Primarily based on **The finite volume method in computational fluid dynamics - an advanced introduction with OpenFOAM and Matlab**.

**Chapter 6 & 7**

1. pointField(vectorField) - a list of all mesh vertices

// i.e.

4678 // total 4678 vertices

(

(-32 -16 0.9377382715)

(-32 -32 0.9377382715)

(-32 -16.30818205 0.9377382715)

/.../

)

1. faceList - a list of identities of the points defining the faces of the mesh

const faceList& fs = faces();

faceList是包含数据类型为labelList的链表, labelList中便储存了构成face的vertex的索引, i.e.

const labelList& f = fs[facei];

// i.e.

8938 // total 8938 faces

(

4(2 2341 2583 244) // a labelList, label of vertex

4(81 244 2583 2420)

4(3 2342 2684 345)

4(244 345 2684 2583)

4(4 2343 2785 446)

/.../

)

1. 数据类型label可以理解为标签, 有点类似于数组的索引

for (label pi = 0; pi < nPoints; pi++){ ... }

1. faceOwner()/faceNeighbour() - 返回internal face所对应的owner cell index或neighbour cell index, 数据类型是labelList

const labelList& own = faceOwner();

cEst[own[facei]] += fCtrs[facei]; //own[facei] represents owner index.

// i.e. face owner, how many faces for how many owners.

8938

(

0 // element label

0

1

/.../

)

1. some functions in primitiveMesh class

// PS: The primitiveMesh does not recognise boundaries or domain interfaces.

const labelListList & cellCells() const

const labelListList & pointCells() const

const cellList & cells() const

const vectorField & cellCentres() const

const vectorField & faceCentres() const

const scalarField & cellVolumes() const

const vectorField & faceAreas() const

1. the polyMesh class introduces the handling of boundary definition and information as blow

const labelUList & owner () const //Internal face owner.

const labelUList & neighbour () const //Internal face neighbour.

// Return old-time cell volumes.

const DimensionedField< scalar, volMesh > & V0 () const

// Return old-old-time cell volumes.

const DimensionedField< scalar, volMesh > & V00 () const

// Return face deltas as surfaceVectorField

tmp< surfaceVectorField > delta () const

OpenFOAM® decomposes the boundary mesh into patches stored in a list

designated by **polyPatchList(List<polyPatch>)** under the class **polyBoundaryMesh**. Then, similar to interior mesh, a specialized class denoted by **fvBoundaryMesh** is derived from

**polyBoundaryMesh** that inherits its functionalities and expands on it to include specific functions and data needed for finite-volume discretization. As for *internal(boundary?)* discretization, a similar hierarchy of classes composed of **primitivePatch**, **polyPatch**, and **fvPatch** is defined for boundary discretization. These classes are specific for the boundary mesh and contain the geometric information of each boundary. But it is the **fvPatch** that is used to implement the boundary conditions during the finite volume discretization.

In a similar way the **fvMesh** is used to access all mesh functionalities. Thus, it is mainly with **fvMesh** and **fvPatch** that the discretization classes and functions interact.

1. Boundary definition in polyMesh

#boundary patch name

{

type #patchtype; // wall, patch, empty, etc.

nFaces #number of face in patch set;

startFace #starting face index for patch;

}

1. Mesh manipulation

// read the element centroids

volVectorField C = mesh.C();

// element volumes

volScalarField V = mesh.V();

// surface centroids

surfaceVectorField Cf = mesh.Cf();

1. Loop over the cell volumes

const DimensionedField< scalar, volMesh >& cellVolumes = mesh.V();

forAll(cellVolumes, cellI)

{

scalar cellVolume = cellVolumes[cellI];

}

1. Access boundary patches

const fvBoundaryMesh& boundaryMesh = mesh.boundary();

forAll(boundaryMesh, patchI)

{

const fvPatch& patch = boundaryMesh[patchI];

forAll(patch, faceI)

{

vector faceNormal = patch.Sf()[faceI];

scalar faceArea = patch.magSf()[faceI];

// patch.nf() returns a tmp<Type>.

vector unitFaceNormal = patch.nf()()[faceI];

vector faceCenter = patch.Cf()[faceI];

label owner = patch.faceCells()[faceI];

}

} // access: mesh -> fvBoundaryMesh -> fvPatch

1. Template class GeometricField<Type,...>

Each data defined using this class is strictly related to the mesh dimensions, both for the boundaries and interior domain.

GeometricField<Type,..>的数据结构

* volField<type>
* surfaceField<type>
* pointField<type>

Properties GeometricField<type,...> class inherits

* Dimensions（单位）
* InternalField & BoundaryField (GeometricBoundaryField, fvPatchField)

Mesh - each GeometricField contains a reference to the corresponding mesh

* Time Values and Previous Values

1. GeometricField Example((i)field name, (ii)field dimension, (iii)initialization, and (iv)boundary conditions)

volScalarField T

(

IOobject

(

"T",

runTime.timeName(),

mesh,

IOobject::NO\_READ,

IOobject::AUTO\_WRITE

),

mesh,

dimensionedScalar("DTVol", dimensionSet(0,0,0,1,0,0,0), 300.0),

"zeroGradient"

);

1. Access InternalField

forAll(T.internalField(), cellI)

{

scalar cellT = T.internalField()[cellI];

}

1. Access BoundaryField

const volVectorField::GeometricBoundaryField& UBoundaryList = U.boundaryField();

// boundary access

forAll(UBoundaryList, patchI)

{

const fvPatchField<vector>& fieldBoundary = UBoundaryList[patchI];

forAll(fieldBoundary, faceI)

{

vector faceU = fieldBoundary[faceI];

}

}

// boundary access in a more compact form

forAll(T.boundaryField(), patchI)

{

forAll(T.boundaryField()[patchI], faceI)

{

scalar faceT = T.boundaryField()[patchI][faceI];

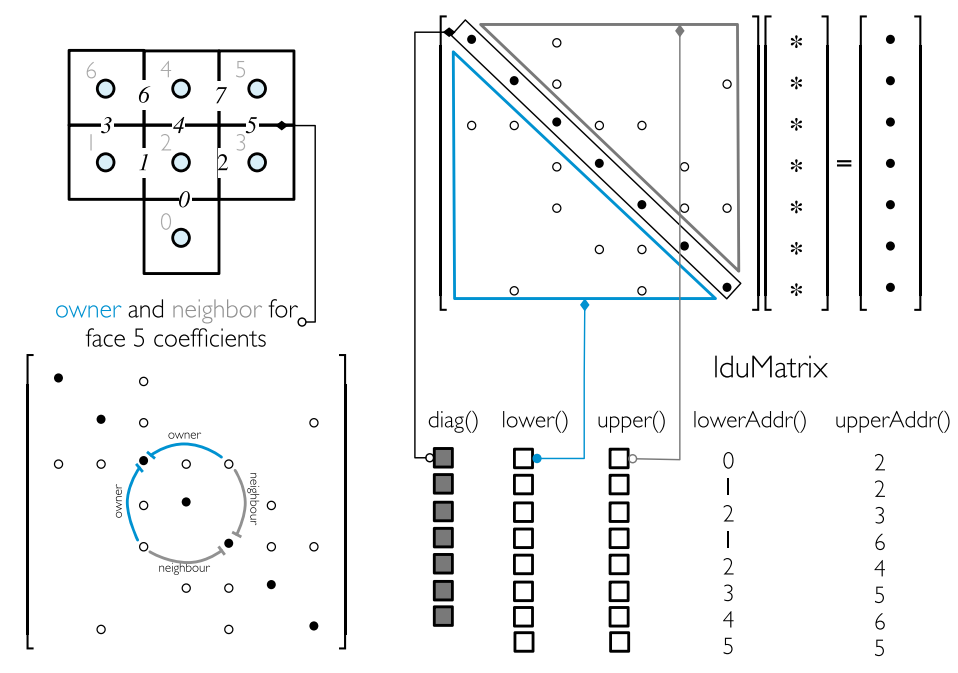
}

}

1. lduAddressing

face addressing scheme - coefficients are stored following the interior face ordering

5 arrays - diag(), lower(), upper(), lowerAddr() - owner, upperAddr() - neighbour



// summation of the off-diagonal coefficients

for (label faceI=0; faceI<l.size(); faceI++)

{

// this is a mistake!

ac[l[faceI]] -= Lower[faceI];

ac[u[faceI]] -= Upper[faceI];

}

**Chapter 8**

1. 类

surfaceMesh， volMesh

deltaCoeffs -

The “deltaCoeffs.internalField()” represents the internal mesh **gDiff** field.

In OpenFOAM the boundary coefficients are stored in “**interalCoeffs**”() and “**boundaryCoeffs**”().

1. laplacianSchemes

The “fvm::laplacian(gamma,phi)” returns instead an **fvMatrix** of coefficients, which is added to the left hand side of the system of equations, in addition to a field containing the non-orthogonal terms (based on a right-now explicit gradient) that is added to the right hand side of the system.

laplacianSchemes

{

laplacian(gamma,phi) Gauss linear corrected;

}

**Gauss** (the only available choice) defines the standard Gauss discretization, **linear** refers to the type of interpolation used for calculating the diffusivity gamma at the face, and **corrected** describes the kind of non-orthogonal correction.

1. gradSchemes

gradSchemes

{

default none;

grad(phi) Gauss;

}

// interpolation schemes for gradient calculation

interpolationSchemes

{

interpolate(phi) linear;

// interpolate(phi) skewCorrected linear

}

// a more compact form for gradient schemes

gradSchemes

{

default none;

grad(phi) Gauss linear;

}

**Chapter 11**

Computational Pointer:

* How the convection schemes was realized in OpenFOAM?
* How to realize fvc::div & fvm::div operator differently?
* The calculation of the divergence of a field involves the following three steps:

1. Evaluating values at the faces of the element.

2. Multiplying the value at the face with the mass flux at the face (i.e., **faceFlux**).

3. Summing the contributions of all cell faces and dividing the sum by the cell volume.

**Boundary Conditions**

<https://www.cfdsupport.com/OpenFOAM-Training-by-CFD-Support/node93.html>

**OpenFOAM的湍流模型构架**

* BasicTurbulenceModel是TurbulenceModel的原则类（既是基类又是模板参数）

BasicTurbulenceModel的可选参数为incompressibleTurbulenceModel和compressibleTurbulenceModel，它们继承自turbulenceModel.

* TurbulenceModel偏特化后派生出IncompressibleTurbulenceModel和CompressibleTurbuenceModel，它们只有一个模板参数TransportModel

完全特化incompressibleTurbulenceModel和CompressibleTurbuenceModel：

incompressibleTurbulenceModel<transportModel> 等价于incompressible::turbulenceModel

ThermalDiffusivity<CompressibleTurbuenceModel<fluidThermo>> 等价于compressible::turbulenceModel

fluidThermo用于描述可压缩流体的热力学模型，继承自basicThermo，派生出两个类：基于可压缩性描述的类psiThermo和基于密度描述的类rhoThermo.

* BasicTurbulenceModel是laminarModel、RASModel和LESModel的原则类

BasicTurbulenceModel的可选参数为incompressible::turbulenceModel和compressible::turbulenceModel.

PS: 当实例化可压缩的RASModel或LESModel时，BasicTurbulenceModel实际为EddyDiffusivity<compressible::turbulenceModel>.

如下：

|  |
| --- |
| *// $FOAM\_SRC/TurbulenceModels/compressible/turbulentFluidThermoModels/turbulentFluidThermoModels.C*  makeBaseTurbulenceModel(  geometricOneField,  volScalarField,  compressibleTurbulenceModel,  CompressibleTurbulenceModel,  ThermalDiffusivity,  fluidThermo);  *// 宏*  *// $FOAM\_SRC/TurbulenceModel/compressible/turbulentFluidThermoModels/makeTurbulenceModel.H*  *#define makeTurbulenceModelTypes( \*  *Alpha, Rho, baseModel, BaseModel, TDModel, Transport) \*  *namespace Foam \*  *{ \*  *typedef TDModel<BaseModel<Transport>> \*  *Transport##BaseModel; \*  *typedef laminarModel<Transport##BaseModel> \*  *laminar##Transport##BaseModel; \*  *typedef RASModel<EddyDiffusivity<Transport##BaseModel>> \*  *RAS##Transport##BaseModel; \*  *typedef LESModel<EddyDiffusivity<Transport##BaseModel>> \*  *LES##Transport##BaseModel; \*  *}* |

宏命令展开后，会定义4个类，分别是：

* fluidThermoCompressibleTurbulenceModel
* laminarfluidThermoCompressibleTurbulenceModel
* RASfluidThermoCompressibleTurbulenceModel
* LESfluidThermoCompressibleTurbulenceModel

OpenFOAM通过一个返回autoPtr<incompressible::turbulenceModel>(或autoPtr<compressible::turbulenceModel>)的selector，实现了运行时选择，从字典文件读取关键字并构造对应的湍流模型实例.

* 所有具体的流动模型都可以看作是由laminarModel、RASModel或LESModel派生出来的类

如Stokes:

|  |
| --- |
| *//$FOAM\_SRC/TurbulenceModels/turbulenceModels/laminar/Stokes/Stokes.H*  template<class BasicTurbulenceModel>  class Stokes  :  public linearViscousStress<laminarModel<BasicTurbulenceModel>>  {  *// ...* |

linearViscousStress也是基于原则设计的类，继承自模板参数laminarModel.

|  |
| --- |
| *//$FOAM\_SRC/TurbulenceModels/turbulenceModels/linearViscousStress/linearViscousStress.H*  template<class BasicTurbulenceModel>  class linearViscousStress  :  public BasicTurbulenceModel  {  *// ...* |

**icoFoam**

adjustPhi()

当压力入口和出口边界条件都是Neumann边界条件时可能会遇到两个问题：

1. 方程系数矩阵奇异，有无穷多个解；
2. 方程相容性问题，需要adjustPhi();

adjusts the inlet and outlet fluxes to obey continuity, which is necessary for creating a well-posed problem where a solution for pressure exists.

**其他**

* Remember, **nNonOrthogonalCorrectors** is used to improve the gradient computation due to mesh quality.
* **geometricField<type>**

1. Internal Field: simplely a **Field<type>** i.e. **scalarField**, **vectorField** etc.
2. Boundary Field: **GeometricBoundaryField** →a Field is defined for the faces of each patch and a Field is defined for the patches of the boundary. This is then a field of fields, stored within an object of the **FieldField<Type>** class. A reference to the **fvBoundaryMesh** is also stored.
3. Diemensions
4. Old Values: The **geometricField<Type>** will store references to stored fields from the previous, or old, time step and its previous, or old-old, time step where necessary.
5. Previous Iteration Values

**geometricField<type>** equivalents by *typedef* declaration:

volField<type>, surfaceField<type>, pointField<type>

* **sampleDict**

1. *sampleDict* is read by the utility *postProcess*. This utility sample field

data (points, lines or surfaces).

1. For details on how to sample, see p117 on *OpenFOAM Tutorials-Wolf Dynamics.*

* **VoF Model**

1. The set of equations of the volume-of-fluid method is in principle equivalent with the set of equations of the homogeneous model.
2. The homogeneous model assumes the phases move with the same velocity(mechanical equilibrium), the relative velocity is zero.
3. In VoF, the velocity is center-of-mass velocity, and the density is mixture density.

PS: 存疑，Description and utilization of interFoam multiphase solver 定义 effective fluid velocity 为 weighted average velocity.

1. But in homogeneous model, the center-of-mass velocity equals to the phase velocity(physical velocity, not volume-average velocity).
2. Two inmiscible fluids are considered as one effective fluid throughout the domain.