Building Blocks and Library Organisation in OpenFOAM

Hrvoje Jasak

hrvoje.jasak@fsb.hr

Faculty of Mechanical Engineering and Naval Architecture

University of Zagreb, Croatia

Outline

FSB

Objective

- Review basic building blocks in OpenFOAM software and their interaction

Topics

- Design rationale
- Basic components
- Mesh handling classes: polyhedral cell support
- Discretisation method: Finite Volume
- Handling physical models and model-to-model interaction
- Modelling and utility libraries
- Top-level solvers and utilities

Design Rationale

FSB

Representing Continuum Mechanics Equations in Software

- Natural language of continuum mechanics: partial differential equations
- Example: turbulence kinetic energy equation

$$\frac{\partial k}{\partial t} + \nabla\bullet(\mathbf{u}k) - \nabla\bullet[(\nu + \nu_t)\nabla k] = \nu_t \left[\frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right]^2 - \frac{\epsilon_o}{k_o}\,k$$

- Objective: **represent differential equations in their natural language**

```
solve
(
   fvm::ddt(k)
 + fvm::div(phi, k)
 - fvm::laplacian(nu() + nut, k)
== nut*magSqr(symm(fvc::grad(U)))
 - fvm::Sp(epsilon/k, k)
);
```

- Correspondence between the implementation and the original equation is clear

Design Rationale

FSB

Software Organisation Model

- Complexity in software organisation stems from data interaction
- **Functional approach** to software organisation
  - Global data and a set of functions operating on it
  - All data is available at all parts of the code: no protection
  - Because of global data model, every new function potentially interacts with all other parts of the software: even for modestly sized software, this is immensely complex!
- There are some software components that can be clearly identified by a combination of data and functionality
- Example: cell volumes
  - Related to the computational mesh: changes when point position changes
  - Can be calculated only from mesh data: points, faces, cells
  - Other parts of the software only use cell volume data
  - . . . but changing it would be an error!
- **Object-oriented approach** to software organisation
  - Encapsulate data and functions into units and protect it from corruption

Object Orientation

Object Oriented Analysis

- Analysis of CFD software from object orientation standpoint:
  "Recognise main objects from the numerical modelling viewpoint"
- Main object = **operator**, *e.g.* time derivative, convection, diffusion, gradient
- . . . but we need many other components to assemble the equation
  - Representation of space and time: computational domain
  - Scalars, vectors and tensors
  - Field representation and field algebra
  - Initial and boundary condition
  - Systems of linear equations and solvers
  - Discretisation methods
- The above does not complete the system
  - Physical models, (*e.g.* turbulence) and model-to-model interaction
  - Pre- and post-processing and related utilities
  - Top-level solvers

Basic Components

Software Organisation

- In OpenFOAM, software layout in directories mirrors its logical organisation
- Libraries
  - `OpenFOAM-2.0-ext/src`: library source
  - `OpenFOAM-2.0-ext/src/OpenFOAM`: foundation library
  - `OpenFOAM-2.0-ext/src/finiteVolume`: finite volume discretisation
  - `OpenFOAM-2.0-ext/src/turbulenceModels/incompressible`: incompressible RANS turbulence models
  - `OpenFOAM-2.0-ext/src/sampling`: data sampling toolkit library
- Top-level executables
  - `OpenFOAM-2.0-ext/applications/solvers`: top-level solvers
  - `.../applications/solvers/basic/scalarTransportFoam`: scalar transport solver
  - `OpenFOAM-2.0-ext/applications/utilities`: top-level utility codes
  - `OpenFOAM-2.0-ext/applications/utilities/mesh/manipulation`: mesh manipulation utilities
- *etc.*

Basic Components

Main Objects

- Computational domain

Object	Software representation	C++ Class
Tensor	(List of) numbers + algebra	vector, tensor
Mesh primitives	Point, face, cell	point, face, cell
Space	Computational mesh	polyMesh
Time	Time steps (database)	time

- Field algebra

Object	Software representation	C++ Class
Field	List of values	Field
Boundary condition	Values + condition	patchField
Dimensions	Dimension set	dimensionSet
Geometric field	Field + mesh + boundary conditions	geometricField
Field algebra	$+ \; - \; * \; / \; tr(), sin(), exp() \ldots$	field operators

Basic Components

Example: `dimensionSet` class

```
class dimensionSet
{
    //- Construct given individual dimension exponents for all
    //  seven dimensions
    dimensionSet
    (
        const scalar mass,
        const scalar length,
        const scalar time,
        const scalar temperature,
        const scalar moles,
        const scalar current,
        const scalar luminousIntensity
    );

    friend dimensionSet operator+
    (
        const dimensionSet&,
        const dimensionSet&
    );
};
```

Basic Components

Example: `Field` class

```
template<class Type>
class Field
:
    public refCount, public List<Type>
{
    //- Construct null
    Field();

    //- Construct given size
    explicit Field(const label);

    //- Construct given size and initial value
    Field(const label, const Type&);

    //- Construct from Istream
    Field(Istream&);

    void operator=(const Field<Type>&);
    void operator*=(const scalar&);
    friend Ostream& operator<< <Type>(Ostream&, const Field<Type>&);
};
```

Basic Components

Main Objects

- Linear equation systems and linear solvers

Object	Software representation	C++ Class
Linear equation matrix	Matrix coefficients	lduMatrix
Solvers	Iterative solvers	lduMatrix::solver

- Linear matrix
  - OpenFOAM uses **arrow matrix format**: easy to identify diagonal-only, symmetric and assymetric matrices
  - `lduAddressing` class defines sparse addressing. Addressing is always symmetric in pattern.
  - Diagonal, upper and lower triangular part stored separately: 3 arrays
  - Matrix is built up component-by-component
  - Coupling interfaces (including parallelisation) handled separately from the global sparse addressing space
- Linear solver
  - Linear solvers answer to common interface: run-time selection
  - Implemented solvers: Krylov space, Algebraic multigrid, CG-AMG, new solvers: extrapolation and deflation algorithms

Mesh Handling

FSB

Polyhedral Mesh Support

- OpenFOAM uses polyhedral mesh format. Mesh definition:
  - List of points
  - List of faces in terms of point labels
  - List of cells in terms of face labels
  - List of boundary patches
- Polyhedral mesh definition provides more freedom in mesh generation: bottleneck in modern CFD simulations
- Mesh analysis classes separated from discretisation support classes
- Built-in support for mesh motion and topological changes
- Simple handling of moving boundary problems: automatic mesh motion solver
- Topological changes support
  - Basic operations: add/modify/remove point/face/cell
  - Mesh modifier classes operate in terms of basic operations. Triggering of topological changes is automatic.
  - Pre-defined mesh modifiers available for standard operations
- Quest for a fully automatic polyhedral mesh generator continues!

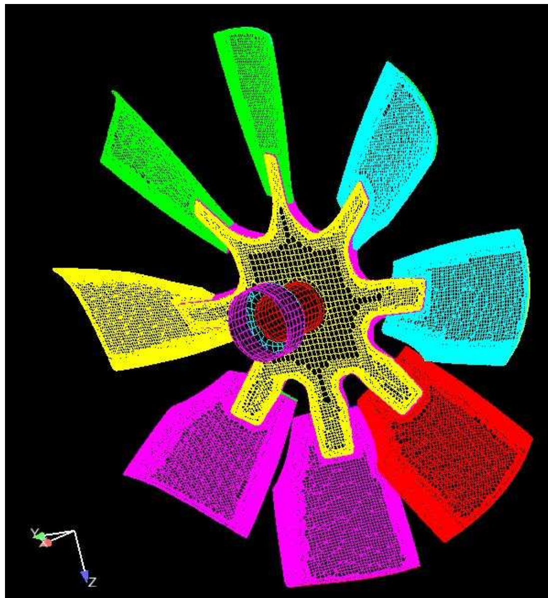
Mesh Handling

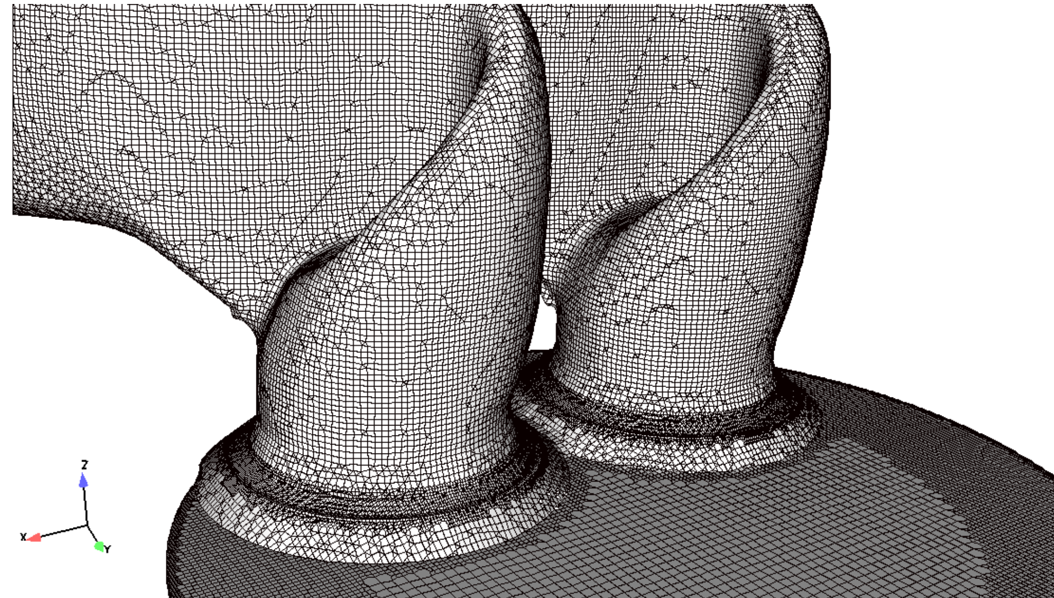
Mesh Conversion Tools

- OpenFOAM does not provide integrated meshing: work in progress
- Polyhedral mesh support makes for easy mesh conversion
- Mesh converters for STAR-CD (Pro-Am), Fluent (Gambit, T-Grid), Ansys *etc.*

Automatic Mesh Generation

- Meshing projects in OpenFOAM: Cartesian base with boundary interaction, polyhedral dual mesh
- Mesh generated from STL surface, with parameters for resolution control

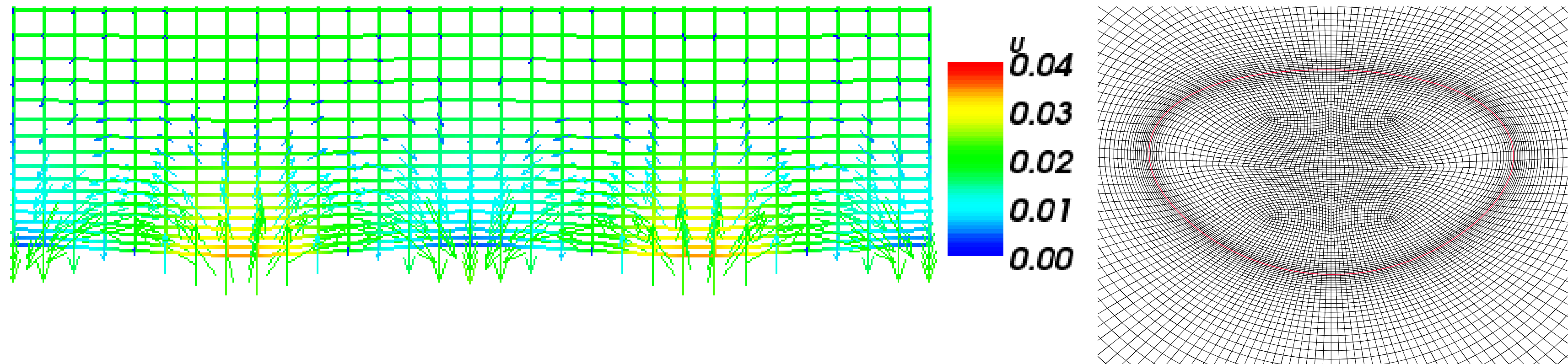




Automatic Mesh Motion

Automatic Mesh Motion Solver

- Typically, only changes in boundary shape are of interest; internal points are moved to accommodate boundary motion and preserve mesh validity
- Internal point motion obtained by solving a “motion equation”
- Prescribed boundary motion provides boundary conditions
- Point-based FEM solver: no interpolation required
- Mesh validity criteria accounted for during discretisation
- Easily handles solution-dependent mesh motion: solving one additional equation to obtain point positions
- Polyhedral support through **mini-element technique**; parallelisation in domain decomposition mode shadows the FVM solver



Discretisation Classes

FSB

Finite Volume Discretisation

- Software is limited in use if it stems from standard form of transport equation
- In Continuum Mechanics, operators can be combined at will
  - Gradient operator (cell and face); divergence operator
  - First and second temporal derivative
  - Convection term: $\nabla \bullet (\phi U)$; diffusion term: $\nabla \bullet (\gamma \nabla \phi)$
  - Source and sink terms
- All other operations reduce to field algebra and interpolation
- Operators implemented as static functions in an appropriate namespace
  - `fvc` for **calculus**: given a field and operator produce a field
  - `fvm` for **method**: given an operator, produce a matrix that discretises it
- Each operator may be discretised in numerous ways. Example: convection differencing schemes
- Discretisation is controlled on a term-by-term basis at run-time

Discretisation Classes

Finite Volume Discretisation: Example

- Scalar transport equation

```
solve
(
    fvm::ddt(T)
  + fvm::div(phi, T)
  - fvm::laplacian(DT, T)
);
```

- Discretisation control: system/fvSchemes

```
ddtSchemes { default    Euler; }
divSchemes
{
    default          none;
    div(phi,T)       Gauss Gamma 1;
}
laplacianSchemes
{
    default          none;
    laplacian(DT,T) Gauss linear corrected;
}
```

Model-to-Model Interaction

Common Interface for Model Classes

- Physical models grouped by functionality, *e.g.* material properties, viscosity models, turbulence models *etc.*
- Each model answers the interface of its class, but its implementation is separate and independent of other models
- The rest of software handles the model through a generic interface: breaking the complexity of the interaction matrix

```
class turbulenceModel
{
    virtual volTensorField R() const = 0;
    virtual fvVectorMatrix divR
    (
        volVectorField& U
    ) const = 0;
    virtual void correct() = 0;
};
```

- New turbulence model implementation : Spalart-Allmaras

```
class SpalartAllmaras : public turbulenceModel{};
```

Model-to-Model Interaction

Common Interface for Model Classes

- Model user only sees the virtual base class
- Example: steady-state momentum equation with turbulence

```
autoPtr<turbulenceModel> turbulence
(
    turbulenceModel::New(U, phi, laminarTransport)
);

fvVectorMatrix UEqn
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  + turbulence->divR(U)
 ==
  - fvc::grad(p)
);
```

- Implementation of a new model does not disturb the existing models
- Consumer classes see no difference, just a new choice

Run-Time Selection

Handling Model Libraries

- Model-to-model interaction handled through common interfaces
- New components do not disturb existing code: fewer new bugs
- Run-time selection tables: dynamic binding for new functionality
- Used for every implementation: “user-coding”
  - Differencing schemes: convection, diffusion, rate of change
  - Gradient calculation
  - Boundary conditions
  - Linear equation solvers
  - Physical models, *e.g.* viscosity, turbulence, evaporation, drag *etc.*
  - Mesh motion algorithms
- Ultimately, there is no difference between pre-implemented models and native library functionality: no efficiency concerns
- Implemented models are examples for new model development

Modelling Libraries

Physical Modelling Libraries

- Physical modelling libraries group models that perform the same function under a same virtual base class (interface)
- Libraries grouped by functionality; some are specific for the purpose. Example:
  - Transport models: single- and multi-phase viscosity models
  - Turbulence models: incompressible RANS turbulence
- Each model is implemented independently and carries its own parameters
- Some application carry their own models, run-time selection and libraries

Libraries

- Thermophysical models: real properties of liquids, gasses and mixtures. Chemistry models combustion thermo-chemistry, interface to JANAF and CHEMKIN packages
- Transport models: fluid viscosity
- Incompressible and compressible RANS turbulence models
- LES models: filter size and filter functions, incompressible and compressible LES models, LES wall functions
- Diesel spray models: atomisation, breakup, collision, dispersion, drag, evaporation heat transfer, wall interaction

Utility Libraries

Utility Libraries

- Common functionality used in several solvers is implemented in generic form. In order to use it in multiple executables, it compiles into a library
- Header files describe the classes: included in the top-level code or another library
- Compiled code obtained by linking the library file
- Beware: utilities are often written as templates: complete functionality in header files

Libraries

- ODE: ordinary differential equation solvers
- Dynamic mesh: topology modifiers, automatic mesh motion
- Engine: internal combustion engine-specific classes
- Error estimation: residual error estimates in operator form
- Miscellaneous: meshTools, randomProcesses, postProcessing
- Public domain tools, *e.g.* parallel communication

Top-Level Physics Solver

Solver Applications Applications

- Top-level solvers implement solution algorithms for specific class of physics
- Directory structure per “class”: `applications/solvers`
- Some solvers will come with own libraries, eg. a class hierarchy for compressibility models in `cavitatingFoam`
- Structure of many solvers is similar
  - Create time, mesh and fields
  - Read material properties and instantiate generic models
  - Time-loop, containing equations and/or model update calls

```
while (runTime.loop())
{
    Info<< "Time = " << runTime.timeName() << nl << endl;
    solve
    (
        fvm::ddt(T)
      + fvm::div(phi, T)
      - fvm::laplacian(DT, T)
    );
    runTime.write();
}
```

Utilities

Utility Applications

- Auxiliary functions, from pre-processing to data analysis are also top-level codes
- Directory structure organised per type:
  `applications/utilities/postProcessing`
  - Pre-processing: FoamX, fields mapping, flow setup
  - Post-processing: graphical tools, data conversion, field manipulation
  - Mesh generation, conversion and manipulation; mesh fixing tools; mesh refinement and simple topological changes
  - Parallel processing: parallel decomposition and reconstruction of mesh and fields
  - Thermophysical properties, *e.g.* simple chemistry
  - A-posteriori error estimation tools; residual fields visualisation

Layered Development

OpenFOAM Software Architecture

- Design encourages code re-use: developing shared tools
- Development of model libraries: easy model extension
- Code developed and tested in isolation
  - Vectors, tensors and field algebra
  - Mesh handling, refinement, mesh motion, topological changes
  - Discretisation, boundary conditions
  - Matrices and solver technology
  - Physics by segment
  - Custom applications
- Custom-written top-level solvers optimised for efficiency and storage
- **Ultimate user-coding capabilities!**
  - Run-time selection tables used in all places where user choice is made
  - Implementation of a new model does not disturb the existing models
  - Addition to selection tables is done automatically: there is no difference between a user-defined model and one provided by OpenFOAM libraries

Summary

FSB

Organisation of OpenFOAM Libraries

- OpenFOAM implements functionality in a library form: re-use of single implementation in multiple executables
- Software is written using object orientation: no excuses!
- Foundation library: `OpenFOAM`: basic components and utilities needed for numerical simulation
- Discretisation functionality implemented in operator form. Multiple choices for discretisation practice, each in its own library
- Generic functionality written in a generic manner. If the same code is needed for two operations (*e.g.* sliding interface and contact detection), it is abstracted: minimal code duplication
- Library is mature: if you think a function should be there, it probably is!

OpenFOAM Solvers

- Multiple top-level solvers in multiple executables. Written and tuned for particular purpose. First step: choose the solver with appropriate physics
- Numerous utility executables, from mesh generation to post-processing
- ... **all available in full source for easy user customisation**