Instructional workshop on OpenFOAM programming
LECTURE # 2

Pavanakumar Mohanamuraly

April 19, 2014

Outline

Day 2

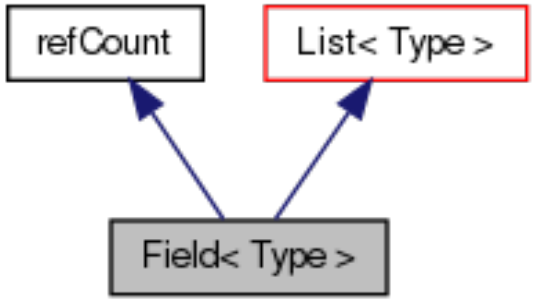
- IOList - List objects with I/O capability
- Mesh conversion from other formats
- *polyMesh* mesh database overview
- *fvMesh* and consequence of inheritance
- *empty* boundary type and $2d/1d$ meshes
- Basics of *GeometricField* class and field boundary conditions
- Creation of volume *GeometricField* fields

Clarifications from Day 2 : Hands on

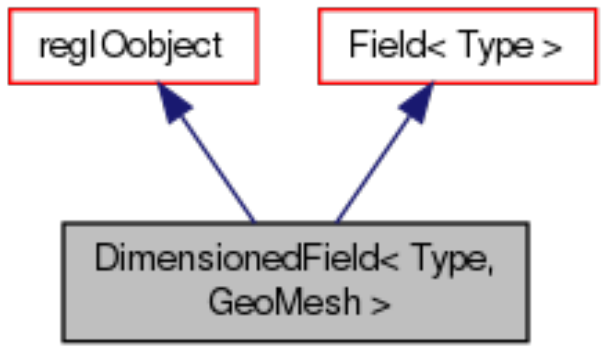
- Field variables can be written to current time folder by forcing *write*() member function
- *AUTO_WRITE* option will write the field variable to the file using the write frequency specified in control dictionary
- Remember that *AUTO_WRITE* will never write for the starting iteration

GeometricField class design

- *Field* class is a *List* class overloaded with arithmetic operators

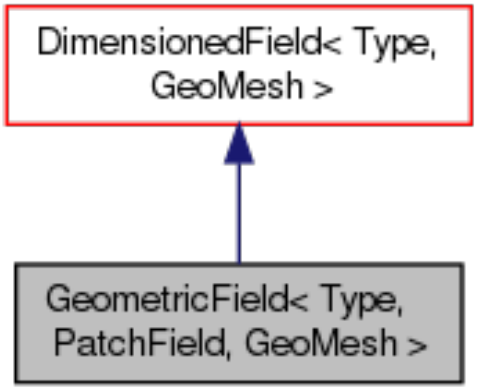


- Field with dimensions and associated with geometry type GeoMesh which is used to size the field and a reference to it is maintained
- Derived from IOobject for input-output and database registration



GeometricField class design

- Dimensioned Field with values associated with mesh patches



- *PatchField* is useful when enforcing boundary conditions
- Helpful to interpolate values for non-confirming patches (coupled simulations)

volumeFields

- Creating new volume field - *Day 2 hands on*
- Constructing volume fields reading input file

```
volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
    /// Missing dimensionedScalar argument
);
```

- Missing arg makes constructor throw error if file not found

Field p dictionary entry

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p;
}
dimensions [1 -1 -1 0 0 0 0]; /// N/m^2
internalField uniform 0.0;
boundaryField
{
  /// Patches should have same names as in
  /// polyMesh/boundary
  patch_1
  {
    type zeroGradient;
  }
  /// other patches
}
```

Field value specification

- A field requires specification of both internal and boundary fields
- Internal fields values can be specified in two ways
  - Use the *uniform* keyword to initialize to constant value
  - Use the *nonuniform* keyword and provide the list
- Boundary appears in two places in FOAM
  - Mesh boundary
  - Field boundary
- Every field created requires boundary conditions
- Operators basically uses this information during calculation

Field boundary conditions

Type	**Condition for field ϕ**	**Data to specify**
empty	-	-
fixedValue	$\phi = value$	*value*
fixedGradient	$\nabla\phi \cdot \hat{\mathbf{n}} = gradient$	*gradient*
zeroGradient	$\nabla\phi \cdot \hat{\mathbf{n}} = 0$	*none*
calculated	Boundary field ϕ derived from other fields	*none*
mixed	Robin BC	*refValue*, *refGradient*, *valueFraction*, *value*
directionMixed	Tensorial *valueFraction*	*refValue*, *refGradient*, *valueFraction*, *value*

- Possible to build derived types from the above basic types
- Will cover building derived BC's during week 2 lecture

Hands on - volume field reading

- Modify Day-2 volume field code - file read constructor
- Create a new dictionary entry for p (with field BC)
- Write the field and run it for the 3 cases

Warm up exercise !

FOAM class - *surfaceField*

- The internal field are simply the internal face values
- The boundary field is always *calculated* or *empty*

```
surfaceScalarField faceLRSum
(
  IOobject
  (
    "faceLRSum",
    runTime.timeName(),
    mesh,
    IOobject::NO_READ,
    IOobject::AUTO_WRITE
  ),
  mesh,
  dimensionedScalar( "dimless", dimless, 0.0 )
);
```

Hands on: *surfaceField* - Give me more !

Create a Left/Right cell value summation kernel

- Create a volumeScalarField p reading from file
- Create a surfaceScalarField *faceLRSum* without reading
- Loop over all internal faces and add the Left/Right cell values to the face
- Ignore boundary faces

Hints

- The dimensionSet object of "p" can be obtained using *p.dimensions*()
- Use *forAll* to loop over *mesh.neighbours*()
- *mesh.owner*()[i] and *mesh.neighbour*()[i] are the left/right cell of i^{th} face

FOAM class - *pointField*

pointFieldFwd.H

```
40 namespace Foam
41 {
42   typedef vectorField pointField;
43 }
```

vectorField.H

```
47
48 typedef Field<vector> vectorField;
49
```

FOAM class - *pointField* creation

- Not same as volume or surface fields
- File I/O using IOobject not possible
- Manually write using IOList object
- Size specified manually while construction

```
pointField pField( mesh.points().size() );
```

- Or use the List constructor

```
vectorList some = vectorList(myDict.lookup("some"));
pointField pField( some );
```

Hands on - *pointField* reading from file

- Create an *IOdictionary* named *fieldDict* in *runTime.timeName()* folder

```
IOdictionary fieldDict
(
    IOobject
    (
        "field",
        runTime.timeName(),
        runTime,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    )
);
```

- Read the vector list from file using a *List* object

```
List<vector> fieldList = List<vector>( fieldDict.
    lookup("field") );
```

Hands on - *pointField* reading from file

- Transfer the list to the *pointField* using *Xfer*

```
pointField field( fieldList.xfer() );
```

- Now try to print the List sizes before and after *xfer*

Short digression - *Xfer* $< T >$ class

- Handy class to copy or transfer data from one container to another
- The wrapped object of type $< T >$ must implement
  - A *transfer*() method and
  - An *operator* = () copy method
- Contents of the *Xfer* object can be transferred unconditionally
- Simplifies defining constructors or methods in other classes with mixed transfer/copy semantics without requiring $2N$ different versions

Accessing boundary field data

- Essential when manipulating *boundaryField*
- User defined boundary conditions
- User defined operators

```
/// First loop over all patches
forAll( field.boundaryField(), ipatch ) {
  /// each face in the the patch
  forAll( field.boundaryField()[ipatch], iface ) {
    /// ...
  }
}
```

- *type*() member gives field BC type string
- Left cell data (internalField) can be accessed using *patchInternalField*() member

Hands on - Complete faceLRSum

- Modify previous example by adding the following
  - Loop over each boundary field
  - Assign the left cell value as the average value of boundary face

Hint

- *patchInternalField*() of *p* will get all the left cell values
- Assign it to the *boundaryField*()[*ipatch*] of field *faceLRSum*

boundaryMesh and *boundaryField*

- We mentioned that FOAM has boundary specification in two places
  - Mesh data
  - Field data
- Field boundary can be accessed using *boundaryField*() member function
- Mesh boundary can be accessed using *boundaryMesh*() member function
- *type*() and *forAll*(*mesh.boundaryMesh*(), *ipatch*) remain the same as field type

Hands on - Bummer !

- Write code to print the number of faces of each patch in *boundaryMesh* and *boundaryField*
- Run this for the $3d$, $2d$ and $1d$ mesh we have created
- What is expected ? What is that you get ?

Hint

- The number of face can be obtained using the *size*() member function
- Use *Info* to print it on the screen

boundaryMesh and *boundaryField* - Watch out !

- *mesh.boundaryMesh*() empty patches have same face count as in *polyMesh/boundary* file
- *field.boundaryField*() empty patches have zero face count

End of Day 3 and Week 1