Centre for Modeling and Simulation
Savitribai Phule Pune University

Master of Technology (M.Tech.)
Programme in Modeling and Simulation

Project Report

Implementation of dynamicMesh In OpenFOAM

Rupesh Wadibhasme
CMS1419

Academic Year 2014-16

Centre for Modeling and Simulation
Savitribai Phule Pune University

Certificate

This is certify that this report, titled

Implementation of dynamicMesh In OpenFOAM,

authored by

Rupesh Wadibhasme (CMS1419),

describes the project work carried out by the author under our supervision during the period from January 2016 to June 2016. This work represents the project component of the Master of Technology (M.Tech.) Programme in Modeling and Simulation at the Center for Modeling and Simulation, Savitribai Phule Pune University.

Dr. Sukratu Barve
Assistant Professr
Centre for Modeling and Simulation
Savitribai Phule Pune University
Pune 411007 India

Anjali Kshirsagar, Director
Centre for Modeling and Simulation
Savitribai Phule Pune University
Pune 411007 India

Centre for Modeling and Simulation
Savitribai Phule Pune University

Author's Declaration

This document, titled

Implementation of dynamicMesh In OpenFOAM,

authored by me, is an authentic report of the project work carried out by me as part of the Master of Technology (M.Tech.) Programme in Modeling and Simulation at the Center for Modeling and Simulation, Savitribai Phule Pune University. In writing this report, I have taken reasonable and adequate care to ensure that material borrowed from sources such as books, research papers, internet, etc., is acknowledged as per accepted academic norms and practices in this regard. I have read and understood the University's policy on plagiarism (http://unipune.ac.in/administration_files/pdf/Plagiarism_Policy_University_14-5-12.pdf).

Rupesh Wadibhasme
CMS1419

Abstract

OpenFOAM is an opensource solver written in C++ for solving fluid dynamics problems.In this thesis, the framework of OpenFOAM is explored through directory tree, solvers, utilities and simple case setup. After general introduction, concept of dynamicMesh is introduced and its importance in modeling the CFD problems has been discussed.Discussion on dyanamicMesh includes various types of dynamicMesh capabilities available in OpenFOAM like Mesh Motion, GGI and dynamicTopoFVMesh with their working principles.The dynamicMeshDict file provided with each type explains the various algorithms implemented to ensure the mesh motion during runtime eg. AdaptiveMeshReconnection,Edge swiping, solidBodyMotion.The case setup provided with dyanmicMesh explains the importance of various parameters of dynamicMesh and constrains of dynamicMeshDict file.

Further explanation on setup of dynamicMesh explains the available predefined solid body motions like linearMotion, Oscillating Rotating Motion MultiMotion etc. and explains the setup of subsequent motion coefficients associated with each motion.Further insight on dynamicTopoFvMesh,which has capability of mesh motion with topology change is given and general understanding about the setup of dynamicMeshdict file is built.

With the lesson learned from algorithms and file setup, Series of problems setup like ballTranslation and projectile are run. The results of ballTranslation case shows mesh topology changes due to motion of solid ball are handled with great accuracy.The projectile case implements pimpleDymFoam solver along with dynamicTopoFvMesh and also uses the 6-DOF motion solver to compute projectile motion at runtime. Projectile case also paves the way for sabot separation problem which need the dynamicMesh with topology change and 6-DOF motion solver to do the CFD analysis.

Acknowledgments

It gives me immense pleasure to have this opportunity to thank and acknowledge all those who made a difference in this endeavor. First and foremost, my sincere thanks to Prof. Sukratu Barve for being a wonderful guide. Prof. Barve supported me a great deal and helped me to shape my ideas. Working under Proof. Barve, I have enjoyed great deal of freedom to think even on very crude ideas and I am grateful to him for having belief in me.

I am indebted to faculty Dr. Mihir Arjunwadkar, Dr. Bhalchandra Gore and Dr. Bhalchandra Pujari and other visiting faculties for introducing me to the world of Modeling and Simulation during the entire coursework.Also, my sincere thanks to Prof. Anjali Kshirsagar, our Director at the Centre for providing wonderful working environment.

Outside CMS, I thank Mr. Akash Rao, Mr. Ashish Kanoje and Mr. Manish Marode. it is through their support and encouragement I stand where I am today!

I extend a token of thanks to all of my friends at CMS those helped me me to make this project successful. Last but not the least I would like to extend my gratitude towards my parents who have been a source of inspiration and motivation.

Contents

Chapter 1

Introduction

1.1 Introduction to OpenFOAM

The Word OpenFOAM stands for Open Source Field Operation and Manipulation. This chapter talks about the History and General structure of OpenFOAM software with the intend to give general idea to the readers who is not familiar with the OpenFOAM, and should not be considered as an standard document of reference. More complete information can be obtained from OpenFOAM user guide which can be found at sourceforge website[11], these documents are probably the best starting points for OpenFOAM beginners.

In following subsection short history of OpenFOAM is given and rest of chapter will talk about how to setup an case in OpenFOAM.

1.1.1 History

OpenFOAM (originally FOAM) was created by Henry Weller from the late 1980s atImperial College London, as a collaboration of Henry Weller and Hrvoje Jasak, who started working on his PhD thesis, Jasak (1996), at that time. The motivation to develop CFD software from scratch was mainly due to dissatisfaction with legacy codes written in Fortran language and the goal to create something reusable by others. Initially, FOAM(Earlier name of OpenFOAM) was developed as closed-source commercial software, before becoming open source in year 2004 with the announcement of OpenFOAM version 1.0.As per the OpenFOAM website ,OpenFOAM is used by research teams in many well known industries, as well as academic institutions, among them Imperial College London, Chalmers University and the Tokyo Institute of Technology[2].

1.2 Features Of OpenFOAM

OpenFOAM is first and foremost a C++ library, used primarily to create executable, known as applications. The applications can be categorized into two categories

- **Solvers:** These are each designed to solve a specific problem in continuum mechanics
- **Utilities:** These are designed to perform tasks that involve data manipulation. The OpenFOAM distribution contains various solvers and utilities covering a wide range of problems.

1.2.1 Solvers

OpenFOAM covers wide range of applications with solvers ranging from a simple potential flow solver. (potentialFoam) over incompressible steady-state (simpleFoam), transient laminar (ico-

Foam) turbulent (turbFoam) or dynamic mesh (icoDyMFoam) solvers, compressible steady-state (rhoSimpleFoam) or trans- and supersonic turbulent (sonicTurbFoam) solvers to multiphase flow solvers (e. g., interFoam), LES solvers (oodles), combustion codes (dieselEngineFoam), electromagnetics (mhdFoam), solid stress analysis (solidDisplacementFoam) and even finance (financialFoam) solvers.

Each type of solver, through proper case setup, solves the set of partial differential equation also called as Navier-Stokes equations. Explanation of each solver is beyond the scope of this thesis but detailed explanation about the solvers can be found in OpenFOAM user guide[4].

1.2.2 Utilities

The utilities can basically be divided into supporting pre- and post-processing tasks.The FoamX tool available in OpenFOAM, which is actually just a GUI to effect changes in the different dictionary files and execute other utilities, instead of calling them directly from the command line.

Pre-processing utilities

Of the many pre-processing utilities that come with OpenFOAM, Some of them are listed below.

- mapFields : maps volume fields from one mesh to another; this is useful for mesh refinement studies to map results from a coarse mesh to a finer one without starting all over.
- blockMesh : is the small included mesh generator. It is quite powerful in principle, but for more complicated geometries, it is recommended to use more sophisticated tools like snappyHexMesh.
- checkMesh : checks the mesh for validity, skewness and gives information about its size.

Post-processing utilities

The following post-processing utilities are the few of widely used post-processing utilities:

- **mach**: Calculates the local Mach number and writes it at each time in a database.
- **sample**: Allows to sample arbitrary quantities at specified locations. This is used to compare to experiments or analytical solutions.
- **foamLog**: Extracts data such as initial residuals, iterations and Courant number from a log file for plotting and observing trends over longer periods of time.

To view and post-process simulations graphically, OpenFOAM comes with paraFoam, a reader module for the open source visualization application ParaView.

1.3 Extensibility

Extensibility is one of key advantage of OpenFOAM. The source code is accessible to all and Users can create custom objects, such as boundary conditions or turbulence models that will work with existing solvers without having to modify or recompile the existing source code[1].

An equation such as

$$\frac{\partial \rho U}{\partial t} + \nabla . \phi U - \nabla . \mu \nabla U = -\nabla p \tag{1.1}$$

Is represented by the code as:

```
  solve
(
      fvm::ddt(rho,U)
    + fvm::div(phi,U)
    - fvm::laplacian(mu,U)
  ==
    - fvc::grad(p)
);
```

1.4 Simple Case Setup in OpenFOAM

Tutorial cases in OpenFOAM describes how set an problem in OpenFOAM based on the types of fluid flow involved in a problem.with the principal aim of introducing a user to the basic procedures of running OpenFOAM. The $FOAMTUTORIALS directory contains many more cases that demonstrate the use of all the solvers and many utilities supplied with Open-FOAM. Before attempting to run the tutorials, the user must first make sure that they have installed OpenFOAM correctly.

The tutorial cases describe the use of the blockMesh pre-processing tool, case setup and running OpenFOAM solvers and post-processing using paraFoam. Copies of all tutorials are available from the tutorials directory of the OpenFOAM installation. The tutorials are organised into a set of directories according to the type of flow and then subdirectories according to solver. For example, all the icoFoam cases are stored within a subdirectory incompressible/icoFoam,where incompressible indicates the type of flow. If the user wishes to run a range of example cases, it is recommended that the user copy the tutorials directory into their local run directory. They can be easily copied by typing:

```
> mkdir -p $FOAMRUN
> cp -r $FOAMTUTORIALS $FOAM_RUN
```

1.4.1 Case structure in OpenFOAM

Most of the cases in OpenFOAM have the following basic case structure:

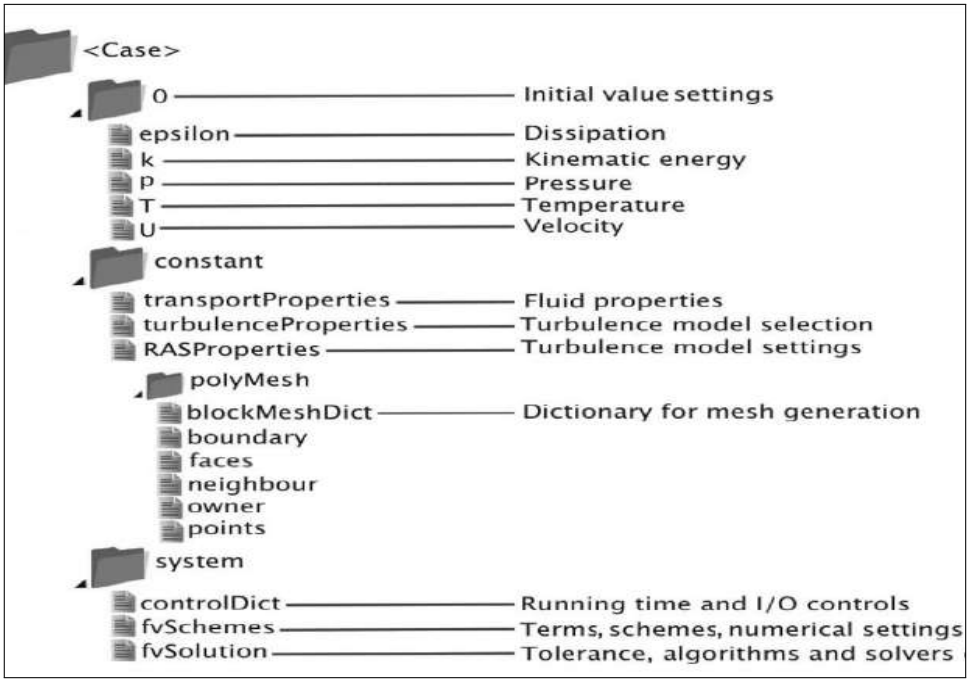

Figure 1.1: Directory tree in OpenFOAM(source [15])

There are three main directories (0, constant, system) in each case foloder. This chapter will cover the test case of cavity located in icoFoam solver which of the type incompressible.

0 Directory

The 0 directory includes the initial conditions for running the simulation. In each file in this folder the initial conditions for one property can be set. The files are named after the property they are standing for, e.g. usually T file includes temperature initial conditions. In the elbow example there are only two files inside the 0 directory, p and U. p stands for pressure and U stands for velocity[15].
The P and U file for the cavity case located in folder 0 are shown here,

The Velocity file:"U"

The U file has to be defined via three components (since velocity is a vector): first one stands for the x component, second one for the y component, and the third one for the z component. For this case setup the z component is always zero because it is a 2D simulation and no calculations will be done in the z direction. The boundaries vertical to z direction have been already set to empty.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
```

```
    object      U;
}

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }
}
```

The Pressure file:"p"

In the dimensions the physical dimension according to SI base units of the quantity is defined, for example here it shows that the p dimension is $(m/s)^2$

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }
```

```
    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Constant directory

The constant directory usually consists of a subdirectory and some files. The files (usually) include material properties, simulation physics and chemistry. In the directory polyMesh the mesh data are stored (in this case the data for converted mesh). The boundary file in this polyMesh directory includes the mesh boundary data,e.g. type, number of faces on this boundary and also starting face number (unique face IDs) for this boundary (for the sake of space, the dictionary headers will not be included in this scope any more):

BlockMeshDict File:

```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
```

```
        );
    }
);

mergePatchPairs
(
);
```

TransportProperties

By opening the transportProperties file, properties dimensions and also the property value can be found and edited, e.g. nu

```
nu [ 0 2 -1 0 0 0 0 ] 0.01;
```

nu is the fluid kinematic viscosity, which is 0.01 m^2/s for this example.

System directory:

Solver and finite volume methods settings can be found and changed in this directory. There are three main files in this directory:

- fvSchemes:The discretization scheme which is used for each term of the equations are set in this file.
- fvSolution:Contains the settings to the coupling method of pressure and velocity, the numerical methods, which are used for solving different quantities, and also the final tolerance for convergence of that quantity.
- controlDict:The time, time step from where simulation starts (startFrom),the time when the simulation finishes (stopAt), the time step (deltaT), the data saving interval (writeInterval), the saved data file format (writeFormat), the saved file data precision (writePrecision), and also if changing the files during the run can affect the run or not (runTimeModifiable) are set in this file.

Note: If the write format is ascii , then the simulation data which is written to the file can be opened and read using any text editor. If the format is binary , the data will be written in binary style and is not readable by text editors. The advantage of binary over ascii is the smaller file size, and consequently faster conversion and writing to disk, for big simulations

The controlDict file:

```
application         icoFoam;
startFrom           startTime;
startTime           0;
stopAt              endTime;
endTime             0.5;
deltaT              0.005;
writeControl        timeStep;
writeInterval       20;
purgeWrite          0;
writeFormat         ascii;
writePrecision      6;
writeCompression    uncompressed;
timeFormat          general;
timePrecision       6;
runTimeModifiable   yes;
```

Running simulation

Before running the simulation the mesh has to be created. In the previous step the mesh and the geometry data were set. For creating it the following command should be executed from case main directory.
>blockMesh

The simulation can be run by typing the solvers name and executing it.
>icoFoam

Note: For running the simulation the solver command (e.g. icoFoam) should be executed inside the copy of the tutorial main folder. For example: The command should be executed in the elbow folder, if it was run at some subfolders or somewhere else, the simulation will fail.

Exporting simulation data

The data files created by OpenFOAM should be exported (converted) by the appropriate tools, to the post processing tools data format. For ParaView:
>foamToVTK

where VTK is the ParaView data format. This command should be also executed in the case main directory, e.g. elbow. Here, ParaView is used as the post-processing tool, for running it:
>paraview &

Note: There is also another option to open the OpenFOAM simulation results with ParaView without converting them to VTK; Create an empty text file in the main case directory, name it ¡someName¿.foam (e.g. foam.foam), and execute the following command. This method is good for fast evaluation of the data in the middle of the simulation or with a decomposed case in parallel simulations:
>paraview foam.foam &

Note: By putting & at the end of command, the command line will remain active and ready for further inputs while that program is running.

Simulation Results

Once simulation is completed, results of simulation can be seen in paraFoam. Return to the paraFoam window and select the Properties panel for the cavity.OpenFOAM case module. If the correct window panels for the case module do not seem to be present at any time, please ensure that cavity.OpenFOAM is highlighted in blue eye button alongside it is switched on to show the graphics are enabled To prepare paraFoam to display the data of interest, we must first load the data at the required run time of 0.5 s. If the case was run while ParaView was open, the output data in time directories will not be automatically loaded within ParaView.To load the data the user should click Refresh Times in the Properties window. The time data will be loaded into ParaView.

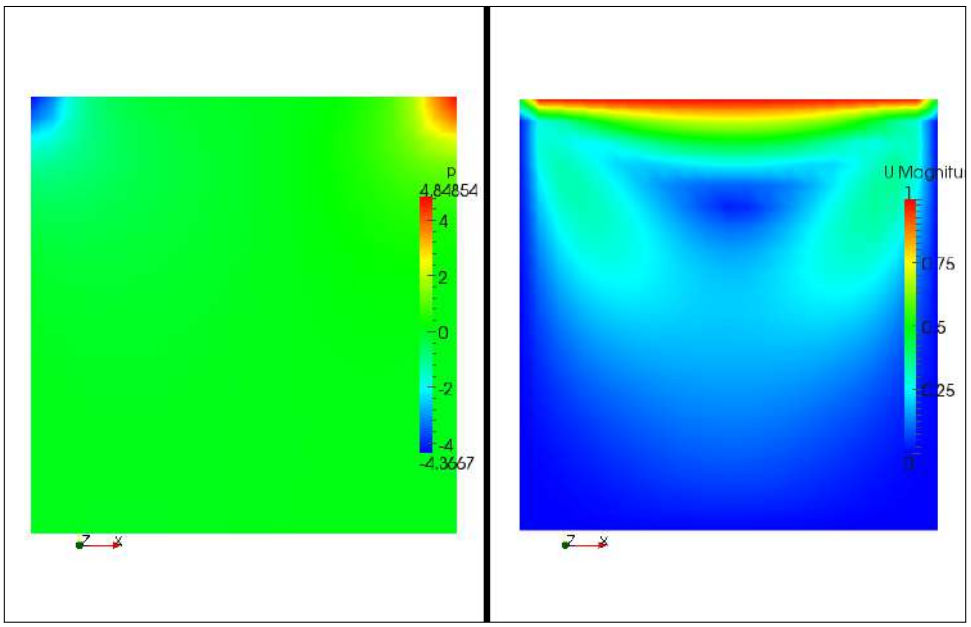

Figure 1.2: Velocity and pressure profiles of cavity case

Chapter 2

Dynamic Mesh in OpenFOAM and Types

Broadly,any problem in Computational fluid dynamics(CFD) involves the replacement of continuous problem domain with a discrete domain using a grid or also called as Mesh. This means the computational domain is divided into finite number of cells before starting to solve the partial differential equations over each cells.This Model works perfectly well when problem dose not involves any solid body motion in a computational domain. But many engineering and scientific problems involves the solid body motion in computational domain which ultimately affects the fluid flow.

`eg.Fluid flow in Piston cylinder assembly, Rotating fan or Blower`

To correctly simulate such cases,there is a need of tool/utility which is able to handle the solid body motion and subsequently the mesh around the solid body in computational domain.There can be various method's/models to simulate such problems and implementation of such method's/models OpenFOAM comes under topic of dynamicMesh.This Chapter will discuss types of dynamicMesh available in OpenFOAM and gives overview about each type. In OpenFOAM, the mesh motions and the topology changes are handled by Dynamic Mesh functionality and the Solvers that can handle these mesh changes have the letters DyM, an abbreviation for Dynamic Mesh, in its name.

`e.g. pimpleDyMFoam, interDyMFoam`

In General there are three main dynamic Mesh handling capabilities in OpenFOAM

1. Mesh Motion (Without topological change)
2. General Grid Interface
3. Dynamic Mesh with Topological Change (dynamicTopoFvMesh)

2.1 Mesh Motion

It is a dynamic Mesh operation that solely involves displacement of mesh points without altering the topological information of the mesh. Topology of mesh describes how the points, edges, faces and cells of the mesh are built, as well as the way those mesh elements relates to each other.

During mesh motion[5] operation mesh points are moved as per the desired mesh motion, eg: linear oscillation, rotational, translational Motion etc. As geometry of mesh faces and cells are based upon the mesh points, they deforms as a result of the motion of the points. After the mesh has been deformed, the fields whose values still relate to initial mesh need to be mapped to the new mesh. This is necessary since the field value in finite volume method represents

average with respect to the cell volume, or face area. Both the cell volume and area of the face may change as a result of mesh motion.
Mesh motion, where mesh points are moved without altering the topology of mesh, is classified in two main types.

- Solid Body Motion
- Mesh deformation

2.1.1 Solid Body Motion:

The solid body mesh motion is defined as a motion of a body, applied to the mesh points, where no relative displacement between any two points occurs.

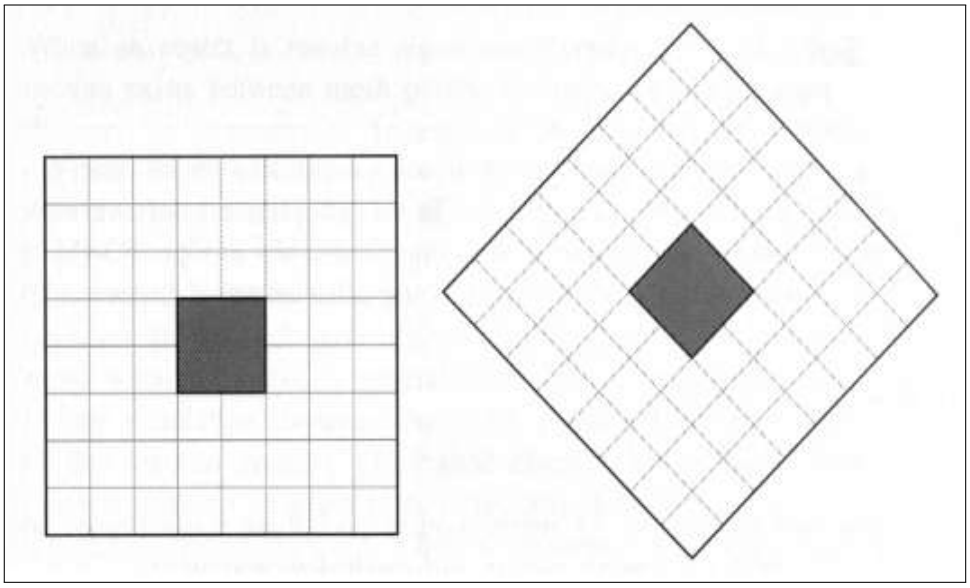

Figure 2.1: The filled body is subjected to solid body motion, so mesh points do not move relatively to each other

In OpenFOAM, the solid body motion is defined by variety of classes, all derived from their common base solidBodyMotionFunction. The motion function returns a septernian which describes the motion of the body. This motion is a combination of a vector for the translation and a quaternion for the rotation. Classes derived from the abstract base class solidBodyMotionFunction define what type of motion is present. Ranging from LinearMotion and rotatingMotion or combination of both to have more complex motions like SDA(Ship design analysis) and SKA(Sea Keeping Analysis).

Class hierarchy diagram is shown in following Figure

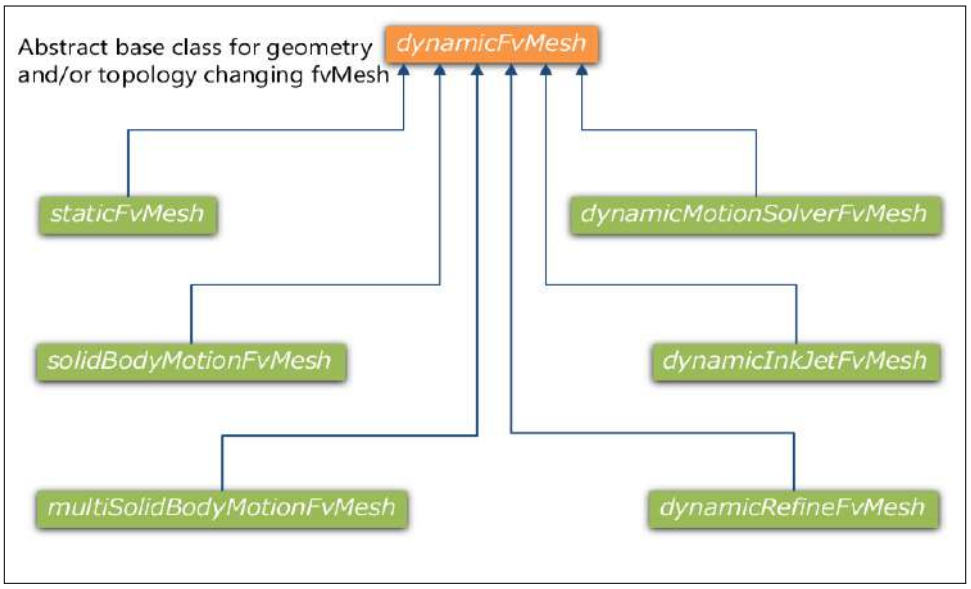

Figure 2.2: Inheritance diagram for dynamicFvMesh models

2.1.2 Mesh Deformation

Mesh Deformation or Mesh Morphing, is performed by applying a different displacement to each mesh points. Mesh deformation will introduce relative motion between mesh points, which will in turn distort mesh cells and modify the mesh in an in-homogeneous way. This usually leads to high distortions, especially for larger motions, which often reduces the quality of Mesh.

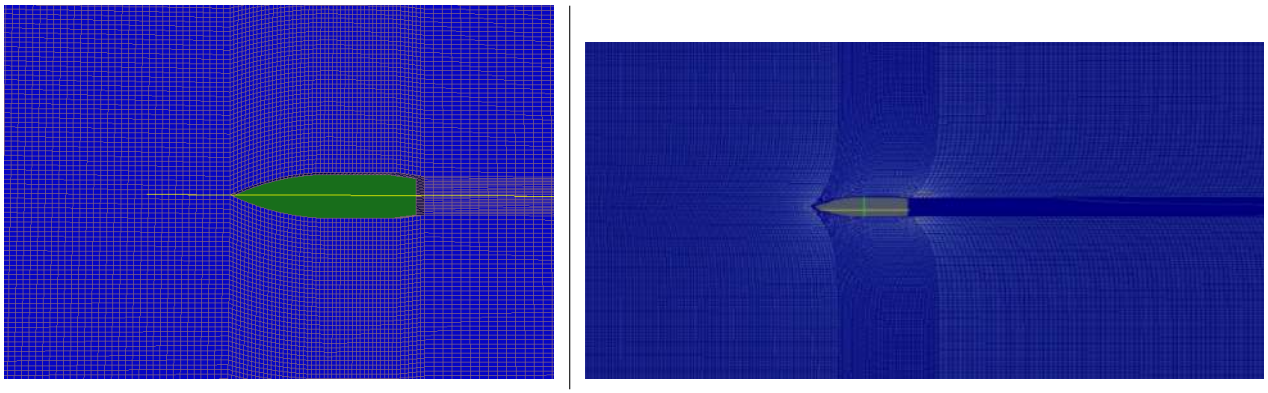

Figure 2.3: Mesh before Deformation(left) and Mesh after deformation(Right)

In order to maintain the mesh quality, mesh deformation need to be used carefully: often only small sub-region of the mesh is deformed strongly, while the deformation for the rest of the mesh is kept as low as possible. This type of approach results in mesh motion in the region where It is necessary, Which is often In the vicinity of the mesh motion boundary or a part of mesh boundary. Additionally, the mesh deformation can be regarded as an optimization problem, where the quality of the mesh represents a domain-global optimized scalar function.

2.2 General Grid Interface(GGI)

OpenFOAM version 1.6-ext, 3.1-ext and 3.2-ext, provides an implementation of the Generalized Grid Interface (GGI) to allow interaction between meshes. It was available before AMI was implemented in the official OpenFOAM distribution, and has thus been more widely used. Certain cases testing GGI have been made available by the Extend-Project and others.

Implicit couplings are present in OpenFOAM in order to join multiple mesh regions into a single contiguous domain. But most of them are built to join conformal mesh regions, where the patches nodes on each side of the interface are matching one to one.The GGI, developed by M. Beaudoin and H. Jasak[8] is a coupling interface used to join multiple non-conformal regions where the patches nodes on each side of the interface do not match. A GGI is commonly used in turbo-machinery, where the flow is simulated through a succession of various complex geometries.

The dynamicMeshDict setup for oscillating cylinders case which uses GGI approach is shown below:

```
dynamicFvMesh      multiTopoBodyFvMesh;

multiTopoBodyFvMeshCoeffs
{
    bodies
    (
        frontCyl
        {
            movingCells     cyl1;
            layerFaces
            (
                topLayerCyl1
                botLayerCyl1
            );

            solidBodyMotionFunction   linearOscillation;
            linearOscillationCoeffs
            {
                amplitude   (0 0.028 0);
                period      2;
            }

            minThickness     0.0015;
            maxThickness     0.004;

            invertMask       true;
        }

        backCyl
        {
            movingCells     cyl2;
            layerFaces
            (
                topLayerCyl2
```

```
            botLayerCyl2
        );

        solidBodyMotionFunction   linearOscillation;
        linearOscillationCoeffs
        {
            amplitude   (0 -0.028 0);
            period      3;
        }

        minThickness      0.0015;
        maxThickness      0.004;

        invertMask        true;
    }
);
}
```

Mesh Motion using GGI

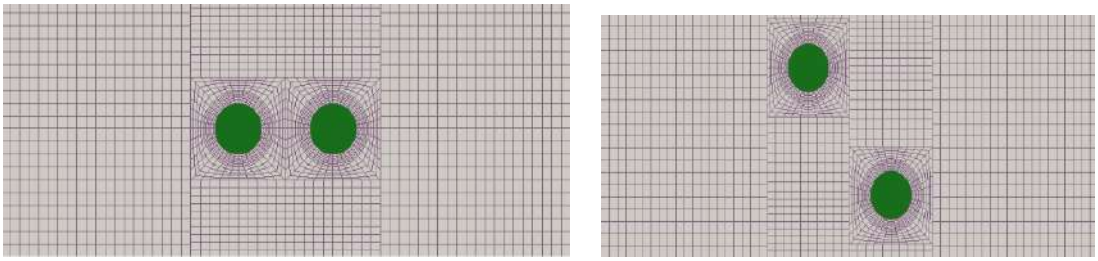

Figure 2.4: Mesh motion using GGI approach

2.3 Dynamic Mesh With topology change(dynamicTopoFvMesh)

As name suggests this type of dynamic mesh involves the mesh motion (motion of node points) but it handles the issues of mesh deformation by allowing the topological changes in the existing mesh through various algorithms like Adaptive Mesh reconnection, local mesh refinement and Variable Remapping.

More emphasis on `dynamicTopoFvMesh`[7] is given, as this is recently developed utility in OpenFOAM and can be implemented over wide verity of problems. Some of the case studies using `dynamicTopoFvMesh` are also presented in a later chapters which otherwise might be extremely difficult or sometime impossible to simulate using other available dynamicMesh.

This dynamicMesh utility is divided in various classes as follows

- **dynamicTopoFvMesh class:** Mesh class that extends dynamicFvMesh functionality to handle dynamic simplical meshes, which consist of triangle prisms in 2D, and tetrahedra in 3D. Adaptation is driven mainly by mesh-quality and mesh refinement criteria. When used in combination with mesh-smoothing methods, this functionality is expected to suit situations where domain deformation characteristics are not known a-priori. Conservative solution remapping after mesh reconnection is performed automatically.

- **fluxCorrector:** Auxiliary library which is used by dynamicTopoFvMesh to perform a correction to velocity fluxes after mesh reconnection.
- **mesquiteMotionSolver:** Class that provides a general interface to the Mesquite mesh smoothing library from Sandia National Labs. The class also performs smoothing for surface meshes using a spring-analogy approach, and is known to work in parallel.
- **conservativeMeshToMesh:** Conservative mesh-to-mesh interpolation class.
- **mapConservativeFields:** Field-mapping utility that works in a manner similar to mapFields in OpenFOAM, using the conservativeMeshToMesh class as a backend. This utility is currently not designed to work in parallel.
- **Asaptive Mesh Reconnection:**

  Adaptive mesh re connection[7] is a fairly broad term that is used to describe the process of re-meshing (or re-gridding) an existing mesh, subject to certain requirements. In the framework of Lagrangian interface-tracking, such methods are often necessary in situations where cells have become excessively distorted, and mesh smoothing can do very little to mitigate the issue. One approach is to re-mesh the domain entirely, using an appropriate mesh-generation algorithm.

There are two major drawbacks to this approach:

- Mesh-generation can be a particularly time-consuming process, and currently automatic mesh generation (i.e., without any user intervention) is not well established.
- Re-meshing requires the interpolation of flow variables to the new mesh, which can frequently induce errors and fluctuations to the flow-field. By re-meshing the entire domain, these errors can often be difficult to contain, and might even magnify as the simulation proceeds over time.

Therefore, a more logical approach is local re-meshing, which works well in minimizing interpolation errors and, given the right algorithms, can be quite efficient. The mesh reconnection algorithms in this work are limited to simplical meshes (triangles in 2D and tetrahedra in 3D), since generalization of these concepts to arbitrary polyhedra is complicated.

The topic of mesh reconnection can also be extended to include refinement and de-refinement of cells. Physical phenomena can often develop near-singular solutions with large gradients in very localized regions of the mesh, and in most cases, the only solution is to resolve these variations using an increased number of cells in the area. The option of uniformly refining the entire mesh is immediately rejected, because the exponential increase in computational effort (particularly in three dimensions) is not really justified, and areas away from the singularity dont have to be resolved that well anyway. Local refinement allows an increase in mesh density around areas that need it most, thus providing improved solution accuracy at an acceptable computational cost.

2.3.1 Mesh Reconnection for Improved Mesh Quality

The quality of a simplical mesh can be locally improved by an operation known as edge-swapping (sometimes also known as edge-flipping), which is applicable in both two- and three-dimensions.In two-dimensions, the condition for edge-swapping is defined by the Delaunay criterion, which specifies that no mesh points are to be contained in the circumcircle of any cell of the mesh. This concept was first introduced by Lawson[6], who also extended edge-swapping to three and higher dimensions. Fig. Shows the Delaunay criterion in 2D. The point marked d

is contained within the circumcircle of the triangle (abc) and so, edge bc must be flipped. The flipped configuration (and new circumcircle) is shown in the figure on the right.

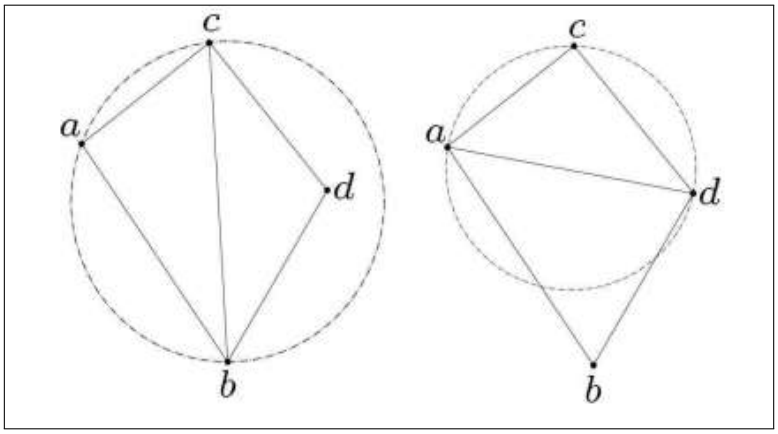

Figure 2.5: Edge flipping operation using the Delaunay criterion

This approach is mathematically guaranteed to provide a mesh of better quality; as it maximizes the minimum angle of a triangulation and is also irreversible, thereby preventing infinite loops. In the current code framework, 2D simulations are performed by extruding a two-dimensional surface-mesh by one cell in the direction normal to the mesh-plane. Thus, two-dimensional simulations are actually performed in 3D. In this context, a 2D simplical mesh is now no longer composed of triangles, but triangular prisms, as shown in following figure

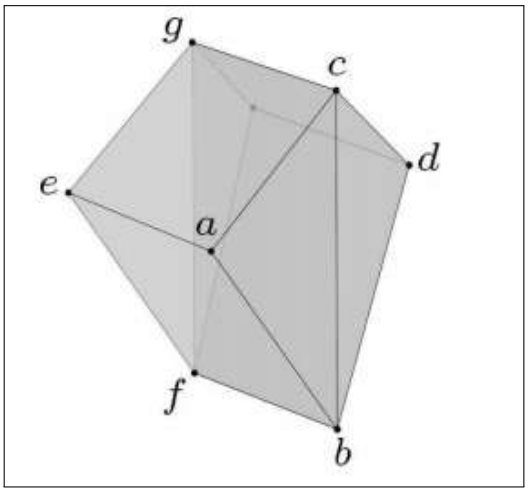

Figure 2.6: Edge flipping operation using the Delaunay criterion

The detailed explonation about how to set dynamicMeshDict, its various parameters and sample case setup's using using `dynamicTopoFvMesh` is given in chapter 3.

Chapter 3

Some Insight into dynamicMesh parameters and setup

Setting up the correct dynamic mesh is an important aspect for using dynamicMesh in OpenFOAM. Hence it is important to understand dynamicMesh from its algorithm point of view. This chapter will discuss some of the key aspects of how to setup dynamicMeshDict file.

Note: This chapter do not include the setup for dynamicMesh with Topology change, which is explained in the Chapter 4 separately.

Following code represents the general structure of dynamicmesh dict which is located in `constant/dynamicMeshdict`.

```
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
location    "constant";
object      dynamicMeshDict;
}
//*** Selection of dynamicFvMesh Type from available options

dynamicFvMesh      solidBodyMotionFvMesh;

motionSolverLibs   ( "libfvMotionSolvers.so" );

//*** Settings for the dynamicFvMesh Motion parameters

solidBodyMotionFvMeshCoeffs
{
        cellZone                rotor;
        solidBodyMotionFunction rotatingMotion;
        rotatingMotionCoeffs
        {
                origin   (0 0 0);
                axis     (0 0 1);
                omega    6.2832; // rad/s
        }
}
```

The solidBodyMotionFvMeshCoeffs represents the setup for the motion coefficients. The code represented above shows parameters like origin,axis and omega for simulating rotating-Motion. Some of the other available motions are shown in later part of the chapter.The source code for the solvers like `pimpleDymFOAM, interDyMFoam` which employs the dynamicMesh implementation is located at

`applications/solvers/incompressible/pimpleFoam/pimpleDyMFoam/pimpleDymFoam.C`

(Source code for pimpleDyMFoam)

```
while (runTime.run())
{
#include "readControls.H"
#include "CourantNo.H"

#include "setDeltaT.H"

runTime++;

Info<<"Time=" << runTime.timeName() << nl << endl;

mesh.update();

phi = mesh.Sf() & Uf;

if (mesh.changing() && correctPhi)
{
  #include "correctPhi.H"
}
// Make the flux relative to the mesh motion

          fvc::makeRelative(phi, U);

           if (mesh.changing() && checkMeshCourantNo)
           {
          #include "meshCourantNo.H"
           }
```

The `mesh.update()` calls the mesh motion library to calculate the new position of points and update the mesh.This is an crucial operation in dynamicMesh implementation. The mesh update may needs the quantities or dependant variables of PDE to be remapped on the new mesh or the mesh created after moving the node points of the mesh. Improper mapping of variables may lead to numerous errors.

3.0.1 Available types of SolidBodyMotion

The type of motion that a solid body in a fluid domain can perform is predefined. Following are the types of available solid body motions in Mesh motion type.

1. Transnational motions
   - **linearMotion**: Uniform linear motion with constant velocity
   - **oscillatingLinearMotion**: Oscillating linear motion
2. Rotational motions
   - **rotatingMotion**
   - **axisRotationMotion**
   - **oscillatingRotatingMotion**
3. Ship Design Analysis(SDA)
4. tabulated6DoFMotion
5. Combination of above motion types
   - **multiMotion**

Each of the motion mentioned above need to be set up with its Motion Coefficients. Following part of code shows the way to set the motion coefficients for the desired solid body motion.

Coefficient for linearMotion

```
dynamicFvMesh       solidBodyMotionFvMesh;

motionSolverLibs    ( "libfvMotionSolvers.so" );

solidBodyMotionFvMeshCoeffs
  {
      solidBodyMotionFunction   linearMotion;

      linearMotionCoeffs
      {
        velocity        (1 0 0);
      }
  }
```

Coefficient for Oscillating linear Motion:

```
dynamicFvMesh      solidBodyMotionFvMesh;
motionSolverLibs   ( "libfvMotionSolvers.so" );

solidBodyMotionFvMeshCoeffs
       {
          cellZone                  inletChannel;
          solidBodyMotionFunction   oscillatingLinearMotion;
```

```
  oscillatingLinearMotionCoeffs
      {
          Amplitude    (0   0.5   0);
          omega        3.14;
      }
       }
```

Coefficient for Oscillating Rotating Motion

```
dynamicFvMesh      solidBodyMotionFvMesh;
motionSolverLibs   ( "libfvMotionSolvers.so" );

solidBodyMotionFvMeshCoeffs
  {
          cellZone                    rotor;
          solidBodyMotionFunction     rotatingMotion;

  rotatingMotionCoeffs
      {
          origin                (0 0 0);
          axis            (0 0 1);
          omega           6.2832;
      }
  }
```

Coefficient for Oscillating Axis Rotation Motion

```
dynamicFvMesh      solidBodyMotionFvMesh;

motionSolverLibs   ( "libfvMotionSolvers.so" );

solidBodyMotionFvMeshCoeffs
  {
      cellZone                  rotor;
      solidBodyMotionFunction   axisRotationMotion;

    axisRotationMotionCoeffs
    {
      origin          (0 0 0);
      radialVelocity  (0 0 360);
    }
  }
```

Coefficient for Multi Motion

Multi motion class is an combination of any of the above mentioned motions. OpenFOAM provides the option for combining multiple motions in one case setup. That means we can have more than one solid body in a case setup and multiple motions to those solid bodys which enables user to have great flexibility in modeling wide verity of problems.

```
dynamicFvMesh    solidBodyMotionFvMesh;
```

```
solidBodyMotionFvMeshCoeffs
{
    solidBodyMotionFunction       multiMotion;
   multiMotionCoeffs
     {
       // Table rotating in z axis
          rotatingTable
            {
              solidBodyMotionFunction          rotatingMotion;
              rotatingMotionCoeffs
                {
                  origin       (0 0.1 0);
                  axis         (0 0 1);
                  omega        6.2832; // rad/s
                }
                    }
// Tube rocking on rotating table
          rotatingBox
            {
              solidBodyMotionFunction        oscillatingRotatingMotion;
              oscillatingRotating            MotionCoeffs
                {
                  origin       (0 0 0);
                  omega        40;
                  amplitude    (45 0 0);
                }
            }

       }
}
```

Chapter 4

Insight into dynamicTopoFvMesh with Case study's

In continuation with introduction given in chapter 2, this chapter will give reader about actual setup of dynamicTopoFvMesh and explanation of various important parameters of the utility.This dynamic meshing class uses a combination of node smoothing and edge re-connection methods to improve mesh quality for deforming domain simulations (e.g. free surface motion, IC cylinder motion, six-DOF motion). This class is applicable on simplicial (tetrahedral & triangle) meshes only. For many of the string based options, an invalid entry will throw a list of valid options at run time similar to typical foam error responses.

4.0.1 dynamicMeshDict options for the dynamicTopoFvMesh class

Choose the node smoother solver of your choice. The Mesquite Mesh Quality Improvement Library developed by Sandia National Labs was used throughout the development of the mesh engine. This solver handles three dimensional smoothing of internal nodes, and tangential smoothing of any patch/boundary nodes.

```
solver          mesquiteMotionSolver;
```

Choose the dynamicFvMesh class to load.

```
dynamicFvMeshLibs    ("libdynamicTopoFvMesh.so");
```

Not all solvers have explicit flux correction in the top level. In this case you can have the re-mesher run a Poisson based flux corrector during mesh-to-mesh field remapping to maintain a divergence free flow.

Load extra flux correction library.

```
fluxCorrectorLibs    ("libincompressibleFluxCorrector.so");
```

Use stand-alone Poisson solver.

```
fluxCorrector        Poisson;
```

If fluxes need to be corrected then corrector will need to be passed the appropriate field names.

```
PoissonCorrector
        {
         correctFluxes   yes;
         U               U;
         p               p;
```

```
        rAU             rAU;
        phi             phi;

    }
```

Mesquite Smoother Options

There are other smoother algorithms available in openFOAM-ext. There are no restrictions on smoother use however the Mesquite lib. seems to do an excellent job and is computationally efficient.

Choose mesquite objection function

```
objFunction     LptoP;
```

Optimization metric

```
optMetric       InverseMeanRatio;
```

Optimization algorithm

```
optAlgorithm    FeasibleNewton;
```

Termination criteria sub-dictionary (takes default values if not specified). Specifying an empty sub-dictionary terminates with available options

```
tcInner
{
  absGradL2   1e-4;
  cpuTime     1.25;
}
```

For composite functions, two objectives need to be specified

```
firstFunction     LptoP;
secondFunction    LInf;
```

A CG solver is used to calculate the tangential motion of surface nodes. Multiple CG sweeps typically improve the smoothness of the surface mesh.

```
tolerance   1e-2;
nSweeps     2;
```

Slip patches are any patches where you want the nodes to move tangentially along the boundary. If a patch is not on this list, nodes will not displace. This does not exclude the patch from face and edge refinement.

```
slipPatches
{
  mySlipPatch;
}
```

If you want to apply pre-determined motion to a patch, specify its name and motion specific options here. Below is an example patch type that will displace the patch in a sinusoidal fashion given the parameters shown. A few derived mesh motion patch types can be found at:
`src/dynamicMesh/meshMotion/fvMotionSolver/pointPatchFields/derived`

```
fixedValuePatches
    {
      myFixedValuePatch
            {
            type          oscillatingDisplacement;
            amplitude     (0 -0.5 0);
            axis          (0 -1 0);
            origin        (0 0.8 0);
            angle0        0.0;
            omega         0.15;
            value         uniform (0 0 0);
            }
    }
```

How often should internal and surface smoothing be performed? A value of 1 is every time step, 2 is every other step etc.

```
surfInterval        1;
```

Large point motions (which correlates to large swept face volumes) can cause flux spikes in the next time step. This will relax points to a fractional distance between the original and the location found by the CG solution.

```
RelaxationFactor      0.6;
```

Small errors in cell volume occur from tangential motion of points on a non-planar surface. This algorithm will correct the domain volume. This is usually only needed for special circumstances.

```
volumeCorrection        false;
volCorrTolerance        1e-20;
volCorrMaxIter          100;
```

dynamicTopoFvMesh Options

A `no` value will allow the simulation to start without all non-vital entries to be set.

```
allOptionsMandatory     no;
```

Higher debug levels will write out progressively more information to both the terminal and case directories. Additional mesh connectivity checks are also performed at higher b levels.

```
Debug       0;
```

Boolean to set sliver removal (sliver type cell removal requires several sub-steps and are difficult to detect)

```
removeSlivers         yes;
```

Specifies the minimum quality criteria for sliver removal.

```
sliverThreshold       0.3;
```

Write the mesh length scale as a volScalarField during case write-outs.

```
dumpLengthScale       yes;
```

Should dynamicTopoFvMesh load and run the motion solver specified above?

`Note: Refinement without smoothing motion will most likely degrade mesh quality quickly.`

```
loadMotionSolver      true;
```

Specify the number of threads to be used (for multi-core machines)

```
threads               1;
```

Specify re-meshing time step interval. Remeshing is typically not required every time step.

```
Interval              3;
```

If the number of bisections/collapses are to be limited in a certain timestep, set this option

```
maxModifications      10000;
```

If edge-swapping is to be avoided at surfaces with high curvature specify the threshold here.

```
swapDeviation         0.85;
```

Perform edge-bisection/collapse? If no, swaps will still occur.

```
edgeRefinement        yes;
```

refinementOptions Sub-Dictionary

How much smaller should an edge be than its target length scale until it is collapsed.

```
collapseRatio         0.4;
```

How much larger should an edge be than its target length scale until it is bisected.

```
bisectionRatio        1.75;
```

The rate at which the target length scale will grow,cell layer by cell layer, out from patches, and toward the interior of the mesh.

```
growthFactor          1.05;
```

Maximum and Minimum length-scales can be specified here.

```
maxLengthScale      2e-05;
minLengthScale      0.25e-5;
```

By default, existing boundary edge-lengths are used for length-scales. These can be fixed for certain patches and has units of meters.

```
fixedLengthScalePatches
{
  myFixedLengthScalePatch;
}
```

Patch length scale will be calculated and set at simulation start. The length scale of freeLengthScalePatches is calculated as the average edge length of the patch.

```
freeLengthScalePatches
{
  myFreeLengthScalePatch;
}
```

Avoid any modifications on these patches

```
noModificationPatches
{
  myNoModificationPatch;
}
```

If curvature-based refinement is required, specify patches here. curvatureDeviation, calculated by $n^1.n^2$ is the normalized dot product between two adjacent faces. A value of 1 is a perfectly flat face pair. This tools should be using in conjunction with minLengthScale to prevent recursive over-refinement.

```
curvaturePatches
{
  myCurvaturePatch;
}

curvatureDeviation     0.96;
```

Similar to the methodology behind interDyMFoam, simplectic refinement can be performed. this can be useful on simulations that do not have deforming domains such as fixed VOF simulations. Comment this section out if you do not want field based refinement.

```
//- Field-based refinement options
  fieldRefinement gamma;
  fieldLengthScale     0.005;
  lowerRefineLevel     0.001;
  upperRefineLevel     0.999;
  maxRefineLevel       4;
  meanScale            0.015;
```

Tetrahedral mesh quality metric. Different methods are available for computing the geometric quality of a tetrahedra. They can be listed by filling an invalid.

```
  TetMetric      Knupp;
```

Avoid 2-2 swapping on certain patches

```
noSwapPatches
  {
    myNoSwappingPatch;
  }
```

Options for dynamic parallel load-balancing. Redistribution will occur during runtime without user intervention.

```
loadBalancing
      {
      enabled              true;      //perform load balancing?
      Interval             100;       //timestep interval to perform redist.
      method               parMetis;  //redistribution algorithm
      numberOfSubdomains   2;         //number of subdomains
      mergeTol             1e-6;      //point pair matching tol.
      }
```

4.0.2 Useful Guidelines

- The mesh that your meshing tool will generate and the mesh that dynamicTopoFvMesh will tend towards will be different (sometimes very different). When generating your initial mesh be sure to keep track of all your patch length scales (and growth functions if your using them) and set them accordingly in the dynamic mesh dictionary. The first topo-change process of the simulation will most likely involve a very large number of topological changes (bisections, collapses, swaps) and drastically re-arrange your mesh. It is recommended that your first step be to run.

  moveDynamicMesh

  through many times steps until the number of topo-changes per time step has tended toward zero. Be sure to shut off all perscribed boundary motion and flow-variable related settings such as flux correction and field remapping before running it. Now reset your initial conditions and start the flow solver from your new mesh with flux correction and field remapping active.

- For large domains, a small change in patch length scale or growth factor can drastically change the number of cells in your final target mesh. Be sure to play around with values and get the mesh you want.

4.1 Case Study's using dynamicTopoFvMesh

This section discusses test cases to demonstrate the efficiency and robustness of the mesh adaptation algorithms, and their general applicability to engineering problems.To test the ability of the described mesh adaptation algorithms to handle large domain deformations, a variety of validation cases were used. The first `BallTranslation` case in this section do not involve fluid flow, and only serve the purpose of demonstrating the versatility of the dynamicTopoFvMesh Utility.

The Second case of projectile includes Mesh Motion along with fluid flow simulated using `pimpleDyMFoam` solver.

4.1.1 Case1: Translating Ball in a Rectangular Domain

This test case involves a three-dimensional tetrahedral mesh with approximately 50,000 cells. The outer rectangle is 5 units wide, 2 units tall and 2 units deep. The inner sphere has a diameter of 0.5 units. The case was run for a total time of 60 units, with a time-step of 0.1. The minimum cell quality, as defined by the quality metric was 0.48 .

The case setup uses the dynamicTopoFvMesh and results shows the outcome of adaptive mesh re-connection method to achieve solid body motion through fluid domain. Interior mesh vertices were smoothed using the Mesquite Mesh Improvement library, while surface vertices were smoothed using the spring-based Laplacian method.

This case setup is just to represent the implementation of Adaptive Mesh Reconnecttion methodology in dynamicTopoFvMesh and dose not involve any solver in it. Following part

explains the case setup and done for `ballTranslation` problem and the results showing the mesh topology changes.

controlDict File

The `controlDict` file controls the simulation time and the write interval.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}

startTime           0;
startFrom           latestTime;
endTime             35;
stopAt              endTime;
deltaT              0.1;
adjustTimeStep      no;
maxCo               0.8;
writeControl        timeStep;
writeInterval       5;
writeFormat         ascii;
writeCompression    uncompressed;
timeFormat          general;
timePrecision       11;
runTimeModifiable   yes;
```

dynamicMeshDict setup

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}

solver          mesquiteMotionSolver;

dynamicFvMesh   dynamicTopoFvMesh;

dynamicFvMeshLibs    ("libdynamicTopoFvMesh.so");

//Mesquite Motin Solver Option
mesquiteOptions
{
```

```
        // Optimization metric
        optMetric     AspectRatioGamma;

        // Objective function
        objFunction    LPtoP;

        // Optimization algorithm
        optAlgorithm   FeasibleNewton;

//Termination criteria sub-dictionary(takes default values if not specified)
//Specifying an empty sub-dictionary terminates with available options
        tcInner
        {
           absGradL2     1e-4;
           cpuTime       1.25;
        }

        // For composite functions, two objectives need to be specified
        firstFunction    LPtoP;
        secondFunction   LInf;

        // Power value for the LPtoP objective function
        pValue      2;
        power       2;

        // Specify a tolerance for the CG solver
        tolerance    1e-2;

        // Specify number of CG sweeps
        nSweeps      1;

        // Set run-time debug level
        debug        0;

        // Specify interval for surface smoothing
        surfInterval   1;

    //- Specify fixedValue patches for the motionSolver
    fixedValuePatches
    {
        ball
        {
            type           oscillatingDisplacement;
            amplitude      (0.1 0 0);
            axis           (1 0 0);
            origin         (2.5 0 0);
            angle0         0.0;
            omega          0.005;
            value          uniform (0 0 0);
        }
```

```
    }

  }

// Options for dynamicTopoFvMesh
dynamicTopoFvMesh
{
    allOptionsMandatory no;

    // Specify the number of threads
    threads             1;

    // Specify re-meshing interval
    interval            1;

    // Options for edge-bisection/collapse
    edgeRefinement      yes;

    refinementOptions
    {
        collapseRatio   0.7;
        bisectionRatio  1.5;
        growthFactor    1.02;

        // In 3D, sliverThreshold specifies the
        // quality criteria for sliver removal.
        sliverThreshold 0.4;

  // By default, existing boundary edge-lengths are used for length-scales
  // Length-scale can be fixed for certain patches.
        fixedLengthScalePatches
        {
           ball         0.1;
        }

        freeLengthScalePatches
        {
           outTop; outBottom; outLeft; outRight; outFront; outBack;

        }

        noModificationPatches
        {
            outTop; outBottom; outLeft; outRight; outFront; outBack;
        }

    }
```

```
    // Tetrahedral mesh quality metric
    tetMetric           Knupp;

    // Avoid 2-2 swapping on certain patches
    noSwapPatches
    {ball;}
}
```

Results

As mentioned earlier this case dose not involve any fluid flow hence there is no need to do case setup for boundary conditions like p and U files.Results of mesh motion are shown at various time interval which shows the changes in mesh topology due to motion of solid ball.

Ball at simulation Time 0

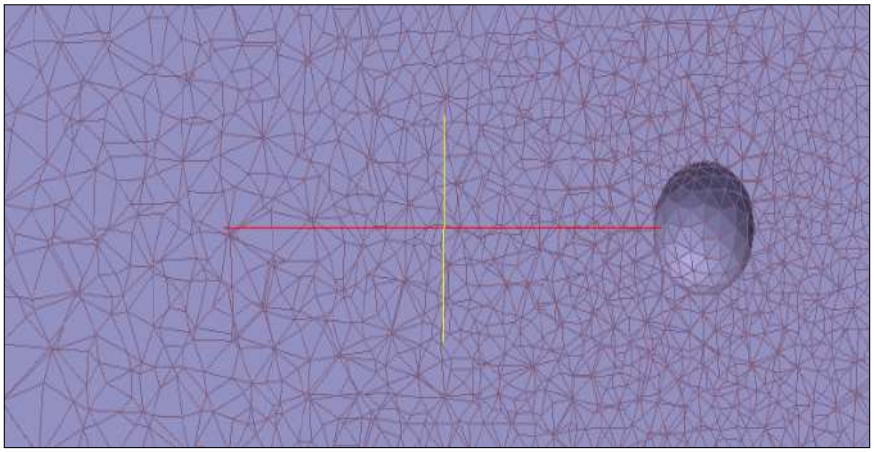

Figure 4.1: Translation of ball in fluid domain

Ball at simulation Time 20

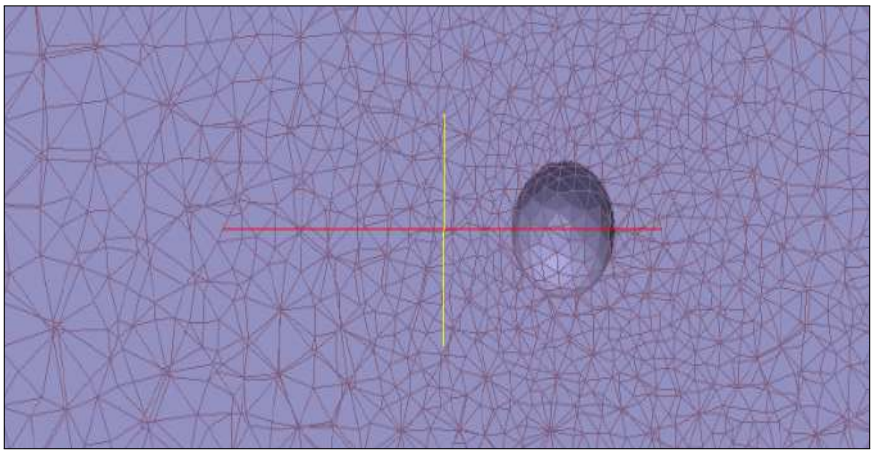

Figure 4.2: Translation of ball in fluid domain

Ball at simulation Time 40

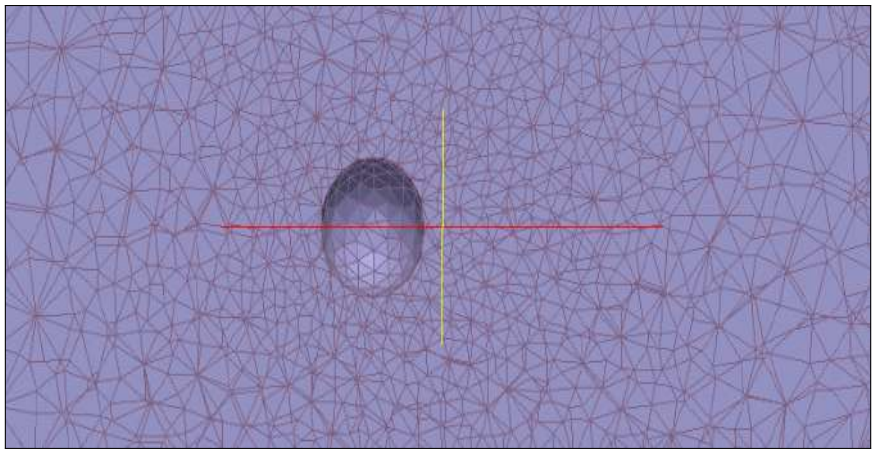

Figure 4.3: Translation of ball in fluid domain

Ball at simulation Time 60

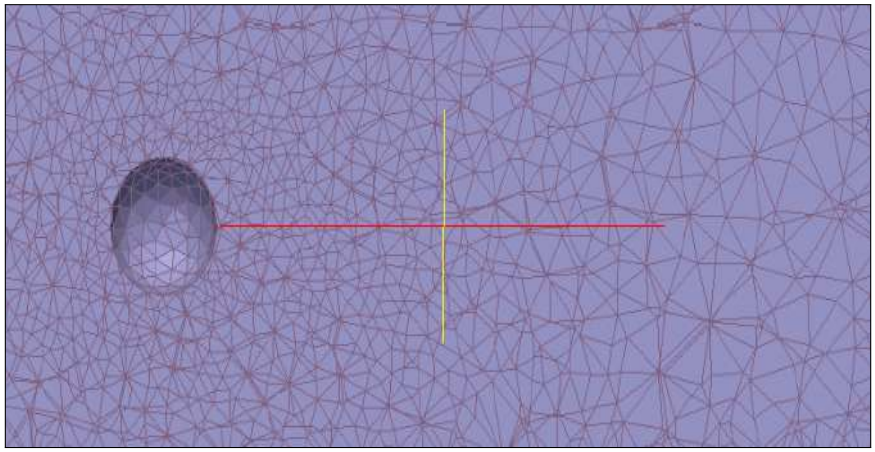

Figure 4.4: Translation of ball in fluid domain

4.1.2 Case2:Projectile case with pimpleDyMFoam Solver

This case setup involves the cubical fluid domain with small projectile placed inside. Unlike the ball translation case, it involved the fluid flow and change in a orientation of projectile is analyzed with pimpleDyMFoam solver. This case also involves implementation of 6-DOF motion solver. The solid body motion is handled by dynamicTopoFvMesh.

Computational Domain

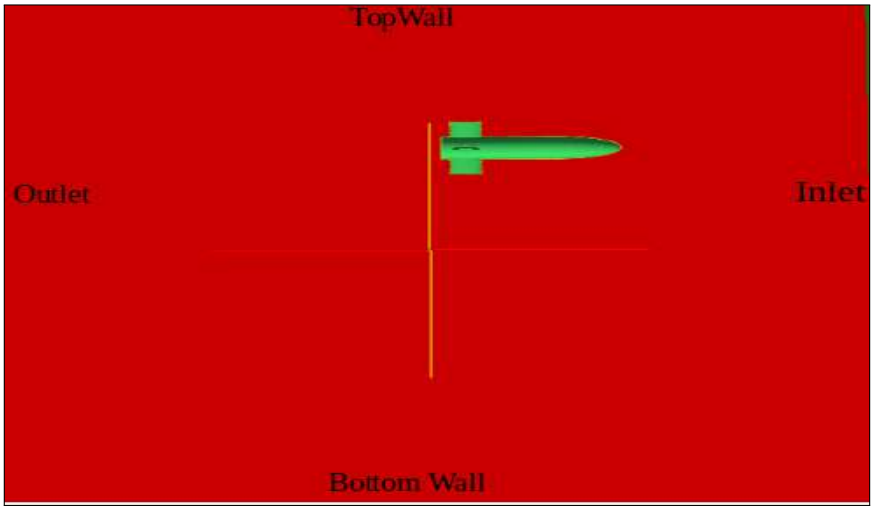

Figure 4.5: Computational Domain

Above figure shows computational domain for the projectile case. Boundary conditions are named as `inlet,Outlet,TopWall,BottomWall`. Projectile is an solid abject with the boundary conditions similar to wall boundary conditions. P,U boundary condition files are shown here which are kept on `0` directory in OpenFOAM.

Velocity: U

`Note:For the sake of making the document crisp the header files of U and p file are not shown here.User can refer any OpenFOAM tutorial case for getting those headers.`

```
boundaryField
{
    hull
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    inlet
    {
        type            fixedValue;
        value           uniform (350 0 0);
    }
    outlet
    {
        type            inletOutlet;
```

```
        inletValue      uniform (0 0 0);
        value           uniform (350 0 0);
    }
    sideWalls
    {
        type slip;
    }
}
```

Pressure :p

```
boundaryField
{
    hull
    {
        type    zeroGradient;
    }
    inlet
    {
        type    zeroGradient;
    }
    outlet
    {
        type    fixedValue;
        value   uniform 0.0;
    }
    sideWalls
    {
        type zeroGradient;
    }
}
```

Pre-Processing

Geometry Creation

FreeCAD is a opensource 3D CAD modeler which offers features similar to Catia,SolidWorks.The 3-D geometry shown in computational domain part is modelled in FreeCAD and is exported in stl format, which then can be imported in ICEM CFD software for generating the mesh.

Meshing

ANSYS ICEM CFD [10]meshing software starts with advanced CAD/geometry readers and repair tools to allow the user to quickly progress to a variety of geometry-tolerant meshers and produce high-quality volume or surface meshes with minimal effort.Advanced mesh diagnostics, interactive and automated mesh editing, output to a wide variety of computational fluid dynamics (CFD) and finite element analysis (FEA) solvers and multiphysics post-processing tools make ANSYS ICEM CFD a complete meshing solution. ANSYS endeavors to provide a variety of flexible tools that can take the model from any geometry to any solver in one modern and fully scriptable environment.

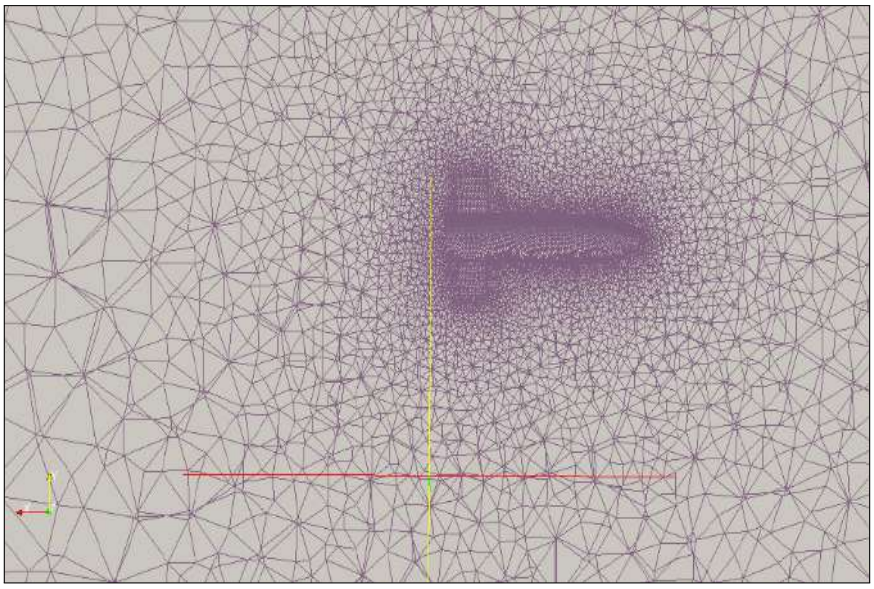

Figure 4.6: Mesh for Projectile case

Solver Setup

Importing Meshes

OpenFOAM can import meshes from a wide range of sources. Mesh is imported with the command `fluentto3Dfoam`. This command import the .msh file into OpenFOAM forming the polyMesh file containing the all the boundary, point, cell etc information into file format. Explored files contain the information about the geometry and it mesh including position of cell, nodes.

DynamicMeshDict Setup

As explained in `ballTranslation` case, correct setup of dynamicMeshDict is an crucial part of the simulation setup. The length scale parameters and local remeshing parameters may need to be changed as per the geometry requirement. The following dynamicMeshDict shows the setup done for the projectile case. This case also uses the 6-DOF motion solver to calculate the solid body motion at run time and setup for the 6-DOF is also shown in dynamicMeshDict File.The header file and some of the options of dynamicMeshDict are not included here as those are same as the one explained in `ballTranslation` case.

mesquiteOptions

```
mesquiteOptions
{

    usePointDisplacement     yes;
    // Optimization metric
    optMetric                AspectRatioGamma;
```

```
    // Objective function
    objFunction             LPtoP;

    // Optimization algorithm
    optAlgorithm            FeasibleNewton;

    // Termination criteria sub-dictionary (takes default values if not speci
    tcInner
    {
       iterationLimit       5;
    }

    sliverThreshold         0.35;

    // For composite functions, two objectives need to be specified
    firstFunction           LPtoP;
    secondFunction          LInf;

    // Power value for the LPtoP objective function
    pValue                  2;
    power                   2;

    // Specify a tolerance for the CG solver
    tolerance               1e-4;

    relaxationFactor 0.2;

    // Specify number of CG sweeps
    nSweeps                 1;

fixedValuePatches
  {
   hull
    {
        type            sixDoFRigidBodyDisplacement;
        mass            0.1;
        centreOfMass    (-0.187872  -0.0013468  0);
        momentOfInertia (0.1 0.1 0.1);
        orientation
        (
            0.9953705935 0.09611129781 0
           -0.09611129781 0.9953705935 0
            0 0 1
        );
        velocity        (0 0 0);
        acceleration    (0 0 0);
        angularMomentum (0 0 -0.5);
        torque          (0 0 0);
        rhoName         rhoInf;
```

```
    rhoInf          1;
    g               (0 -9.8 0);
    report          on;
    value           uniform (0 0 0);
    translationSpringConstant   (0 0 0);
    translationDampingConstant  (0 0 0);
    rotationSpringConstant      (0 0 0);
    rotationDampingConstant     (0 0 0);

constraints
    {
        maxIterations   5000;
        fixedLine1
        {
            sixDoFRigidBodyMotionConstraint fixedLine;
            tolerance               1e-9;
            relaxationFactor        0.7;
            fixedLineCoeffs
            {
                refPoint      (0.25 0.007 0.125);
                direction     (0 1 0);
            }
        }

        fixedAxis1
        {
            sixDoFRigidBodyMotionConstraint fixedAxis;
            tolerance               1e-06;
            relaxationFactor        0.7;
            fixedAxisCoeffs
            {
                axis            ( 0 0 1 );
            }
        }
    }
    restraints
    {
        verticalSpring
        {
            sixDoFRigidBodyMotionRestraint linearSpring;

            linearSpringCoeffs
            {
                anchor              (0.25 0.007 0.125);
                refAttachmentPt     (0.25 0.007 0.125);
                stiffness           1;
                damping           0.5;
                restLength    0;
            }
        }
```

```
            axialSpring
            {
              sixDoFRigidBodyMotionRestraint    linearAxialAngularSpring;

              linearAxialAngularSpringCoeffs
                {
                  axis              (0 0 1);
                  stiffness         1;
                  damping           0.5;
                  referenceOrientation $orientation;
                }
            }
        }
      value         uniform (0 0 0);
      }

   }
    // Specify interval for surface smoothing
    surfInterval    2;
}

//For the sake of making the document crisp,
//only the refinementOptions of dynamicTopoFvMesh are shown here,
//Remaining options are same as the one in ballTranslation case.

dynamicTopoFvMesh
{
        edgeRefinement      yes;

    refinementOptions
    {
        collapseRatio    0.5;
        bisectionRatio   1.6;
        growthFactor    1.05;

        fixedLengthScalePatches
        {
            hull        0.015;
        }

        freeLengthScalePatches
        {
            inlet;
            outlet;
            sideWalls;
        }
        maxLengthScale    0.266;
        minlengthScale    0.015;
        noModificationPatches
        {
```

```
            hull;
            inlet;
            outlet;
            sideWalls;
        }
    }
}
```

4.1.3 Simulation Results

Simulation carried over the solver `pimpleDyMFoam`, for the wide range of simulation to obtain the specified flow case is subjected to the different time step. Results shows the changes in a position of projectile with time and subsequent mesh Motion is handled by dynamicTopoFvMesh allowing the projectile to move in fluid domain. Results are shown at various time intervals.

The results of simulation are presented to serves the purpose of technology demonstration. The combination of 6-DOF motion solver,dynamic mesh with topology changes at run time opens the way to simulate the many engineering problems which involves the motion of solid body in fluid domain due to fluid forces.Which would otherwise be difficult to model. One of such possible application, sabot separation problem is introduced in chapter 5.

4.1.4 Velocity Field

Velocity profile and projectile orientation at simulation time 0

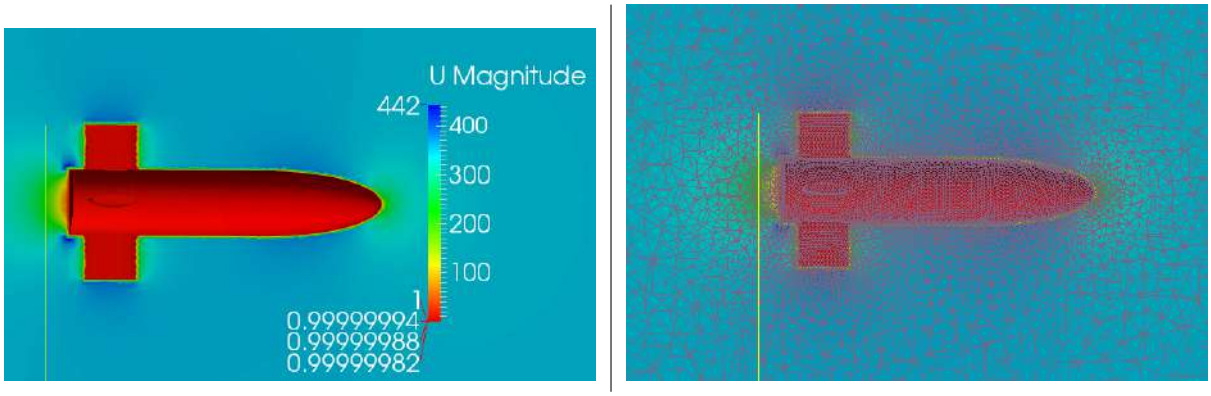

Figure 4.7: Velocity profile(left) and Mesh around projectile(Right)

Velocity profile and projectile orientation at simulation time 10

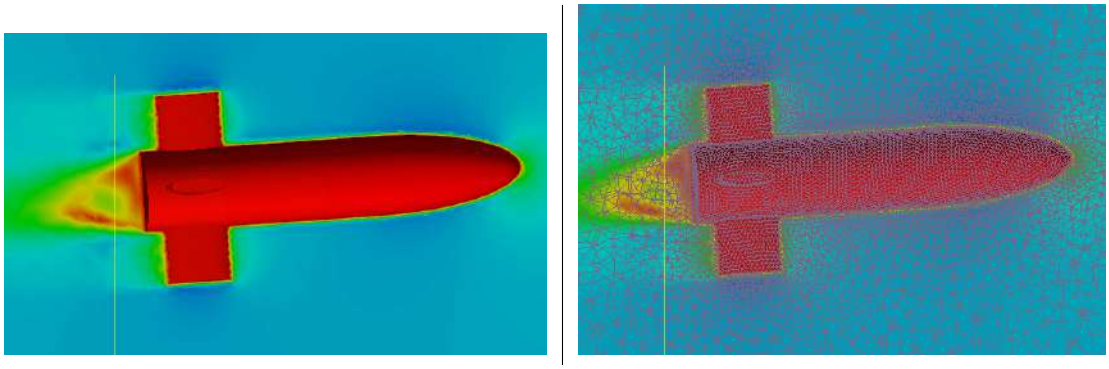

Figure 4.8: Velocity profile(left) and Mesh around projectile(Right)

Velocity profile and projectile orientation at simulation time 20

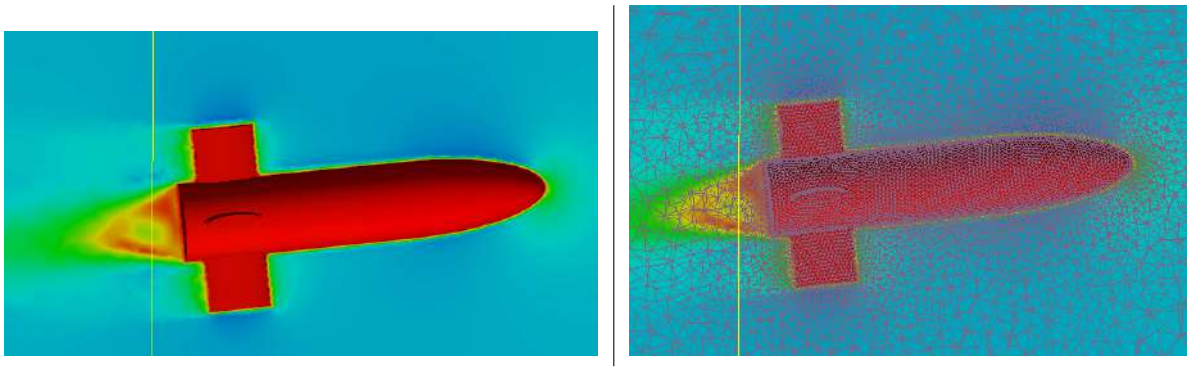

Figure 4.9: Velocity profile(left) and Mesh around projectile(Right)

4.1.5 Pressure Field

Pressure profile and projectile orientation at simulation time 10

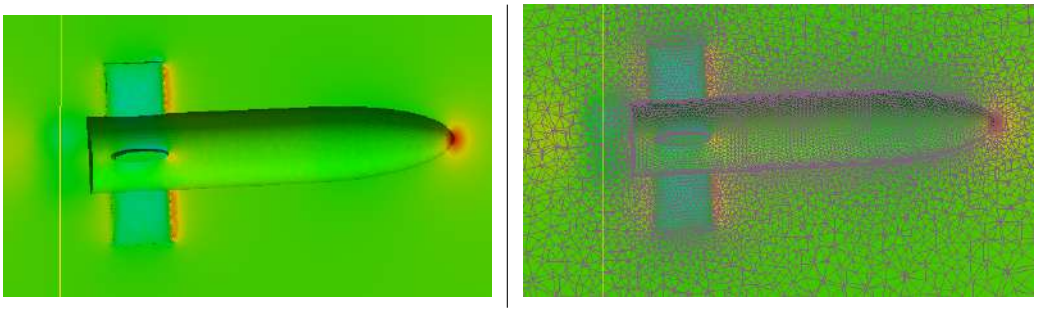

Figure 4.10: Pressure profile(left) and Mesh around projectile(Right)

Pressure profile and projectile orientation at simulation time 20

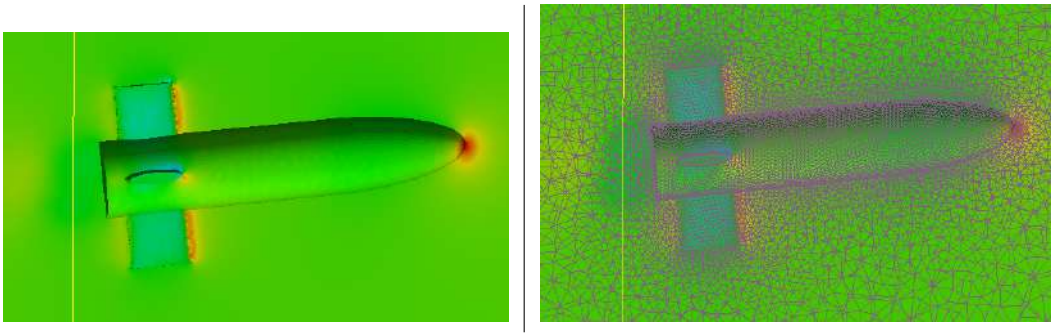

Figure 4.11: Pressure profile(left) and Mesh around projectile(Right)

Chapter 5

Introduction to Sabot separation problem for future applications

5.1 Armour-piercing discarding sabot(APDS)

Armour-piercing discarding sabot[13] is a type of kinetic energy projectile fired from a gun to attack armoured targets. APDS rounds are sabot rounds and were commonly used in large calibre tank guns, but have now been superseded by `armour-piercing fin stabilized discarding sabot`(APFSDS) projectiles in such guns.

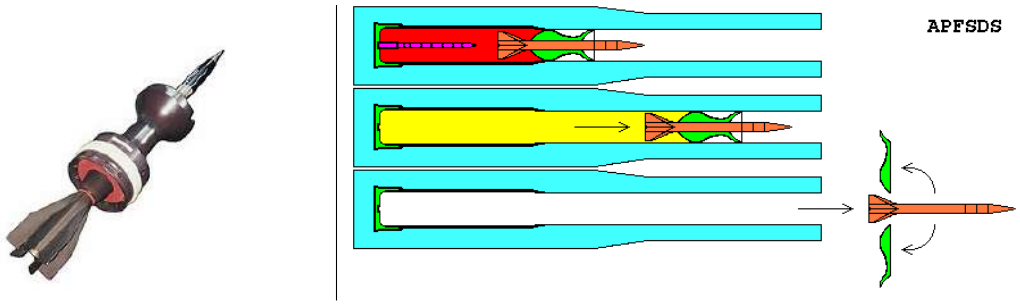

Figure 5.1: Anti-tank round with its sabot(left).A diagram of a fin stabilised discarding sabot showing its operation.(Right) (source [12])

5.1.1 History and development

APDS was developed by engineers working for the French Edgar Brandt company, and was fielded in two calibers(75mm/57mm for the Mle1897/33 75mm anti-tank cannon, 37mm/25mm for several 37mm gun types) just before the French-German armistice of 1940.The Edgar Brandt engineers, having been evacuated to the United Kingdom, joined ongoing APDS development efforts there, culminating in significant improvements to the concept and its realisation.The APDS projectile type was further developed in the United Kingdom between 19411944 by Permutter and Coppock, two designers with the Armaments Research Department. In mid-1944 the APDS projectile was first introduced into service for the UK's QF 6 pounder anti-tank gun and later in September 1944 for the QF 17 pounder anti-tank gun.

The reason for the development of the APDS was the search for anti-tank projectiles with increased penetrating performance. It was known that high impact (terminal) velocity, or a larger diameter projectile would be required to improve penetration. A larger projectile would require a completely new weapon system and may have been too heavy to retrofit onto existing

armoured fighting vehicles. Increasing the velocity of the current projectiles was also a problem due to the impact velocity limitations of steel armour-piercing (AP) projectiles, which would shatter at velocities above 850m/s when uncapped.

To allow increased impact velocity, a stronger penetrator material was required. The chosen new penetrator material was tungsten carbide (WC), due to its greater hardness and its ability to withstand the greater shock and pressure generated during a higher velocity impact. As the density of WC (15.7 g/cm) is twice that of steel (7.86 g/cm), such a shot was too heavy at full bore to be accelerated to a sufficient muzzle velocity. To overcome this, a lightweight full diameter carrier shell was developed to sheathe the inner high density core. The name given to this projectile type was the Armour-Piercing Composite Rigid (APCR). The APCR projectile was about half the weight of a standard AP shot, but of the same diameter. Due to the large surface area for the gases to impinge upon the lightweight APCR projectile, it experienced a higher average acceleration in the gun barrel, in turn imparting a higher muzzle velocity. Unfortunately the low sectional density of the APCR resulted in poor carrying power (high aerodynamic drag), losing velocity and penetration rapidly over distance.To overcome these limitations the British devised a way for the outer sheath to be discarded after leaving the bore. The name given to the discarded outer sheath was the sabot (a French word for a wooden shoe). For APDS projectiles the sabot is also known as a pot, as the sabot resembles a flower pot in shape. The APDS has the advantages of the lightweight projectile with regards to bore acceleration and high muzzle velocity, but does not suffer from the high drag of the APCR in flight.

5.1.2 Sabot construction

The sabot of a large calibre APDS consists of a light high strength alloy full diameter pot and base unit, which is screwed together. The front part of the pot has 3-4 petals (sabots) which are covered with a centering band (often a nylon derivative). The rear half has a rubber obturator and driving band (again nylon) held in place by the screw-in base unit. The base unit, if a tracer element is attached to the sub-projectile, has a hole located at the centre. Before firing, the sub-projectile and sabot are locked together. Due to the high setback forces, friction between the pot and sub-projectile allows spin to be transferred, so stabilizing the sub-projectile. Small/medium calibre APDS use a lightweight high strength alloy base pot and three or more plastic petals. To transfer the spin to the core in small/medium calibre weapons, the core tends to have a notch at its base. Under bore acceleration, which can be higher than 100,000 g, the uneven base is forced into the softer pot material, locking the sub-projectile to the pot and imparting spin. Not all small/medium calibre APDS rely on this technique, another method for spin coupling is by using the forward plastic petals. The petals are of a slightly larger diameter than the lands in the rifled bore. This forces the petals tightly against the core, increasing the friction between them and allowing the spin to be transferred[13].

5.1.3 Future Work: Need of CFD Analysis

The kinetic energy penetrator is a widely used anti-tank munitions. Its lethality is due to the kinetic energy imparted by the penetrator to the target with impact velocities of between 1.4 to 1.8 km/s. The penetrator is launched by means of a sabot assembly, consisting of three aluminium sabot petals, required to minimize the penetrators in bore balloting. The process of sabot separation begins as the projectile leaves the gun tube. Because of the transverse motion of the projectile within the gun, energy is stored in the elastic sabot petals. As the projectile leaves the muzzle, the constraints of the gun tube are released and the sabot elements are able to move laterally outward.

Aerodynamic forces acting on each of the sabot petals causes them to lift up and disengage from the buttress grooves of the penetrator rod to permit its unconstrained, low drag flight to the target. It has been demonstrated by Schmidt and Shear[14] that aerodynamic interference generated by the sabot components can be a significant source of projectile launch disturbance leading to unacceptable loss of accuracy at the target. Perturbations to the projectile trajectory are magnified by geometric asymmetry in the discard pattern and by the extended periods during launch when the sabot components are in close proximity to the projectile.

Hence analysis of discarding pattern of sabot petals becomes most critical parameter in deciding the accuracy of projectile. As aerodynamic forces plays important role in the separation of sabot petals, CFD analysis can be brought in to better understand the separation process. As this process includes solid body motion due to the aerodynamic forces, it is clear that CFD simulation will need dynamicMesh to achieve effective CFD analysis.

Also the solid body motion (separation of petals) involved in the problem is a result of aerodynamic forces generated/calculated during run time, it is quickly understood that there will be a need of a dynamicMesh which will have a capability to adjust the mesh motion and allow the topological changes in mesh whenever required. This leads to the requirement of DynamicMesh with topological changes and adaptive mesh reconnection. Moreover it is clear that the dynamicMesh's like `Mesh motion` or `General grid interface(GGI)` will not serve the purpose of this problem. Hence the dynamicMesh called `dynamicTopoFvMesh` found to be a best option to model the sebot separation problem because of its notable features like adaptive Mesh reconnection, local mesh refinement and mesh quality optimization algorithms which maintains the mesh quality throughout the simulation.

Bibliography

[1] https://en.wikipedia.org/wiki/OpenFOAM

[2] Master Thesis by Benjamin Wuthrich-Simulation and validation of compressible flow in nozzle geometries and validation of OpenFOAM for this application

[3] http://openfoamwiki.net/index.php/Main_Page

[4] OpenCFD. 2007a (Apr.). OpenFOAM: The Open Source CFD Toolbox. Programmers Guide Version 3.0.1 OpenCFD Limited

[5] Tomislav Maric,Jens Hopken,Kyle Mooney-The OpenFOAM technology primer-Sourceflux UG,first edition

[6] Lawson, C. L. Software for C1 surface interpolation. Mathematical Software III (1977), 161194.

[7] Sandeep Menon. A numerical study of droplet formation and behavior using interface tracking methods. PhD thesis, University of Massachusetts Amherst, 2011.

[8] Beaudoin M. and Jasak H., "Development of a Generalized Grid Interface for Turbomachinery simulations with OpenFOAM", Open Source CFD International Conference 2008.

[9] M. Brewer, L. Diachin, P. Knupp, T. Leurent, and D. Melander. The Mesquite Mesh Quality Improvement Toolkit. In 12th International Meshing Roundtable, Sandia National Laboratories report SAND 2003-3030P, Sept. 2003, 2003.

[10] https://www.academia.edu/3196227/ANSYS_ICEM_CFD_14_Tutorial_Manual

[11] https://sourceforge.net/projects/

[12] https://en.wikipedia.org/wiki/Sabot

[13] https://en.wikipedia.org/wiki/Armour-piercing_discarding_sabot

[14] Schmidt E.M., Shear D.D., Aerodynamic Interface During Sabot Discard,Journal of Spacecraft and Rockets, AIAA, Vol 15, No 3, May-June, 1978,pp. 235-240.

[15] http://cfd.at/downloads/2014_OFoam_Tut_Complete.pdf