OpenFOAM Workshop

Daniele Trimarchi
daniele.trimarchi@soton.ac.uk
University of Southampton, 01-11-2010
Table of contents:
Table of contents:

- OpenFOAM, UNIX AND THE MAC
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
- UNSTEADY RANSE
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
- UNSTEADY RANSE
  - Imposing unsteady BC using ramp files
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
- UNSTEADY RANSE
  - Imposing unsteady BC using ramp files
  - forces, coefficients and samples extraction
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
- UNSTEADY RANSE
  - Imposing unsteady BC using ramp files
  - forces, coefficients and samples extraction
  - Airfoil case study: the pisoFOAM solver, SST turb. model & unsteady BC
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
- UNSTEADY RANSE
  - Imposing unsteady BC using ramp files
  - forces, coefficients and samples extraction
  - Airfoil case study: the pisoFOAM solver, SST turb. model & unsteady BC
- DYNAMIC MESH HANDLING
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
- UNSTEADY RANSE
  - Imposing unsteady BC using ramp files
  - forces, coefficients and samples extraction
  - Airfoil case study: the pisoFOAM solver, SST turb. model & unsteady BC
- DYNAMIC MESH HANDLING
  - Principles of mesh motion: ALE => the PimpleDyMFOAM solver, and the velocityLaplacianMotionSolver
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
- UNSTEADY RANSE
  - Imposing unsteady BC using ramp files
  - forces, coefficients and samples extraction
  - Airfoil case study: the pisoFOAM solver, SST turb. model & unsteady BC
- DYNAMIC MESH HANDLING
  - Principles of mesh motion: ALE => the PimpleDyMFOAM solver, and the velocityLaplacianMotionSolver
  - tutorial case study: the moving beam
<table>
<thead>
<tr>
<th>Section</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenFOAM, UNIX AND THE MAC</td>
<td></td>
</tr>
<tr>
<td>MESH GENERATION</td>
<td></td>
</tr>
<tr>
<td>presentation of the routine airfoil.exe</td>
<td></td>
</tr>
<tr>
<td>GMSH: tutorial on the meshing of a simple geometry</td>
<td></td>
</tr>
<tr>
<td>GMSH conversion: the utility gmshToFOAM and CreateBaffles</td>
<td></td>
</tr>
<tr>
<td>UNSTEADY RANSE</td>
<td></td>
</tr>
<tr>
<td>Imposing unsteady BC using ramp files</td>
<td></td>
</tr>
<tr>
<td>forces, coefficients and samples extraction</td>
<td></td>
</tr>
<tr>
<td>Airfoil case study: the pisoFOAM solver, SST turb. model &amp; unsteady BC</td>
<td></td>
</tr>
<tr>
<td>DYNAMIC MESH HANDLING</td>
<td></td>
</tr>
<tr>
<td>Principles of mesh motion: ALE =&gt; the PimpleDyMFOAM solver, and the velocityLaplacianMotionSolver</td>
<td></td>
</tr>
<tr>
<td>tutorial case study: the moving beam</td>
<td></td>
</tr>
<tr>
<td>WRITING AND COMPILING USER APPLICATIONS AND LIBRARIES</td>
<td></td>
</tr>
</tbody>
</table>
Table of contents:

- OpenFOAM, UNIX AND THE MAC
- MESH GENERATION
  - presentation of the routine airfoil.exe
  - GMSH: tutorial on the meshing of a simple geometry
  - GMSH conversion: the utility gmshToFOAM and CreateBaffles
- UNSTEADY RANSE
  - Imposing unsteady BC using ramp files
  - Forces, coefficients and samples extraction
  - Airfoil case study: the pisoFOAM solver, SST turb. model & unsteady BC
- DYNAMIC MESH HANDLING
  - Principles of mesh motion: ALE => the PimpleDyMFOAM solver, and the velocityLaplacianMotionSolver
  - Tutorial case study: the moving beam
- WRITING AND COMPILING USER APPLICATIONS AND LIBRARIES
- READING IN THE CODE: the library ‘Force.C’
<table>
<thead>
<tr>
<th>Table of contents:</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenFOAM, UNIX AND THE MAC</td>
</tr>
<tr>
<td>MESH GENERATION</td>
</tr>
<tr>
<td>presentation of the routine airfoil.exe</td>
</tr>
<tr>
<td>GMSH: tutorial on the meshing of a simple geometry</td>
</tr>
<tr>
<td>GMSH conversion: the utility gmshToFOAM and CreateBaffles</td>
</tr>
<tr>
<td>UNSTEADY RANSE</td>
</tr>
<tr>
<td>Imposing unsteady BC using ramp files</td>
</tr>
<tr>
<td>forces, coefficients and samples extraction</td>
</tr>
<tr>
<td>Airfoil case study: the pisoFOAM solver, SST turb. model &amp; unsteady BC</td>
</tr>
<tr>
<td>DYNAMIC MESH HANDLING</td>
</tr>
<tr>
<td>Principles of mesh motion: ALE =&gt; the PimpleDyMFOAM solver, and the velocityLaplacianMotionSolver</td>
</tr>
<tr>
<td>tutorial case study: the moving beam</td>
</tr>
<tr>
<td>WRITING AND COMPILING USER APPLICATIONS AND LIBRARIES</td>
</tr>
<tr>
<td>READING IN THE CODE: the library ‘Force.C’</td>
</tr>
<tr>
<td>PATCH DEFORMATIONS: a modified version of PimpleDyMFOAM for FSI</td>
</tr>
</tbody>
</table>
OpenFOAM and the MAC
OpenFOAM and the MAC

- MAC core is UNIX, therefore it is possible to build OF
OpenFOAM and the MAC

- MAC core is UNIX, therefore it is possible to build OF
- It is however less easy than in Ubuntu....
OpenFOAM and the MAC

- MAC core is UNIX, therefore it is possible to build OF.
- It is however less easy than in Ubuntu....
- Very good guidance in the forum:
  - [http://www.cfd-online.com/Forums/openfoam-installation/77570-patches-openfoam-1-7-macos-x.html](http://www.cfd-online.com/Forums/openfoam-installation/77570-patches-openfoam-1-7-macos-x.html)
OpenFOAM and the MAC

- MAC core is UNIX, therefore it is possible to build OF.
- It is however less easy than in Ubuntu....
- Very good guidance in the forum:
  - [http://www.cfd-online.com/Forums/openfoam-installation/77570-patches-openfoam-1-7-macos-x.html](http://www.cfd-online.com/Forums/openfoam-installation/77570-patches-openfoam-1-7-macos-x.html)
- You need Xode
OpenFOAM and the MAC

- MAC core is UNIX, therefore it is possible to build OF.
- It is however less easy than in Ubuntu....
- Very good guidance in the forum:
  - http://www.cfd-online.com/Forums/openfoam-installation/77570-patches-openfoam-1-7-macos-x.html
- You need Xode
- And macPorts is a very confortable tool..!
  - www.macports.org
Unix: the use of the terminal
Unix: the use of the terminal
Unix: the use of the terminal

- `ls`
Unix: the use of the terminal

- ls
- cd
Unix: the use of the terminal

- `ls`
- `cd`
- `pwd`
Unix: the use of the terminal

- `ls`
- `cd`
- `pwd`
- `mkdir`
Unix: the use of the terminal

- `ls`
- `cd`
- `pwd`
- `mkdir`
- `>`
Unix: the use of the terminal

- ls
- cd
- pwd
- mkdir
- >
- man
Unix: the use of the terminal

- `ls`
- `cd`
- `pwd`
- `mkdir`
- `>`
- `man`
- `rm / rm -r` [!!!]
Unix: the use of the terminal

- ls
- cd
- pwd
- mkdir
- >
- man
- rm / rm -r  [!!!]
- the .bash files
Unix: the use of the terminal

- `ls`
- `cd`
- `pwd`
- `mkdir`
- `>`
- `man`
- `rm / rm -r` [!!!]
- the `.bash` files
- `make ➔ wmake`
Unix: the use of the terminal

- `ls`
- `cd`
- `pwd`
- `mkdir`
- `>`
- `man`
- `rm / rm -r` ![Warning]
- The `.bash` files
- `make ➔ wmake`

http://info.ee.surrey.ac.uk/Teaching/Unix/
Airfoil.exe

- Fortran program for creating a BlockMesh input file
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types (http://www.ae.illinois.edu/m-selig/ads.html)
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types ([http://www.ae.illinois.edu/m-selig/ads.html](http://www.ae.illinois.edu/m-selig/ads.html))
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types (http://www.ae.illinois.edu/m-selig/ads.html)
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types ([http://www.ae.illinois.edu/m-selig/ads.html](http://www.ae.illinois.edu/m-selig/ads.html))
- 2 Input files: airfoil.data ; input.data
**Airfoil.exe**

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types ([http://www.ae.illinois.edu/m-selig/ads.html](http://www.ae.illinois.edu/m-selig/ads.html))
- 2 Input files: airfoil.data ; input.data
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types (http://www.ae.illinois.edu/m-selig/ads.html)
- 2 Input files: airfoil.data ; input.data
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types (http://www.ae.illinois.edu/m-selig/ads.html)
- 2 Input files: airfoil.data ; input.data
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types ([http://www.ae.illinois.edu/m-selig/ads.html](http://www.ae.illinois.edu/m-selig/ads.html))
- 2 Input files: airfoil.data ; input.data
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types (http://www.ae.illinois.edu/m-selig/ads.html)
- 2 Input files: airfoil.data ; input.data
- compile the program: Makefile
**Airfoil.exe**

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types ([http://www.ae.illinois.edu/m-selig/ads.html](http://www.ae.illinois.edu/m-selig/ads.html))
- 2 Input files: airfoil.data ; input.data
- compile the program: Makefile

```plaintext
FFLAGS= -O
.DEFAULT: .f
.SUFFIXES: .f

.f.o: $*.f
    gfortran -c $(FFLAGS) $*.f

#-- source file(s) name
OBJ=$(Airfoil.o max.o mid_A.o quA.o mid_B.o quB.o cGrid.o ruot.o \    inv.o inv2.o simm.o mirror.o

airfoil: $(OBJ)
gfortran $(FFLAGS) $(OBJ) -o airfoil.exe
```
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types (http://www.ae.illinois.edu/m-selig/ads.html)
- 2 Input files: airfoil.data ; input.data
- compile the program: Makefile
Airfoil.exe

- Fortran program for creating a BlockMesh input file
- Geometry: 2D airfoil types ([http://www.ae.illinois.edu/m-selig/ads.html](http://www.ae.illinois.edu/m-selig/ads.html))
- 2 Input files: airfoil.data; input.data
- Compile the program: Makefile

Gmsh

- OpenSource meshing program with GUI and scripting language (for Mac, Unix, Win)
- Pro: Easy to use, cross plattform
- Cons: difficult for complex geometries, and no hybrid meshes (..?)
- Online: http://geuz.org/gmsh/  =>  gmsh2.4
Gmsh: basic scripting commands
Gmsh: basic scripting commands

- Point(0) = \{x, y, z\};
Gmsh: basic scripting commands

- Point(0) = \{x, y, z\};
- Line(0) = \{0, 1\};
Gmsh: basic scripting commands

- Point(0) = \{x, y, z\};
- Line(0) = \{0, 1\};
- Transfinite Line \{0\} = N Using Progression P;
Gmsh: basic scripting commands

- Point(0) = {x, y, z};
- Line(0) = {0, 1};
- Transfinite Line {0} = N Using Progression P;
- Line Loop(2) = {0, 1, -2, -3};
- Plane Surface(1) = {2};
Gmsh: basic scripting commands

- `Point(0) = {x, y, z};`
- `Line(0) = {0, 1};`
- `Transfinite Line {0} = N Using Progression P;`
- `Line Loop(2) = {0, 1, -2, -3};`
- `Plane Surface(1) = {2};`
Gmsh: basic scripting commands

- \text{Point}(0) = \{x, y, z\};
- \text{Line}(0) = \{0, 1\};
- \text{Transfinite Line} \{0\} = N \text{ Using Progression P};
- \text{Line Loop}(2) = \{0, 1, -2, -3\};
  \text{Plane Surface}(1) = \{2\};
- \text{Transfinite Surface} \{1\};
Gmsh: basic scripting commands

- Point(0) = \{x, y, z\};
- Line(0) = \{0, 1\};
- Transfinite Line \{0\} = N Using Progression P;
- Line Loop(2) = \{0, 1, -2, -3\};
  Plane Surface(1) = \{2\};
- Transfinite Surface \{1\};
Gmsh: basic scripting commands

- Point(0) = {x, y, z};
- Line(0) = {0, 1};
- Transfinite Line {0} = N Using Progression P;
- Line Loop(2) = {0, 1, -2, -3};
  Plane Surface(1) = {2};
- Transfinite Surface {1};
- Recombine Surface {1};
Gmsh: basic scripting commands

- `Point(0) = {x, y, z};`
- `Line(0) = {0, 1};`
- `Transfinite Line {0} = N Using Progression P;`
- `Line Loop(2) = {0, 1, -2, -3};`
  Plane Surface(1) = {2};
- `Transfinite Surface {1};`
- `Recombine Surface {1};`
Gmsh: basic scripting commands

- `Point(0) = {x, y, z};`
- `Line(0) = {0, 1};`
- `Transfinite Line {0} = N Using Progression P;`
- `Line Loop(2) = {0, 1, -2, -3};`
  Plane Surface(1) = {2};
- `Transfinite Surface {1};`
- `Recombine Surface {1};`
- `Extrude {0, 0, H}
  {
    Surface{1};
    Layers{{n1,n2,n3,n4},{.25,.5,.75,1}};
    Recombine;
  }`
Gmsh: basic scripting commands

- Point(0) = {x, y, z};
- Line(0) = {0, 1};
- Transfinite Line {0} = N Using Progression P;
- Line Loop(2) = {0, 1, -2, -3};
  Plane Surface(1) = {2};
- Transfinite Surface {1};
- Recombine Surface {1};
- Extrude {0, 0, H}
  {
    Surface{1};
    Layers{{n1,n2,n3,n4},{.25,.5,.75,1}};
    Recombine;
  }
- Physical Surface("Inlet") = {1};
Gmsh: basic scripting commands

- Point(0) = {x, y, z};
- Line(0) = {0, 1};
- Transfinite Line {0} = N Using Progression P;
- Line Loop(2) = {0, 1, -2, -3};
  Plane Surface(1) = {2};
- Transfinite Surface {1};
- Recombine Surface {1};
- Extrude {0, 0, H}
  {
    Surface{1};
    Layers{{n1,n2,n3,n4}, {.25,.5,.75,1}};
    Recombine;
  }
- Physical Surface("Inlet") = {1};
- Physical Volume("AIR") = {1,2, 3, 4};
gmshToFoam
gmshToFoam

- Once created in .msh format, the mesh has to be converted for OpenFOAM
gmshToFoam

- Once created in .msh format, the mesh has to be converted for OpenFOAM
- Create directory System, which contains:
gmshToFoam

- Once created in .msh format, the mesh has to be converted for OpenFOAM
- Create directory System, which contains:
  - ‘ControlDict’, ‘FvSchemes’ and ‘Fvsolution’
Once created in .msh format, the mesh has to be converted for OpenFOAM

Create directory System, which contains:

- ‘ControlDict’, ‘FvSchemes’ and ‘Fvsolution’

Type in terminal: `gmshToFoam MyMesh.msh`
CreateBaffles
CreateBaffles

- Example of the 2D spinnaker section: download the file
  (http://www-roc.inria.fr/MACS/spip.php?rubrique69)
CreateBaffles

- Example of the 2D spinnaker section: download the file (http://www-roc.inria.fr/MACS/spip.php?rubrique69)
- Create dirs and the files; run gmshToFoam
CreateBaffles

- Example of the 2D spinnaker section: download the file (http://www-roc.inria.fr/MACS/spip.php?rubrique69)
- Create dirs and the files; run gmshToFoam
- look at the output in terminal:
CreateBaffles

- Example of the 2D spinnaker section: download the file (http://www-roc.inria.fr/MACS/spip.php?rubrique69)
- Create dirs and the files; run gmshToFoam
- look at the output in terminal:
  .....  
  Patch 0 gets name walls  
  Patch 1 gets name outlet  
  Patch 2 gets name wing  
  Patch 3 gets name inlet  
  .....  
  Writing zone 0 to cellZone Air and cellSet  
  .....  
  Writing zone 2 to faceZone faceZone_2 and faceSet
CreateBaffles

- Create dirs and the files; run `gmshToFoam`
- Look at the output in terminal:
  
  ```
  .....  
  Patch 0 gets name walls  
  Patch 1 gets name outlet  
  Patch 2 gets name wing  
  Patch 3 gets name inlet  
  .....  
  Writing zone 0 to cellZone Air and cellSet  
  .....  
  Writing zone 2 to faceZone faceZone_2 and faceSet  
  ```
- The sail surface, called (in gmsh) ‘wing’ has the facezone label ‘2’ assigned.
CreateBaffles

- Example of the 2D spinnaker section: download the file (http://www-roc.inria.fr/MACS/spip.php?rubrique69)
- Create dirs and the files; run gmshToFoam
- look at the output in terminal:
  .....  
  Patch 0 gets name walls  
  Patch 1 gets name outlet  
  Patch 2 gets name wing  
  Patch 3 gets name inlet  
  .....  
  Writing zone 0 to cellZone Air and cellSet  
  .....  
  Writing zone 2 to faceZone faceZone_2 and faceSet  
- The sail surface, called (in gmsh) ‘wing’ has the facezone label ‘2’ assigned.
- type in terminal:  createBaffles faceZone_2 wing
...and finally the end...
...and finally the end...

- The new mesh is written in a new directory, called as the time-step in the ControlDict (es: 0.02)
...and finally the end...

- The new mesh is written in a new directory, called as the time-step in the ControlDict (es: 0.02)
- Still a modification has to be done in constant/polymesh
...and finally the end...

- The new mesh is written in a new directory, called as the time-step in the ControlDict (es: 0.02)
- Still a modification has to be done in `constant/polymesh`
- All surfaces are now patches, therefore walls and empty entries are to be substituted:
...and finally the end...

- The new mesh is written in a new directory, called as the time-step in the ControlDict (es: 0.02)
- Still a modification has to be done in constant/polymesh
- All surfaces are now patches, therefore walls and empty entries are to be substituted:
Unsteady BC with ramp files
Unsteady BC with ramp files

- Very easy strategy for imposing variable boundary conditions
  Ex: variation in inlet velocity or body motion
Unsteady BC with ramp files

- Very easy strategy for imposing variable boundary conditions
  Ex: variation in inlet velocity or body motion
- Use a ‘ramp’ file for describing the variation in time
Unsteady BC with ramp files

- Very easy strategy for imposing variable boundary conditions
  Ex: variation in inlet velocity or body motion
- Use a ‘ramp’ file for describing the variation in time

Ramp

( 0.0 ( 0 0 0 ) )
( 0.015 ( .5 0 0 ) )
( 0.025 ( 0 0 0 ) )
( 0.05 ( -1 0 0 ) )
Unsteady BC with ramp files

- Very easy strategy for imposing variable boundary conditions
  Ex: variation in inlet velocity or body motion
- Use a ‘ramp’ file for describing the variation in time

```plaintext
Ramp
(
  (0.0 (0 0 0))
  (0.015 (.5 0 0))
  (0.025 (0 0 0))
  (0.05 (-1 0 0))
)
```

```plaintext
U (cellMotionU, PointMotionU.....)
boundaryField
{
  inlet
  {
    type
timeVaryingUniformFixedValue;
    fileName "ramp"
    outOfBounds repeat;
  }
  ...
  ...
}
```
Force extraction
Force extraction

- Add in the `system/controlDict` file the entry for force extraction
Force extraction

- Add in the `system/controlDict` file the entry for force extraction
- Forces can be extracted with ‘libforce’, lift and drag coeffs with ‘libforceCoeffs’
Force extraction

- Add in the `system/controlDict` file the entry for force extraction
- Forces can be extracted with ‘libforce’, lift and drag coeffs with ‘libforceCoeffs’

```plaintext
controlDict
....
forces
{
  type forces;
  functionObjectLibs ("libforces.dylib"); // .dylib on Mac and .so on Linux
  outputControl outputTime;
patches (wing); // Name of patche to integrate forces
  pName p;
  Uname U;
rhoName rhoInf;
rhoInf 1.2; // Reference density for fluid
  pRef 0;
  CofR (0 0 0); // Origin for moment calculations
}
```
Sampling

- Add in workdir/system a file called sampleDict
Sampling

- Add in workdir/system a file called sampleDict
- Samples can be extracted during the execution, or post-execution, typing; ‘sample’
Sampling

- Add in workdir/system a file called sampleDict
- Samples can be extracted during the execution, or post-execution, typing; ‘sample’

```plaintext
sampleDict
interpolationScheme cellPoint;
setFormat       raw;
sets
{
  MySample //Filename
  {
    type    uniform;
    axis    xyz;
    start   ( 3.8 1.5 0.005 );
    end     ( 4 1.5 0.005 );
    nPoints 3;
  }
}
surfaces   ()
fields     ( U );
```
Sampling

- Add in workdir/system a file called sampleDict
- Samples can be extracted during the execution, or post-execution, typing; ‘sample’

```plaintext
sampleDict
interpolationScheme cellPoint;
setFormat raw;
sets
{
    MySample //Filename
    {
        type uniform;
        axis xyz;
        start ( 3.8 1.5 0.005 );
        end   ( 4 1.5 0.005 );
        nPoints 3;
    }
}
);
surfaces()
fields ( U );
```
Sampling

- Add in workdir/system a file called sampleDict
- Samples can be extracted during the execution, or post-execution, typing; ‘sample’

```plaintext
sampleDict
interpolationScheme cellPoint;
setFormat raw;
sets
{
    MySample //Filename
    {
        type uniform;
        axis xyz;
        start ( 3.8 1.5 0.005 );
        end ( 4 1.5 0.005 );
        nPoints 3;
    }
}
);
surfaces ();
fields ( U );
```
Sampling

- Add in workdir/system a file called sampleDict
- Samples can be extracted during the execution, or post-execution, typing; ‘sample’

```plaintext
sampleDict
interpolationScheme cellPoint;
setFormat raw;
sets
{
    MySample //Filename
    {
        type uniform;
        axis xyz;
        start ( 3.8 1.5 0.005 );
        end ( 4 1.5 0.005 );
        nPoints 3;
    }
}

surfaces
{
    ()
}
fields ( U );
```

www.openfoam.com/docs/user
Sampling

- Add in workdir/system a file called sampleDict
- Samples can be extracted during the execution, or post-execution, typing; ‘sample’

```
sampleDict
interpolationScheme cellPoint;
setFormat    raw;
sets
{
    MySample  //Filename
    {
        type    uniform;
        axis    xyz;
        start   ( 3.8 1.5 0.005 );
        end     ( 4 1.5 0.005 );
        nPoints 3;
    }
}
); surfaces    ();
fields         ( U );
```

www.openfoam.com/docs/user
Sampling

- Add in workdir/system a file called sampleDict
- Samples can be extracted during the execution, or post-execution, typing; ‘sample’

```plaintext
sampleDict
interpolationScheme cellPoint;
setFormat    raw;
sets
{
    MySample   //Filename
    {
        type    uniform;
        axis    xyz;
        start   ( 3.8 1.5 0.005 );
        end     ( 4 1.5 0.005 );
        nPoints 3;
    }
};
surfaces    ()
fields       ( U );
```
Sampling

- Add in workdir/system a file called sampleDict
- Samples can be extracted during the execution, or post-execution, typing; ‘sample’

```
sampleDict
interpolationScheme cellPoint;
setFormat raw;
sets
{
    MySample //Filename
    {
        type uniform;
        axis xyz;
        start ( 3.8 1.5 0.005 );
        end ( 4 1.5 0.005 );
        nPoints 3;
    }
}
);
surfaces ();
fields ( U );
```
Airfoil case study:
pisoFOAM, SST turbulence model & unsteady BC
Airfoil case study:
pisoFOAM, SST turbulence model & unsteady BC

- pisoFOAM: unsteady solver of the SIMPLE family (Semi-Implicit Method for Pressure Linked Equations) => iterative method:
Airfoil case study:
pisoFOAM, SST turbulence model & unsteady BC

- pisoFOAM: unsteady solver of the SIMPLE family (Semi-Implicit Method for Pressure Linked Equations) => iterative method:
  
  guess $p^*$ and solve $u^*$
Airfoil case study:
pisoFOAM, SST turbulence model & unsteady BC

- pisoFOAM: unsteady solver of the SIMPLE family (Semi-Implicit Method for Pressure Linked Equations) => iterative method:

  guess $p^*$ and solve $u^*$

  find correction for $p^*$ (by mean of $\nabla \cdot u^* = 0$)
Airfoil case study:
pisoFOAM, SST turbulence model & unsteady BC

- pisoFOAM: unsteady solver of the SIMPLE family (Semi-Implicit Method for Pressure Linked Equations) => iterative method:
  
  guess \( p^* \) and solve \( u^* \)

  find correction for \( p^* \) (by mean of \( \nabla \cdot u^* = 0 \))
Airfoil case study:
pisoFOAM, SST turbulence model & unsteady BC

- pisoFOAM: unsteady solver of the SIMPLE family (Semi-Implicit Method for Pressure Linked Equations) => iterative method:

  guess \( p^* \) and solve \( u^* \)

  find correction for \( p^* \) (by mean of \( \nabla \cdot u^* = 0 \))

- SST: blend between \( k-\varepsilon \) (far from the wall) and \( k-\omega \) (close to the wall). Good model for engineering type applications
Airfoil case study:
pisoFOAM, SST turbulence model & unsteady BC

- **pisoFOAM**: unsteady solver of the SIMPLE family (Semi-Implicit Method for Pressure Linked Equations) => iterative method:
  
  \[
  \text{guess } p^* \text{ and solve } u^* \\
  \text{find correction for } p^* \text{ (by mean of } \nabla \cdot u^* = 0) \]

- **SST**: blend between k-\(\varepsilon\) (far from the wall) and k-\(\omega\) (close to the wall). Good model for engineering type applications

- Set variable inlet velocity with a ramp file
Airfoil case study: pisoFOAM, SST turbulence model & unsteady BC

- pisoFOAM: unsteady solver of the SIMPLE family (Semi-Implicit Method for Pressure Linked Equations) => iterative method:
  - guess p* and solve u*
  - find correction for p* (by mean of $\nabla \cdot u^* = 0$)

- SST: blend between k-ε (far from the wall) and k-ω (close to the wall). Good model for engineering type applications

- Set variable inlet velocity with a ramp file

- Extract forces on the wing
Airfoil case study: directory listing

Working Directory
Airfoil case study: directory listing

- Working Directory
  - '0'
  - constant
  - system
Airfoil case study: directory listing

Working Directory

'0' constant system

k omega U p
Airfoil case study: directory listing

\[ k \approx \frac{3}{2} \left( \frac{5}{100} u_{in} \right)^2 \]
\[ \omega \approx \frac{\sqrt{k}}{L} \]

Working Directory

‘0’ constant system

k omega U p
Airfoil case study: directory listing

\[ k \approx \frac{3}{2} \left( \frac{5}{100} u_{in} \right)^2 \]

\[ \omega \approx \frac{\sqrt{k}}{L} \]

Working Directory

\('0'\) constant system

k omega U p

ControlDict fvSchemes fvSolutions
Airfoil case study: directory listing

\[ k \approx \frac{3}{2} \left( \frac{5}{100} u_{in} \right)^2 \]
\[ \omega \approx \frac{\sqrt{k}}{L} \]

- Working Directory
- constant
- system
  - '0'
- ControlDict
- fvSchemes
- fvSolutions
- polyMesh
- RASProperties
- transportProperties
- turbulenceProperties
- k
- omega
- U
- p
Airfoil case study: directory listing

\[ k \approx \frac{3}{2} \left( \frac{5}{100} u_{in} \right)^2 \]

\[ \omega \approx \frac{\sqrt{k}}{L} \]

Working Directory

\( '0' \), constant, system

k, omega, U, p

ControlDict, fvSchemes, fvSolutions

polyMesh, RASProperties, transportProperties, turbulenceProperties

boundary, cellZone, faces, faceZones, ...

...
Case:
undeformable Spinnaker geometry in gusts
Case: undeformable Spinnaker geometry in gusts
Principles of mesh motion: ALE and the PimpleDyMFOAM solver
Principles of mesh motion: ALE and the PimpleDyMFOAM solver

- Arbitrary Lagrangian Eulerian methods:
Principles of mesh motion: ALE and the PimpleDyMFOAM solver

- Arbitrary Lagrangian Eulerian methods:
  - Deform the Eulerian fluid mesh in lagrangian way on the moving interface
Principles of mesh motion: ALE and the PimpleDyMFOAM solver

- Arbitrary Lagrangian Eulerian methods:
  - Deform the Eulerian fluid mesh in lagrangian way on the moving interface
  - Mesh boundaries remain unchanged
**Principles of mesh motion:**
ALE and the PimpleDyMFOAM solver

- Arbitrary Lagrangian Eulerian methods:
  - Deform the Eulerian fluid mesh in lagrangian way on the moving interface
  - Mesh boundaries remain unchanged
  - In the domain all is allowed, provided that some regularity is respected
Principles of mesh motion:
ALE and the PimpleDyMFOAM solver

- Arbitrary Lagrangian Eulerian methods:
  - Deform the Eulerian fluid mesh in lagrangian way on the moving interface
  - Mesh boundaries remain unchanged
  - In the domain all is allowed, provided that some regularity is respected
Principles of mesh motion:
Principles of mesh motion:
The Grid has now its own velocity => this should be considered in order to be conservative
Principles of mesh motion:

The Grid has now its own velocity => this should be considered in order to be conservative

Navier-Stokes equation are reformulated in the ALE framework, by taking into account the mesh velocity w
Principles of mesh motion:

The Grid has now its own velocity $\Rightarrow$ this should be considered in order to be conservative.

Navier-Stokes equation are reformulated in the ALE framework, by taking into account the mesh velocity $w$

\[
\begin{align*}
\frac{\partial u}{\partial t} + (u \cdot \nabla) u - \nu \Delta u + \nabla p &= f \\
\nabla \cdot u &= 0
\end{align*}
\]
Principles of mesh motion:

The Grid has now its own velocity $\Rightarrow$ this should be considered in order to be conservative. Navier-Stokes equation are reformulated in the ALE framework, by taking into account the mesh velocity $w$.

$$\begin{align*}
\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu \Delta u + \nabla p &= f \\
\nabla \cdot u &= 0 \\
\Omega &\subset \mathbb{R}^d
\end{align*}$$

$$\begin{align*}
\frac{\partial u}{\partial t} \bigg|_\xi + [(u - w) \cdot \nabla_x] u - \nu \Delta_x u + \nabla_x p &= f \\
\nabla_x \cdot u &= 0 \\
\Omega &\subset \mathbb{R}^d.
\end{align*}$$
Principles of mesh motion:

The Grid has now its own velocity => this should be considered in order to be conservative. Navier-Stokes equation are reformulated in the ALE framework, by taking into account the mesh velocity $w$

\[
\begin{aligned}
\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu \Delta u + \nabla p &= f \\
\nabla \cdot u &= 0
\end{aligned}
\]

in $\Omega \subset \mathbb{R}^d$

Formally, the convective term only changes

\[
\begin{aligned}
\frac{\partial u}{\partial t} \bigg|_\xi + [(u - w) \cdot \nabla] u - \nu \Delta_x u + \nabla_x p &= f \\
\nabla_x \cdot u &= 0
\end{aligned}
\]

in $\Omega \subset \mathbb{R}^d$. 
Principles of mesh motion:

The Grid has now its own velocity => this should be considered in order to be conservative.

Navier-Stokes equation are reformulated in the ALE framework, by taking into account the mesh velocity $w$.

Formally, the convective term only changes

This is equivalent to make velocities relative to the mesh motion. In pimpleDyMFOAM:

```cpp
// Make the fluxes relative to the mesh motion
fvc::makeRelative(phi, U);
```
Set up the case for the moving beam:
Set up the case for the moving beam:
Set up the case for the moving beam:

<table>
<thead>
<tr>
<th></th>
<th>Beam</th>
<th>Top</th>
<th>Bottom</th>
<th>Inlet</th>
<th>Outlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>ramp</td>
<td>0</td>
<td>0</td>
<td>pressureInletOutletVelocity</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>ZeroG.</td>
<td>ZeroG.</td>
<td>ZeroG.</td>
<td>totalPressure - 0</td>
<td>ZeroG.</td>
</tr>
</tbody>
</table>
Set up the case for the moving beam:

A dynamic mesh type and a mesh motion solver have to be chosen.

<table>
<thead>
<tr>
<th></th>
<th>Beam</th>
<th>Top</th>
<th>Bottom</th>
<th>Inlet</th>
<th>Outlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>ramp</td>
<td>0</td>
<td>0</td>
<td>pressureInletOutletVelocity</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>ZeroG.</td>
<td>ZeroG.</td>
<td>ZeroG.</td>
<td>totalPressure - 0</td>
<td>ZeroG.</td>
</tr>
</tbody>
</table>
Set up the case for the moving beam:

A dynamic mesh type and a mesh motion solver have to be chosen

This is done in **Constant/dynamicMeshDict**
Set up the case for the moving beam:

A dynamic mesh type and a mesh motion solver have to be chosen. This is done in Constant/dynamicMeshDict. This choice determines the additional input files to be added.

<table>
<thead>
<tr>
<th></th>
<th>Beam</th>
<th>Top</th>
<th>Bottom</th>
<th>Inlet</th>
<th>Outlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>ramp</td>
<td>0</td>
<td>0</td>
<td>pressureInletOutletVelocity</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>ZeroG.</td>
<td>ZeroG.</td>
<td>ZeroG.</td>
<td>totalPressure - 0</td>
<td>ZeroG.</td>
</tr>
</tbody>
</table>

A dynamic mesh type and a mesh motion solver have to be chosen.
Choice of the dynamic mesh class:

**dynamicInkJetFvMesh**: moves the mesh using analytical expression
   (See tutorial from Gonzales, Chalmers university)

**dynamicMotionSolverFvMesh**: prescribes mesh motions, for example on boundaries, and it allows the direct specification of mesh points motions (velocities or displacements).
Choice of the motion solver:
Choice of the motion solver:
Choice of the motion solver:
Choice of the motion solver:

```
20    dynamicFvMesh  dynamicMotionSolverFvMesh;
21    motionSolverLibs ( "libfvMotionSolvers.so" );
22    solver          velocityLaplacian;
23    diffusivity     directional ( 1 200 0 );
```

Constant/dinamicMeshDict

```
regIOobject
```

```
IOdictionary
motionSolver
fvMotionSolver
```

```
20    dynamicFvMesh  dynamicMotionSolverFvMesh;
21    motionSolverLibs ( "libfvMotionSolvers.so" );
22    solver          velocityLaplacian;
23    diffusivity     directional ( 1 200 0 );
```
Using the velocityLaplacianMotionSolver:
Using the velocityLaplacianMotionSolver:

- Basic Idea: define the motion of the boundaries in terms of velocity, and solve a laplacian (therefore apply a diffusion) in the rest of the domain
Using the `velocityLaplacianMotionSolver`:

- **Basic Idea**: define the motion of the boundaries in terms of velocity, and solve a laplacian (therefore apply a diffusion) in the rest of the domain.
- The motion can be defined as a constant velocity or via a ‘ramp’ file.
Using the `velocityLaplacianMotionSolver`:

- **Basic Idea**: define the motion of the boundaries in terms of velocity, and solve a laplacian (therefore apply diffusion) in the rest of the domain.

- The motion can be defined as a constant velocity or via a ‘ramp’ file.

- Two files have to be added in the ‘0’ directory: `cellMotionU` and `pointMotionU`. Their definition is as usual for FOAM files, and in this case they are equal:

```plaintext
dimensions [0 1 -1 0 0 0];
internalField uniform (0 0 0);
boundaryField
{
    beam
    {
        type timeVaryingUniformFixedValue;
        fileName "ramp";
        outOfBounds repeat;
    }
    topmoving
    {
        type slip;
    }
    outlet
    {
        type fixedValue;
        value uniform (0 0 0);
    }
}
```
And finally some results:
And finally some results:
WRITING AND COMPILING USER APPLICATIONS AND LIBRARIES

Tutorial from:
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/

Report, files and presentation by: Andreu Oliver Gonzalez
Before to get into the code...
Before to get into the code...

C++ is an object oriented programming language, OpenFOAM is build up to objects and classes

www.cplusplus.com/doc
Deitel & Deitel: C++ How to program
Before to get into the code...

C++ is an object oriented programming language, OpenFOAM is build up to objects and classes

www.cplusplus.com/doc
Deitel & Deitel: C++ How to program

A reference: & indicates an address in the computer memory, and it can be seen as a bookmark. Treat variables as references is good, since the object is not copied or transferred in the memory
Before to get into the code...

C++ is an object oriented programming language, OpenFOAM is build up to objects and classes

www.cplusplus.com/doc
Deitel & Deitel: C++ How to program

A reference: & indicates an address in the computer memory, and it can be seen as a bookmark. Treat variables as references is good, since the object is not copied or transferred in the memory

Classes have member functions, the same named function applied to different objects produces different calls. Functions should be (generally) searched in the object’s class, or in the parent classes
Before to get into the code... Inheritance
Before to get into the code... Inheritance

CLASS POLYGONS
{B, H}

CLASS TRIANGLE
AREA=B*H/2

CLASS RECTANGLE
AREA=B*H
Before to get into the code... Inheritance

//Define objects:
Triangle Tria{2,3}
Rectangle Rec{3,4}

//Calculate area:
Tria.area();
Rec.area();
OpenFOAM makes a large use of inheritance, therefore it is necessary to use the doxygen (http://foam.sourceforge.net/doc/Doxygen/html/)
Before to get into the code... Inheritance

CLASS POLYGONS {B, H}

CLASS TRIANGLE

CLASS RECTANGLE

AREA = B * H / 2

//Define objects:
Triangle Tria{2,3}
Rectangle Rec{3,4}

//Calculate area:
Tria.area();
Rec.area();
Before to get into the code... Inheritance

CLASS POLYGONS
{B, H}

CLASS TRIANGLE

CLASS RECTANGLE

AREA = B * H / 2

//Define objects:
Triangle Tria{2,3}
Rectangle Rec{3,4}

//Calculate area:
Tria.area();
Rec.area();

Public Member Functions

ClassName ("fvMesh")
fvMesh (const IOobject &io)
Construct from IOobject.
fvMesh (const IOobject &io, const Xfer<IOobject>&)
Construct from components without boundary.
fvMesh (const IOobject &io, const Xfer<IOobject>&)
Construct without boundary from cells

virtual ~fvMesh ()
void addFvPatches (const List< polyPatch * > &)
Add boundary patches. Constructor helps.

virtual readUpdateState readUpdate ()
Update the mesh based on the mesh file.
const Time & time () const
Return the top-level database.

virtual const objectRegistry & thisDb () const
Return the object registry - resolve conflict.
const word & name () const
Return reference to name.
const fvBoundaryMesh & boundary () const
Return reference to boundary mesh.

virtual const IduAddressing & IduAddr () const
Return Idu addressing.
virtual IduInterfacePtrsList interfaces () const
Return a list of pointers for each patch.

const unallocLabelList & owner () const
Internal face owner.
const unallocLabelList & neighbour () const
Internal face neighbour.
const DimensionedField< scalar, volMesh > & V () const
Return cell volumes.
How to find entries:
How to find entries:

Looking into the code:

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 225 to 299

// **************************** Member Functions ************************** //

void Foam::forces::read(const dictionary& dict)
{
    ...  
    const fvMesh& mesh = refCast<const fvMesh>(obr_);
    patchSet_ =
        mesh.boundaryMesh().patchSet(wordList(dict.lookup("patches")));  
    ...
    pName_ = dict.lookupOrDefault<word>("pName", "p");
    UName_ = dict.lookupOrDefault<word>("UName", "U");
    rhoName_ = dict.lookupOrDefault<word>("rhoName", "rho");
    ...
    rhoRef_ = readScalar(dict.lookup("rhoInf"));
    pRef_ = dict.lookupOrDefault<scalar>("pRef", 0.0);
    CofR_ = dict.lookup("CofR");
}
How to find entries:

Looking into the code:

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 225 to 299

// * * * * * * * * * * * * * Member Functions * * * * * * * * * * * * * /*

void Foam::forces::read(const dictionary& dict)
{
    ...
    const fvMesh& mesh = refCast<const fvMesh>(obr_);
    patchSet_ = mesh.boundaryMesh().patchSet(wordList(dict.lookup("patches")));
    ...
    pName_ = dict.lookupOrDefault<word>("pName", "p");
    UName_ = dict.lookupOrDefault<word>("UName", "U");
    rhoName_ = dict.lookupOrDefault<word>("rhoName", "rho");
    ...
    rhoRef_ = readScalar(dict.lookup("rhoInf"));
    pRef_ = dict.lookupOrDefault<scalar>("pRef", 0.0);
    CofR_ = dict.lookup("CofR");
    ...
}
How to find entries:

Looking into the code:
OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 225 to 299

// * * * * * * * * * * * * * * * Member Functions * * * * * * * * * * * * * //

void Foam::forces::read(const dictionary& dict)
{
  ...
  const fvMesh& mesh = refCast<const fvMesh>(obr_);
  patchSet_ =
    mesh.boundaryMesh().patchSet(wordList(dict.lookup("patches")));
  ...
  pName_ = dict.lookupOrDefault<word>("pName", "p");
  UName_ = dict.lookupOrDefault<word>("UName", "U");
  rhoName_ = dict.lookupOrDefault<word>("rhoName", "rho");
  ...
  rhoRef_ = readScalar(dict.lookup("rhoInf"));
  pRef_ = dict.lookupOrDefault<scalar>("pRef", 0.0);
  CofR_ = dict.lookup("CofR");
}
How to find entries:

Looking into the code:

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 225 to 299

// * * * * * * * * * * * * * * * Member Functions * * * * * * * * * * * * * //

void Foam::forces::read(const dictionary& dict)
{
    ...
    const fvMesh& mesh = refCast<const fvMesh>(obr_);
    patchSet_ =
        mesh.boundaryMesh().patchSet(wordList(dict.lookup("patches")));
    ...
    pName_ = dict.lookupOrDefault<word>("pName", "p");
    UName_ = dict.lookupOrDefault<word>("UName", "U");
    rhoName_ = dict.lookupOrDefault<word>("rhoName", "rho");
    ...
    rhoRef_ = readScalar(dict.lookup("rhoInf"));
    pRef_ = dict.lookupOrDefault<scalar>("pRef", 0.0);
    CofR_ = dict.lookup("CofR");
}
How to find entries:

Looking into the code:

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 225 to 299

// ****************************************** Member Functions ****************************************** //

```cpp
void Foam::forces::read(const dictionary & dict)
{
    ...
    const fvMesh& mesh = refCast<const fvMesh>(obr_);
    patchSet_ =
        mesh.boundaryMesh().patchSet(wordList(dict.lookup("patches")));
    ...
    pName_ = dict.lookupOrDefault<word>("pName", "p");
    UName_ = dict.lookupOrDefault<word>("UName", "U");
    rhoName_ = dict.lookupOrDefault<word>("rhoName", "rho");
    ...
    rhoRef_ = readScalar(dict.lookup("rhoInf"));
    pRef_ = dict.lookupOrDefault<scalar>("pRef", 0.0);
    CofR_ = dict.lookup("CofR");
}
```
How to find entries:

Looking into the code:

OpenFOAM/src/postProcessing/functionObjects/forces/forces.

// * * * * * * * * * * * * * * * Member Functions * * * *

void Foam::forces::read(const dictionary& dict)
{
  ...
  const fvMesh& mesh = refCast<const fvMesh>(obr_);
  patchSet_ = mesh.boundaryMesh().patchSet(wordList(dict.lookup("patches")));
  ...
  pName_ = dict.lookupOrDefault<word>("pName", "p");
  UName_ = dict.lookupOrDefault<word>("UName", "U");
  rhoName_ = dict.lookupOrDefault<word>("rhoName", "rho");
  ...
  rhoRef_ = readScalar(dict.lookup("rhoInf"));
  pRef_ = dict.lookupOrDefault<scalar>("pRef", 0.0);
  CofR_ = dict.lookup("CofR");
}
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fmm
    ...

    const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
    const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
    const fvMesh& mesh = U.mesh();
    const surfaceVectorField::GeometricBoundaryField& Sfb =
        mesh.Sf().boundaryField();
 ...

And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
...

    { const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
      const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
      const fvMesh& mesh = U.mesh();
      const surfaceVectorField::GeometricBoundaryField& Sfb = 
          mesh.Sf().boundaryField();
      ...

    was read from the controlDict
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
  forcesMoments fm
  ...
  {
    const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
    const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
    const fvMesh& mesh = U.mesh();
    const surfaceVectorField::GeometricBoundaryField& Sfb =
      mesh.Sf().boundaryField();
    ...
  }
}
And how the calculation is actually done (1):

```
OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
    ...

    const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
    const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
    const fvMesh& mesh = U.mesh();
    const surfaceVectorField::GeometricBoundaryField& Sfb =
        mesh.Sf().boundaryField();
    ...
```
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
...
{
    const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
    const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
    const fvMesh& mesh = U.mesh();
    const surfaceVectorField::GeometricBoundaryField& Sfb =
        mesh.Sf().boundaryField();
    ...
}
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
    ...
}

    const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
    const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
    const fvMesh& mesh = U.mesh();
    const surfaceVectorField::GeometricBoundaryField& Sfb =
        mesh.Sf().boundaryField();
    ...

    mesh() has to be a function of the class
    volScalarField, returning a reference to the
    mesh associated with the field U. But
    volScalarField is class template... Where is it?
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
    ...
    {
        const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
        const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
        const fvMesh& mesh = U.mesh();
        const surfaceVectorField::GeometricBoundaryField& Sfb =
            mesh.Sf().boundaryField();
        ...
}
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
    ...
{
    const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
    const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
    const fvMesh& mesh = U.mesh();
    const surfaceVectorField::GeometricBoundaryField& Sfb =
        mesh.Sf().boundaryField();
    ...

    mesh is object of the class fvMesh.
    searching in the Doxygen for this class:
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
    ...
}

    const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
    const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
    const fvMesh& mesh = U.mesh();
    const surfaceVectorField::GeometricBoundaryField& Sfb =
        mesh.Sf().boundaryField();

    ...

mesh is object of the class fvMesh.
searching in the Doxygen for this class:
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
    ...
    {
        const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
        const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
        const fvMesh& mesh = U.mesh();
        const surfaceVectorField::GeometricBoundaryField& Sfb =
            mesh.Sf().boundaryField();
        ...
    }
}
And how the calculation is actually done (1):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 393 to 447

Foam::forces::forcesMoments Foam::forces::calcForcesMoment() const
{
    forcesMoments fm
    ...
}

const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
const volScalarField& p = obr_.lookupObject<volScalarField>(pName_);
const fvMesh& mesh = U.mesh();
const surfaceVectorField::GeometricBoundaryField& Sfb =
    mesh.Sf().boundaryField();
...

Sfb is then a reference to the face area vector of the mesh boundaryField (inlet, outlet, walls, wing, ...) as it is defined in the mesh and in the boundary files (as p or U)
And how the calculation is actually done (2):
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

```c++
...
forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;  
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);
    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);
    ...
}
...
return fm;
```
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

... 

forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;

    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

    fm.first().first() += rho(p)*sum(pf);

    fm.second().first() += rho(p)*sum(Md ^ pf);

    ...

} 

... 

return fm; 

OpenFOAM iterator, as a ‘for’ cycle.
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

...  
forallConstIter(labelHashSet, patchSet_, iter) 
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;  
<vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);>
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);  

    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);

    ...
}
...  
return fm;  

OpenFOAM iterator, as a ‘for’ cycle.

It cycles (using the index ‘iter’) over the objects in the patchSet
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

```c
forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);

    ...
}
```

OpenFOAM iterator, as a ‘for’ cycle.

It cycles (using the index ‘iter’) over the objects in the patchSet

This has been previously defined as the list of the patches on which we want to integrate forces (coming from the controlDict)
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

... 
forAllConstIter(labelHashSet, patchSet_, iter) 
{ 
label patchi = iter.key();

vectorField Md = mesh.C().boundaryField()[patchi] - CofR_; 
vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

fm.first().first() += rho(p)*sum(pf);
fm.second().first() += rho(p)*sum(Md ^ pf);
...
}
...
return fm;
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

... 
forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField \(\text{Md} = \text{mesh.C().boundaryField()[patchi]} - \text{CofR}_\);

    vectorField \(\text{pf} = \text{Sfb[patchi]*(p.boundaryField()[patchi]} - \text{pRef})\);

    \(\text{fm.first().first() += rho(p)*sum(pf)}\);

    \(\text{fm.second().first() += rho(p)*sum(Md ^ pf)}\);

    ...
}

return \(\text{fm}\);
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

... forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;

    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);

    ...
}
...

mesh is object of the class fvMesh

From the Doxygen then:

return fm;
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

... forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;  
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);
    ...
}

mesh is object of the class fvMesh

From the Doxygen then:

... return fm;
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

```c
...
forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);
}
...
return fm;
```

mesh is object of the class fvMesh

From the Doxygen then:
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

... forAllConstIter(labelHashSet, patchSet_, iter)
{
  label patchi = iter.key();

  vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;  

  vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

  fm.first().first() += rho(p)*sum(pf);

  fm.second().first() += rho(p)*sum(Md ^ pf);

  ...
}

mesh is object of the class fvMesh

From the Doxygen then:

The entire expression returns a vector with the cell centres of the
patch that we chose for integrating forces
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

```cpp
...
forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;

    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

    fm.first().first() += rho(p)*sum(pf);

    fm.second().first() += rho(p)*sum(Md ^ pf);

    ...
}
...
return fm;
```
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

... forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);

    ...
}
}

... return fm;

Sfb is the (reference to) the face area vector
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

```c
...
forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);

    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);
...
}
...
return fm;
```

Sfb is the (reference to) the face area vector

It is here multiplied for the pressure boundaryField $\Rightarrow$ pf returns the vector of forces insisting on the chosen patch
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

```cpp
...
forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();

    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);
    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);
...
}
...
return fm;
```
And how the calculation is actually done (2):

OpenFOAM/src/postProcessing/functionObjects/forces/forces/Force.C, lines 461 to 490

```c++
...
forAllConstIter(labelHashSet, patchSet_, iter)
{
    label patchi = iter.key();
    vectorField Md = mesh.C().boundaryField()[patchi] - CofR_;
    vectorField pf = Sfb[patchi]*(p.boundaryField()[patchi] - pRef);
    fm.first().first() += rho(p)*sum(pf);
    fm.second().first() += rho(p)*sum(Md ^ pf);
    ...
}
return fm;
```
Modifying pimpleDyMFOAM for Fluid Structure interactions: imposing patch deformations:
Modifying pimpleDyMFOAM for Fluid Structure interactions: imposing patch deformations:

- Until now, it is easy to impose rigid body movements, in terms of velocities of displacements.
Modifying pimpleDyMFOAM for Fluid Structure interactions: imposing patch deformations:

- Until now, it is easy to impose rigid body movements, in terms of velocities of displacements.
- The solver has then be modified in order to read the cell displacements from a file, coming from a structural solver.
Modifying pimpleDyMFOAM for Fluid Structure interactions: imposing patch deformations:

- Until now, it is easy to impose rigid body movements, in terms of velocities of displacements.
- The solver has then be modified in order to read the cell displacements from a file, coming from a structural solver.
- Three modifications:
Modifying pimpleDyMFOAM for Fluid Structure interactions: imposing patch deformations:

- Until now, it is easy to impose rigid body movements, in terms of velocities of displacements.
- The solver has then be modified in order to read the cell displacements from a file, coming from a structural solver.
- Three modifications:
  - 1. In the initial phase an output file is created with the coordinates of the face-centres of the interface, namely the sail surface. This is used for calculating the connectivity file.
Modifying pimpleDyMFOAM for Fluid Structure interactions: imposing patch deformations:

- Until now, it is easy to impose rigid body movements, in terms of velocities of displacements.
- The solver has then be modified in order to read the cell displacements from a file, coming from a structural solver.
- Three modifications:
  - 1. In the initial phase an output file is created with the coordinates of the face-centres of the interface, namely the sail surface. This is used for calculating the connectivity file.
  - 2. Before the mesh update, displacements of the mesh (cells) are read from a file, which comes from the structural calculation.
Modifying pimpleDyMFOAM for Fluid Structure interactions: imposing patch deformations:

- Until now, it is easy to impose rigid body movements, in terms of velocities of displacements.
- The solver has then be modified in order to read the cell displacements from a file, coming from a structural solver.
- Three modifications:
  - 1. In the initial phase an output file is created with the coordinates of the face-centres of the interface, namely the sail surface. This is used for calculating the connectivity file.
  - 2. Before the mesh update, displacements of the mesh (cells) are read from a file, which comes from the structural calculation.
  - 3. At the end of the routine, pressures at the interface are written to an output file.
IOdictionary coupling

(IOobject

  "CouplingDict",

  runTime.constant(),

  mesh,

  IOobject::MUST_READ,

  IOobject::NO_WRITE)

word sail=coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];

int nFaces=SailMesh.size()/2;

Output the patch face centres to file /1:
IOdictionary coupling
(
 IOobject
 ( "CouplingDict",
   runTime.constant(),
   mesh,
   IOobject::MUST_READ,
   IOobject::NO_WRITE
 )
);

word sail = coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];

int nFaces = SailMesh.size()/2;
Output the patch face centres to file /1:

```cpp
IOdictionary coupling 
(
  IOobject 
  (  
    "CouplingDict",  
    runTime.constant(),  
    mesh,  
    IOobject::MUST_READ,  
    IOobject::NO_WRITE
  )
);  

word sail = coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];

int nFaces = SailMesh.size()/2;
```
Output the patch face centres to file /1:

```
IOdictionary coupling
  (
    IOobject
      ("CouplingDict",
       runTime.constant(),
       mesh,
       IOobject::MUST_READ,
       IOobject::NO_WRITE
      )
  );

word sail = coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];

int nFaces = SailMesh.size() / 2;
```

Searches in the Dict the entry “wing”, and assigns its (char) value to the variable “sail”. This should correspond to a physical surface in the mesh.
Output the patch face centres to file /1:

1  IOdictionary coupling
2   (  
3     IOobject
4       (  
5         "CouplingDict",
6           runTime.constant(),
7               mesh,
8         IOobject::MUST_READ,
9         IOobject::NO_WRITE
10       )  
11   );  
12  
13  word sail=coupling.lookup("wing");
14  
15  label sailL = mesh.boundaryMesh().findPatchID(sail);
16  
17  const polyPatch &SailMesh=mesh.boundaryMesh()[sailL];
18  
19  int nFaces=SailMesh.size()/2;
Output the patch face centres to file /1:

1  IOdictionary coupling
2    (  
3       IOobject
4         (  
5           "CouplingDict",
6             runTime.constant(),
7             mesh,
8         IOobject::MUST_READ,
9         IOobject::NO_WRITE
10       )
11    );
12
13   word sail=coupling.lookup("wing");
14
15   label sailL = mesh.boundaryMesh().findPatchID(sail);
16
17   const polyPatch &SailMesh=mesh.boundaryMesh()[sailL];
18
19   int nFaces=SailMesh.size()/2;
Output the patch face centres to file /1:

1  IOdictionary coupling
2   ( 
3     IOobject
4       ( 
5       "CouplingDict",
6         runTime.constant(),
7         mesh,
8       IOobject::MUST_READ,
9       IOobject::NO_WRITE
10     )
11   );

12  word sail=coupling.lookup("wing");
13
14  label sailL = mesh.boundaryMesh().findPatchID(sail);
15
16  const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];
17
18  int nFaces = SailMesh.size()/2;
IOdictionary coupling
(
 IOobject
   ( "CouplingDict",
     runTime.constant(),
     mesh,
     IOobject::MUST_READ,
     IOobject::NO_WRITE
   )
);

word sail = coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh(sailL);

int nFaces = SailMesh.size()/2;
\begin{verbatim}
ioDictionary coupling

  IOobject
    (
      "CouplingDict",
      runTime.constant(),
      mesh,
      IOobject::MUST_READ,
      IOobject::NO_WRITE
    )

  word sail = coupling.lookup("wing");

  label sailL = mesh.boundaryMesh().findPatchID(sail);

  const polyPatch &SailMesh = mesh.boundaryMesh(sailL);

  int nFaces = SailMesh.size()/2;
\end{verbatim}

Output the patch face centres to file /1:

\begin{itemize}
  \item boundaryMesh() is not a member function of fvMesh, BUT a class itself. (..?).
  \item boundaryMesh() is its constructor
  \item findPatchID is function member of this class
\end{itemize}
IOdictionary coupling
(
  IOobject
  ("CouplingDict",
   runTime.constant(),
   mesh,
   IOobject::MUST_READ,
   IOobject::NO_WRITE
  )
)

word sail = coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];

int nFaces = SailMesh.size() / 2;
Output the patch face centres to file /1:

1  IOdictionary coupling
2   (
3     IOobject
4       ('CouplingDict',
5       runTime.constant(),
6       mesh,
7       IOobject::MUST_READ,
8       IOobject::NO_WRITE
9     )
10   );
11
12  word sail = coupling.lookup("wing");
13
14  label sailL = mesh.boundaryMesh().findPatchID(sail);
15
16  const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];
17
18  int nFaces = SailMesh.size()/2;

Globally, sailL is the label (the numbering) of the surface defined in the CouplingDict
IOdictionary coupling

IOobject
  ( "CouplingDict",
    runTime.constant(),
    mesh,
    IOobject::MUST_READ,
    IOobject::NO_WRITE
  );

word sail=coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh=mesh.boundaryMesh()[sailL];

int nFaces=SailMesh.size()/2;
IOdictionary coupling
(
    IOobject
    ("CouplingDict",
     runTime.constant(),
     mesh,
     IOobject::MUST_READ,
     IOobject::NO_WRITE
    )
)

word sail = coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];

int nFaces = SailMesh.size()/2;

and SailMesh a reference to the boundary-Mesh defining the surface in the mesh.
IOdictionary coupling

IOobject

"CouplingDict",

runTime.constant(),
mesh,

IOobject::MUST_READ,

IOobject::NO_WRITE

);}

word sail=coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh=mesh.boundaryMesh()[sailL];

int nFaces=SailMesh.size()/2;
Output the patch face centres to file /1:

```cpp
IOdictionary coupling
(
    IOobject
    ("CouplingDict",
     runTime.constant(),
     mesh,
     IOobject::MUST_READ,
     IOobject::NO_WRITE
    )
);

word sail = coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh(sailL);

int nFaces = SailMesh.size() / 2;
```

Sail is function of a template class, therefore it is multi-purpose, applicable to different objects.
IOdictionary coupling

IOobject

"CouplingDict",
runTime.constant(),
mesh,
IOobject::MUST_READ,
IOobject::NO_WRITE

word sail = coupling.lookup("wing");

label sailL = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];

int nFaces = SailMesh.size()/2;

Sail is function of a template class, therefore it is multi-purpose, applicable to different objects.

It is /2, since the zero thickness surface is described as a series of superposed faces.
Output the patch face centres to file /1:

```plaintext
1   IOdictionary coupling
2    ( 
3      IOobject
4      ( 
5        "CouplingDict",
6          runTime.constant(),
7          mesh,
8          IOobject::MUST_READ,
9          IOobject::NO_WRITE
10     )
11    );
12
13   word sail = coupling.lookup("wing");
14
15   label sailL = mesh.boundaryMesh().findPatchID(sail);
16
17   const polyPatch &SailMesh = mesh.boundaryMesh()[sailL];
18
19   int nFaces = SailMesh.size()/2;
```
Output the patch face centres to file /2:

23  std::ofstream myfile;
24  myfile.open("FluidMesh_centres.txt", std::ios::app);
25
26 myfile<<nFaces<<"\n";
27 for (int n=0; n<nFaces; n++)
28   {
29     myfile<<SailMesh.faceCentres()[n][0]<<
30       " " <<SailMesh.faceCentres()[n][1]<<
31       " " <<SailMesh.faceCentres()[n][2]<<"\n";
32   }
33
34  myfile.close();
std::ofstream myfile;
myfile.open("FluidMesh_centres.txt", std::ios::app);

myfile<<nFaces<<"\n";
for (int n=0; n<nFaces; n++)
{
    myfile<<SailMesh.faceCentres()[n][0]<<" "
        <<SailMesh.faceCentres()[n][1]<<" 
        " << SailMesh.faceCentres()[n][2]<<"\n";
}

myfile.close();

faceCentres() is again a template function, applied here to an object of the class boundaryMesh, it returns the vector (x,y,z) with the coordinates of the centres of the mesh faces.
Output the patch face centres to file /2:

```cpp
23 std::ofstream myfile;
24 myfile.open("FluidMesh_centres.txt", std::ios::app);
25 myfile<<nFaces<<"\n";
26 for (int n=0; n<nFaces; n++)
27 {
28     myfile<<SailMesh.faceCentres()[n][0]<<" "
29     <<SailMesh.faceCentres()[n][1]<<" "
30     <<SailMesh.faceCentres()[n][2]<<"\n";
31 }
32 myfile.close();
```

faceCentres() is again a template function, applied here to an object of the class boundaryMesh, it returns the vector (x,y,z) with the coordinates of the centres of the mesh faces.

The operator [] is used in C++ for accessing the vector entries.
std::ofstream myfile;
myfile.open("FluidMesh_centres.txt", std::ios::app);

myfile<<nFaces<<"\n";
for(int n=0; n<nFaces; n++)
{
    myfile<<SailMesh.faceCentres()[n][0]<<" ">
        " <<SailMesh.faceCentres()[n][1]<<" ">
        " << SailMesh.faceCentres()[n][2]<<"\n";
}

myfile.close();
Read the cell displacements from file /1:

```c
#include "readCellDisp.h"

volVectorField::GeometricBoundaryField &meshDisplacement =
    const_cast<volVectorField&>(
        (mesh.objectRegistry::lookupObject<volVectorField>("cellDisplacement"))
            .boundaryField();
```
Read the cell displacements from file /1:

```cpp
#include "readCellDisp.h"

volVectorField::GeometricBoundaryField &meshDisplacement =
    const_cast<volVectorField&>(
        mesh.objectRegistry::lookupObject<volVectorField>("cellDisplacement")
    ).boundaryField();
```

Searches in the objectRegistry of the mesh the entry “cellDisplacement”, which is an object of the class volVectorField. ObjectRegistry is function member of the class fvMesh.
Read the cell displacements from file /1:

```c++
#include "readCellDisp.h"

volVectorField::GeometricBoundaryField &meshDisplacement =
    const_cast<volVectorField&>
    (mesh.objectRegistry::lookupObject<volVectorField>("cellDisplacement"))
    .boundaryField();
```

Searches in the objectRegistry of the mesh the entry “cellDisplacement”, which is an object of the class volVectorField. ObjectRegistry is function member of the class fvMesh.

From the cellDisplacement vector, searches the entries corresponding to the boundaryField (inlet, outlet, walls, sail... )
Read the cell displacements from file /1:

```cpp
#include "readCellDisp.h"

volVectorField::GeometricBoundaryField &meshDisplacement =
    const_cast<volVectorField&>(
        mesh.objectRegistry::lookupObject<volVectorField>("cellDisplacement")
    ).boundaryField();
```

Whatever is returned, forces it to be a reference to a volVectorField. We have then the address in the computer memory where displacements of the boundary field cells are stored.

Searches in the objectRegistry of the mesh the entry “cellDisplacement”, which is an object of the class volVectorField. ObjectRegistry is function member of the class fvMesh.

From the cellDisplacement vector, searches the entries corresponding to the boundaryField (inlet, outlet, walls, sail... )
Read the cell displacements from file /1:

```cpp
#include "readCellDisp.h"

volVectorField::GeometricBoundaryField &meshDisplacement = 
   const_cast<volVectorField&>(
      mesh.objectRegistry::lookupObject<volVectorField>("cellDisplacement")
   ).boundaryField();
```
Read the cell displacements from file /1:

```cpp
#include "readCellDisp.h"

volVectorField::GeometricBoundaryField &meshDisplacement =
    const_cast<volVectorField&>(
        (mesh.objectRegistry::lookupObject<volVectorField>("cellDisplacement"))
    .boundaryField();
```
Read the cell displacements from file /2:

```cpp
label fluidSideI = mesh.boundaryMesh().findPatchID(sail);

const polyPatch &fluidMesh = mesh.boundaryMesh()[fluidSideI];

labelList interfacePointLabels = mesh.boundaryMesh()[fluidSideI].meshPoints();

vectorField &mDisp = refCast<vectorField>(meshDisplacement[fluidSideI]);
```

We need now to find the address of the moving patch (identified by the variable “sail”) in the global cellDisplacement Boundary-Field.
Read the cell displacements from file /3:

```cpp
std::ifstream fin("Cell_Displ_out");
int pSize=fluidMesh.size();
vector move;
fin>>pSize;

double u;
double v;
double w;

List<vector> dispVals(fluidMesh.size());
```
Read the cell displacements from file /3:
Read the cell displacements from file /3:

```cpp
forAll(fluidMesh,i) {
    fin >> u;
    fin >> v;
    fin >> w;  // Values from the structural solver

dispVals[i][0]=u;
dispVals[i][1]=v;
dispVals[i][2]=w;

vector dd = dispVals[i];

mDisp[i]=dd;
}
```