Instructional workshop on OpenFOAM programming
LECTURE # 1

Pavanakumar Mohanamuraly

April 16, 2014

Outline

Recap of day 1

OpenFOAM mesh file structure

Finite volume mesh - *fvMesh*

Introduction to Field variables

OpenFOAM classes discussed so far

- Primitive types
- Dimensioned types
- **Info** stream output
- **argList** Command-line parsing
- **Time** object
- **IOdictionary** Input file parsing

Clarifications from yesterday

- Foam::MUST_READ reads only once during construction
- Only OpenFOAM-2.x version supports Foam::MUST_READ_IF_MODIFIED
- Foam::AUTO_WRITE is used to enable the write trigger for *IOobject*

OpenFOAM source file structure - story so-far

Example: solver.cpp

```
#include "fvCFD.H"

int main(int argc, char *argv[])
{
  #include "setRootCase.H"
  #include "createTime.H"

  /// Create mesh and fields
  /// ...

  while( runTime.loop() ) {
    Info << "Time : " << runTime.timeName() << "\n";
    /// ... solver stuff
    runTime.write(); /// The write trigger
  }

  return 0;
}
```

Behind the *.H* scene

Example: solver.cpp

```
/// #include "setRootCase.H"
Foam::argList args(argc, argv);
if (!args.checkRootCase()) {
  Foam::FatalError.exit();
}

/// #include "createTime.H"
Foam::Time runTime
(
  Foam::Time::controlDictName,
  args.rootPath(),
  args.caseName()
);
```

OpenFOAM classes - *IOList* for list file I/O

- Similar to *IOdictionary*
  - Does not use *keyword* → *value* style parsing
  - Array with I/O capability
- Extensively used by *mesh* object

```
labelIOList some = labelIOList
(
  IOobject
  (
    "some",
    "",
    runTime,
    IOobject::MUST_READ,
    IOobject::NO_WRITE,
    false /// Does not register with objectRegistry
  )
);
```

OpenFOAM classes - *IOList* for list file I/O

Input file format

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       labelList;
    location    "";
    object      some;
}

4
(
11
12
13
14
)
```

Hands on

- Create a labelListIO object
- Create a sample input ListIO file
- Read the file and print the read list

Warm up complete

Hands on - Mesh conversion and setup

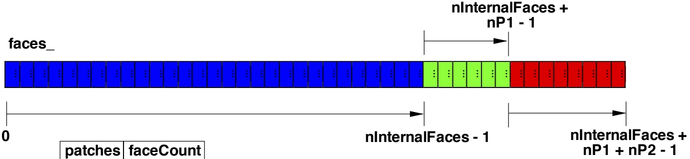
- Get a copy of the fluent mesh file
- Create a new case folder and controlDict
- Convert mesh using *fluentMeshToFoam*
- Remove unwanted files

polyMesh database

- *constant/polyMesh*
  - *points*
  - *faces*
  - *owner*
  - *neighbour*
  - *boundary*
- File format is face-based with polyhedral cell support
- Calculation of volume, area, centroid, etc, performed using just face information
- FOAM obtains all other connectivity information using this data alone

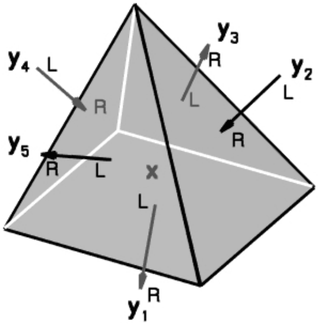
faces file

- Contains the list of node index forming faces
- Nodes ordering consistent with owner (left) and neighbor (right) cell
- faces segregated contiguously according to type



patches	faceCount
P1	nP1
P2	nP2

owner / *neighbour* file



Edge distance = $x - y_i$

faceID	Left cell	Right cell
1	x	y_1
2	y_2	x
3	x	y_3
4	y_4	x
5	x	y_5

L = left cell
R = right cell

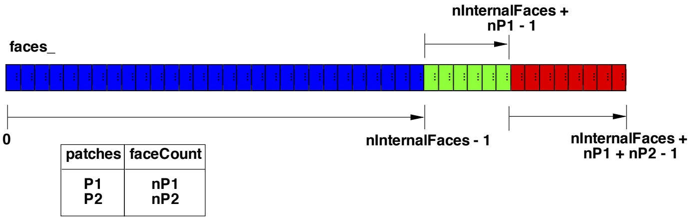
Left Cell (Owner Cell) The cell attached to a face such that the normal is pointing away from that cell.

Right Cell (Neighbour Cell) The cell attached to a face such that the normal is towards (inward) that cell.

owner/*neighbour* file

- The owner cell of all internal and patch faces are found in *owner* file
- The neighbour cell of all internal faces are found in *neighbour* file
- Remember that a boundary (patch) face will have only *owner* and no neighbour cell

boundary file



- *boundary* contains the patch boundaries of the mesh
- Each patch has the following attributes set
  - patch name
  - *type* - patch type
  - *nFaces* - Number of faces forming the patch
  - *startFace* - The starting index of the face

OpenFOAM basic patch types [1]

Selection Key	Description
`patch`	generic patch
`symmetryPlane`	plane of symmetry
`empty`	front and back planes of a 2D geometry
`wedge`	wedge front and back for an axi-symmetric geometry
`cyclic`	cyclic plane
`wall`	wall — used for wall functions in turbulent flows
`processor`	inter-processor boundary

[1] source: `http://openfoam.org/docs/user/boundaries.php`

Creating 2*d* and 1*d* meshes using *empty* patch

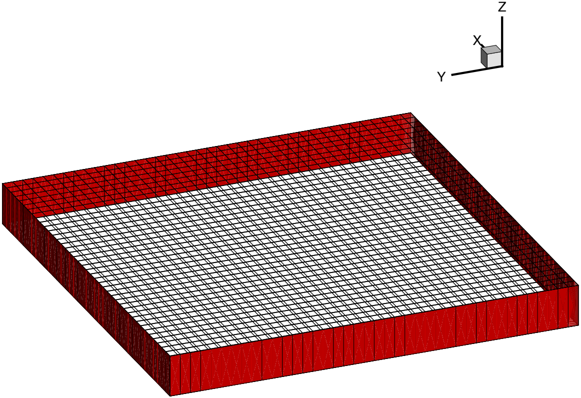
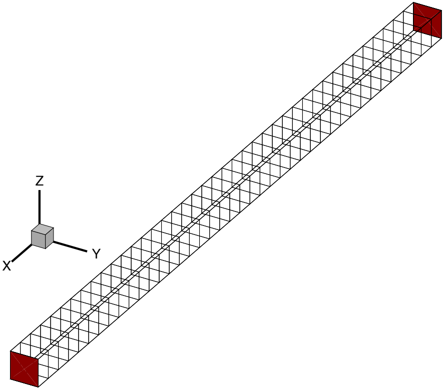


Figure : Sample 2*d* mesh in FOAM

- Extrude 2*d* grid one cell thick ($dz = 1.0$)
- Except the red patches make all others *empty* patches

Creating 2d and 1d meshes using *empty* patch

Figure : Sample 1d mesh in FOAM



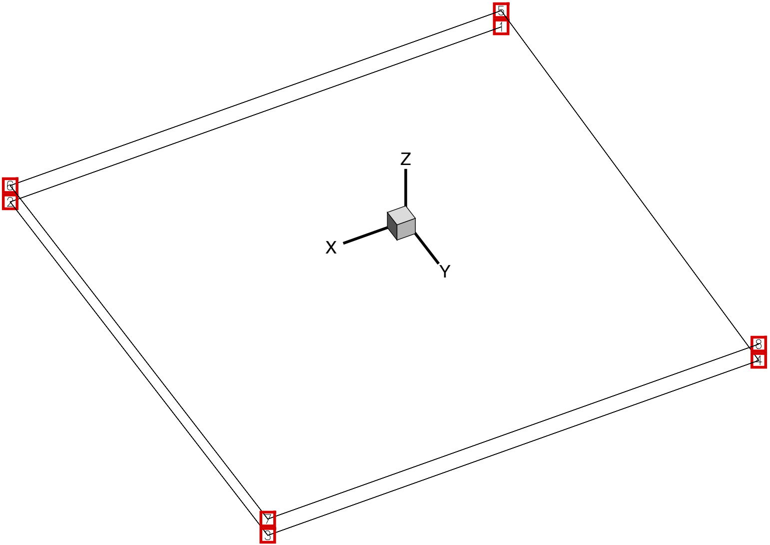
- Make $dy = 1.0$ and $dz = 1.0$ (one cell thick along y and z)
- Except the red patches all others are made *empty* patches

Hands on - 2d and 1d mesh

- Use the supplied blockMeshDict and generate the grids
- Visualize the generated mesh

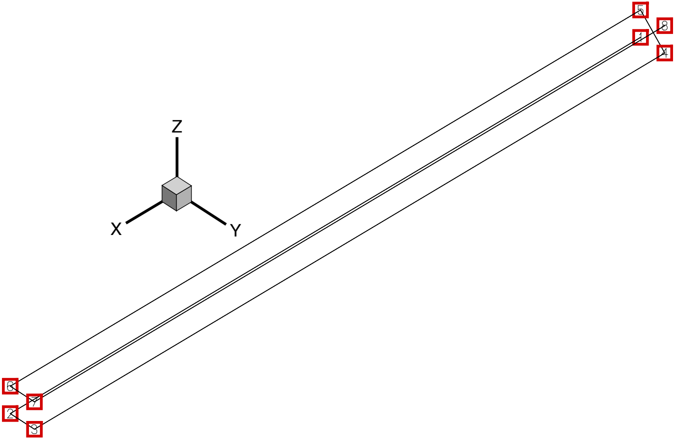
Hands on - 2d mesh

Figure : Sample 1d mesh in FOAM

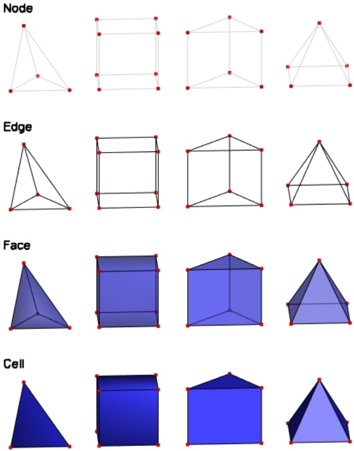


Hands on - 1d mesh

Figure : Sample 1d mesh in FOAM

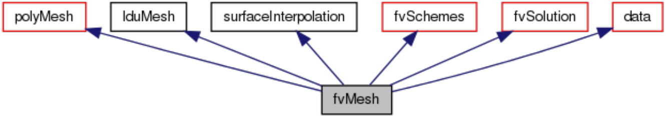


finite volume mesh data-structure



- Points $\rightarrow$ Edges $\rightarrow$ Faces $\rightarrow$ Cells
- FV operators require the above topology primitive information (and dependency)

fvMesh class overview



- *polyMesh* requires the complete polyMesh data-base for object construction
- *fvSchemes* and *fvSolution* classes requires dictionary files *fvSchemes* and *fvSolution* in the *system* folder for object construction

fvSchemes and *fvSolution*

- *fvSchemes* is the fundamental class, which registers all finite volume schemes
- Its constructor requires the scheme definition for the following operators
  - gradSchemes - The gradient scheme
  - divSchemes - The divergence scheme
  - laplacianSchemes - The laplacian scheme
- *fvSolution* does not require any solution scheme definition
- But requires the dictionary file to be present while constructing the object

Minimal *fvSchemes*

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

gradSchemes { default none;}
divSchemes { default none; }
laplacianSchemes { default none;}
```

Minimal *fvSolution*

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

Constructing *fvMesh* object

fvMesh constructor

```
Foam::fvMesh mesh
(
  Foam::IOobject
  (
    Foam::fvMesh::defaultRegion,
    runTime.timeName(),
    runTime,
    Foam::IOobject::MUST_READ
  )
);
```

- Simplified mesh creation by including header file *createMesh.H*

```
#include "createMesh.H"
```

Hands on - Complete *fvMesh* example

```
#include "fvCFD.H"

int main(int argc, char *argv[])
{
   #include "setRootCase.H"
   #include "createTime.H"
   #include "createMesh.H"
   return 0;
}
```

- Remember to create *fvSchemes* and *fvSolution* files (minimal)
- The *createTime.H* requires *controlDict* file

OpenFOAM classes - geometricField variables

- Class ties field to an fvMesh topology (can also be typedef volField, surfaceField, pointField)
  - volField - Volumetric field variable tied to the cell average value (centroid)
  - surfaceField - Field variable tied to faces of the domain (Left/Right)
  - pointField - Nodal field variables tied to mesh nodes/discrete points(lagrangian)
- Inherits all the operators of its corresponding field type
- Has dimension consistency checking
- Discrete operators are available to calculate gradients, divergence, etc

Field variables - Primitive operators

Table : Vector/Tensor primitive operations

Operator	FOAM notation
Addition	a+b
Inner Product	a & b
Cross Product	a ^ b
Outer Product	a * b
Vector magnitude	mag(a)
Transpose	A.T()
Determinant	det(A)

Useful fields in *fvMesh*

- mesh.C() - *volVectorField* storing cell centroids
- mesh.points() - *pointField* storing mesh nodes
- mesh.V() - *volScalarField* storing cell volumes
- mesh.Sf() - *surfaceVectorField* storing face area vector
- mesh.magSf() - *surfaceScalarField* storing face area magnitude
- mesh.Cf() - *surfaceVectorField* storing face centroid

Hands on - Create unit face normals

- Re-use the fvMesh example from previous hands-on
- Divide the mesh.Sf() and mesh.magSf() to obtain the unit normals at each face
- Remember that the resulting unit normal is a *surfaceVectorField*

Mesh connectivity information

- mesh.owner()/neighbour() - Access owner/neighbour information (labelList)
- mesh.pointPoints() - Node-to-node connectivity (labelListList)
- mesh.cellCells() - Cell-to-cell connectivity (labelListList)
- mesh.pointCells() - Node-to-cell connectivity (labelListList)

Looping through connectivity

- FOAM provides convenient way to loop through lists using *forAll* macro
- The syntax is as follows

```
forAll( object, loop_var )
{ /* object[loop_var] ... */ }
```

- loop_var is the loop variable and object is FOAM object

```
Foam::labelListList pp = mesh.pointPoints();
forAll( pp , i )
{
  /// ...
  forAll( pp[i] , j )
  {
    /// p[i][j] ...
  }
}
```

volume fields

- Field variables are mostly tied to the Time object as they vary with iteration
- Hence FOAM stores the field variables in time folders (0, dt, $2dt$, ...)
- A field has the following attributes
  - dimesionSet object
  - internalField values
  - boundaryField value/type
- Each field variables requires field specific boundary condition
- Despite additional patch definition made in *polyMesh*
- FOAM designed to work for segregated solvers
- The last week of lecture we will discuss coupled solvers

volume field object construction

```
volScalarField testFun
(
  IOobject
  (
    "testFun",
    runTime.timeName(),
    mesh,  /// Just to get objectRegistry
    IOobject::NO_READ, /// Read trigger
    IOobject::AUTO_WRITE /// Write trigger
  ),
  mesh, /// The mesh to which testFun is tied
  dimensionedScalar( "testFun", dimless , 0.0 )
);
```

Writing/Reading fields

- Fields with AUTO_WRITE attribute set can be written by simply invoking *runTime.write*()
- One can specifically write a particular field by invoking the *write*() member function

Hands on - Create 2^{nd} order polynomial field

- Take the dot product of cell centorid to get 2^{nd} order scalar field
- Write it to file and plot

End of Day 2