Object-Oriented Continuum Mechanics Simulations with OpenFOAM

Hrvoje Jasak

h.jasak@wikki.co.uk

Wikki Ltd, United Kingdom
University of Zagreb, Croatia
26/Apr/2006
Objective

- Present a novel way of handling software implementation in numerical mechanics

Topics

- Design of modern CFD software
- A new approach to model representation
- Introduction to OpenFOAM: object-oriented software for Computational Continuum Mechanics (CCM)
- OpenFOAM library capabilities and pre-implemented solvers
- Review of running research projects using OpenFOAM
Background

State of the Art: CFD

- Numerical modelling is becoming part of product design
  - Improvements in computer performance
  - Improved physical modelling and numerics
  - Sufficient validation and experience
- Two-fold requirements
  - Integration into the CAD-based design process
  - Quick and reliable implementation of new models
- Complex geometry support, high-performance computing, automatic meshing, dynamic mesh capabilities etc. needed across the spectrum
- Opening new areas of CFD simulation
  - Non-traditional physics: complex heat and mass transfer models, electromagnetics, fluid-structure interaction
  - New solution requirements, e.g. optimisation and robust design
Design of Modern Solvers

- Monolithic implementation and integrated approach
- Fortran, maybe C; batch-model, 1985-1990 “vintage”
- Cover the selected physics: fluids, plasticity etc.
- A single discretisation method (FVM, FEM), parallelism and vectorisation
- User-defined modifications inefficient and limiting
- Model-to-model interaction matrix is becoming too complex
- Difficulties in development, maintenance and support
- Complex solver-to-solver interaction or embedding virtually impossible: two solvers, solver and mesh generator, embedding in a CAD environment
- Some new simulation techniques cannot be accommodated at all

Based on the above, change of paradigm is overdue!
How to handle complex models in software?

- Natural language of continuum mechanics: partial differential equations

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

- Main object = **operator**, e.g. time derivative, convection, diffusion, gradient
FOAM (Field Operation and Manipulation):
Represent equations in their natural language

```cpp
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.
Analysis of CFD software from object orientation standpoint:
“Recognise main objects from the numerical modelling viewpoint”

- **Computational domain**

<table>
<thead>
<tr>
<th>Object</th>
<th>Software representation</th>
<th>C++ Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensor</td>
<td>(List of) numbers + algebra</td>
<td>vector, tensor</td>
</tr>
<tr>
<td>Mesh primitives</td>
<td>Point, face, cell</td>
<td>point, face, cell</td>
</tr>
<tr>
<td>Space</td>
<td>Computational mesh</td>
<td>polyMesh</td>
</tr>
<tr>
<td>Time</td>
<td>Time steps (database)</td>
<td>time</td>
</tr>
</tbody>
</table>

- **Field algebra**

<table>
<thead>
<tr>
<th>Object</th>
<th>Software representation</th>
<th>C++ Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>List of values</td>
<td>Field</td>
</tr>
<tr>
<td>Boundary condition</td>
<td>Values + condition</td>
<td>patchField</td>
</tr>
<tr>
<td>Dimensions</td>
<td>Dimension set</td>
<td>dimensionSet</td>
</tr>
<tr>
<td>Geometric field</td>
<td>Field + mesh + boundary conditions</td>
<td>geometricField</td>
</tr>
<tr>
<td>Field algebra</td>
<td>+ − * / tr(), sin(), exp() . . .</td>
<td>field operators</td>
</tr>
</tbody>
</table>
Object Orientation

- Matrix and solvers

<table>
<thead>
<tr>
<th>Object</th>
<th>Software representation</th>
<th>C++ Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear equation matrix</td>
<td>Matrix coefficients</td>
<td>lduMatrix</td>
</tr>
<tr>
<td>Solvers</td>
<td>Iterative solvers</td>
<td>lduMatrix::solver</td>
</tr>
</tbody>
</table>

- Numerics

<table>
<thead>
<tr>
<th>Object</th>
<th>Software representation</th>
<th>C++ Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolation</td>
<td>Differencing schemes</td>
<td>interpolation</td>
</tr>
<tr>
<td>Differentiation</td>
<td>ddt, div, grad, curl</td>
<td>fvc, fec</td>
</tr>
<tr>
<td>Discretisation</td>
<td>ddt, d2dt2, div, laplacian</td>
<td>fvm, fem, fam</td>
</tr>
</tbody>
</table>

Implemented Methods: Finite Volume, Finite Element, Finite Area and Lagrangian particle tracking (discrete particle model)

- Top-level code organisation

<table>
<thead>
<tr>
<th>Object</th>
<th>Software representation</th>
<th>C++ Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model library Application</td>
<td>Library main()</td>
<td>e.g. turbulenceModel</td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td>–</td>
</tr>
</tbody>
</table>
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 generic interface: breaking the complexity of the interaction matrix

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

- New turbulence model implementation: Spalart-Allmaras

```cpp
class SpalartAllmaras : public turbulenceModel{};
```
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
Complex Geometry Handling

- Complex geometry is a rule, not exception
- Polyhedral cell support
  - A cell is a polyhedron bounded by polygons
  - Consistent handling of all cell types
  - More freedom in mesh generation
- Interfaces to all major mesh generators

Automatic Mesh Motion Solver

- Supporting cases with variable geometrical shape
- Based on the prescribed boundary deformation, re-calculate the point position

Supporting Topological Changes

- For cases with considerable mesh deformation, mesh topology changes during simulation
- Automatic handling of field mapping
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!**
Implemented Capabilities

Model and Utility Libraries
- Thermo-physical models (liquids and gasses)
- Chemical reaction library interface (Chemkin)
- Non-Newtonian viscosity models
- Turbulence models (RANS and LES)
- Dynamic mesh and topology changes
- A-posteriori error estimation
- Diesel spray (atomisation, dispersion, heat transfer, evaporation, spray-wall etc.)

Top-Level Solvers
- Basic: Laplace, potential flow, transport
- Incompressible flow, compressible flow
- Heat transfer: buoyancy-driven flows
- Multiphase: Euler-Euler, surface capturing and surface tracking
- DNS and LES turbulent flows, aero-acoustics
- Pre-mixed and Diesel combustion, spray and in-cylinder flows
- Stress analysis, fluid-structure interaction, electromagnetics, MHD, etc.
Automatic Mesh Motion

Handling Shape Change: Problem Specification

- Initial valid mesh is available
- Time-varying boundary motion
  - Prescribed in advance: e.g. IC engines
  - Part of the solution: surface tracking
- Need to determine internal point motion based on prescribed boundary motion
- Mesh in motion must remain valid: face and cell flip must be prevented by the algorithm

Solution Technique

- Point position will be provided by solving an equation given boundary motion conditions
- Cell-based methods fail in interpolation; spring analogy unreliable
- Choosing vertex-based (FEM) discretisation with polyhedral cell support: mini-element technique
- Choice of equation: Laplace or pseudo-solid

Solution-dependent mesh motion: solving equations for mesh deformation
Automatic Mesh Motion

Free surface tracking

- 2 phases = 2 meshes
- Surface pressure, momentum and deformation is a part of the solution
- Mesh adjusted for interface motion

\[ \mathbf{v}_b = -\mathbf{v}_F \]
Coupled Simulation

Free-Rising Air Bubble with Surfactants

Complex coupling problem

- FVM flow solver
- FEM automatic mesh motion
- FAM for surfactant transport
OpenFOAM in Research

- Open architecture and extensive capabilities make a good research platform
- Currently, downloads on over 200 Universities worldwide
- First OpenFOAM Workshop, Zagreb Jan/2006: 80 attendees
- Leading research/development centres: Chalmers University, Sweden; Politecnico di Milano, University College Dublin, TU Freiberg, Germany
- Major development on multi-phase flows: MFIX-NG NETL, US Dept. of Energy

OpenFOAM in Industry

- An open platform for in-house or specialist software development is required
- Interest greatly increased in the last year, sometimes following PhD projects or joint development
- Pilot projects or active use: ABB Corporate Research, Audi, BAE Systems, Caldeyrs, Danone SA, Hydro Quebec, Scania, Shell Global Solutions, SKF, Volkswagen, TTP and others
- Wikki Ltd. is a premier provider of support, consultancy and collaborative development
Free Surface LES

LES of a Diesel Injector

- Injection of Diesel fuel into the atmosphere and subsequent breakup
- \( d = 0.2 \text{mm}, \text{high velocity and surface tension} \)
- Mean injection velocity: \( 460 \text{m/s} \)
- Diesel fuel injected into air, \( 5.2 \text{MPa}, 900K \)
- Turbulent subsonic flow
  - 1-equation LES model with no free surface correction
  - Fully developed pipe flow inlet
- Cavitation and compressibility effects not taken into account
Free Surface LES

- Mesh size: 1.2 to 8 million cells
- Aggressive local mesh refinement close to pipe exit
- 50k time-steps
- 6µs initiation time: starting transient
- 20µs averaging time for mean properties
Fluid-Structure Interaction

Plastic Pipeline Failure

- Rapid crack propagation in pressurised plastic pipes
- Internal pressure drives crack propagation; crack opening depressurises the pipeline
- Example of strong two-way coupling
Pipeline failure: crack propagation and leakage
Enlarged deformation of the pipe
Topological Changes on Polyhedral Meshes

- Mesh modifications implemented in terms of basic operation: add/modify/remove a point/face/cell
- Topology modifiers specify and trigger mesh changes
  - Mesh attach/detach
  - Cell layer addition/removal
  - Sliding interfaces
- Data mapping handled automatically
Topological Changes

In-Cylinder Flow Simulation

- Exhaust and intake stroke in a 2-valve internal combustion engine
- Moving piston and operating valves using topological changes

Exhaust stroke, incompressible flow
Summary

- Object-oriented approach facilitates model implementation: layered design + re-use
- Equation mimicking opens new grounds in Computational Continuum Mechanics
- Extensive capabilities already implemented
- Open design for easy user customisation

Acknowledgements and Further Info

- Željko Tuković, University of Zagreb
- Eugene de Villiers, Imperial College London
- Aleksandar Karač and Alojz Ivanković, University College Dublin
- Tommaso Luccini and Gianluca d'Errico, Politecnico di Milano
- For more info on OpenFOAM, please visit http://www.openfoam.org
FOAM: CCM in C++

Main characteristics
- Wide area of applications: all of Computational Continuum Mechanics
- Shared tools and code re-use
- Libraries of pre-implemented models
- Custom top-level solvers, all available in source
- Mature software under active development

Versatility
- Unstructured meshes, automatic mesh motion + topological changes
- Finite Volume, Finite Element, Lagrangian tracking and Finite Area methods
- Efficiency through massive parallelism
- Extensive validation in research projects