A Moving Mesh Strategy to Perform Adaptive Large Eddy Simulation of IC Engines in OpenFOAM®

F. Piscaglia, A. Montorfano, A. Onorati
Dipartimento di Energia, Politecnico di Milano, via Lambruschini 4, 20156 Milano (Italy)

The paper focuses on a mesh moving method based on non-conformal topologically changing grids for Adaptive Large Eddy Simulation of IC engines in OpenFOAM®. With respect to authors’ previous work, a more efficient algorithm to handle the connectivity of non-conformal interfaces and a mesh-motion solver supporting multiple layer addition/removal of cells, to decouple the timestep constraints of the mesh motion and of the fluid dynamics, have been developed. Other new features include automatic decomposition of large multiple region domains to preserve processors load balance with topological changes and additional tools for automatic parallel pre- and post-processing. Finally, a transient solver for compressible flow of Newtonian fluids using a merged PISO-SIMPLE algorithm has been extended to work with the proposed mesh motion. The resulting framework allows to perform simulations with moving grids where initial mesh properties (skewness and non-orthogonality) are preserved during the engine cycle, favoring an improved accuracy of the fluid-dynamic solution and a fast convergence of the fluid dynamic solver. Also, when LES simulation is performed, the filter width results to be constant during the integration of the engine cycle. The paper describes the new developments by the authors and includes some validation tests on full-cycle simulation of a single cylinder engine with Transparent Combustion Chamber (TCC) under motored conditions.

INTRODUCTION

Among all physical phenomena that occur inside an engine cylinder, turbulence has certainly a direct impact on thermodynamic efficiency, brake power and emissions of the engine since its influence extends from volumetric efficiency to air/fuel mixing, combustion and heat transfer.

The computational tool used in this paper is the open-source CFD software OpenFOAM®, which has been extended by the authors with SGS models, boundary conditions, pre- and post-processing applications to perform LES simulation of ICE [1, 2, 3, 4, 5, 6]. In [1], the implemented code was used to simulate the complex unsteady flow features like the laminar-to-turbulent transition and the evolution of the tumble vortexes by time-resolved analysis and Proper Orthogonal Decomposition (POD) on an engine-like geometry consisting of a flat-top cylinder head with a fixed, axis-centered valve and a low-speed piston. The finite volume grid used in [1] was a purely hexahedral mesh with about 4.6 million elements; mesh motion was based on a point-stretching concept in order to avoid remapping of the fields onto different grids. A second-order backward differencing scheme was used for discretizing the temporal derivatives, whereas momentum convection was performed with the Linear-Upwind Stabilized Transport (LUST) scheme, a low-dissipation method specifically developed for LES [7]. For the remaining differential terms, pure second-order differencing schemes were used, with the exception of energy, for which an upwind-biased method was employed for stability.

The procedure presented in [1] proved to be accurate both in the prediction of average quantities and turbulence dynamics; on the other hand, when cell stretching is applied for mesh motion, the cell volume changes. In particular, during cylinder compression:

- decrease in the mesh size $\Delta x$ results in a reduction of the discretization error, yielding an improved numerical accuracy;

- if the cutoff length $\bar{\Delta}x$ is tied to $\Delta x$, mesh refinement will also induce a decrease in $\bar{\Delta}$ and make the subgrid model less influential on the results (since a larger part of the exact solution is directly captured).

A dynamic reduction of the filter the filter size requires to find an error estimate and a bound for the algorithm, in order to guarantee a sufficient resolution of the grid to resolve the main turbulent scales of the engine [8]. For this reasons, runtime checking on grid resolution was performed by introducing the LSR parameter [4, 9]:

\[
\text{LSR} = \frac{\bar{\Delta}}{\ell_{DI}}
\]  

which for engine simulation should be lower than five [4].

In Eq. (1), $\ell_{DI}$ is the lower limit of the inertial sub-range, which is usually estimated [10] as $\ell_{DI} \approx 60 \eta$ [10]; $\eta$ is the Kolmogorov scale. Although it is difficult to determine whether resolution of the turbulent scales improves with the reduction of the filter size during compression, since the geometry of the engine is changing in time and projection error, discretization error and modeling error are present at the same time, it is quite natural to think that the solution at a time step is correlated to the solution of the earlier time step. Hence in ICE simulation, if a mesh motion strategy based on cell stretching [1] is applied, it is important that the mesh at the bottom dead center (BDC) before compression has a sufficient resolution to capture...
the main turbulent scales. During the expansion phase, when cells are stretched, the issue is even bigger since the resolution decreases and this need to be controlled. As a result, to have sufficient resolution at the BDC with this moving mesh strategy, the resulting mesh at the top dead center (TDC) will result to be strongly limited in the time step by the CFL constraint. If remapping over multiple meshes between different time steps is adopted, the change of filter size and shape is even more difficult to handle and it will be impossible to determine any criterion for error estimation, that will result uncontrolled.

The aim of this paper is to define a fully automatic mesh motion strategy to perform Adaptive LES Simulation of IC engines, where the filter cell size is unchanged during the all engine cycle. This may be achieved by handling moving boundaries by a wide use of so called topology modifiers, namely:

- sliding interface, to connect dynamically different mesh regions through non conformal interfaces;
- dynamic addition/removal of cell layers, to keep optimum size of the cells during piston and valve motion;
- attach/detach of boundaries, to automatically simulate the valve closure event.

All the algorithms and models described have been developed by the authors to work with the mesh definition of the official release of OpenFOAM® [11] and they have been included in LibICE, a C++ object oriented library in the OpenFOAM® technology developed at Politecnico di Milano.

CASE DEFINITION AND SETUP

The test case chosen to test and validate the developed code is an optical engine with Transparent Combustion Chamber (TCC) from Engine Combustion Network [12], which was was developed for the specific purpose of supporting the development and validation of LES approaches. The engine features a two-valve head with simple intake and exhaust port/runner geometries and a pancake-shape combustion chamber. Experimental data were acquired with optical multi-dimensional high-speed diagnostics techniques. In addition, the engine has been fully instrumented with pressure and temperature sensors for high-fidelity measurements of boundary conditions.

The aim of the development presented in this work was to use topological changes to perform LES simulations on moving grids where initial mesh properties (skewness and non-orthogonality) were preserved during the all engine cycle. The implications of this strategy on the simulation are several: first, since the quality of the mesh is preserved during the all engine cycle, controlling the quality of the initial mesh ensure an accurate fluid-dynamic solution and a fast convergence of the fluid dynamic solver. Also, when LES simulation is carried out, the proposed mesh motion strategy allows for maintain the LES filter size constant.

Mesh generation

For the simulation of the TCC engine, a single hybrid mesh (tetrahedral + hexahedral), where topological changes are automatically applied during the engine cycle, has been generated. The mesh motion strategy followed needs as in input one single mesh for the all simulation (Fig. 1-a), that is usually defined at 0° after top dead center of the expansion piston (ATDCE), in order to ensure a very high cell quality during the all simulation. In Fig. 1-b, the strategy to generate the mesh is shown.

The initial computational grid is divided into three regions (the cylinder region, the intake duct and the exhaust duct) by two attach-detach boundaries, which are used to disconnect the valve port from the cylinder mesh at the valve closure time. The definition of attach-detach boundaries consists of an oriented surface of mesh faces and a time- or event controlled trigger which can be made solution-dependent.

<table>
<thead>
<tr>
<th>Engine geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bore</td>
</tr>
<tr>
<td>Stroke</td>
</tr>
<tr>
<td>Connecting rod length</td>
</tr>
<tr>
<td>Crank radius</td>
</tr>
<tr>
<td>Engine Speed</td>
</tr>
<tr>
<td>Compression ratio</td>
</tr>
<tr>
<td>Displacement</td>
</tr>
<tr>
<td>Clearance volume</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valve timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of valves</td>
</tr>
<tr>
<td>IVO/IVC</td>
</tr>
<tr>
<td>EVO/EVC</td>
</tr>
<tr>
<td>Intake peak lift</td>
</tr>
<tr>
<td>Exhaust peak lift</td>
</tr>
</tbody>
</table>

Table 1: Geometrical features and valve timing for the TCC engine [12].

![Figure 1: a) Initial mesh 0° ATDCE for LES of the TCC engine; the computational grid is made of about 4 M of elements. b) detail of the non non conformal (sliding) interfaces [6, 13] used to simulate the motion of the valves.](image)
Boundary conditions are set accordingly to the following scheme:

- **piston**: patch defining the boundary that will be moved according to a prescribed motion;
- **liner**: boundary representing the cylinder surface, whose point definition will be changed according to a prescribed motion law;
- **cylinderHead**: set of points of the engine geometry that will not be moved. In this set there are points that are shared with the liner patch;

Additionaly, for each moving object of the geometry (either it is a *piston* or a *valve*), user must define:

- the list of cells (**cellSet**) that will be rigidly moved along a prescribed direction by a prescribed velocity
- one or multiple lists of faces (**faceSets**), where dynamic layer addition/removal will be applied. For any given faceSet, the algorithm calculates run time the average cell height of the cell layer (hexahedral or prismatic) to check whether layer addition and removal must be triggered or not. Cells in a deleted layer are merged into neighboring cells, with the value in the resulting cell being the volume-average of deleted and neighboring cells; cells of added layers have a constant thickness that is defined by a dictionary;
- one or multiple lists of faces (at least one per valve bank) where attach/detach of the boundaries is applied.

In Fig. 2, the blue area represents the set of cells of a valve that are moving according to a prescribed motion law (lift profile); cells belonging to this cellSet can be of any kind (hex, tet, prism, hybrid) and they are surrounded in the upper part by a layer of hexahedral cells. Fig. 2 shows in detail how the topology modifiers are used for valve motion and how cellSets/faceSet must be set.

Sliding interface topology modifier [6, 13] is applied on the sides of the cellSet, because the static mesh of the engine head and the sliding valve are connected by non-conformal hybrid interfaces.

Additionaly, for each moving object of the geometry (either it is a *piston* or a *valve*), user must define:

- the list of cells (**cellSet**) that will be rigidly moved along a prescribed direction by a prescribed velocity
- one or multiple lists of faces (**faceSets**), where dynamic layer addition/removal will be applied. For any given faceSet, the algorithm calculates run time the average cell height of the cell layer (hexahedral or prismatic) to check whether layer addition and removal must be triggered or not. Cells in a deleted layer are merged into neighboring cells, with the value in the resulting cell being the volume-average of deleted and neighboring cells; cells of added layers have a constant thickness that is defined by a dictionary;
- one or multiple lists of faces (at least one per valve bank) where attach/detach of the boundaries is applied.

In Fig. 2, the blue area represents the set of cells of a valve that are moving according to a prescribed motion law (lift profile); cells belonging to this cellSet can be of any kind (hex, tet, prism, hybrid) and they are surrounded in the upper part by a layer of hexahedral cells. Fig. 2 shows in detail how the topology modifiers are used for valve motion and how cellSets/faceSet must be set.

Sliding interface topology modifier [6, 13] is applied on the sides of the cellSet, because the static mesh of the engine head and the sliding valve are connected by non-conformal hybrid interfaces.

**CODE DEVELOPMENT**

At the time the paper is written, despite the official standard release of OpenFOAM® [14] results to be potentially designed to handle dynamic meshes with topological changes, it is still missing in some features that were needed for this work. The C++ classes in the software for dynamic mesh handling, namely `slidingInterface`, `layerAdditionRemoval` and `attachDetach` and some mesh motion solver were significantly modified, to make them suitable for use with engine (and non-engine) complex meshes.

**Layer addition/removal**

Layer addition/removal – or dynamic cell layering – consists in adding or removing layers of cells in accordance with the motion of a translating boundary (e.g. the engine piston, see Fig. 4), whereas the majority of the grid remains fixed. This strategy allows for keeping the mesh quality (size and shape of cells) constant during the whole simulation. Actual layer thickness is continuously monitored during the simulation: cells are added when the thickness of the stretched layer rises above a specified threshold (this happens, e.g., during the expansion stroke) and they are removed when the deforming layer thickness falls below a different threshold (e.g. during the compression stroke). A C++ class for addition/removal of cell layers is available in OpenFOAM® since the earliest releases; however, it needed some improvements to behave properly in association with other mesh modifiers and in
case of parallel computations. The major developments carried out on the layer A/R class are:

- implementation of inter-processor communication. Relevant mesh data are exchanged between different processors in case of parallel computing. These quantities include mean layer thickness and face set used to add/remove layers.

- update of face zones affected by layer A/R. If a face zone extends perpendicularly to the master layering face zone (even through it), newly inserted stick-out faces automatically are included into the face zone, thus preserving its topology (see Fig. 5-a).

- checking for boundary proximity: during removal phase, if only one layer of cells exists between the master face zone and a mesh boundary, removal is automatically deactivated, to avoid deletion of a boundary face and a subsequent topological inconsistency (Fig. 5-b).

Addition/removal of cell layers is applied also for piston motion. To move the piston, the user has to define in the engineGeometry file the name of the patch, the cellSet to move, the faceSet where dynamic layer addition removal must be applied and the parameters to control the maximum and the minimum thickness of the cell layers to add or remove.

Attach/detach of boundaries

Attach/detach mesh modifier is applied to simulate the valve closure event and it consists in a reversible interface between two conformal mesh regions. It is used to temporarily join or split different parts of the mesh starting from a prescribed and arbitrary set of internal faces (detachFaces), that will be used by the dynamic mesh solver to be transformed into boundary walls. In ICE simulation, the attachDetach C++ class separates the intake/exhaust ports from the combustion chamber at the valve closure (Fig. 7).

The original version of the class, as released in the standard release of OpenFOAM®, calculates the face matching between the two sides by implying that the point ordering is the same; since this seldom happens in complex meshes (like in IC engines), the original class has been extended to calculate the face matching on the basis of point projection. The followed strategy results to be very similar to the one employed for the sliding interface algorithm [6, 13]. Despite the new version of algorithm results to be a slightly slower, it has proven to be more robust and it succeeded in all test cases performed so far.
Sliding Interface

A `slidingInterface` topology modifier allows for the dynamic stitching and splitting of mesh regions with different mesh structures (see Fig. 8), so that a reversible, non-conformal interface can be created; the sliding interface procedure generates a seamless joint, thus requiring no particular numerical technique to solve the equations across the interface. In this aspect, `slidingInterface` differs from AMI (Arbitrary Mesh Interface technique [15, 16, 17], already available in the standard distribution of the code), because in AMI fluid-dynamic coupling is achieved by interpolation of cell fluxes among two topologically separated mesh regions facing each other. The merging procedure of `slidingInterface` can be reversed without any loss of information to detach the interface, provided that all information about the coupling process have been appropriately stored. This part of the algorithm was implemented by the authors and it has been already presented in previous works [6, 13], according to the theory already described in [18].

![Figure 8: Point projection of slave patch (red) onto master. Solid dots are master points (which are retained), hollow circles are slave points added due to direct hits or intersections.](image)

In this work, a further extension of the original theory of the `slidingInterface` algorithm is proposed, in order to allow it for work with non-conformal interfaces presenting sharp corners (e.g. 90°) and to improve its robustness. In particular, the algorithm for the detection of the `stickout` faces over the two arbitrary-shape non conformal interfaces has been completely redesigned, to overcome some inherent shortcomings of the original implementation and to assign faces to the appropriate mesh region. A `stickout` face is a face that does not lie on the non-conformal interface to be formed, but that is however affected by the coupling process due to insertion of points on its edges (see Fig. 8). In the original formulation, detection of `stickout` faces is performed on a cell-face basis: starting from a face belonging to master (or slave) interface, the algorithm detects the owner cell and classifies the `stick-out` faces as those sharing an edge with that face.

This procedure does have tight constraints about the shape of the interfaces to couple:

- the surface must be as smooth as possible (ideally speaking, planar);
- on both sides, all cells adjacent to the surface must be hexahedra or pyramids;

In the original implementation, if the above requirements are not fulfilled (e.g. when there is an unstructured meshes on one side), the coupling algorithm will fail, since there can be `stick-out` faces other than those belonging to the master owner cell (see Fig. 9). To overcome this problem, a new procedure based on a point-face seeking has been developed for detection of `stick-out` faces. The algorithm consists of the following steps:

- a master (slave) face is selected;
- for all the points of the master (slave) face, all faces sharing a point with the selected master (or slave) face and are classified as `stick-out` faces.

![Figure 9: Stick-out faces of an unstructured mesh. They include also faces that does not belong to owner cells.](image)

The novel algorithm for detection of `stick-out` faces now has only one tight requirement: no points must be shared by the master and the slave side. For this reason, a pre-processing application (`splitSharedPoints`) to split possibly shared points has been developed in the OpenFOAM® technology, to fix meshes generated by external mesh generators that do no fulfill the above constraint. The new algorithm for detecting `stick-out` faces has proven to be indispensable when complex interfaces need to be stitched. An example of use of the algorithm is shown in Fig. 10: the cylinder and the spark plug region of the TCC engine are merged through non-conformal interfaces: in this way, a perfectly structured mesh (cylinder region) can be merged with a highly refined tetrahedral grid (spark plug), in order to have a very high quality mesh. The algorithm is written in a general form and the functionality to keep the old mesh topology is perfectly working and tested.

Variable topology-driven time-stepping

Topological changes, being triggered in dependency of the current mesh status, pose some limits on the maximum time step size needed to guarantee topological consistency. Using too a large temporal integration step may cause the mesh handling algorithm to skip the point when a topological changes would be triggered, hence leading to a wrong mesh configuration. A typical example is when the piston upward displacement in a single time step is larger than the threshold value for dynamic layer removal. To avoid
problematic situations and to ensure dynamic mesh consistency, adaptive topology-driven time stepping has been implemented. The expected displacements for all the moving components (piston, valves) $\Delta z'$ are computed before they are actually executed. As an example, for a piston moving by a velocity $u_p$ during the compression phase:

$$\Delta z'_{\text{piston}} = u_p \cdot \Delta t$$

If the predicted displacement is larger than the average height of the cell layer to remove:

$$\Delta z'_{\text{piston}} \geq \Delta z_{\text{max layer}}$$

then the time step is recalculated as:

$$\Delta t_{\text{lim}} = \Delta z_{\text{max layer}} / u_p$$

In Eq. (3), $\Delta z_{\text{max layer}}$ is the layer removal threshold and $\Delta t_{\text{lim}}$ the maximum allowed time step. The multi-stepping on topology modifiers implies a proper handling of the contributions of the single step to the grid velocity vector on the conservation equations for scalar quantities, as it will be explained further.

PARALLEL COMPUTING

Parallelism is implemented in OpenFOAM by the so-called domain decomposition technique. The whole mesh is divided into several sub-domains, each assigned to a separate process. Communication between sub-domains is carried out by specific boundary conditions based on the MPI protocol: this ensures physical consistency disregarding the specific equations and models implemented in the solver. Insertion of topology modifiers in a decomposed mesh is straightforward, provided that sufficient care is taken about the decomposition strategy, to comply with some restrictions to make the topology modifiers work in parallel [6, 13]. Since topology modifiers change the addressing and definition of faces in the local mesh, so topology must be synchronized over inter-processor boundaries. Alternatively, another way to proceed is to ensure during decomposition that pair of patches defining the topology modifiers are contained in one single sub-domain and that no inter-processor boundaries can lie on the interface itself. An extensive work has been done to allow automatic decomposition of the domains, including:

- extensions of the capabilities of the meshTools class, to extract cellSets and faceSets from closed volumes and surfaces provided in the stereolithography (STL) format, in order to easily define in a complex mesh the cellSets and faceSets needed by the topological modifiers;
- the algorithm for automatic domain decomposition in the application decomposePar has been extended to handle multiple topology modifiers in the mesh, by considering the inter-dependencies between them. Since the implemented technique is based on cell addressing rather than on the mesh topology, the load balance between processors results to be improved.

NOVEL CLASS STRUCTURE FOR FLEXIBLE DYNAMIC MESH HANDLING

Dynamic handling of finite volume (FV) grids needs, together with the definitions for space discretization (points, faces, cells, boundaries), some additional information to handle its dynamic behavior:

- the motion law(s) of the deforming boundaries;
- a motion solver for points displacement, as a consequence of deforming boundaries;
- the definitions of the topological changes.

The number and the kind of such inputs vary accordingly to the engine to be simulated (four stroke, two-stroke) to the kind of valve motion accoring to the geometry of the cylinder head (flat-top, pent-roof..) and to the point motion strategy (layer A/R, cell stretching). In an object oriented code, this reflects into a different number and type of classes to be loaded to achieve the desired functionality. To improve code usability and extension and to allow for easy case-setup, all the features needed for dynamic mesh handling have been grouped on the basis of the physical
component (piston, valve, . . . ) they refer to. As a consequence, the structure of the classes implemented in the library will follow the “structure” of a real engine. In the present work, the following components have been defined so far:

- **class engineTime**: class to handle engine geometry and global features (revolution speed, stroke, connecting rod, crank radius);
- **class engineMesh**: class to handle space discretization;
- **class enginePiston**: class to handle piston motion with/without dynamic addition/removal of cell layers;
- **class engineValve**: class to handle valve geometry (lift profile, axis) and valve motion strategy (use of dynamic layer addition/removal on top and bottom sides, attach/detach boundary, sliding interfaces).

In the novel implementation, all topology modifier are automatically linked to the class to handle the dynamic mesh by the newly developed class (**topoManager**). When the **engineMesh** is used, **topoManager** handles all communications between engine components (pistons, valves) and the global mesh. The final code structure is reported in Fig. 12.

**Figure 12**: Collaboration diagram of **engineTopoMesh** class.

All the management of the topological changes in the dynamic mesh (definition, parallelization, synchronization among parallel domain and interpolation methods for transported scalars) has been included in a low-level class (**topoManager**) in the code structure, which is linked to the base class for dynamic mesh handling (e.g. **dynamicMesh**), as shown in Fig. 12. Code implementation involving topological changes results therefore completely transparent to the developers of mesh motion solvers, that with the novel structure can concentrate only on the definition of the mesh motion law, that is included in an overloaded function at the upper level. In engine simulation, any further extension to the engine class (**engineTopoMesh**) with respect to the already implemented typical applications (two-stroke or four stroke engines) can be done very easily, by adding new physical components or implementing new strategies for point motion, since any combination of the existing functionality results now extremely simple. Also, the interface between the dynamic mesh class and the Navier-Stokes solver is minimal, being limited to a single function call that performs all mesh changes when required. On the other hand, thanks to the run-time selection capabilities of OpenFOAM®, the user intervention when setting up a new case is restricted to the compilation of a input dictionary which contains all engine specifications. No modifications of the code are required in case a new engine family has to be simulated.

**CFD SOLVER FOR DYNAMIC MESH**

The compressible dynamic solver used for the simulation is **coldTopoEngineFoam**, which is an extension of the already existing transient solver for compressible flows (using a merged PISO-SIMPLE algorithm), with some modifications to work with topological changing meshes. In particular, the solver has been extended by the authors with:

- a correction term to account for the contribution to the mass fluxes through the control volume (CV) face, due to the relative velocity between the grid and the flow that is coupled to the **dynamicMesh** class;
- an equation to correct the face fluxes, to use when significant topological modifications of the grid occur.

Conservation of mass and momentum for a compressible flow over a moving grid are written as [19]:

\[
\frac{\partial}{\partial t} \int_{x_1(t)}^{x_2(t)} \rho dV + \int_{x_1(t)}^{x_2(t)} \frac{\partial}{\partial x} \left( \rho \bar{v} - \bar{v_b} \right) dx = 0 \quad (5)
\]

\[
\frac{\partial}{\partial t} \int_{V} \rho dV + \int_{V} \frac{\partial}{\partial x} \left( \rho \bar{v} - \bar{v_b} \right) n dS = 0 \quad (6)
\]

where \( \bar{v_b} \) are the velocities the control volume boundaries move with.

As shown by Eq. (5), a cell mass source appears in the mass conservation equation as cell faces move:

\[
\Delta \bar{m} = \frac{\rho \Delta V}{\Delta t} \quad (7)
\]

To ensure mass conservation, the *space conservation law* must be enforced, i.e. a continuity equation in case of a zero fluid velocity [19]:

\[
\frac{d}{dt} \int_{V} dV - \int_{S} \bar{v} \cdot \bar{n} dS = 0 \quad (8)
\]

Discretization of (8) depends on the temporal integration scheme and it allows for calculating the mesh motion flux \( (\phi_M) \) on the basis of the swept volume \( \tilde{V}_f \); in the simplest case of Euler implicit integration, the mesh motion flux can be calculated as:

\[
\phi_M = (\bar{v}_b \cdot \bar{n}) f S_f = \tilde{V}_f \quad (9)
\]

where \( \tilde{V}_f = \Delta V / \Delta t \) is the volume swept by a cell face in a single time step. In case of a higher order scheme, a different discrete equation for \( \phi_M \) must be used. In OpenFOAM®, the calculation of \( \phi_M \) according to the time discretization scheme is done by the virtual function \texttt{fvc::meshPhi()} by the run-time selection of temporal discretization scheme.

For a cell face with a generic shape, the swept volume is calculated as follows: first, the face is decomposed into several triangles, one for each edge, that have as common
vertex the face centroid; then, the swept volume is calculated for each triangle, as the difference between its new point coordinates \( T \) and the old ones \( T^o \):

\[
V_f = f(T - T^o) \tag{10}
\]

Since a face is stored as a list of point IDs, and not as a list of point coordinates, Eq. (10) does hold as long as every point maintains its own ID during the mesh change (i.e., in the case of point motion without topological changes). When topological changes occur, points are renumbered and there is no correspondence between old and new point IDs, so the correlation between \( T \) and \( T^o \) is no longer valid. In this case, a workaround consists in rewriting the topological map generated during the topological changes. However, if a face is added during a topology modification, the mesh flux must be always zero. Since Eq. (10) is obviously not applicable (because there are no “old point positions”) mesh flux on these faces is handled by explicitly setting its value.

To even more enforce mass conservation at the occurrence of a topological change, the absolute flux (sum of advective fluxes and mesh motion fluxes) \( \Phi = \phi + \phi_M \) is corrected by solving a Poisson equation for a correction pressure \( p_{corr} \), which is used to adjust \( \Phi \):

\[
\nabla^2 p_{corr} = \nabla \cdot \Phi \tag{11}
\]

\[
\Phi^N = \Phi^{N-1} - \sum \frac{1}{A_p} \nabla p_{corr} \tag{12}
\]

Eqs. (11) and (12) are solved iteratively any time the mesh changes, until convergence on pressure is reached; this strategy is particularly useful when dynamic layer addition/removal and attach/detach of boundaries are performed, since the calculation of the face fluxes is critical and it requires strong correction.

VALIDATION

Dynamic layer addition/removal. A benchmark test to verify if mass and energy are properly conserved by the dynamic solver with addition/removal of cell layers is the simulation of an adiabatic compression and expansion of air in a closed cylinder volume. A piston moving at 2000 rpm in a cylinder (with volumetric compression ratio \( r = 10 \)) was simulated: initial conditions of \( p = 101325 \) Pa, \( T = 292 \) K.

![Figure 13: Adiabatic compression/expansion of air by a piston in a constant vessel: comparison between simulations and theory.](image)

The comparison between the expected theoretical values coming for the adiabatic compression/expansion \( pV^k = \) const and the results from the simulation is reported in Fig. 13 (left). For the simulation the compressible solver coldEngineFoam was used; dynamic layer A/R was performed on the third layer of cells above the piston (Fig. 13, right). The comparison between simulations and theory shows a very good agreement: the discrepancy on pressure prediction is lower than 0.3\%, while the discrepancy on the calculation of temperature is lower than 0.5\%. Mass within the cylinder is conserved with an error that is lower than \( 10^{-8} \).

Engine simulation. At the time the current paper is written, validation of the physical models has been carried out on a flat-top cylinder head with a fixed, axis-centered valve and a low-speed piston [1], while the application of the mesh motion strategy presented in this paper has been performed for the simulation of the Transparent Combustion Chamber (TCC) engine [12].

![Figure 14: Detail of the velocity and temperature fields across the topology modifiers in the TCC engine [12].](image)

Plots of Fig. 14-a and 14-b show some preliminary results from a RANS simulation. The aim of this simulation was to verify the robustness of the implemented mesh motion algorithm and its performance when used together with the fluid dynamic solver, when applied to the simulation of the engine cycle. Checks were performed to monitor that pressure, temperature and velocity were correctly conserved both through the non-conformal moving interfaces and on the faces where dynamic layer addition/removal was performed: usually, errors in the calculation of the mesh motion fluxes appear as small discontinuities in the solution of the conserved variables (velocity and temperature in particular) where topological changes occur. As evidenced by Fig. 14-a and 14-b, fields across the interfaces does not show any discontinuity, which means that variables are fully conserved during mesh motion and that the proposed method is fully conservative and, therefore, suitable for LES of IC engines.

CONCLUSIONS

A summary of the recent implementation carried out in the OpenFOAM® technology to perform simulations of
compressible flows with dynamic mesh has been presented. The implementation is compatible with the mesh definition employed in the versions of OpenFOAM® released by OpenCFD®. A novel structure of the dynamic mesh class has been implemented in order to facilitate the implementation of dynamic solvers with topological changing meshes and their integration with the flow solver. Also, an extensive work has been done to allow the code to work with strong parallelization, by minimizing the unbalancing over the processors. Together with the SGS models and boundary conditions for LES and a series of applications for data post-processing, the implemented framework is currently being used to perform adaptive LES simulation of the TCC engine; thanks to the adopted mesh strategy, it is possible to keep the filter size almost constant during the all engine cycle. Results shown on fluid-dynamics of the TCC engine must be intended as preliminary, since at the current time simulations are still running.

REFERENCES


