Numerical Modelling in Continuum Mechanics

Hrvoje Jasak

h.jasak@wikki.co.uk

Wikki Ltd. United Kingdom

22/Mar/2005
Outline

Objective

- Present a new way of handling Continuum Mechanics models in numerical software

Topics

- A new approach to model representation
- Object-orientation in numerical simulation software: code re-use and layered design
- Examples of complex model implementation
State of the Art

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

- Two-fold requirements
  - Ease of use and process integration
  - Quick and reliable model implementation
Numerics for CCM

How to handle complex models in software?

- Natural language of continuum mechanics: partial differential equations

\[
\frac{\partial k}{\partial t} + \nabla \cdot (u_k) - \nabla \cdot [(\nu + \nu_t) \nabla k] =
\]

\[
\nu_t \left[ \frac{1}{2} (\nabla u + \nabla u^T) \right]^2 - \frac{\epsilon_o}{k_o} k
\]
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)
);
```
# 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>
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{};
Model Interaction

Model-to-model interaction

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

New components do not disturb existing code
Geometry Handling

Complex geometry requirements

- Complex geometry is a rule, not exception
- Polyhedral cell support
  - Cell described as a polyhedron bounded by polygons
  - Consistent handling of all cell types
  - More freedom in mesh generation
- Recent developments: polyhedral FVM provides equivalent accuracy at lower cost
Geometry Handling

[Images of 3D grid structures]
Geometry Handling

Time-varying geometry cases

- Automatic mesh motion
- Topological mesh changes with poly support
Speed of Execution

Handling large-scale computations

- Efficient and accurate numerics
  - Best discretisation practice for a given problem
  - Iterative solvers almost inevitable
  - Careful analysis of non-linearity and inter-equation coupling

- Massive parallelism: domain decomposition
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!**
Examples of Application

Illustrates examples of FOAM library in use

- Concentrating on complex physics modelling: this is what FOAM is best at!
- Foundation work: numerics, mesh handling, accuracy, efficiency, validation etc.
- Examples chosen from recent work/interests
- Some other research and application areas not fairly represented
Droplet Splash

Two-phase incompressible system

\[ \frac{\partial \gamma}{\partial t} + \nabla \cdot (u\gamma) = 0 \]

\[ \nabla \cdot u = 0 \]

\[ \frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho uu) - \nabla \sigma = -\nabla p + \rho f + \sigma \kappa \nabla \gamma \]

\[ u = \gamma u_1 + (1 - \gamma)u_2 \]

\[ \mu, \rho = \gamma \rho_1 + (1 - \gamma)\rho_2 \]
Droplet Splash

Droplet impact into a wall film, 1.3 million cells

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

Droplet impact into a wall film, cutting plane

Splash, $u = 50\text{m/s}$, $d = 0.3\text{mm}$
- Ice represented as a 2-D continuum: \((h, A)\)
- Ice interaction model: Hibler 1979

\[
\sigma = 2\eta \dot{\varepsilon} + I (\zeta - \eta) \left[ tr(\dot{\varepsilon}) - \frac{P(h, A)}{2} \right]
\]

\[
\zeta = \frac{P(h, A)}{2\Delta}; \quad \eta = \frac{\zeta}{e^2}
\]

\[
\Delta = \sqrt{\left(1 - \frac{1}{e^2}\right) tr(\dot{\varepsilon})^2 + \frac{2}{e^2} \dot{\varepsilon} : \dot{\varepsilon}}
\]
Ice Modelling

- Wind + ocean current forcing
- Coriolis force, mean water surface gradient
- Simple melting and freezing model
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
Diesel Combustion in Scania D-12 Engine
Fluid-Solid Coupling

Pipeline failure: crack propagation and leakage

[Diagram showing RCP Along Gas Pressurised Pipe with time at 4.8 ms, showing velocity and absolute pressure distributions]
Fluid-Solid Coupling

Enlarged deformation of the pipe
Surface tracking

Free surface tracking

- 2 phases = 2 meshes
- Mesh adjusted for interface motion
- Coupled b.c.

Air-water system

- 2-D: \( r_b = 0.75 \text{ mm} \)
- 3-D: \( r_b = 1 \text{ mm} \)
Surfactant Effect

Clean surface

Pollution by surfactant chemicals
3-D Rising Bubble

Complex coupling problem: FVM flow solver + FEM mesh motion + FAM surfactant transport
Current and Future Work

Grocery list for future research

- Free surface tracking, solid-fluid interaction
- LES and LES free surface, multi-phase flows
- Polyhedral mesh generation, topological changes, internal combustion engines
- Non-linear stress analysis, crack propagation
- Algorithmic differentiation, adjoints (discrete and continuous) and error estimation
Summary

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

Acknowledgements

- Ice Modelling: Dr. Jenny Hutchings University of Fairbanks, Alaska
- Cracking pipe: Dr. Vlado Tropša, prof. Alojz Ivanković, UC Dublin
- Surface tracking: Dr. Željko Tuković, University of Zagreb, Croatia
- Foam and OpenFOAM are released under GPL: http://www.openfoam.org
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