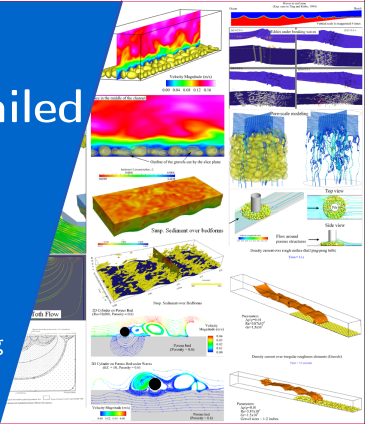
CE597D Special Topics, Fall, 2014

Chapter 2, Part 4: Detailed walk-through of OpenFOAM®

Xiaofeng Liu, Ph.D., P.E.
Assistant Professor
Department of Civil and Environmental Engineering
Pennsylvania State University
xliu@engr.psu.edu


Velocity Magnitude (m/s)
Eddies under breaking waves
Pore-scale modeling
Outline of the gravels cut by the slice plane
Susp. Sediment over bedforms
Top view
Side view
Flow around porous structures
Toth Flow
2D Cylinder on Porous Bed
Porous bed (Porosity = 0.6)
2D Cylinder on Porous Bed under Waves
Density current over irregular roughness elements (Gravels)

Outline

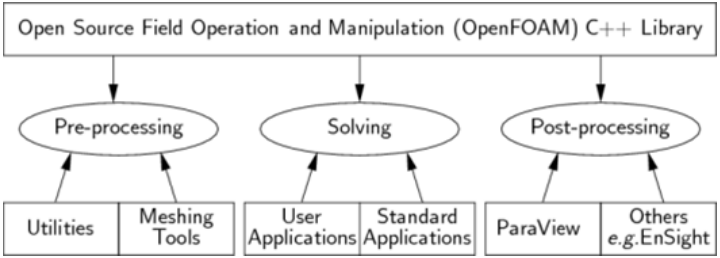
Level of users

- Beginner level:
  - A brief introduction of OpenFOAM®
  - Demonstration of OpenFOAM® usages
  - Installation of OpenFOAM®
  - Hands-on exercises for the basics
  - Detailed walk-through of the code
  - ...
- Intermediate and advanced levels (later):
  - Demonstration of OpenFOAM® development
    - Solver
    - Boundary conditions
    - Tools
  - Hands-on exercise
  - Discussion of specific applications

What we have done so far:

- *A brief introduction of OpenFOAM®*
- *Sample applications of OpenFOAM®*
- *Installation of OpenFOAM®*
  - *Brief introduction of Linux system*
  - *Installation of OpenFOAM®*
  - *Test the installation*
  - *Initial browsing of the OpenFOAM® code*
- *Hands-on exercises for the basics*
- Detailed walk-through of the code

Basic code structure of OpenFOAM®



Structure of the OpenFOAM® platform, OpenFOAM® User Guide

Basic code structure of OpenFOAM®

- Three code categories:
  - Solvers (alias sol)
    - Basic CFD: laplacianFoam, potentialFoam, scalarTransportFoam, etc.
    - Incompressible flow: icoFoam, nonNewtonianIcoFoam, pisoFoam, simpleFoam, pimpleFoam, pimpleDyMFoam, etc.
    - Compressible flow: rhoCentralFoam, rhoPimpleFoam, etc.
    - Multiphase flow: bubbleFoam, interFoam, settlingFoam, etc.
    - Combustion
    - Heat transfer
    - Stress analysis
    - ...
  - Utilities (alias util): pre-processing, post-processing, mesh processing
  - Libraries (alias lib, src)
    - Turbulence: RANS, LES
    - Transport models (rheology)
    - Lagrangian particle tracking
    - Thermal dynamics
    - Mesh motion
    - Parallel computation
    - Numerical methods
    - ...
- The underlying source code (alias src)

Basic code structure of OpenFOAM®

Demonstration of code walk-through

Basic code structure of OpenFOAM®

So how well should I know the details?

- Fact: OpenFOAM® system is very complicated
- The details you should know depend on how you will use OpenFOAM®
  - Basic usage: Just run some simulations with existing solvers
  - Intermediate: Make some minor changes to suit your needs
  - Advanced: Want to make major changes, create new solvers, libraries, utilities, etc.

Basic code structure of OpenFOAM®

Basic CFD Elements

- Basic Elements:
  - Mesh: Discrete representation of physical domain
  - Field variables: velocity, pressure, concentration, etc
  - Discretization of equations: how to discretize the governing equations (such as advection-diffusion equation, N-S equations)
  - Solution of linear system: $[A][x] = [b]$
- OpenFOAM® uses C++ Object-Oriented programming
  - As a user: you should be aware of this
  - As a developer: you should know the details

Basic code structure of OpenFOAM®

A Brief Overview of Object-Oriented Programming

- Class: protect your data
  - Data and operations are grouped together
  - Interface: set of available operations

Account
Withdraw
Deposit

```
class Account {
  public:
    void withdraw(fload amount);
    void deposit(float amount);
  private:
    float balance;
);
```

- Go to a simple example of class definition in OpenFOAM® : Class line

```
[xxx@xxxx ] cat
$FOAM_SRC/OpenFOAM/meshes/primitiveShapes/line/line.H
```

Basic code structure of OpenFOAM®

A Brief Overview of Object-Oriented Programming

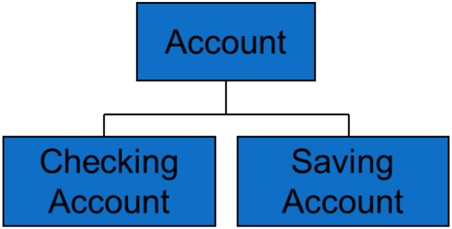
- Object: instance of a class
- Classes reflect concepts, objects reflect instances that embody those concepts
- Example: define and use an ‘Account‘ object:

```
Account my_account;
my_account.deposit(100.1);
my_account.withdraw(10);
```

Basic code structure of OpenFOAM®

A Brief Overview of Object-Oriented Programming

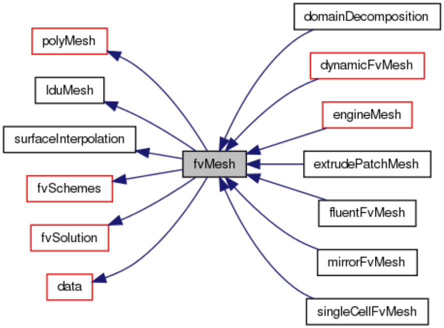
- Inheritance ("IS " relationship)
  - A class which is a subtype of a more general class is said to be inherited from it.
  - The sub-class inherits the base class data and member functions
  - A sub-class has all data of its base-class plus its own
  - A sub-class has all member functions of its base class (with changes) plus its own



Basic code structure of OpenFOAM®

A Brief Overview of Object-Oriented Programming

- Inheritance ("IS " relationship)

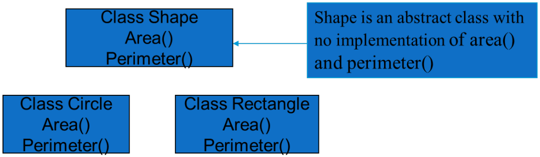


- Doxygen can be run on the source code of OpenFOAM® to automatically generate documentation of the classes
- I usually go the OpenFOAM® source code documentation webpage and search: http://www.openfoam.org/docs/cpp/

Basic code structure of OpenFOAM®

A Brief Overview of Object-Oriented Programming

- Polymorphism: polymorphism is the ability of objects belonging to different classes to respond to method calls of the same name, each one according to an appropriate type-specific behavior



Circle and Rectangle are concrete classes with their own separate implementations of the methods Area() and Perimeter()

Basic code structure of OpenFOAM®

A Brief Overview of Object-Oriented Programming

- Types of polymorphism:
  - Overloading of names, e.g., sqrt(int a), sqrt (float a)
  - Overloading of operators (e.g. integer + integer, complex + complex)
- Examples in OpenFOAM®
  - Example classes have the same member function name but with different parameter types
  - The operator '+' are defined differently for the class scalarField and vectorField
    - scalarField + scalarField
    - vectorField + vectorField

Basic code structure of OpenFOAM®

A Brief Overview of Object-Oriented Programming

- Templates:
  - A feature of the C++ programming language that allow functions and classes to operate with different types.
  - In the definition of a template, only generic data type is needed.
  - The data type is specified at compilation time.
  - This allows a function or class to work on many different data types without being rewritten for each one.
  - Example:

```
template <class a_type, class b_type, ...>
class a_class
{
...
a_type a_var;
b_type b_var;
...
};
```

Basic code structure of OpenFOAM®

- Examples of class templating in OpenFOAM®
  - UList: A 1D array class $FOAM_SRC/OpenFOAM/containers/Lists/UList

```
template<class T>
class UList
{
    // Private data

        //- Number of elements in UList.
        label size_;

        //- Vector of values of type T.
        T* __restrict__ v_;
...
}
```

Basic code structure of OpenFOAM®

- Examples of function templating in OpenFOAM®
  - 1D interpolation: $FOAM_SRC/OpenFOAM/interpolations/interpolateXY

```
template<class Type>
Field<Type> interpolateXY
(
    const scalarField& xNew,
    const scalarField& xOld,
    const Field<Type>& yOld
)
{
    Field<Type> yNew(xNew.size());

    forAll(xNew, i)
    {
        yNew[i] = interpolateXY(xNew[i], xOld, yOld);
    }
    return yNew;
}
```

Basic code structure of OpenFOAM®

Templating can be used in very complicated way to achieve advanced functionalities. For example, one of the most important classes in OpenFOAM® is used to define a field variable, i.e., the GeometricField class:

$FOAM_SRC/OpenFOAM/fields/GeometricFields/GeometricField

```
template<class Type, template<class> class PatchField, class GeoMesh>
class GeometricField
:
    public DimensionedField<Type, GeoMesh>
```

Basic code structure of OpenFOAM®

Last but not least: *namespace*

- Namespace is designed to solve problems with global name clashes.

```
// some_lib.h
namespace lib_one {
   int func(int);
   class point { ... };
}

// another_lib.h
namespace lib_two {
   int func(int);
   class point { ... };
}
```

  We can distinguish them by `lib_one::point` and `lib_two::point`. Or define a scope:

```
using namespace lib_one {
   point a_point;
}
```

- namespace can be nested.

Basic code structure of OpenFOAM®

Last but not least: *namespace*

- OpenFOAM® defines many namespaces, which can be viewed at: `http://foam.sourceforge.net/docs/cpp/namespaces.html`
- Examples: `Foam`, `fvc`, `fvm`, `incompressible`, `compressible`
- `Foam` is the general namespace defined in OpenFOAM® to isolate the names from outside (to avoid possible clashes with others).
- `fvc` and `fvm`: two name spaces for explicit and implicit discretizations of differential operators. Example, `fvc::div(...)` and `fvm::div(...)`.
- `incompressible` and `compressible`: two name spaces for two different type of fluids. For example, we have (also note the namespace nesting):

  ```
  Foam::compressible::RASModels::kEpsilon
  Foam::incompressible::RASModels::kEpsilon
  ```

Basic code structure of OpenFOAM®

Further readings on C++ and object-oriented programming:

- *Teach Yourself C++ in 10 minutes*,J. Liberty, SAMS 1999.
- *C++ - How to program*, Deitel & Deitel, Prentice Hall, 2001.
- *Object Oriented Programming with C++*, David Parson, Letts Educational, London 1997.
- *Solving PDEs in C++*

Important classes in OpenFOAM®

Five basic classes:

- Time and space (mesh): Time, polyMesh, fvMesh
- Field (data): Field, DimensionedField, and GeometricField
- Boundary conditions: fvPatchField and derived classes
- Finite volume discretization: classes in fvc and fvm namespaces
- Sparse matrices: lduMatrix, fvMatrix and linear solvers

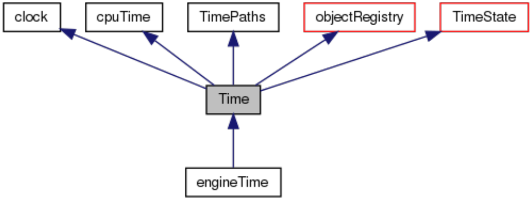
Reference: H. Jasak's OF workshop tutorial slides.

Classes for time

The time class *Time*: controls time during OpenFOAM® simulation.

- Definition in `$FOAM_SRC/OpenFOAM/db/Time`
- Upon construction, it reads in the information in the `system/controlDict` file
- Controls time step marching, time step adjustment, timing for write, etc.
- Construct dynamically loaded libraries in `system/controlDict` file
  - `libs ("libMyLibrary.so");`
- How to create a Time object? Usually include the `createTime.H` file

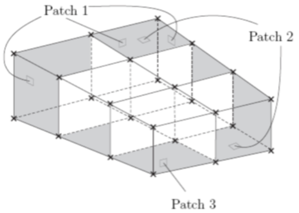
```
Foam::Info<< "Create time\n" << Foam::endl;

Foam::Time runTime(Foam::Time::controlDictName, args);
```

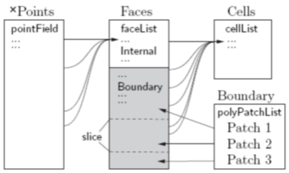


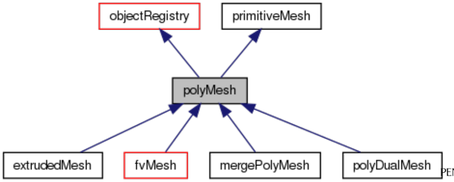
Classes for mesh

The most basic mesh class *polyMesh*: Mesh consisting of general polyhedral cells

- Points, Faces, Cells
- Boundary
- Definition in `$FOAM_SRC/OpenFOAM/meshes/polyMesh`



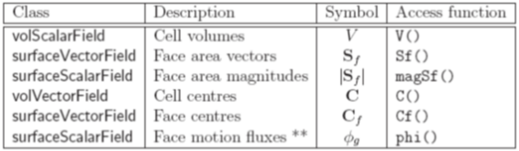




Classes for mesh

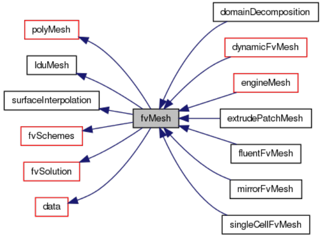
Mesh for finite volume: *fvMesh*

- Stores additional data for FVM discretization
- Definition in `$FOAM_SRC/finiteVolume/fvMesh`



Class	Description	Symbol	Access function
volScalarField	Cell volumes	V	`V()`
surfaceVectorField	Face area vectors	$\mathbf{S}_f$	`Sf()`
surfaceScalarField	Face area magnitudes	$\lvert\mathbf{S}_f\rvert$	`magSf()`
volVectorField	Cell centres	$\mathbf{C}$	`C()`
surfaceVectorField	Face centres	$\mathbf{C}_f$	`Cf()`
surfaceScalarField	Face motion fluxes **	ϕ_g	`phi()`

Table 2.1: fvMesh stored data.



Classes for mesh

Mesh for finite volume: *fvMesh*

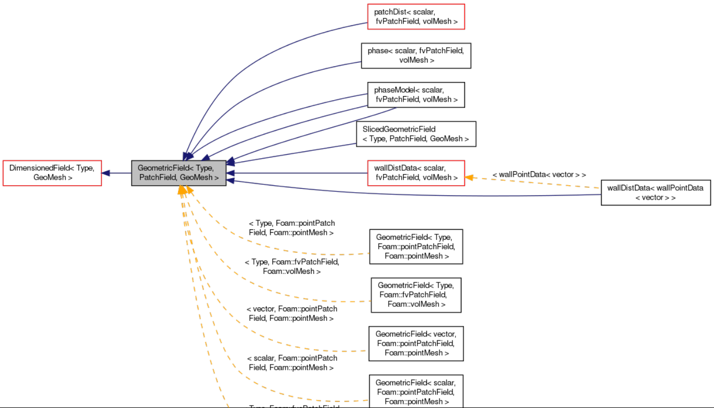
- How to create the mesh in the code? Usually include the `createMesh.H` file.

```
Foam::Info
    << "Create mesh for time = "
    << runTime.timeName() << Foam::nl << Foam::endl;

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

Classes for field data

Field data class *GeometricField*:

- Field = data on a *fvMesh*
  - Definition in `$FOAM_SRC/OpenFOAM/fields/GeometricFields/GeometricField`
  - Internal field and boundary field
  - Field is templated with scalar, vector, or tensor.
    - `typedef GeometricField<scalar, fvPatchField, volMesh> volScalarField`
    - `typedef GeometricField<vector, fvPatchField, volMesh> volVectorField`
    - `typedef GeometricField<tensor, fvPatchField, volMesh> volTensorField`


patchDist< scalar, fvPatchField, volMesh >
phase< scalar, fvPatchField, volMesh >
phaseModel< scalar, fvPatchField, volMesh >
SlicedGeometricField < Type, PatchField, GeoMesh >
DimensionedField< Type, GeoMesh >
GeometricField< Type, PatchField, GeoMesh >
wallDistData< scalar, fvPatchField, volMesh >
< wallPointData< vector > >
wallDistData< wallPointData < vector > >
< Type, Foam::pointPatch Field, Foam::pointMesh >
GeometricField< Type, Foam::pointPatchField, Foam::pointMesh >
< Type, Foam::fvPatchField, Foam::volMesh >
GeometricField< Type, Foam::fvPatchField, Foam::volMesh >
< vector, Foam::pointPatch Field, Foam::pointMesh >
GeometricField< vector, Foam::pointPatchField, Foam::pointMesh >
< scalar, Foam::pointPatch Field, Foam::pointMesh >
GeometricField< scalar, Foam::pointPatchField, Foam::pointMesh >

Classes for field data

Field data class *GeometricField*:

- Dimensions (e.g., velocity m/s)
- Old values at previous time steps
- How to create a field variable in the code? Example in a typical `createFields.H` file:

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

How are PDEs solved in OpenFOAM?

Equations are essentially a group of operations on fields:

- Mathematical Language for Partial differential equation (PDE)

$$\frac{\partial k}{\partial t} + \nabla \cdot (\mathbf{u}k) - \nabla \cdot [(\nu + \nu_t)\nabla k] = \nu_t \left[\frac{1}{2}\left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right)\right]^2 - \frac{\epsilon_0}{k_0} k$$

- Pseudo-Natural Language in OpenFOAM®

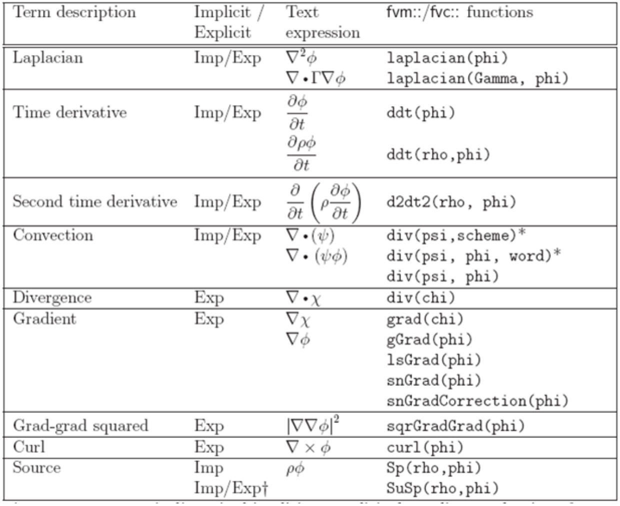
$Ax = B$

```
tmp<fvScalarMatrix> kEqn
(
    fvm::ddt(k_)
  + fvm::div(phi_, k_)
  - fvm::laplacian(DkEff(), k_)
 ==
    nut*magSqr(symm(fvc::grad(U)))
  - fvm::Sp(epsilon_/k_, k_)
);

kEqn().relax();
solve(kEqn);
```

How are PDEs solved in OpenFOAM?

Equations are essentially a group of operations on fields:

- Differential operators available in OpenFOAM®



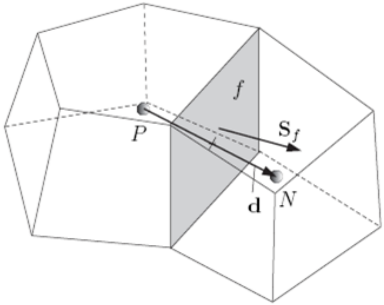
Term description	Implicit / Explicit	Text expression	fvm::/fvc:: functions
Laplacian	Imp/Exp	$\nabla^2\phi$	`laplacian(phi)`
		$\nabla\cdot\Gamma\nabla\phi$	`laplacian(Gamma, phi)`
Time derivative	Imp/Exp	$\frac{\partial\phi}{\partial t}$	`ddt(phi)`
		$\frac{\partial\rho\phi}{\partial t}$	`ddt(rho,phi)`
Second time derivative	Imp/Exp	$\frac{\partial}{\partial t}\left(\rho\frac{\partial\phi}{\partial t}\right)$	`d2dt2(rho, phi)`
Convection	Imp/Exp	$\nabla\cdot(\psi)$	`div(psi,scheme)`*
		$\nabla\cdot(\psi\phi)$	`div(psi, phi, word)`*
			`div(psi, phi)`
Divergence	Exp	$\nabla\cdot\chi$	`div(chi)`
Gradient	Exp	$\nabla\chi$	`grad(chi)`
		$\nabla\phi$	`gGrad(phi)`
			`lsGrad(phi)`
			`snGrad(phi)`
			`snGradCorrection(phi)`
Grad-grad squared	Exp	$\|\nabla\nabla\phi\|^2$	`sqrGradGrad(phi)`
Curl	Exp	$\nabla\times\phi$	`curl(phi)`
Source	Imp	$\rho\phi$	`Sp(rho,phi)`
	Imp/Exp†		`SuSp(rho,phi)`

How are PDEs solved in OpenFOAM?

But how are these operators really discretized in OpenFOAM® ?:

- The core is the finite volume method and the Gauss theorem

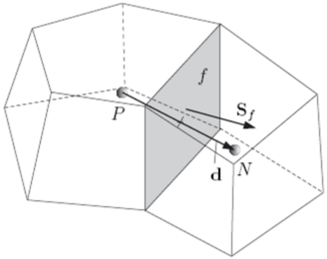
Gauss's theorem	$\int_V \nabla\phi \, \mathrm{d}V = \sum_f S_f \phi_f$
Laplacian	$\int_V \nabla \cdot (\Gamma\nabla\phi) \, \mathrm{d}V = \int_S dS \cdot (\Gamma\nabla\phi) = \sum_f \Gamma_f S_f \cdot (\nabla\phi)_f$
Divergence	$\int_V \nabla \cdot \phi \, \mathrm{d}V = \int_S \mathrm{d}S \cdot \phi = \sum_f S_f \cdot \phi_f$
Convection	$\int_V \nabla \cdot (\rho U\phi) \, \mathrm{d}V = \int_S \mathrm{d}S \cdot (\rho U\phi) = \sum_f S_f \cdot (\rho U)_f \phi_f = \sum_f F\phi_f$
Gradient	$\int_V \nabla\phi \, \mathrm{d}V = \int_S \mathrm{d}S\phi = \sum_f S_f\phi_f$
Source term	$\int_V \rho\phi \, \mathrm{d}V = \rho_P V_P \phi_P$

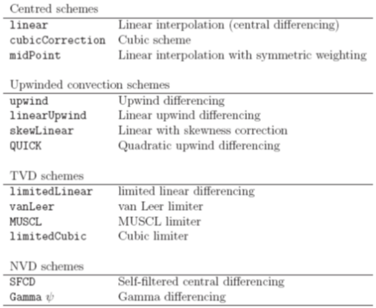


How are PDEs solved in OpenFOAM?

Options for spatial discretization:

- Options are available for choice when discretize the equations
- Options are specified in the file `system/fvSchemes`
- Example: interpolate the values from cell center to face center
- Options for other discretizations are documented in the `UserGuide`





Centred schemes	
`linear`	Linear interpolation (central differencing)
`cubicCorrection`	Cubic scheme
`midPoint`	Linear interpolation with symmetric weighting
Upwinded convection schemes	
`upwind`	Upwind differencing
`linearUpwind`	Linear upwind differencing
`skewLinear`	Linear with skewness correction
`QUICK`	Quadratic upwind differencing
TVD schemes	
`limitedLinear`	limited linear differencing
`vanLeer`	van Leer limiter
`MUSCL`	MUSCL limiter
`limitedCubic`	Cubic limiter
NVD schemes	
`SFCD`	Self-filtered central differencing
`Gamma` ψ	Gamma differencing

How are PDEs solved in OpenFOAM?

Options for temporal discretization:

Scheme	Description
Euler	First order, bounded, implicit
localEuler	Local-time step, first order, bounded, implicit
CrankNicholson	Second order, bounded, implicit
backward	Second order, implicit
steadyState	Does not solve for time derivatives

Example: Euler scheme

$$\frac{\partial}{\partial t}\int_V \rho\phi \,\mathrm{d}V = \frac{(\rho_P\phi_P V)^n - (\rho_P\phi_P V)^o}{\Delta t}$$

backward scheme

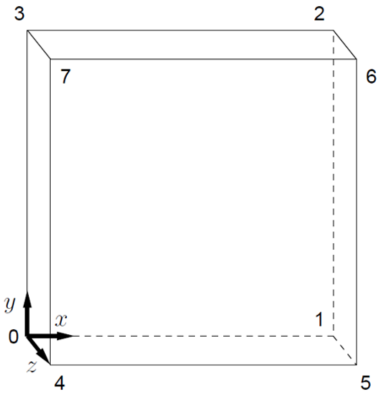
$$\frac{\partial}{\partial t}\int_V \rho\phi \,\mathrm{d}V = \frac{3\,(\rho_P\phi_P V)^n - 4\,(\rho_P\phi_P V)^o + (\rho_P\phi_P V)^{oo}}{2\Delta t}$$

How are PDEs solved in OpenFOAM?

- Boundary conditions (Specified in the field file):
  - Generic BCs: fixed value, fixed gradient, mixed
  - Physical BCs: inlet, outlet, no slip wall, slip wall, etc.
  - Other BCs: symmetry, periodic, empty, processor (for parallel computation), etc.
- OpenFOAM® solves the governing equations without considering whether the problem is 1D, 2D, or 3D
- The dimensionality of the problem is defined in your simulation case through boundary condition, i.e., the use of `empty` BC.

How are PDEs solved in OpenFOAM?

Boundary condition `empty` and problem dimensionality:



- 2D: if faces 0123 and 4567 are defined as `empty`
- 1D: if faces 0123, 4567, 2376, and 0154 are all defined as `empty`

How are PDEs solved in OpenFOAM?

Now what?

- After the discretization and applying BCs and ICs, what we got is an algebraic system of equations:
$$[A][x] = [b]$$
- *A* is matrix, *x* is the unknown value vector at the cell centers of the mesh, *b* is RHS
- *A* could be symmetric, banded, diagonal, etc. It depends on the governing equation and the discretization scheme.
- Usually this algebraic system is HUGE. Most of the computational time of the code is spent here to solve this.
- User has the option to choose the scheme to solve the system
    - Specified in file `system/fvSolution`.

How are PDEs solved in OpenFOAM?

From `fvSolution` file

```
solvers
{
    p PCG
    {
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
    };

    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0;
    };
}
```

Linear system solver choices

Solver	Keyword
Preconditioned (bi-)conjugate gradient	`PCG`/`PBiCG`†
Solver using a smoother	`smoothSolver`
Generalised geometric-algebraic multi-grid	`GAMG`

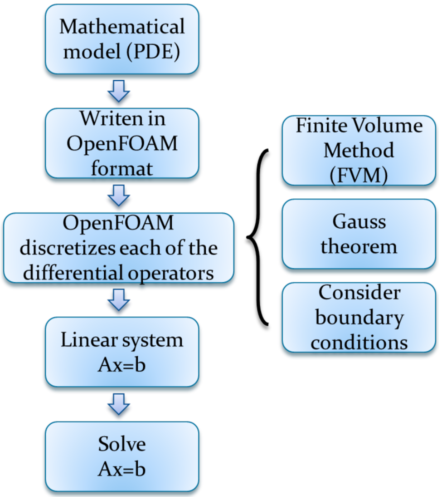
†`PCG` for symmetric matrices, `PBiCG` for asymmetric

Options for preconditioners

Preconditioner	Keyword
Diagonal incomplete-Cholesky (symmetric)	`DIC`
Faster diagonal incomplete-Cholesky (`DIC` with caching)	`FDIC`
Diagonal incomplete-LU (asymmetric)	`DILU`
Diagonal	`diagonal`
Geometric-algebraic multi-grid	`GAMG`
No preconditioning	`none`

How are PDEs solved in OpenFOAM?

In summary:


Mathematical model (PDE)
Writen in OpenFOAM format
OpenFOAM discretizes each of the differential operators
Linear system Ax=b
Solve Ax=b
Finite Volume Method (FVM)
Gauss theorem
Consider boundary conditions

How are PDEs solved in OpenFOAM?

Further readings:

- Jasak's Ph.D. thesis, 1996
  - Many details about the numerical schemes in OpenFOAM
  - How Navier-Stokes equations are solved
- Versteeg and Malalasekera, An Introduction to Computational Fluid Dynamics, The Finite Volume Method, 2nd edition, 2007