

# In-Cylinder Flows with OpenFOAM

**Hrvoje Jasak**

`h.jasak@wikki.co.uk`

**Wikki Ltd, United Kingdom**

**FSB, University of Zagreb, Croatia**

**18/Nov/2005**

## Objective

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

## Topics

- OpenFOAM: Object-oriented software for Computational Continuum Mechanics (CCM)
- A new approach to model representation
- Mesh handling and discretisation support
- In-cylinder flow (related) capabilities

## 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.*

## Simulation Challenges: Very Demanding!

- Complex mathematical model
  - Compressible (transonic) fluid flow
  - Turbulence, combustion, chemistry
  - Lagrangian particles for spray
  - Wall interaction and wall film
- Complex geometry handling: moving mesh, topological changes (valve action)
- Model-to-model interaction: complex coupling

How to handle complex models in software?

- Natural language of continuum mechanics: partial differential equations

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

- Main object = **operator**, e.g. time derivative, convection, diffusion, gradient

## 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

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

# Object Orientation

- Field algebra

Object	Software representation	C++ Class
Field	List of values	Field
Boundary condition	Values + condition	patchField
Dimensions	Dimension Set	dimensionSet
Geometric field	Field + boundary conditions	geometricField
Field algebra	+ - * / <i>tr()</i> , <i>sin()</i> , <i>exp()</i> ...	field operators

- Matrix and solvers

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

# Object Orientation

- Numerics

Object	Software representation	C++ Class
Interpolation	Differencing schemes	interpolation
Differentiation	ddt, div, grad, curl	fvc, fec
Discretisation	ddt, d2dt2, div, laplacian	fvm, fem, fam

Implemented Methods: Finite Volume, Finite Element, Finite Area and Lagrangian tracking

- Top-level organisation

Object	Software representation	C++ Class
Model library	Library	turbulenceModel
Application	main()	–

# 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. evaporation model, *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 + topo 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!**

## Challenges in Simulating In-Cylinder Flows

- Robust FVM solver, turbulence and combustion, chemistry, gas properties *etc.*
- Mesh handling: motion and topo changes
- Lagrangian particle tracking
- Numerics to support wall film: Finite Area
- Efficiency: massive parallelism with domain decomposition

... and then lots and lots of models!

## Handling Shape Change: Problem Specification

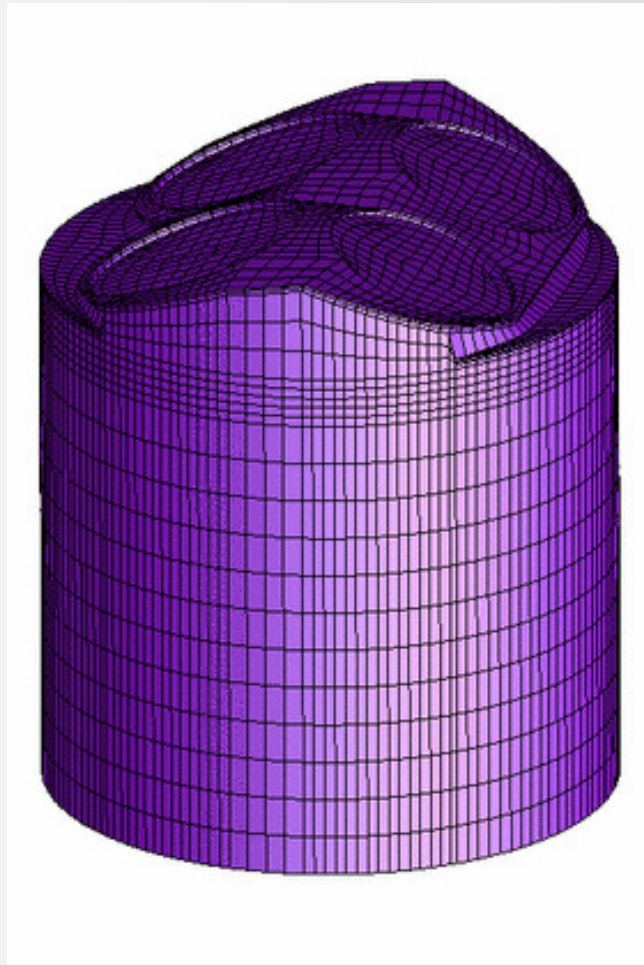
- Initial valid mesh is available
- Time-varying boundary motion
  - Prescribed in advance: *e.g.* IC engines
  - Part of the solution: surface tracking
- Need to determine internal point motion based on prescribed boundary motion
- Mesh in motion must remain valid: face and cell flip must be prevented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

## Solution Technique

- Point position will be provided by solving an equation given boundary motion conditions
- Variants of the motion solver
  - Cell-based methods fail: interpolation
  - Spring analogy: unreliable
  - Vertex-based (FEM) with polyhedral cell support: mini-element technique
- Choice of equation: Laplace or pseudo-solid

## Implementation: Polyhedral FEM Solver

- Substantial code re-use
  - Mesh support, vectors and tensors
  - Geometric fields and field algebra
  - Linear matrix support and solvers
- Additional implementation required
  - Patch fields and matrix interaction
  - FE calculus, *e.g.* gradient calculation
  - FE discretisation: making matrices



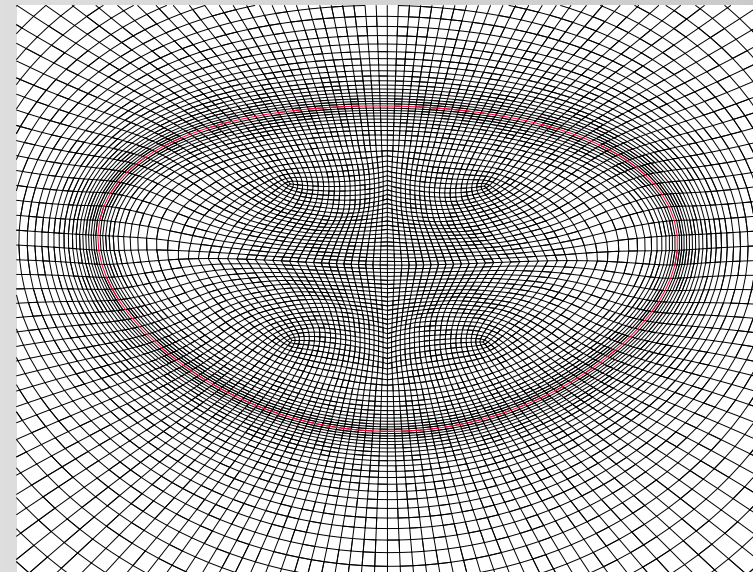
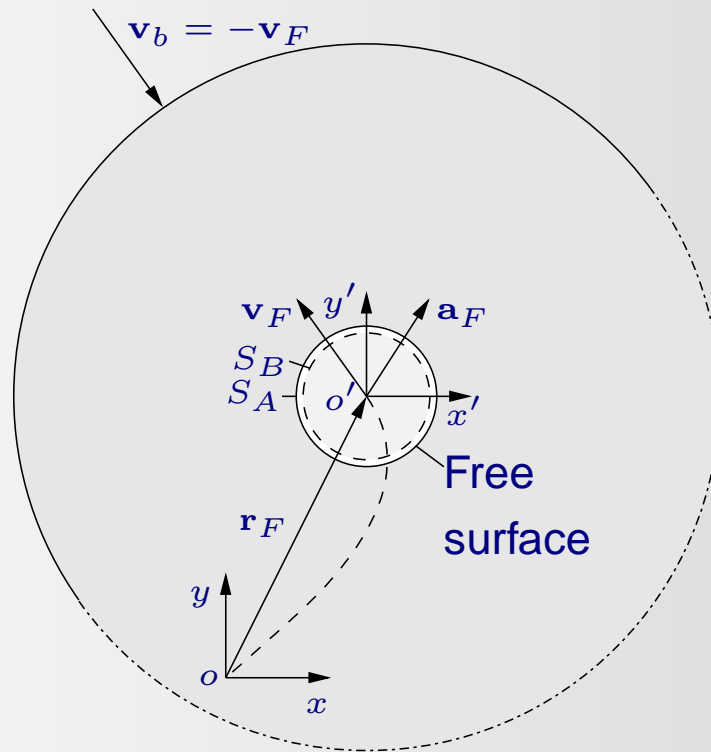
## Automatic Mesh Motion

- Easy to set up
- Improved robustness
- Control over mesh quality and spacing: diffusivity
- Pseudo-solid approach better but more expensive: component coupled vs. segregated approach

# Automatic Mesh Motion

## Free surface tracking

- 2 phases = 2 meshes
- Mesh adjusted for interface motion



# Topological Changes

## Supporting Topological Changes

1. Define primitive mesh operations:  
Add/modify/remove point/face/cell
2. Support primitive operations in mesh classes
3. Describe mesh modifiers in terms of primitive mesh operations
4. Collect changes from mesh modifiers into topo change request
5. Execute topo change, including field mapping!

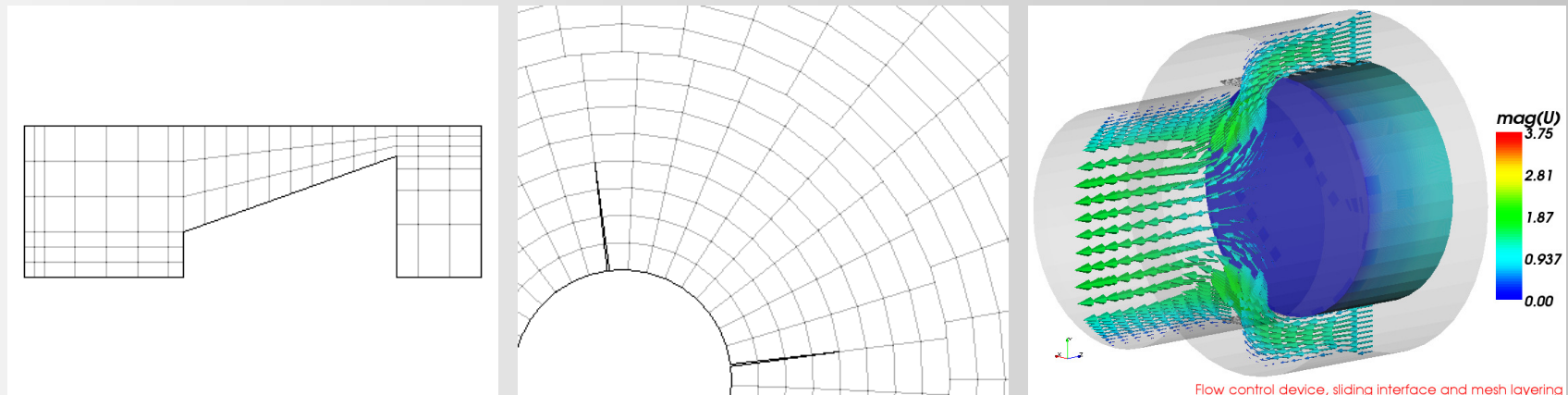
# Topological Changes

## Topology Engine

- Topology engine = list of “mesh modifiers”
  - Attach/detach boundary
  - Layer addition/removal
  - Sliding interface
- Each mesh modifier self-contained, including triggering condition
- Flow solver presented with valid mesh and mapped data + mesh motion info

# Topological Changes

## Topological Changes on Polyhedral Meshes



- This is the current point of development
- Intersecting topo modifiers, parallelisation and bugs (sliding), unified solver support

## Class Hierarchy

- `Particle`: a point recording its position and location in a mesh
- `Cloud`: *is-a* group of `particles` with addition, removal and tracking capabilities
- `Spray Parcel` = `Particle` *is-a* with properties: diameter, population, temperature
- `Spray tracking` *is-a* cloud of `parcels`

## Diesel Spray Model

- Spray tracking + sub-models: atomisation, breakup, collision, dispersion, drag, evaporation heat transfer, wall interaction
- Each class of sub-models answers to generic interface for its class
- `spray::evolve()`: track + update models

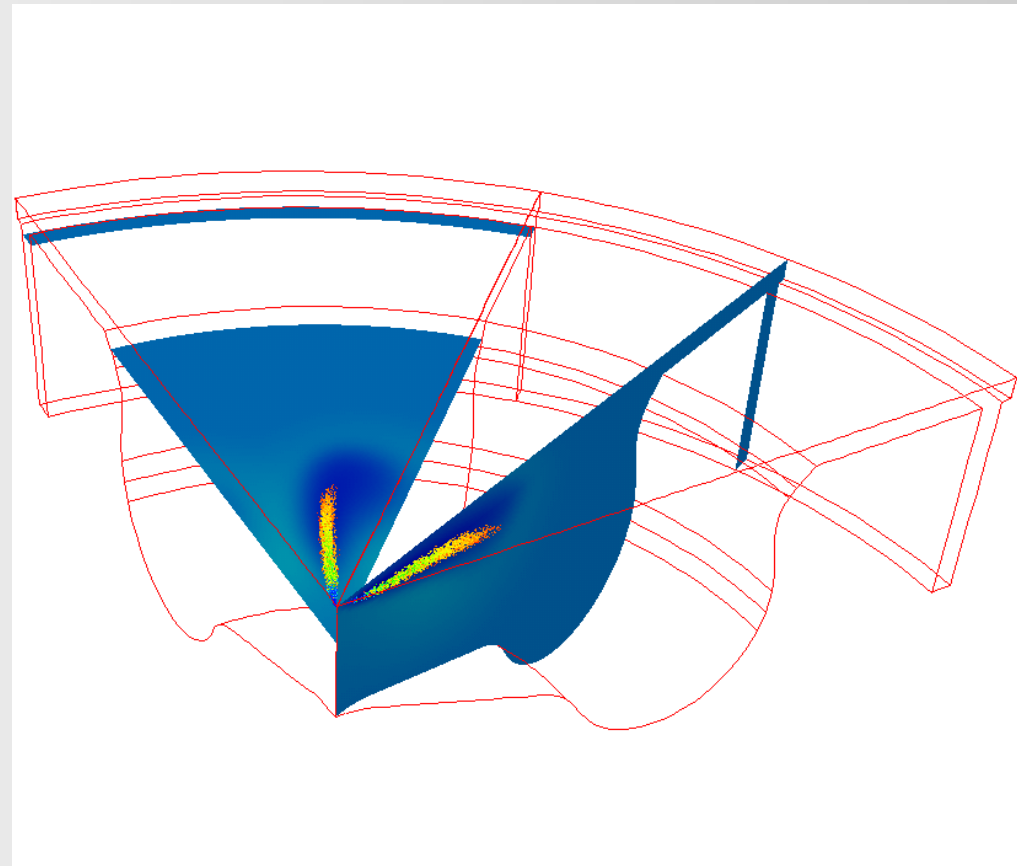
Hierarchical implementation with code re-use:  
object orientation + generic programming

## 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

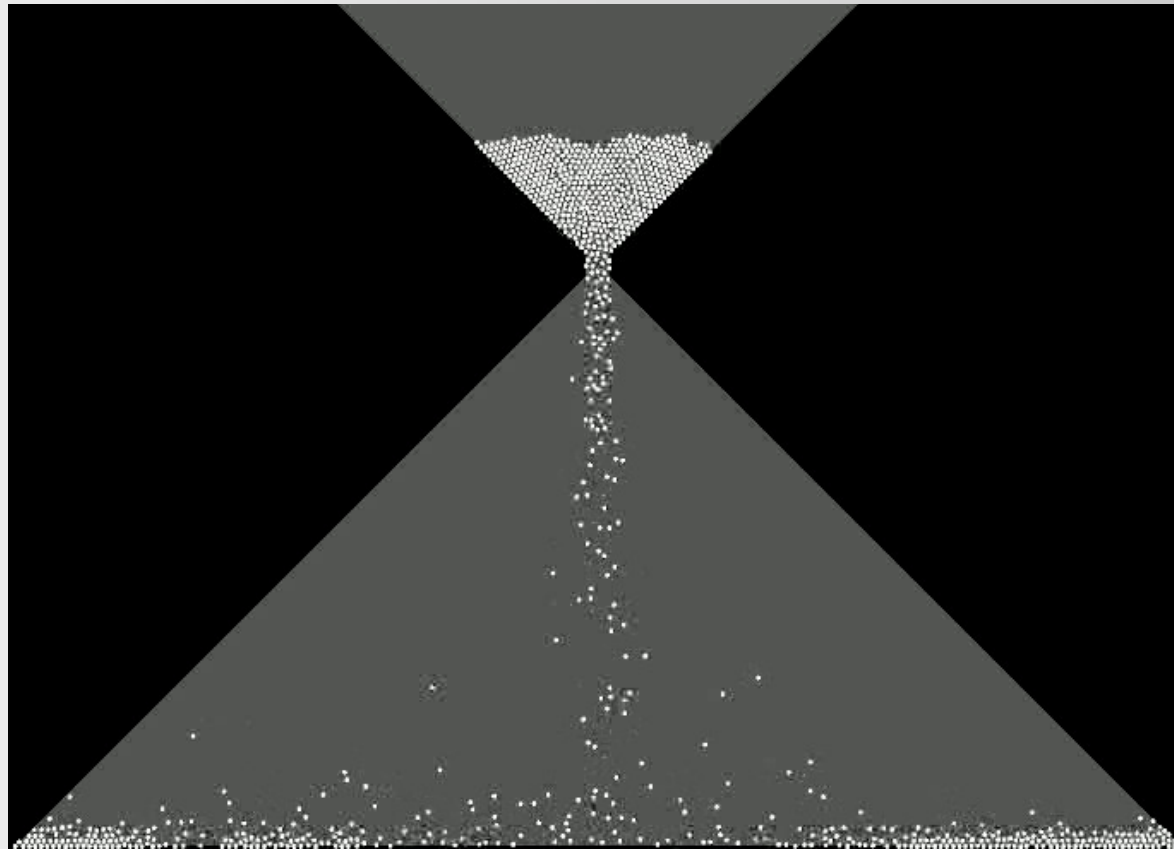
# Diesel Combustion

## Diesel Combustion in Scania D-12 Engine



# Lagrangian Particles

Hour-Glass: same tracking, different particles



# Wall Film Model

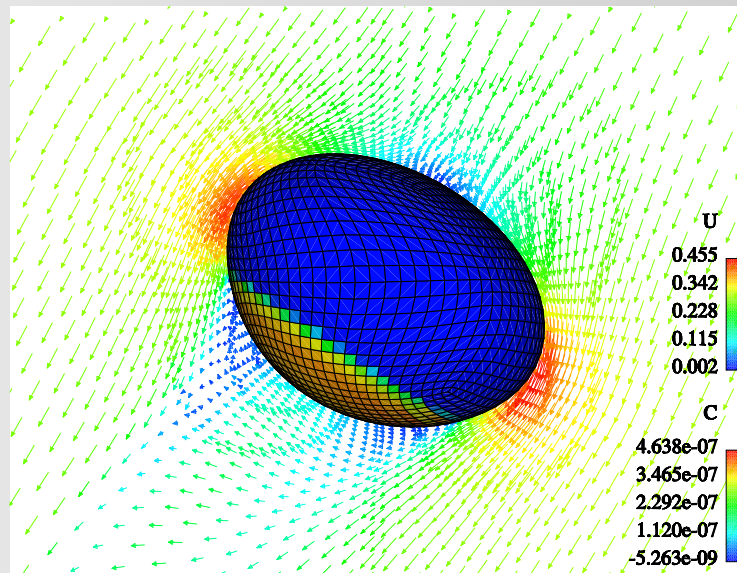
## Finite Area Method (FAM)

- Need to solve transport equations on a curved 2-D surface in 3-D
- Polygonal mesh support: boundary of a polyhedral mesh
- Discretisation almost identical to 3-D FVM
- ... but with corrections for surface curvature

Implementation effort identical to the FEM solver;  
additional code re-use with 3-D FVM

# Coupled Simulation

## Free-Rising Air Bubble with Surfactants



Complex coupling problem:

FVM flow solver + FEM automatic mesh motion +  
FAM for surfactant transport

# 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 and Further Info

- Željko Tuković, University of Zagreb, Niklas Nordin, Scania and Chalmers Uni
- For more info on OpenFOAM, please visit <http://www.openfoam.org>
- Free OpenFOAM Workshop in Zagreb, Croatia 26-28/Jan/2006:  
<http://powerlab.fsb.hr/ped/kturbo/FsbOpenFOAMWorkshop/>

# 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