Features in OpenFOAM-extend

Henrik Rusche and Hrvoje Jasak

`h.rusche@wikki-gmbh.de, h.jasak@wikki.co.uk`

Advanced OpenFOAM Training ISPRAS

Moscow, 13. December 2011

Background

Objective

- Review of features unique to OpenFOAM-extend

Topics

1. Domain Coupled Solution Algorithmns: **Coupled Matrices**
2. Equation Coupled Solution Algorithmns: **Block Matrices**
3. Summary

Domain Coupling Test Case

- Steady-state conjugate heat transfer to an incompressible, laminar fluid
- Fluid:

$$\nabla \bullet (\mathbf{u}\mathbf{u}) - \nabla \bullet \nu \nabla \mathbf{u} = -\nabla p \quad (1)$$

$$\nabla \bullet \mathbf{u} = 0 \quad (2)$$

$$\nabla \bullet (\mathbf{u}T) - \nabla \bullet K(\nabla T) = 0 \quad (3)$$

- Solid:

$$-\nabla \bullet K_s(\nabla T_s) = 0 \quad (4)$$

- Interface:

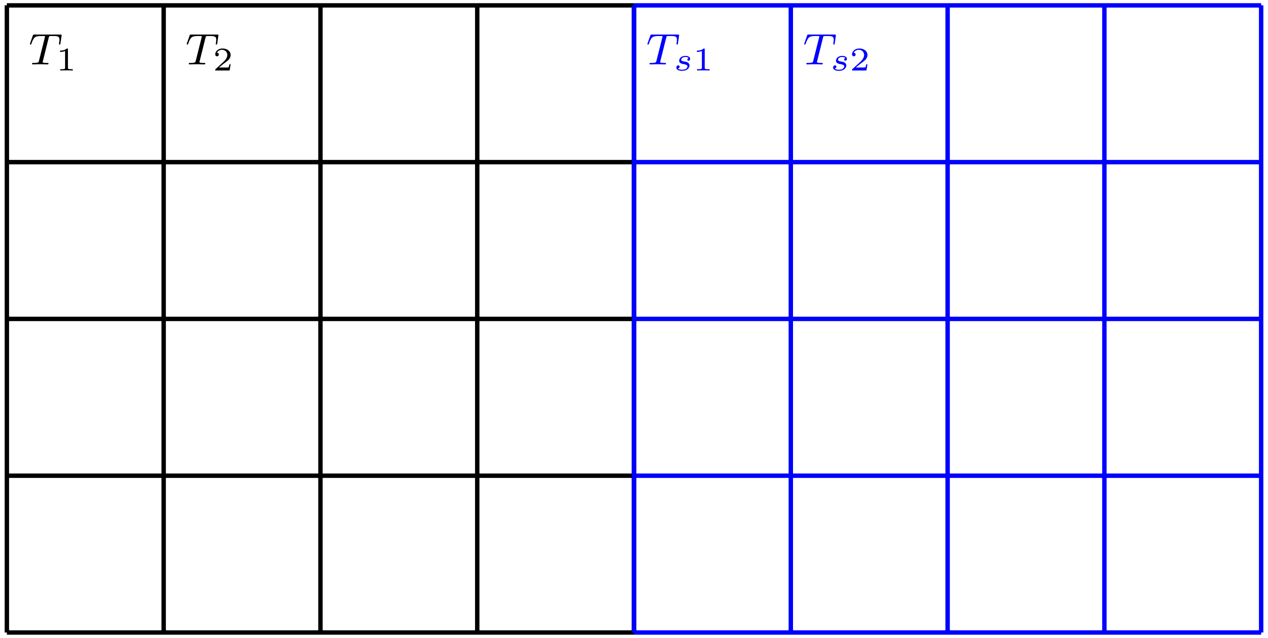
$$T = T_s \quad (5)$$

$$K \nabla T = K_s \nabla T_s \quad (6)$$

Implicit Domain Coupling

- Explict Implementation in OpenFOAM is straight-forward using its multi-domain capabilities
- But in many cases, explicit coupling (Picard iterations) simply does not work or it is too slow
- Discretisation machinery in OpenFOAM is satisfactory and needs to be preserved
- Multi-domain support must allow for some variables/equations to be coupled, while others remain separated
- Example: conjugate heat transfer
  - Fluid flow equations solved on fluid only
  - Energy equation discretised separately on the fluid and solid region but solved in a single linear solver call
- Combining variables or addressing spaces into implicit coupling requires special practices and tools
- Historically, conjugate heat transfer in many CFD codes is “hacked” as a special case: we need a **general arbitrary matrix-to-matrix coupling**
- The problem was insufficient flexibility of matrix support

Mesh and Matrix for Domain Coupling

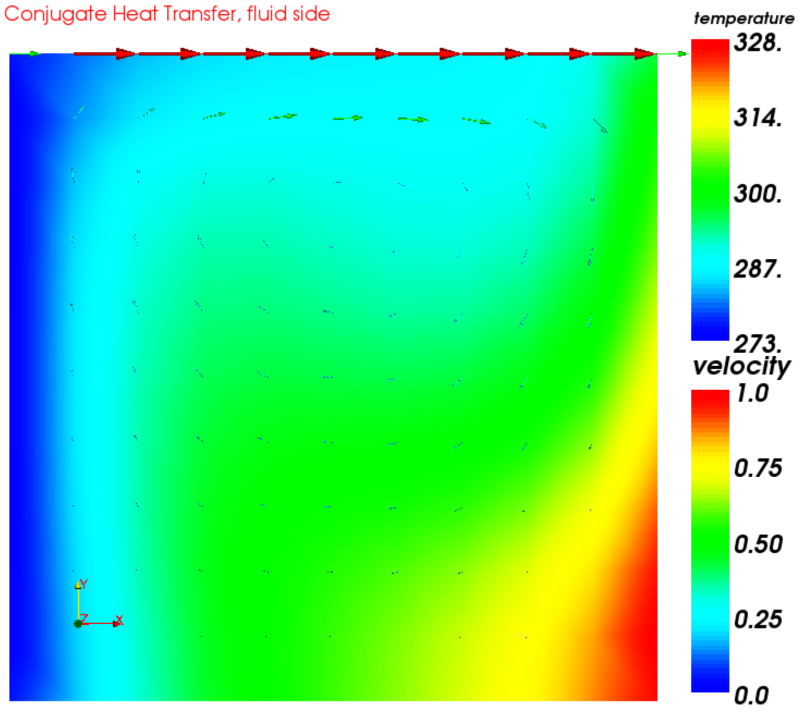


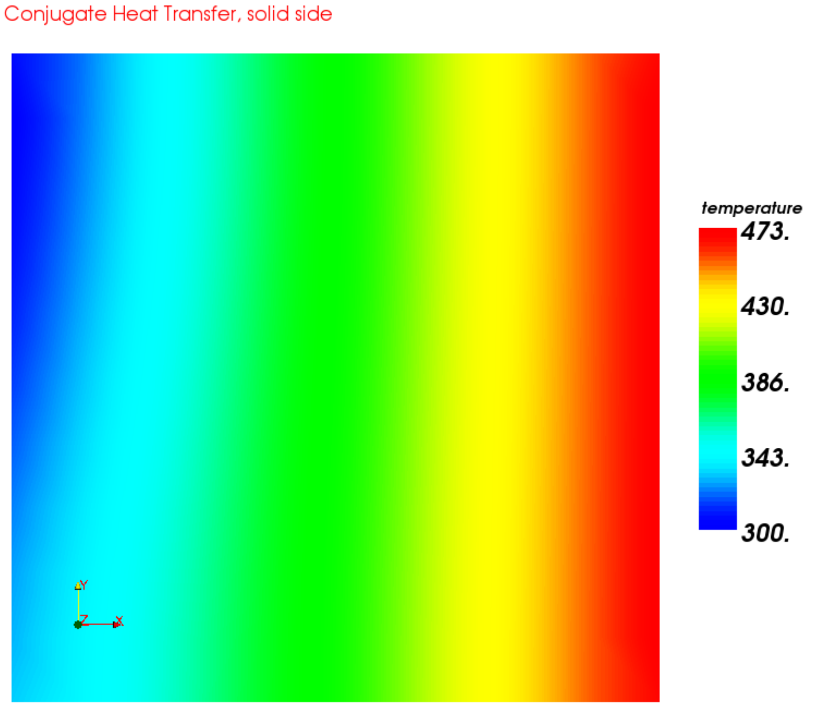
$$
\begin{bmatrix}
a & \cdot & & \\
 & & & \cdot \\
\cdot & \ddots & & \\
 & & a & \cdot \\
 & \cdot & & \\
 & & \cdot & \ddots
\end{bmatrix}
\begin{bmatrix}
T_1 \\ \vdots \\ T_{s1} \\ \vdots
\end{bmatrix}
=
\begin{bmatrix}
b_2 \\ \vdots \\ b_{s1} \\ \vdots
\end{bmatrix}
\tag{7}
$$

Domain Coupled Solution Algorithms

Example: Conjugate Heat Transfer

- Coupling may be established geometrically: adjacent surface pairs
- Each variable is stored only on a mesh where it is active: (U, p, T)
- Choice of conjugate variables is completely arbitrary: e.g. catalytic reactions
- Coupling is established only per-variable: handling a general coupled complex physics problem rather than conjugate heat transfer problem specifically

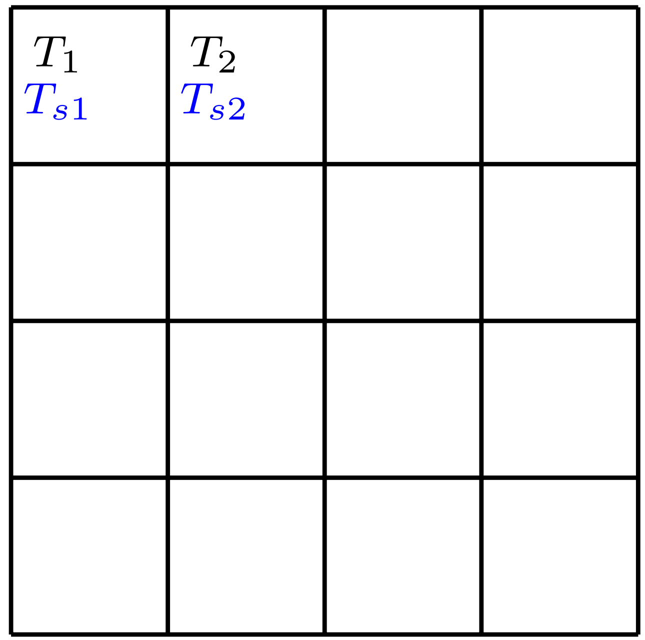




Equation Coupling Test Case

- Steady-state conjugate heat transfer between a porous medium and a fluid flowing through it - Frozen flow field
- Fluid: $$\nabla\bullet(\mathbf{u}T) - \nabla\bullet K(\nabla T) = \alpha(T_s - T) \quad (8)$$
- Solid: $$-\nabla\bullet K_s(\nabla T_s) = \alpha(T - T_s) \quad (9)$$
- Frozen flow field: $$\mathbf{u} = (0, 0, -1) \times (\mathbf{x} - \mathbf{x}_0) \quad (10)$$

Variable Layout Domain Coupling



Segregated Algorithmn

- Implementation is trivial: This is what OpenFOAM was designed for!

```
fvScalarMatrix TEqn
(
    fvm::div(phi, T)
  - fvm::laplacian(DT, T)
  ==
    alpha*Ts - fvm::Sp(alpha, T)
);

TEqn.relax(); TEqn.solve();

fvScalarMatrix TsEqn
(
  - fvm::laplacian(DTs, Ts)
  ==
    alpha*T - fvm::Sp(alpha, Ts)
);

TsEqn.relax(); TsEqn.solve();
```

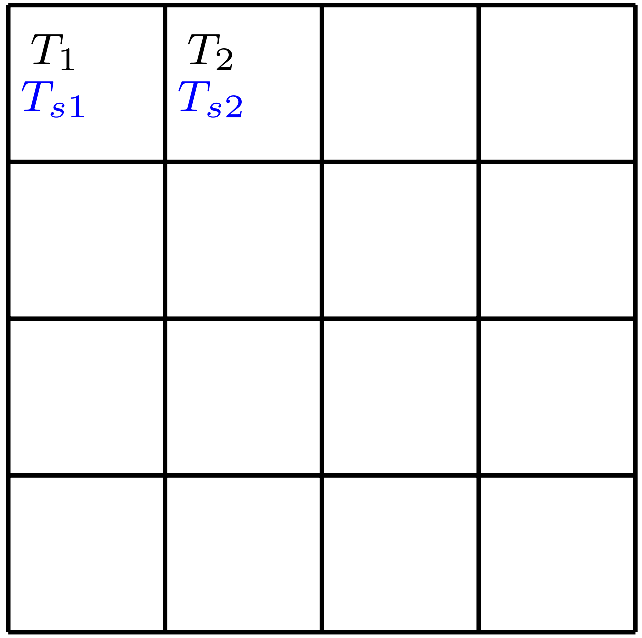
Equation Coupling Idea

- How to couple T and T_s implicitly? They depend on each other in a single cell, through source term linearisation
- Introducing a vector variable at each cell!

  $$\mathbf{\Phi} = \begin{bmatrix} T \\ T_s \end{bmatrix}$$

- Matrix coefficients become tensors, as presented in the block matrix structure ... How does this look like?

Mesh and Matrix Equation (Block)Coupling



$$\begin{bmatrix} \begin{pmatrix} a_{ff} & a_{fs} \\ a_{sf} & a_{ss} \end{pmatrix} & . & \cdots \\ . & \begin{pmatrix} a_{ff} & a_{fs} \\ a_{sf} & a_{ss} \end{pmatrix} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} T_1 \\ T_{s1} \\ T_1 \\ T_{s2} \\ \vdots \end{bmatrix} = \begin{bmatrix} b_1 \\ b_{s1} \\ b_2 \\ b_{s2} \\ \vdots \end{bmatrix} \tag{11}$$

Block Coupled Solution Algorithms

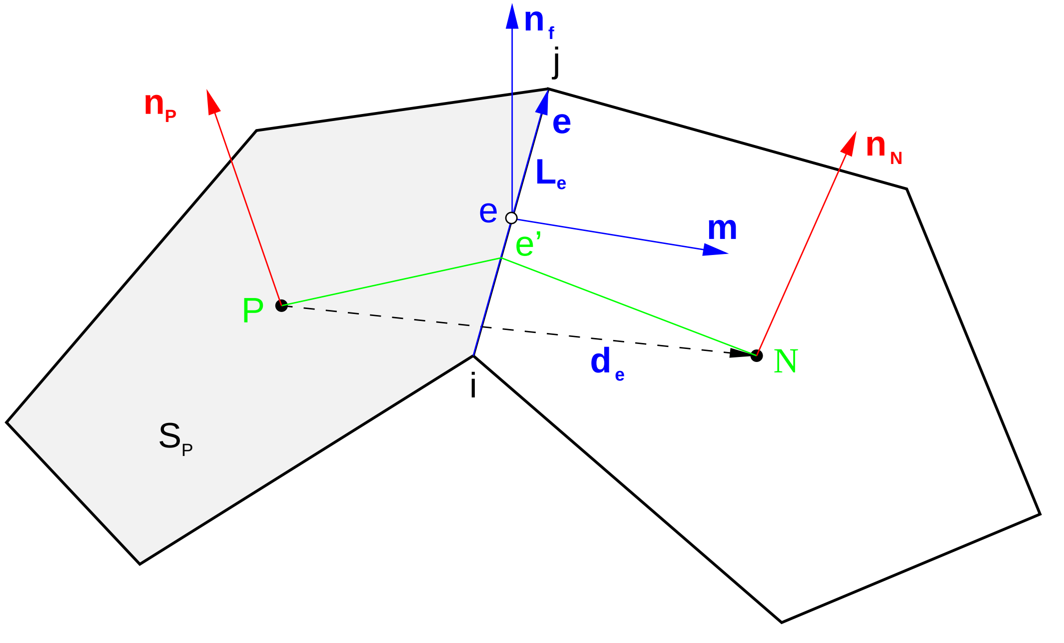
Block Matrix Implementation

- Implementation is general and includes off-diagonal coefficients
- Arbitrary number of equations can be coupled. a_P and a_N may be $n \times n$ tensors
- For vector components coupled in the same cell, a_P is a tensor
- For a vector cross-coupled to its neighbourhood, (e.g. x-to-y), a_N is a tensor
- Matrix algebra generalises to block coefficients, including linear solvers
- . . . and global sparseness pattern of the matrix is still dictated by the mesh!
- For efficiency, coefficient arrays are morphed: `scalar->linear->square` type

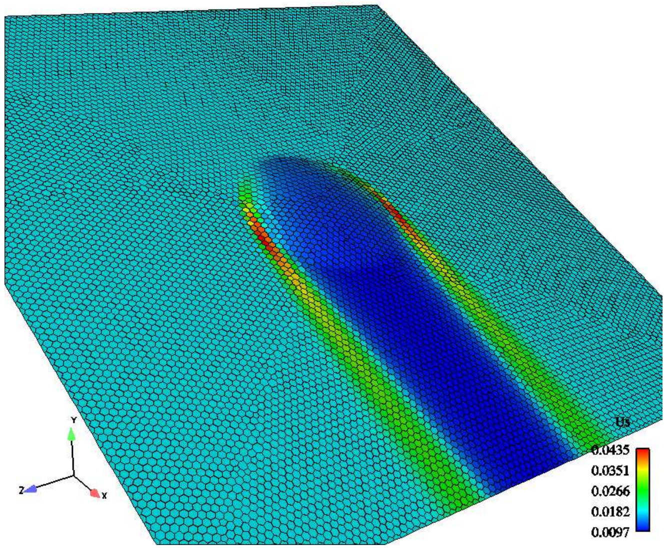
Finite Area Method

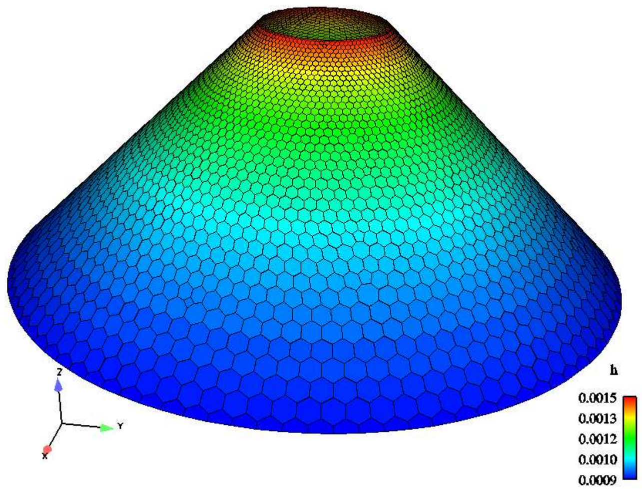
Background

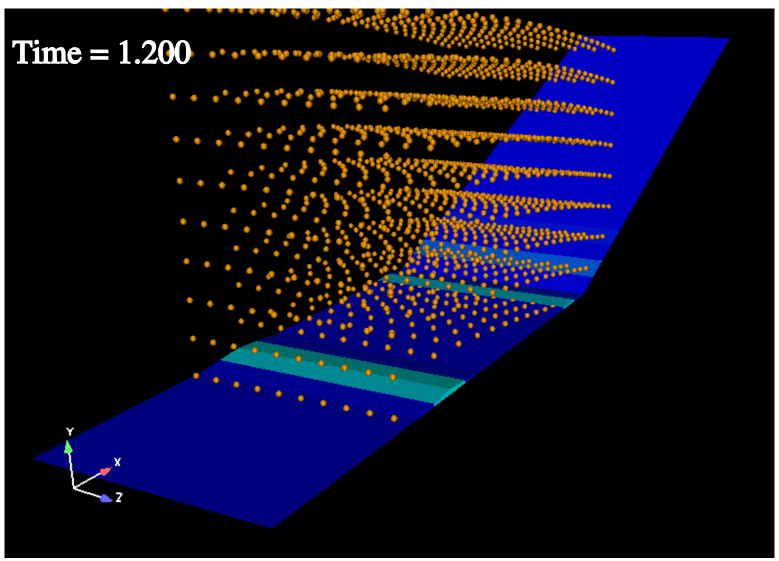
- Finite Area Method discretised equations on a curved surface in 3-D
- Surface is discretised using polygonal faces. Discretisation takes into account **surface curvature**. A level of smoothness is assumed in calculation of curvature terms
- Surface motion is allowed: decomposed into normal and tangential motion
- Nomenclature for a surface element P and its neighbour N



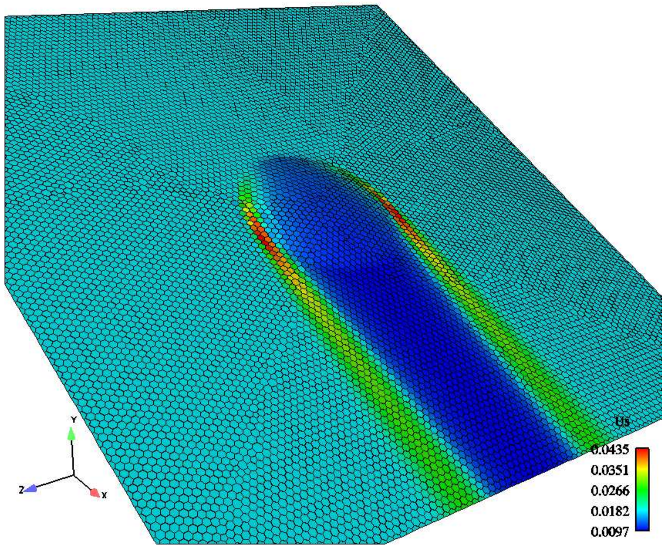
Liquid Film Model

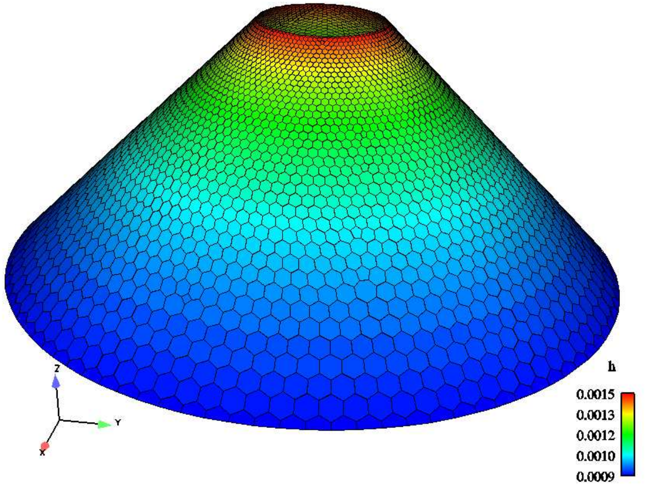


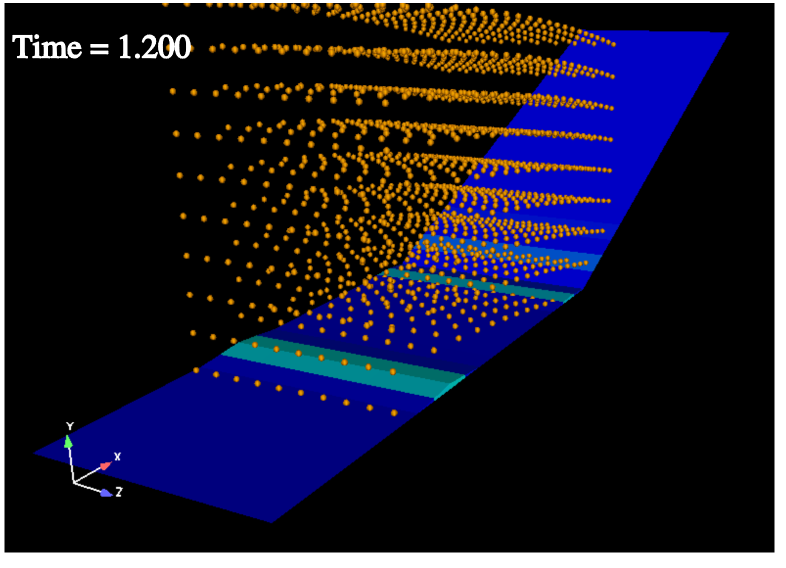




Liquid Film Model

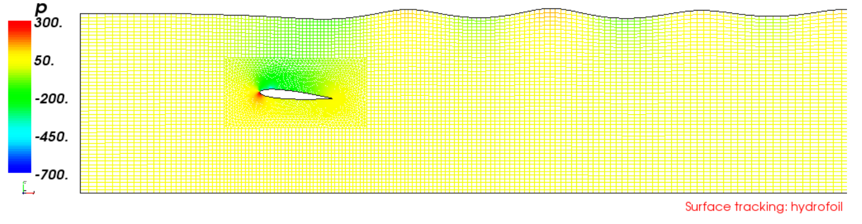




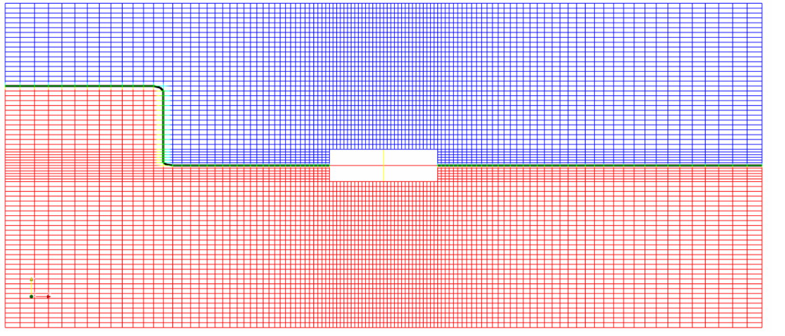


Automatic Motion – Examples

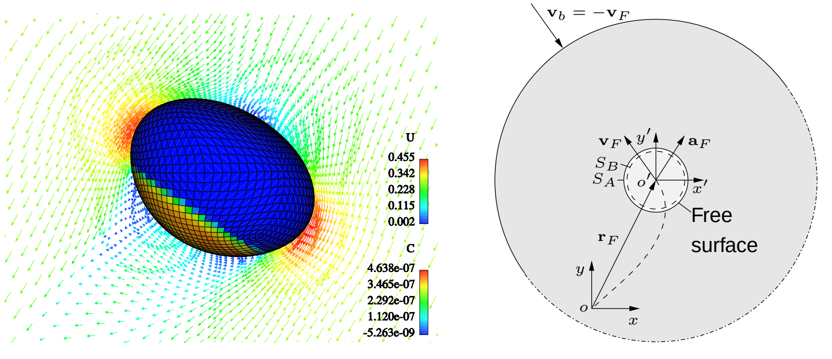
Hydrofoil Under a Free Surface



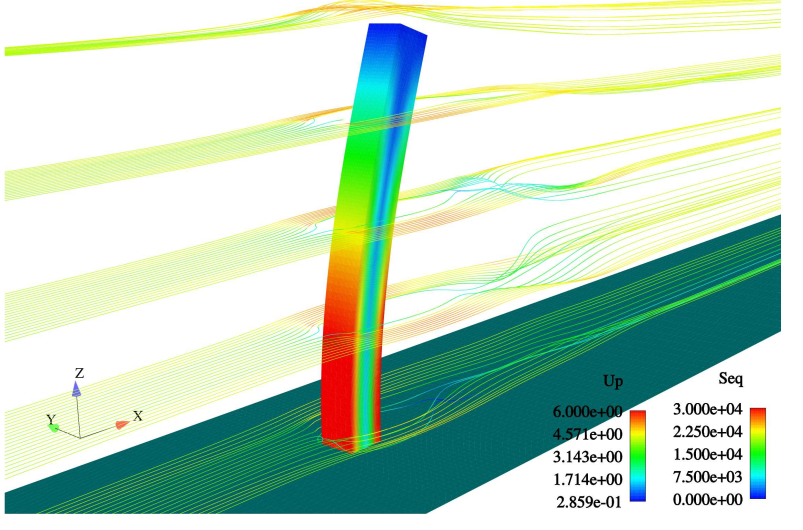
Floating body (6DOF)



Free-Rising Air Bubble with Surfactants



Vibration of a 3-D Beam

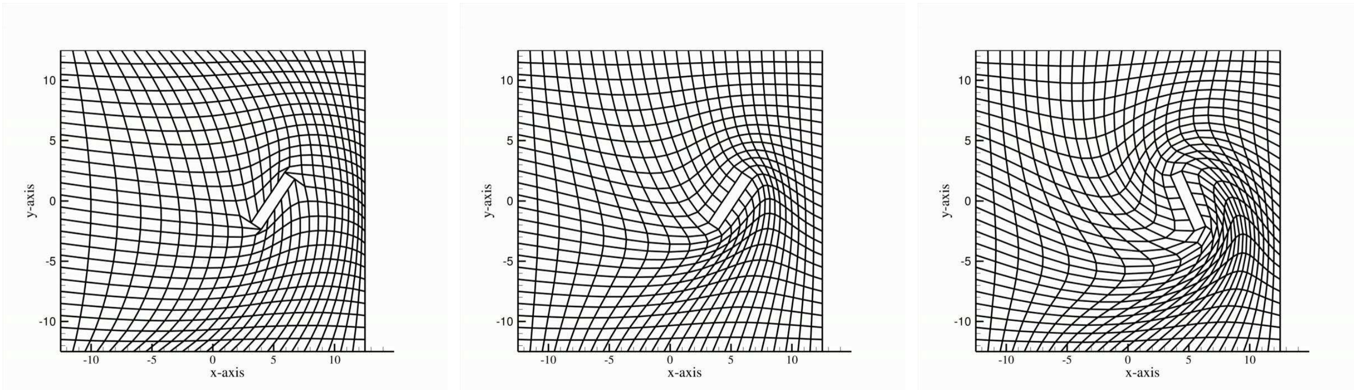


Radial Basis Function

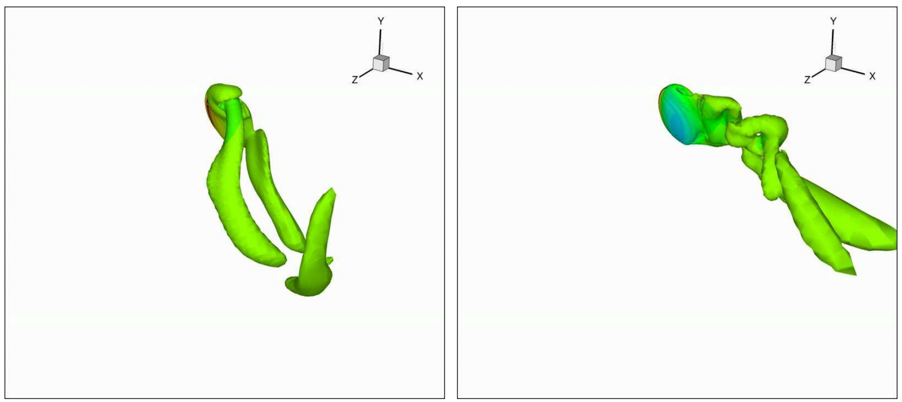
- RBF interpolation defines the interpolation directly from the sufficient smoothness criterion on the interpolation
- Deforming space as a function of motion of control points
- Method requires the solution of a dense matrix by direct solution. Only a small number of control points feasible
- Alternative use: Geometry morphing defined without reference to mesh or CAD

Radial Basis Function – Examples

Flapping wing test



Insect flight



Topological Mesh Changes

Topological Changes on Polyhedral Meshes

- For extreme cases of mesh motion, changing point positions is not sufficient to accommodate boundary motion and preserve mesh quality
- In a **topological change** the number or connectivity of points, faces or cells in the mesh is changed during the simulation
- Motion can be handled by the FVM with no error (moving volume), while a topological change requires additional algorithmic steps
- Cell insertion and deletion will formally be handled as a combination of mesh motion (collapsing cells and faces to zero volume/area) and a change in connectivity after the face and cell collapse

Implementation of topo changes

- **Primitive mesh operations**
  - Add/modify/remove a point, a face or a cell
  - This is sufficient to describe all cases, even to to build a mesh from scratch
  - . . . but using it directly is very inconvenient
- **Topology modifiers**
  - All mesh operations can be described in terms of primitive operations
  - Adding a user-friendly definition and triggering logic creates a “topology modifier” class
  - Examples: Attach-detach boundary, Cell layer additional-removal interface, Sliding interface, Error-driven adaptive mesh refinement

Examples of Topology Modifiers

"Set-and-Forget" Definition of Topology Modifiers

- `layerAdditionRemoval` mesh modifier removes cell layers when the mesh is compressed and adds cells when the mesh is expanding. Definition:
  - Oriented face zone, defining an internal surface
  - Minimum and maximum layer thickness in front of the surface
  - Both internal and patch faces are allowed
- `slidingInterface` allows for relative sliding of components. Definition:
  - A master and slave patch, originally external to the mesh
  - Allows uncovered master and slave faces to remain as boundaries

```
right
{
    type layerAdditionRemoval;
    faceZoneName rightExtFaces;
    minLayerThickness 0.0002;
    maxLayerThickness 0.0005;
    active on;
}
```

```
mixerSlider
{
    type slidingInterface;
    masterPatchName outsideSlider;
    slavePatchName insideSlider;
    projection visible;
    active on;
}
```

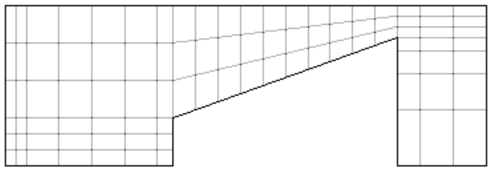
- Even for simple cases, it is easier to speak about problem classes (mixer vessels, engines, 6-DOF bodies) rather than working out individual topology modifiers

Topological Mesh Changes

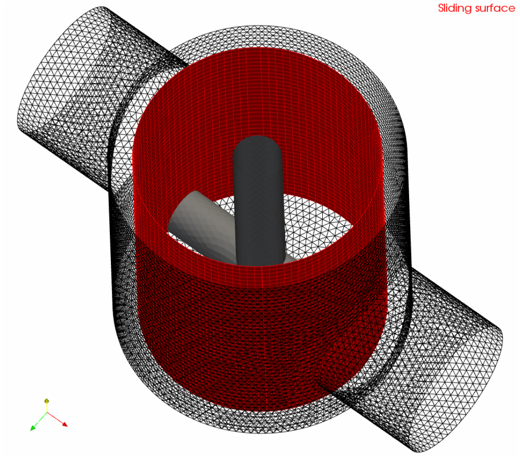
- **Primitive mesh operations**
  - Add/modify/remove a point, a face or a cell
  - This is sufficient to describe all cases, even to to build a mesh from scratch
  - . . . but using it directly is very inconvenient
- **Topology modifiers**
  - All mesh operations can be described in terms of primitive operations
  - Adding a user-friendly definition and triggering logic creates a “topology modifier” class
  - Examples: Attach-detach boundary, Cell layer additional-removal interface, Sliding interface, Error-driven adaptive mesh refinement
- **Dynamic meshes**
  - For complex topological changes, multiple interacting topology modifiers are used, need to be synchronised and used in unison with mesh motion
  - Combining topology modifiers and user-friendly mesh definition creates a “dynamic mesh” class
  - A dynamic mesh class talks the “language of the problem”
  - Examples: mixer mesh, 6-DOF motion, IC engine mesh (valves + piston), solution-dependent crack propagation in solid mechanics

Topological Mesh Changes – Examples

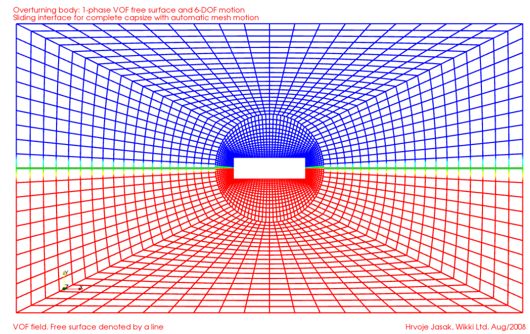
Moving Cone



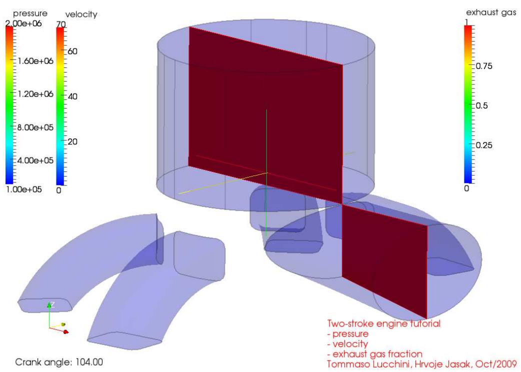
3D Mixer



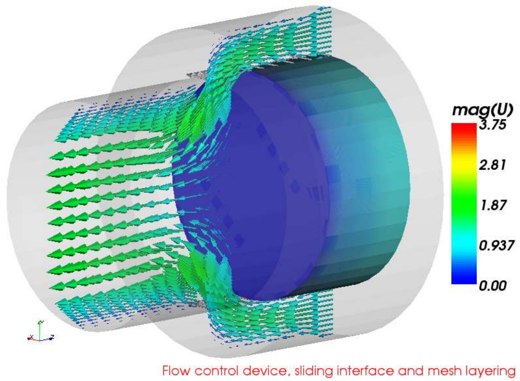
Overturning Floating Body



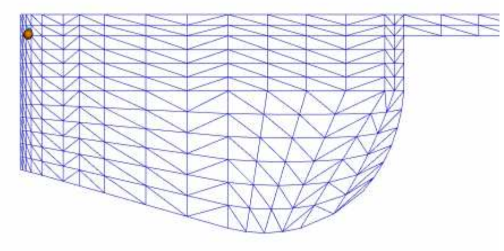
Two-Stroke Engine



Flow Control Device



Auto-refined Diesel Engine



Automatic Mesh Motion

Handling Shape Change: Problem Specification

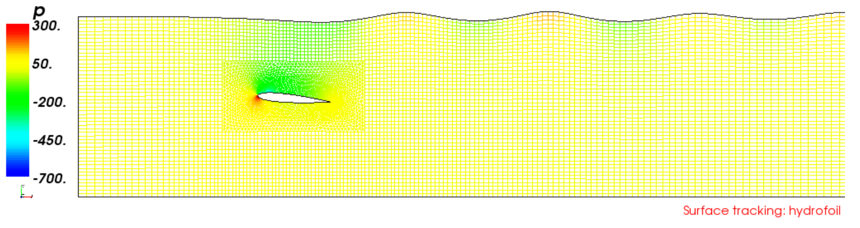
- Initial valid mesh is available
- Time-varying boundary motion
  - Prescribed in advance: *e.g.* IC engines
  - Part of the solution: surface tracking
- Need to determine internal point motion based on prescribed boundary motion
- Mesh in motion must remain valid: face and cell flip must be prevented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

Solution Technique

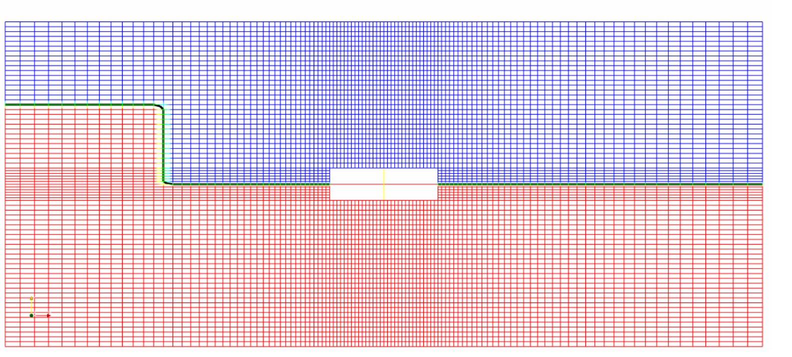
- Point position provided by solving an equation where motion of the boundary acts as the boundary condition for the motion equation
- Choice of motion equation: Laplace or pseudo-solid equation
- Details of mesh grading controlled by variable diffusivity
- Experience shows cell-based methods fail in interpolation; variants of spring analogy technique proved unreliable for large deformation
- **Vertex-based (FEM) mini-element discretisation** with polyhedral cell support

Automatic Motion – Examples

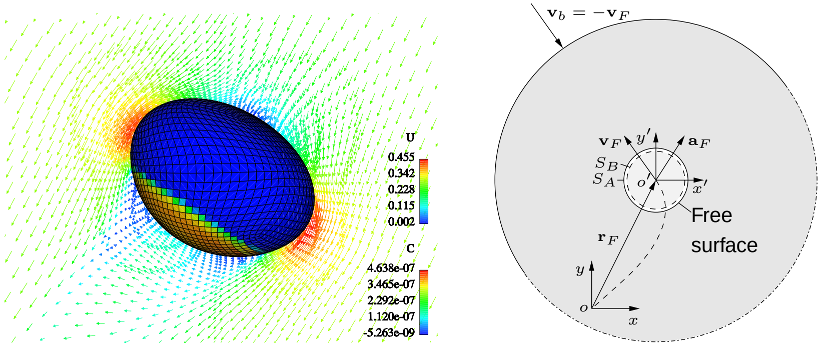
Hydrofoil Under a Free Surface



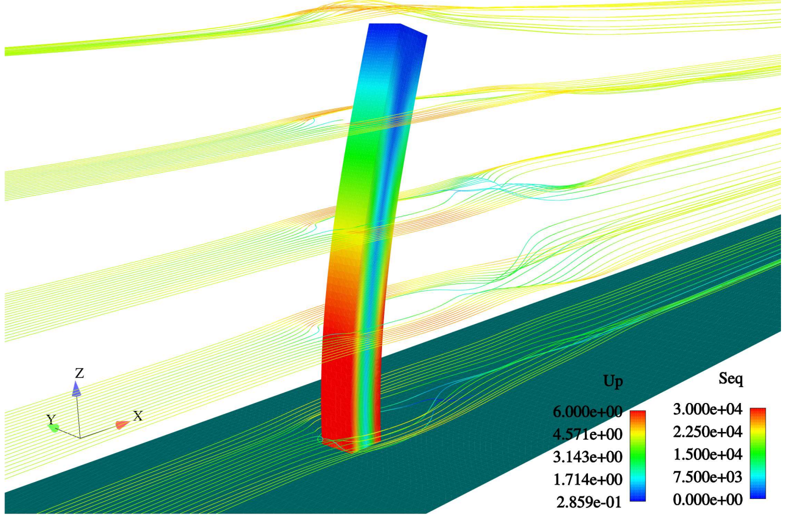
Floating body (6DOF)



Free-Rising Air Bubble with Surfactants



Vibration of a 3-D Beam

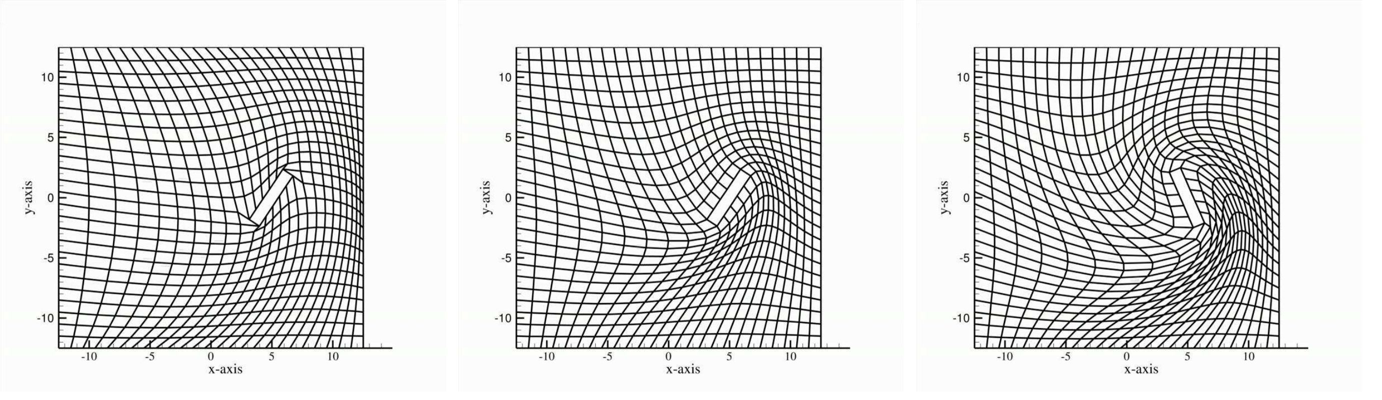


Radial Basis Function

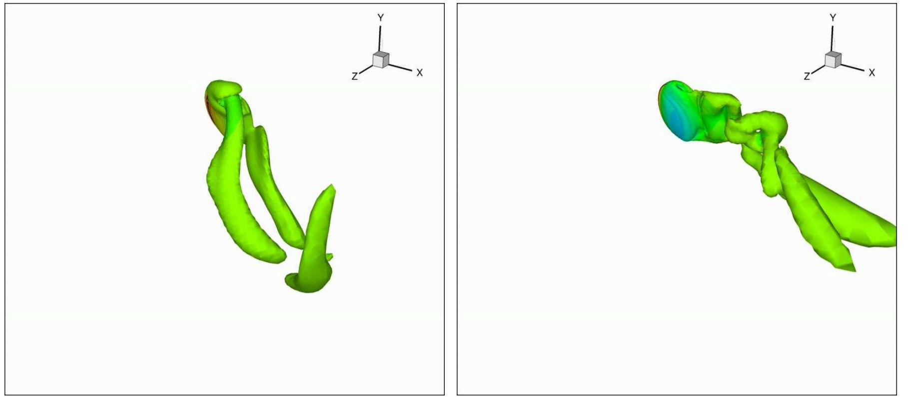
- RBF interpolation defines the interpolation directly from the sufficient smoothness criterion on the interpolation
- Deforming space as a function of motion of control points
- Method requires the solution of a dense matrix by direct solution. Only a small number of control points feasible
- Alternative use: Geometry morphing defined without reference to mesh or CAD

Radial Basis Function – Examples

Flapping wing test



Insect flight



Topological Mesh Changes

Topological Changes on Polyhedral Meshes

- For extreme cases of mesh motion, changing point positions is not sufficient to accommodate boundary motion and preserve mesh quality
- In a **topological change** the number or connectivity of points, faces or cells in the mesh is changed during the simulation
- Motion can be handled by the FVM with no error (moving volume), while a topological change requires additional algorithmic steps
- Cell insertion and deletion will formally be handled as a combination of mesh motion (collapsing cells and faces to zero volume/area) and a change in connectivity after the face and cell collapse

Implementation of topo changes

- **Primitive mesh operations**
  - Add/modify/remove a point, a face or a cell
  - This is sufficient to describe all cases, even to to build a mesh from scratch
  - . . . but using it directly is very inconvenient
- **Topology modifiers**
  - All mesh operations can be described in terms of primitive operations
  - Adding a user-friendly definition and triggering logic creates a “topology modifier” class
  - Examples: Attach-detach boundary, Cell layer additional-removal interface, Sliding interface, Error-driven adaptive mesh refinement

Examples of Topology Modifiers

"Set-and-Forget" Definition of Topology Modifiers

- `layerAdditionRemoval` mesh modifier removes cell layers when the mesh is compressed and adds cells when the mesh is expanding. Definition:
  - Oriented face zone, defining an internal surface
  - Minimum and maximum layer thickness in front of the surface
  - Both internal and patch faces are allowed
- `slidingInterface` allows for relative sliding of components. Definition:
  - A master and slave patch, originally external to the mesh
  - Allows uncovered master and slave faces to remain as boundaries

```
right
{
    type layerAdditionRemoval;
    faceZoneName rightExtFaces;
    minLayerThickness 0.0002;
    maxLayerThickness 0.0005;
    active on;
}
```

```
mixerSlider
{
    type slidingInterface;
    masterPatchName outsideSlider;
    slavePatchName insideSlider;
    projection visible;
    active on;
}
```

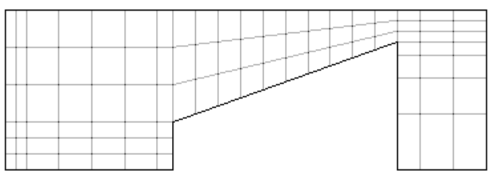
- Even for simple cases, it is easier to speak about problem classes (mixer vessels, engines, 6-DOF bodies) rather than working out individual topology modifiers

Topological Mesh Changes

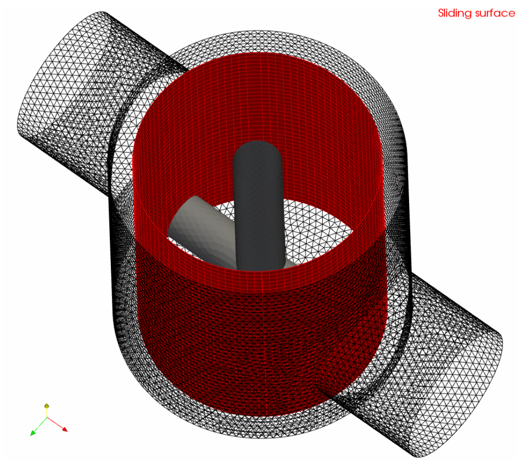
- **Primitive mesh operations**
  - Add/modify/remove a point, a face or a cell
  - This is sufficient to describe all cases, even to to build a mesh from scratch
  - . . . but using it directly is very inconvenient
- **Topology modifiers**
  - All mesh operations can be described in terms of primitive operations
  - Adding a user-friendly definition and triggering logic creates a “topology modifier” class
  - Examples: Attach-detach boundary, Cell layer additional-removal interface, Sliding interface, Error-driven adaptive mesh refinement
- **Dynamic meshes**
  - For complex topological changes, multiple interacting topology modifiers are used, need to be synchronised and used in unison with mesh motion
  - Combining topology modifiers and user-friendly mesh definition creates a “dynamic mesh” class
  - A dynamic mesh class talks the “language of the problem”
  - Examples: mixer mesh, 6-DOF motion, IC engine mesh (valves + piston), solution-dependent crack propagation in solid mechanics

Topological Mesh Changes – Examples

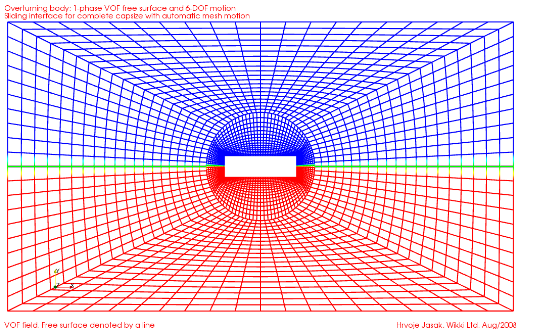
Moving Cone



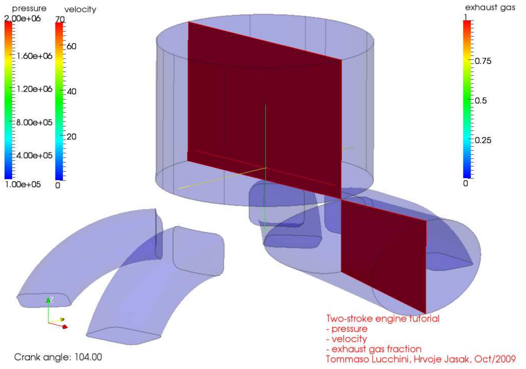
3D Mixer



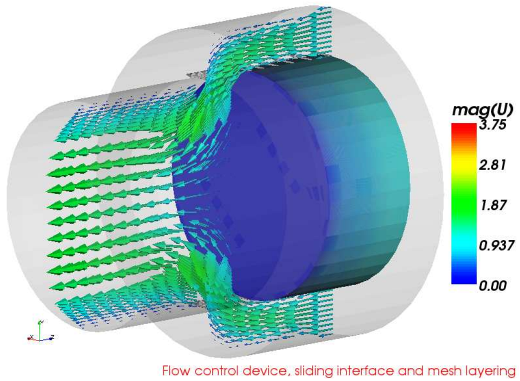
Overturning Floating Body



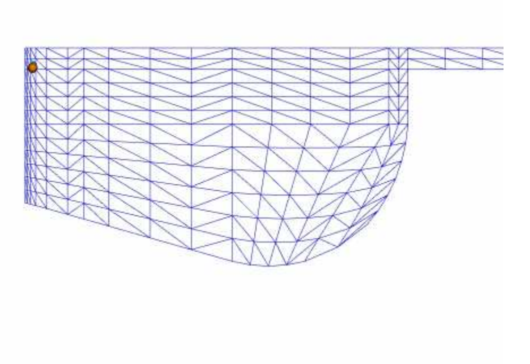
Two-Stroke Engine



Flow Control Device

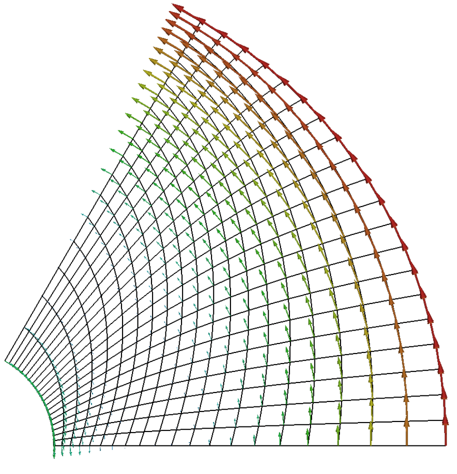


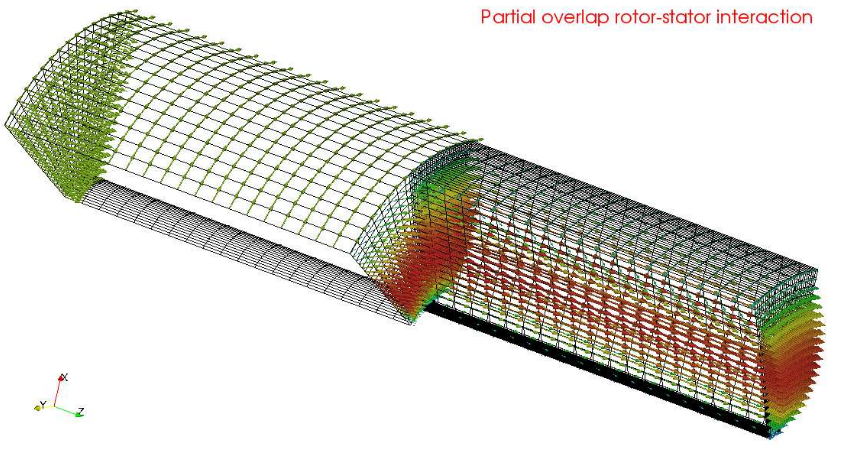
Auto-refined Diesel Engine



Generalised Grid Interface (GGI)

- Objective: mimic behaviour of sliding interface without changing the mesh
- Calculation of weighting factors used for implict coupling on matrix level
- Apart from “fully overlapped” cases, turbomachinery meshes contain similar features that should employ identical methodology, but are not quite the same
    - **Non-matching cyclics** for a single rotor passage
    - **Partial overlap** for different rotor-stator pitch
    - **Mixing plane**: perform averaging instead of coupling directly
- In such cases, the behaviour is closer to a **coupled boundary condition**, but the numerics is similar to sliding interface





GGI – Examples