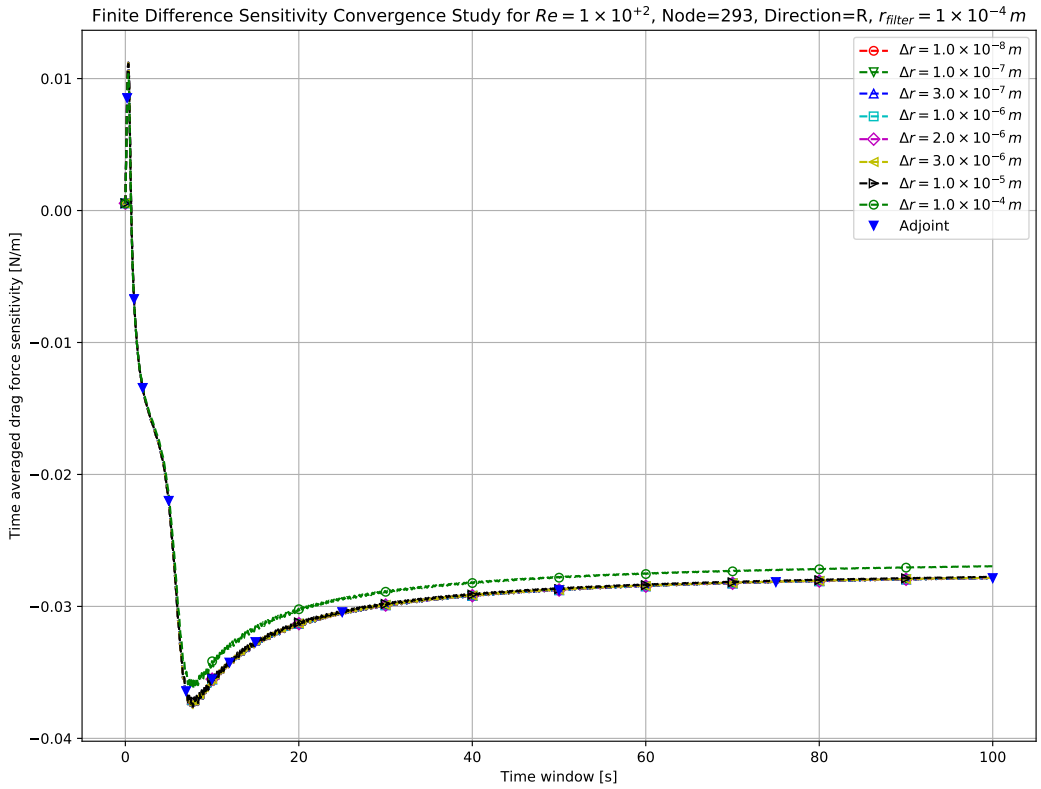


Figure 16: Drag force variation w.r.t. perturbation step sizes in radial direction for node 293 with $Re = 100$

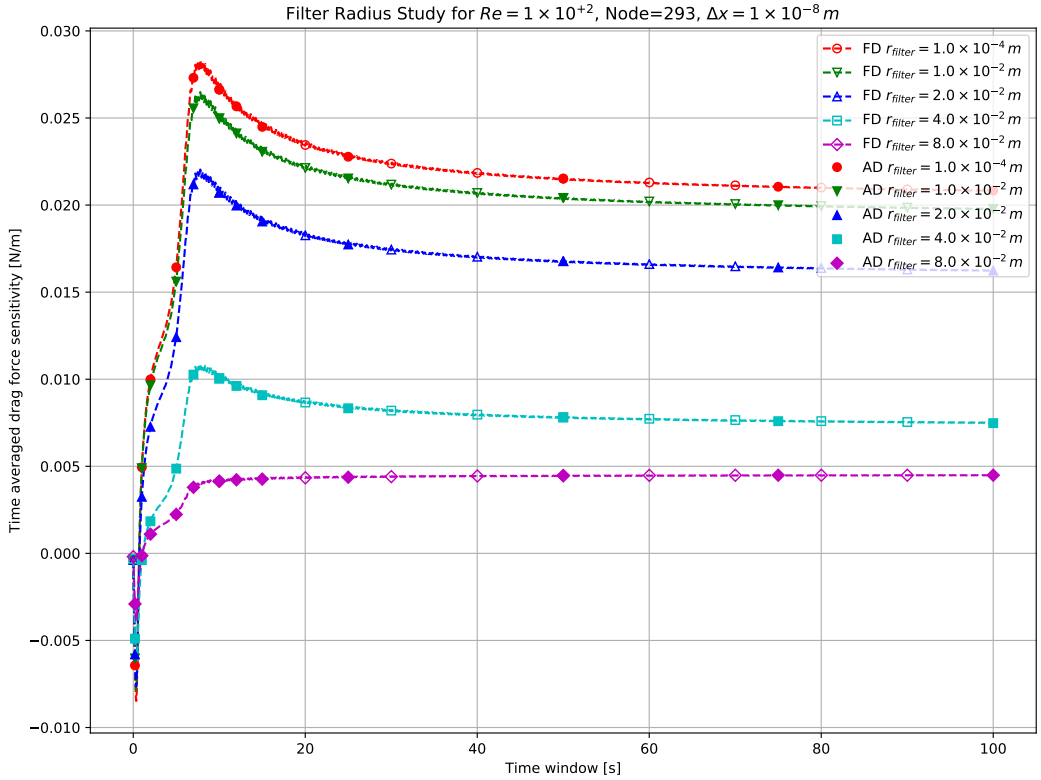


Figure 17: Time averaged drag sensitivity filter radius varitaions for $Re = 100$ at Node=293 with $\Delta x = 1 \times 10^{-8}\, m$

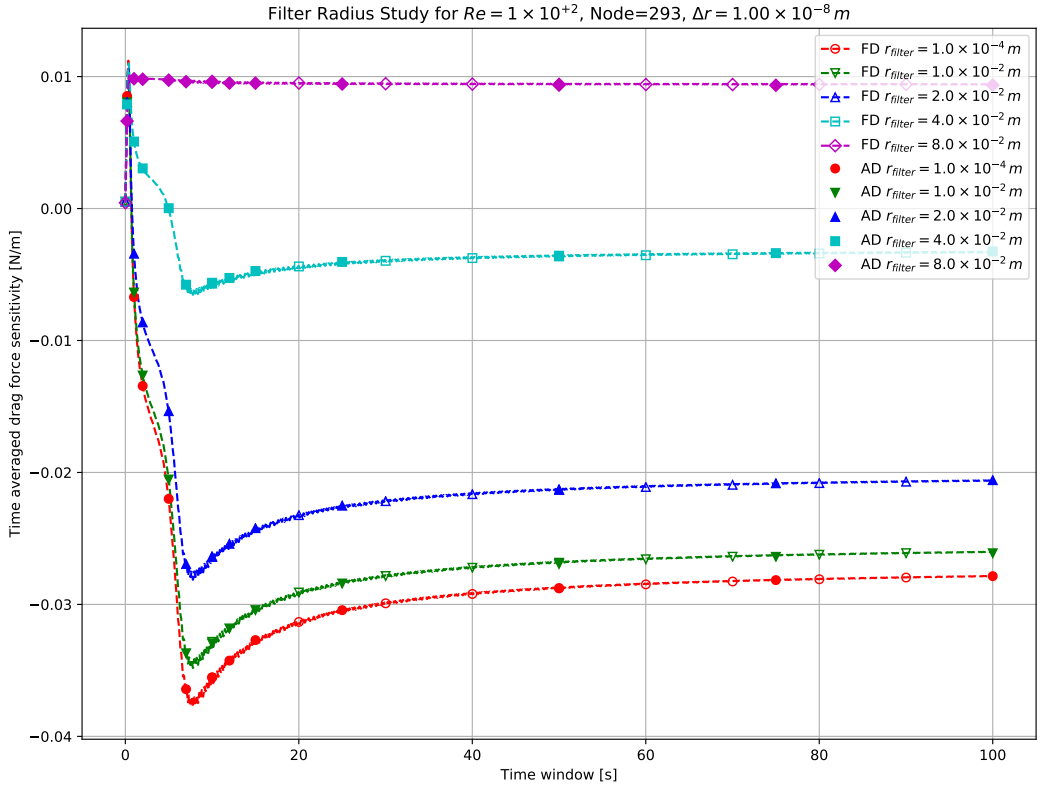


Figure 18: Time averaged drag sensitivity filter radius varitaions for $Re = 100$ at Node=293 with $\Delta r = 1 \times 10^{-8}\, m$

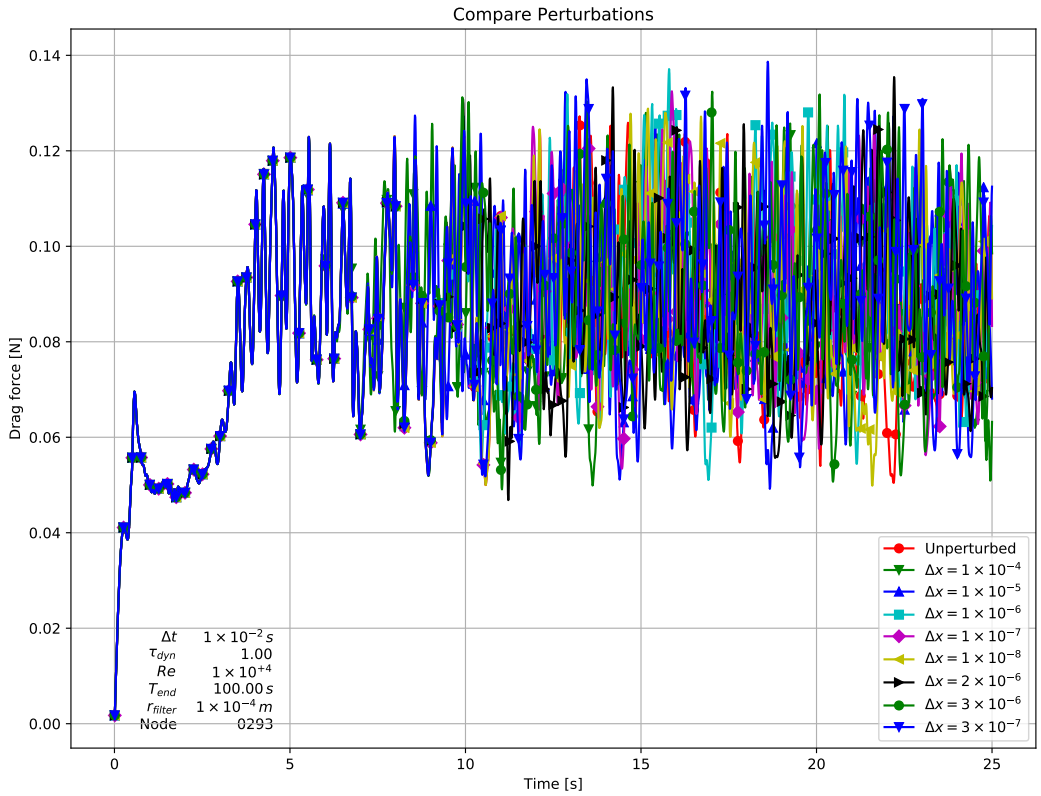


Figure 19: Drag force variation w.r.t. perturbation step sizes in "x" direction for node 293 with $Re = 10000$

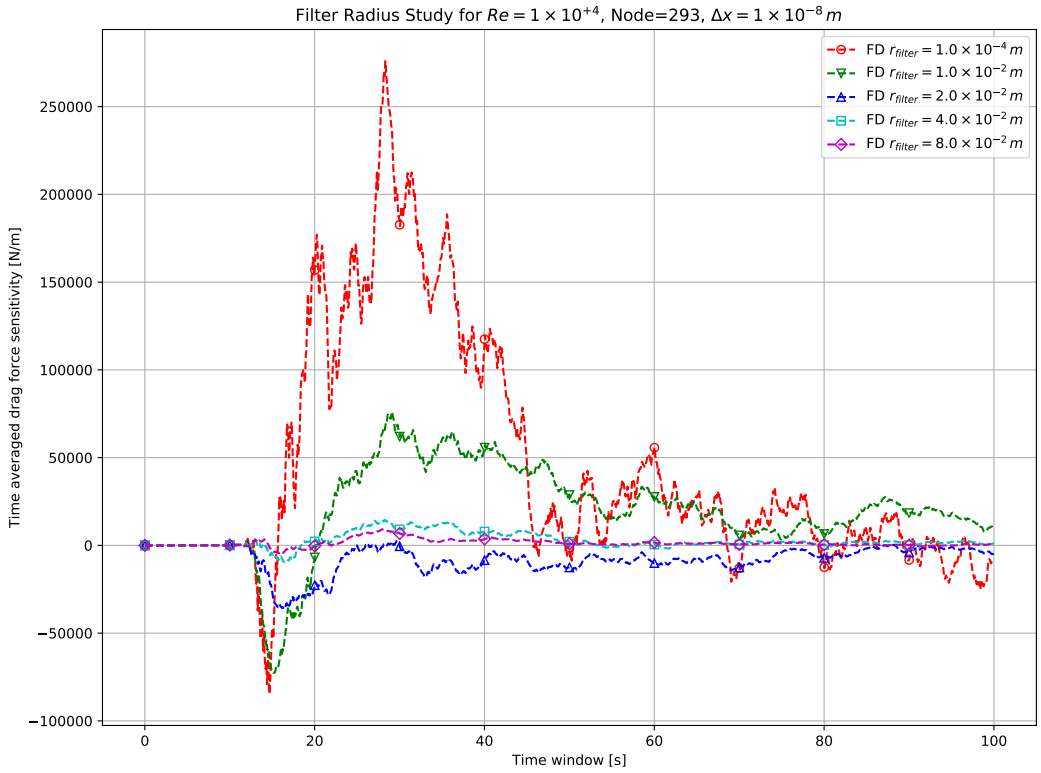


Figure 20: Time averaged drag sensitivity filter radius varitaions for $Re = 10000$ at Node=293 with $\Delta x = 1 \times 10^{-8}\,m$

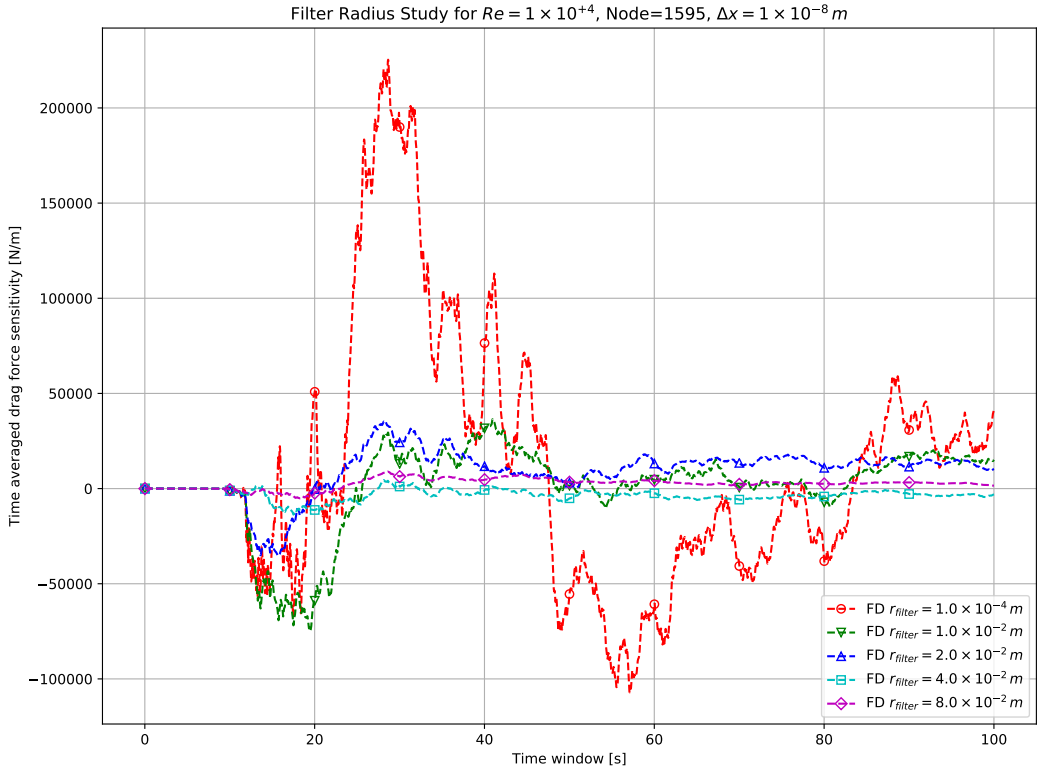


Figure 21: Time averaged drag sensitivity filter radius varitaions for $Re = 10000$ at Node=1595 with $\Delta x = 1 \times 10^{-8}\, m$

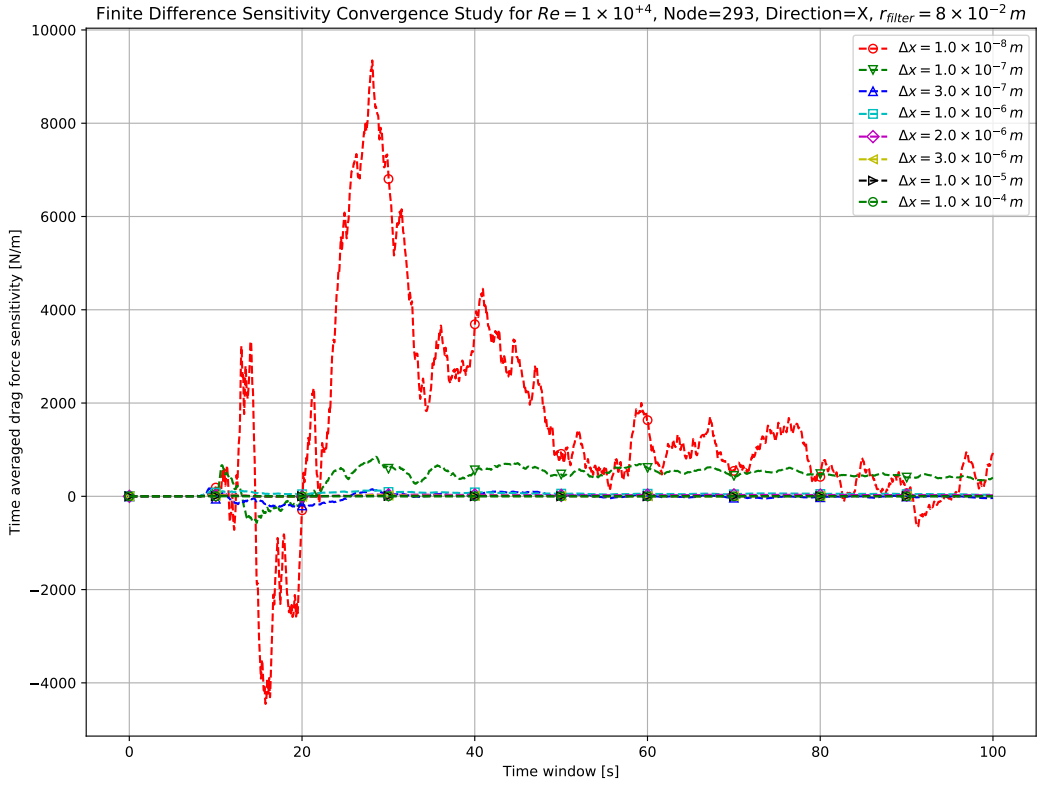


Figure 22: Time averaged drag sensitivity convergence for $Re = 10000$ at Node=293 with $r_{filter} = 8\, cm$

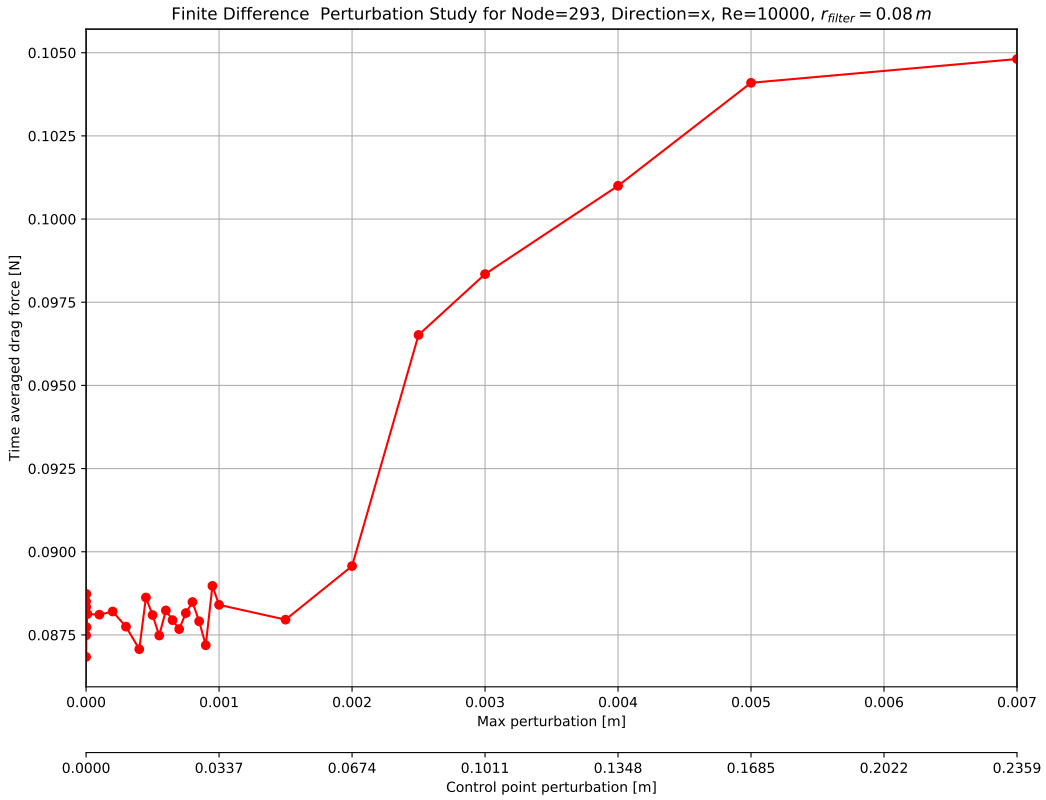


Figure 23: Time averaged drag perturbation study for $Re = 10000$ at Node=293 with $r_{filter} = 8\,cm$ in "x" direction

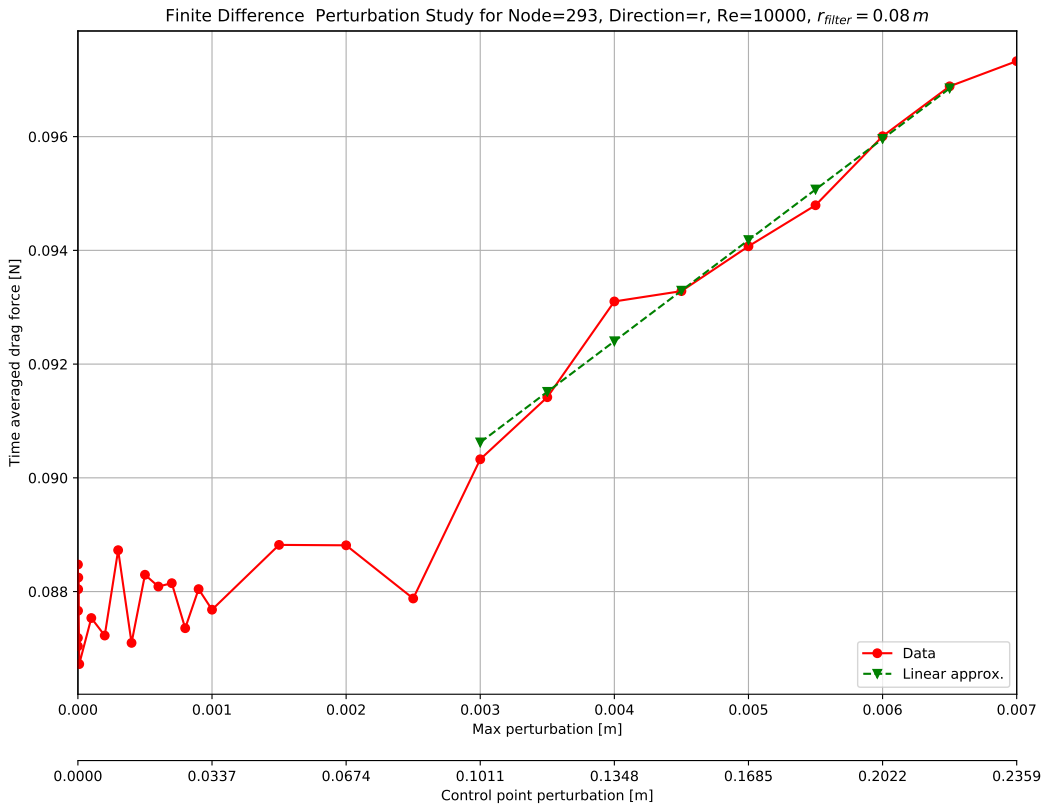


Figure 24: Time averaged drag perturbation study for $Re = 10000$ at Node=293 with $r_{filter} = 8\,cm$ in radial direction

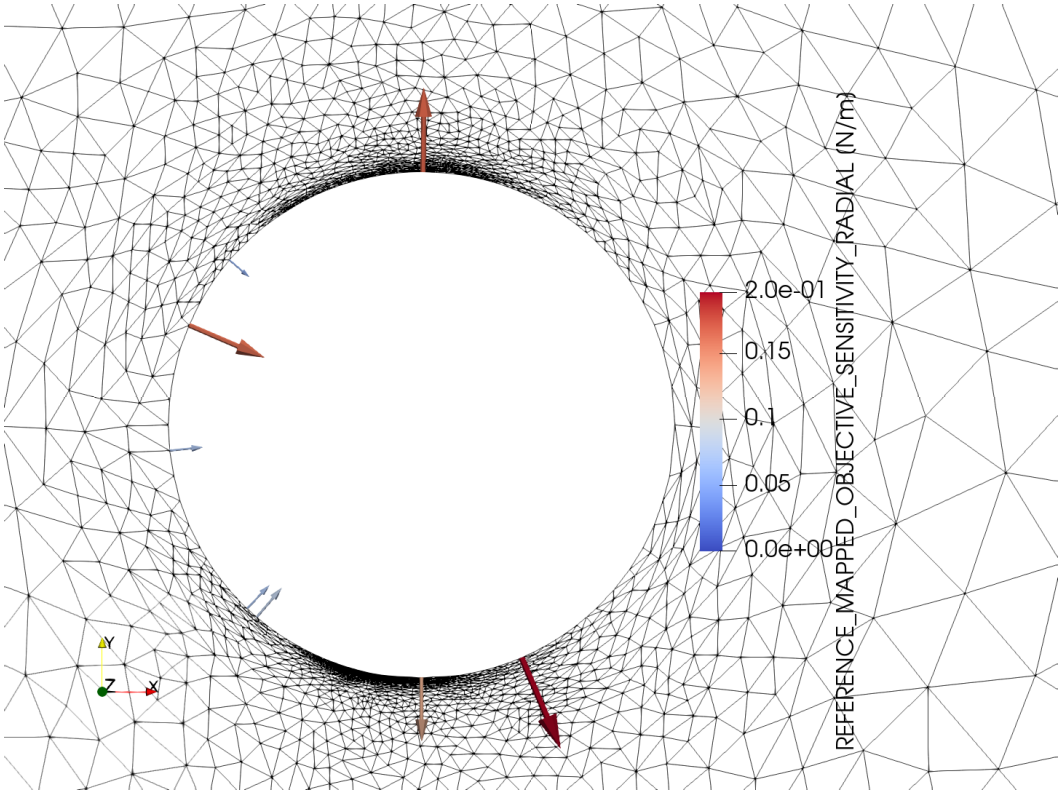


Figure 25: Finite difference time averaged drag sensitivity for $Re = 10000$ with $r_{filter} = 8\,cm$ in radial direction