Modelling Multi-Phase Flows in OpenFOAM

Hrvoje Jasak

h.jasak@wikki.co.uk

Wikki Ltd. United Kingdom

15/Apr/2005
Objective

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

Topics

- Representing physical models in software
- Object-oriented approach in Computational Continuum Mechanics
- Multi-phase flow modelling in OpenFOAM
State of the Art

- Numerical modelling part of product design
  - Improvements in computer performance
  - Improved physical modelling and numerics
  - Sufficient validation and experience

- Two-fold requirements
  - Quick and reliable model implementation
  - Complex geometry, high-performance computing, automatic meshing etc.
How to handle complex models in software?

- Natural language of continuum mechanics: partial differential equations

\[
\frac{\partial k}{\partial t} + \nabla \cdot (uk) - \nabla \cdot [(\nu + \nu_t)\nabla k] = \\
\nu_t \left[\frac{1}{2}(\nabla u + \nabla u^T)\right]^2 - \frac{\epsilon_o}{k_o} k
\]
FOAM (Field Operation and Manipulation): Represent 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)
);
```
Object Orientation

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>Time</td>
<td>Time steps (database)</td>
<td>time</td>
</tr>
<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>
</tbody>
</table>
Object Orientation

- **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 + boundary conditions</td>
<td>geometricField</td>
</tr>
<tr>
<td>Field algebra</td>
<td>+ - * / tr(), sin(), exp() ...</td>
<td>field operators</td>
</tr>
</tbody>
</table>

- **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>
Object Orientation

- 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 tracking

- Top-level organisation

<table>
<thead>
<tr>
<th>Object</th>
<th>Software representation</th>
<th>C++ Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model library</td>
<td>Library</td>
<td>turbulenceModel</td>
</tr>
<tr>
<td>Application</td>
<td>main()</td>
<td>–</td>
</tr>
</tbody>
</table>
Model Interaction

Common interface for related models

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

class SpalartAllmaras : public turbulenceModel{};
Run-Time Selection

- Model-to-model interaction through common interfaces (virtual base classes)
- New components do not disturb existing code
- Run-time selection tables: dynamic binding
- Used for every implementation: “user-coding”
  - Convection differencing schemes
  - Gradient calculation
  - Boundary conditions
  - Linear equation solvers
  - Physical modelling, e.g. non-Newtonian viscosity laws, etc.
Geometry Handling

Complex geometry, mesh motion and morphing

- Complex geometry is a rule, not exception
- Polyhedral cell support
  - Cell is a polyhedron bounded by polygons
  - Consistent handling of all cell types
  - More freedom in mesh generation
- Integrated mesh motion and topo changes
- Automatic motion solver + morph engine
Layered Development

- Design encourages code re-use: shared tools
- Code developed and tested in isolation
  - Vectors, tensors and field algebra
  - Mesh handling, refinement, topo changes
  - Discretisation, boundary conditions
  - Matrices and solver technology
  - Physics by segment
  - Custom applications
- **Ultimate user-coding capabilities!**
Multi-Phase Modelling

Examples of FOAM library in use
- Multi-phase Eulerian models
- Free surface flows
  - Surface capturing method
  - Surface tracking method
- Lagrangian particle model

Extensions and integration benefits:
poly mesh, motion/topo changes, multi-physics, other CFD/CCM: LES, combustion, (nuclear?)
Eulerian Multi-Phase

Bubbly flow in water

- Bubble column: Gomes et al. 1998
- Air bubbles are injected at bottom plate
- Includes free surface: $\gamma = 0, \gamma = 1$
Surface Capturing

Droplet impact into a wall film, 1.3 million cells

Splash, $u = 50\text{m/s}, d = 0.3\text{mm}$
LES Surface Capturing

LES of a Diesel Injector

- $d = 0.2\text{mm}$, high velocity and surface tension
- Mean injection velocity: $460\text{m/s}$
- Diesel fuel injected into air, $5.2\text{MPa}$, $900\text{K}$
- Turbulent and subsonic flow, no cavitation
  - 1-equation LES model with no free surface correction
  - Fully developed pipe flow inlet
LES Surface Capturing

- Mesh size: 1.2 to 8 million CVs, aggressive local refinement, 50k time-steps
- $6\mu s$ initiation time, $20\mu s$ averaging time
Surface Tracking

Free surface tracking
- 2 phases = 2 meshes
- Mesh adjusted for interface motion
- Surfactant transport

Air-water system
- 2-D: $r_b = 0.75 \text{ mm}$
- 3-D: $r_b = 1 \text{ mm}$
Surface Tracking

Clean surface

Pollution by surfactant chemicals
Surface Tracking

Complex coupling problem: FVM flow solver + FEM mesh motion + FAM surfactant transport
Lagrangian Particles

Hour-glass simulation
Lagrangian Particles

Diesel Combustion in Scania D-12 Engine

- 1/8 sector with 75 % load and n-heptane fuel
- RANS, $k - \epsilon$ turbulence model, simplified 5-species chemistry and 1 reaction, Chalmers PaSR combustion model
- Temperature on the cutting plane
- Spray droplets coloured with temperature
Lagrangian Particles

Diesel Combustion in Scania D-12 Engine
Fluid-Solid Coupling

Pipeline failure: crack propagation and leakage
Fluid-Solid Coupling

Enlarged deformation of the pipe
Summary

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

Acknowledgements

- Henrik Rusche and Eugene de Villiers, Imperial College
- Željko Tuković, University of Zagreb; Niklas Nordin, Chalmers University
- Vlado Tropša (Imperial College), Alojz Ivanković, UC Dublin
FOAM: CCM in C++

Main characteristics

- Wide area of applications: all of CCM!
- Shared tools and code re-use

Versatility

- Unstructured meshes, automatic mesh motion + topological changes
- Finite Volume, Finite Element, Lagrangian tracking and Finite Area methods
- Efficiency through massive parallelism