pyMeshFOAM: Automated meshing for CFD and fluid–structure simulations

Fabien Salmon, Ludovic Chatellier

SoftwareX 23 (2023) 101431

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

pyMeshFOAM: Automated meshing for CFD and fluid–structure simulations

Fabien Salmon *, Ludovic Chatellier

Institut Pprime, CNRS - Université de Poitiers - ISAE-ENSMA, UPR 3346, Poitiers, France

ARTICLE INFO

Article history:
Received 23 November 2021
Received in revised form 3 April 2023
Accepted 1 June 2023

Keywords:
Computational fluid dynamics
Fluid–structure interaction
Meshing tool
Boundary layers
OpenFOAM

ABSTRACT

The generation of high-quality meshes is of paramount importance for accurate numerical fluid simulations. As this can be a tedious task, we have developed pyMeshFOAM, an easy-to-use code based on existing free meshing tools that automatically generates different types of structured or unstructured meshes in OpenFOAM format. From a 2D contour, pyMeshFOAM can generate a 2D mesh of the fluid around it or a 3D mesh around the shape extruded from the contour. CAD files can also be used for the geometry. Boundary layer meshing is also available. In addition, pyMeshFOAM manages the reciprocal solid meshes compatible with CalculiX in order to perform coupled fluid–structure simulations.

Code metadata

Current code version	*1.1*
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00220
Code Ocean compute capsule	
Legal Code License	*GNU GPL v3 or later*
Code versioning system used	*git*
Software code languages, tools, and services used	*Python, shell*
Compilation requirements, operating environments & dependencies	
If available Link to developer documentation/manual	https://github.com/FabienSalmon/pyMeshFOAM/tree/main/docs
Support email for questions	Salmon.Fabien@yahoo.com

1. Motivation and significance

Both applied and fundamental studies of fluid flows often rely on computational fluid dynamics (CFD). This approach consists of numerically solving the equations governing fluid mechanics. In parallel to commercial and industrial CFD suites, several open-source software packages have been developed. OpenFOAM [1] is one of the most widely used free open-source codes in various fields of fluid mechanics [2–4]. The OpenFOAM community is heavily involved in its improvement through numerical developments [5,6]. OpenFOAM is also embedded in a free open-source fluid–structure interaction solver, which also consists of CalculiX (solid solver) [7] and preCICE (interface coupling) [8].

OpenFOAM provides all the necessary tools to perform complete fluid simulations, from mesh generation (subdivision of the studied geometry into cells) to visualisation of the results. A weakness of this code is the meshing process, especially when the meshed geometry is rather complex. The native meshing tools can be tedious and even fail to produce the quality of mesh required for a successful simulation. Note that fluid–structure simulations are even more sensitive to mesh quality due to the inherent difficulties in ensuring numerical stability, which can be exacerbated by the difference in cell size between the two meshes.

To overcome this difficulty, the researchers developed cfMesh [9], another meshing tool compatible with OpenFOAM. It allows users to generate meshes of better quality than the OpenFOAM meshing libraries. Unfortunately, the free version of cfMesh does not include boundary layer meshing. And boundary layer meshing is often required in such fluid simulations where high accuracy is needed close to physical boundaries. The anisotropic nature of the meshes near the boundaries is replaced by a regular mesh, which is better suited to simulating the high gradients that occur in this zone. Building the mesh of a geometry can therefore be a difficult

* Corresponding author.
E-mail address: Fabien.Salmon@u-bordeaux.fr (Fabien Salmon).

https://doi.org/10.1016/j.softx.2023.101431

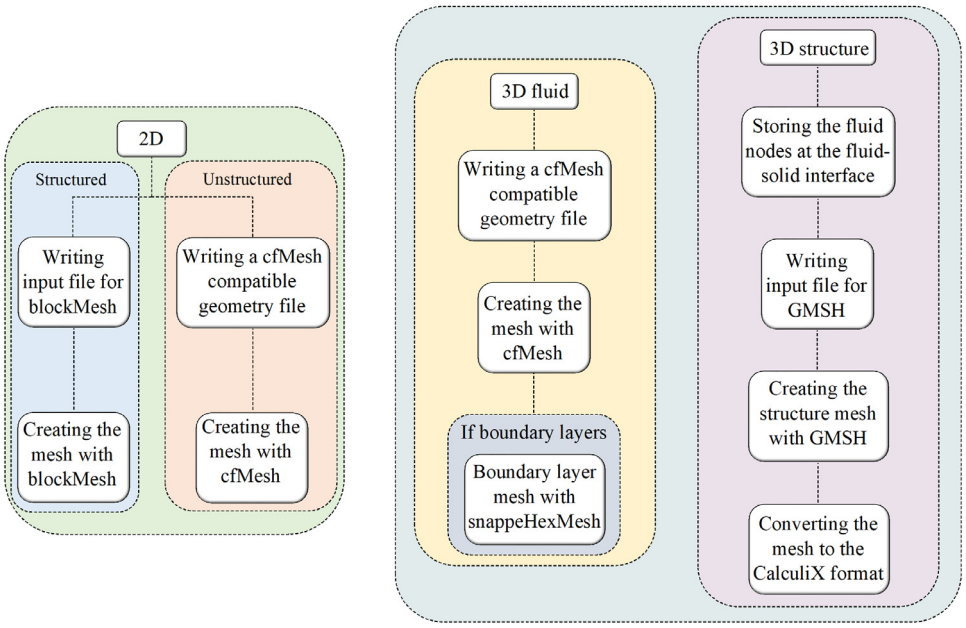

Fig. 1. Logical scheme of the pyMeshFOAM usage.

problem. Users often either use commercial meshing software or struggle to create suitable meshes.

In order to efficiently produce high quality OpenFOAM compatible meshes using free tools, we have developed pyMeshFOAM, a set of Python scripts that make the best use of free meshing tools: native OpenFOAM meshers (blockMesh and snappyHexMesh), cfMesh [9] and GMSH [10]. These tools will be involved in the meshing process as follows: blockMesh for simple enough geometries, cfMesh for complex geometries and snappyHexMesh to mesh boundary layers after the first step achieved by cfMesh. For fluid–structure simulations, an additional solid mesh is generated using GMSH for use in finite element software such as CalculiX. The software code manages any shape based on a 2D contour. This contour is given as an input file containing a list of points. It corresponds to a polygon constructed from these points and can represent any desired shape: circle, square, irregular polygons, etc. Some examples are given in Section 3. The code also includes symmetrical NACA wing profiles as a special case, for which the contour is given directly by an equation, without the need for a list of points. For 3D meshing pyMeshFOAM handles two cases. Either a CAD file is provided or the 2D contour is extruded in the third direction with a specified length.

It is worth noting that although pyMeshFOAM makes fluid meshes compatible with OpenFOAM, existing conversion tools can convert the resulting meshes into various other formats compatible with other CFD software. Furthermore, pyMeshFOAM consists of several Python scripts that are independent of each other and can be used separately.

This tool was used to generate finite element and finite volume meshes in and around an extruded NACA0015 profile for fluid–structure simulations in turbulent flows [11].

2. Software description

pyMeshFOAM consists of several Python scripts that automatically performs the meshing process for OpenFOAM simulations based on existing free meshing software: OpenFOAM native tools (blockMesh and snappyHexMesh), cfMesh [9] and GMSH [10]. pyMeshFOAM also manages the meshing process of solid meshes for fluid–structure simulations.

2.1. Software architecture

Fig. 1 shows an explanatory scheme of pyMeshFOAM. The user parameters are provided in an executable file (“mesh_fluid” or “mesh_solid”). Except when the user wants to obtain the mesh of a symmetric NACA profile or a 3D mesh based on a file, an input file corresponding to a 2D contour (list of points) must be provided. Then pyMeshFOAM generates a 2D mesh or a 3D mesh with a given extrusion length for the 2D contour, according to the user’s choice (cell size, boundary layers, mesh type, etc.). If a CAD file is provided, the extrusion step is skipped and pyMeshFOAM goes directly to the meshing process with cfMesh and possibly snappyHexMesh. The properties of the desired mesh are given in the executable file.

In 2D, a structured mesh may be required. In such a case, the developed tool creates the input file for blockMesh. Otherwise the same work is done for cfMesh. In both configurations, the user can request the mesh of a boundary layer created by either blockMesh (structured) or cfMesh (unstructured) during mesh generation.

In 3D, the global mesh is necessarily created with cfMesh. If required, snappyHexMesh takes care of the meshing of the boundary layers. If the user wants to run a fluid–structure simulation, pyMeshFOAM will also create the solid mesh. The input file for GMSH is written before the solid mesh is meshed with it. Since the tool is initially designed for fluid–structure simulations with OpenFOAM and CalculiX, a Python script converts the mesh into the format read by CalculiX.

The Python scripts and the executable files must be placed in the case directory as shown in the tutorial cases. The “mesh_fluid” executable must be run in the OpenFOAM directory to generate the fluid mesh. The “mesh_solid” executable must be run in the finite element software directory (e.g. CalculiX) to generate the solid mesh.

2.2. Software functionalities

2.2.1. 2D mesh

Structured Fluid mesh

The structured mesh is generated by the OpenFOAM utility blockMesh. The Python script divides the domain into several

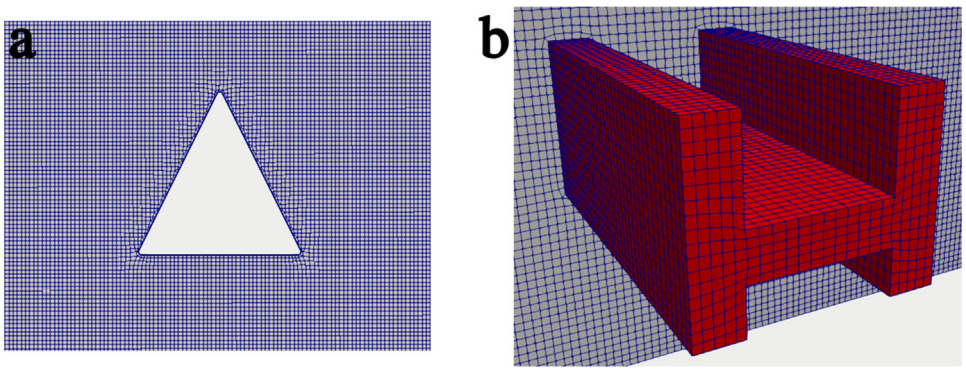

Fig. 2. (a) Fluid mesh around a triangle. (b) Mesh of a 3D non-symmetrical H-shape.

parts so that blockMesh can generate a regular and a structured mesh. Users can specify the general cell size of the mesh. The ratio between cells can also be modified to have coarser cells far away from the area of interest. No refinement box can be requested by the user due to the inherently Cartesian nature of the blockMesh utility. There are two ways to generate a structured mesh with pyMeshFOAM.

The first is to continue the refinement at the geometry boundary up to the edge of the meshed domain (Fig. 4a). In order to maintain a structured mesh, this approach is necessary in the case of sharp geometry such as the trailing edge of a wing profile. The Python script "Mesh_pointe.py" writes this input file to blockMesh which then generates the mesh.

The second is to create a refinement that surrounds the profile (Fig. 4b). This approach is preferred for geometries without sharp angles. The Python script 'Mesh_block.py' writes this input file to blockMesh. As in the previous approach, there is no additional parameter.

Unstructured Fluid mesh

Based on the 2D contour, the script "2D2ftr.py" creates a file in *.ftr* format, a cfMesh compatible format [9]. This file simply contains a list of the points that make up the 2D contour and the vertices of the numerical domain. cfMesh can then read this file to build the mesh. The properties of the mesh must be provided in the standard input dictionary for cfMesh (meshDict). Additional refinement boxes can thus be provided in this dictionary. Fig. 2a shows an unstructured mesh around a triangle.

2.2.2. 3D mesh

Fluid mesh

The choice of cfMesh is motivated by the high quality of this software code for 3D meshes. "dat2gmsh.py" converts the 2D contour into an extruded 3D geometry, creates the input file for GMSH and then runs GMSH to generate the stl file of the geometry. "CreateBox.py" creates the stl file of the computational domain. cfMesh functions then convert both stl files into an ftr file and create the 3D mesh.

Since the free version of cfMesh does not manage satisfactorily 3D boundary layers, snappyHexMesh is used for this step. The meshing process achieved by snappyHexMesh does not modify the first mesh created by cfMesh far enough from the boundary, but only replaces the mesh close to the boundary of the geometry with a refined mesh adapted to the resolution of the boundary layers. The input parameters of the boundary layer mesh are the exponential ratio R, the desired y^+, s the size of the cell close to the shape in the cfMesh mesh, the properties of the fluid (density ρ, viscosity μ), its velocity U_∞ before the shape and a characteristic length of the shape L. From these parameters, the size of the first cell $y_{1^{st}}$ and the number of layers N_{layers} are computed as follows:

$$\begin{cases} y_{1^{st}} = \dfrac{2\mu y^+}{\sqrt{\rho \tau_w}} & \text{(a)} \\ N_{layers} = \left\lfloor \dfrac{\ln s - \ln y_{1^{st}}}{\ln R} \right\rfloor & \text{(b)} \end{cases} \tag{1}$$

with $\tau_w = \frac{1}{2} C_f \rho U_\infty^2$, $C_f = 0.026 \times Re^{-1/7}$ and $Re = \frac{\rho U_\infty L}{\mu}$. $\lfloor \rfloor$ refers to the floor function. The number of layers, the exponential ratio and the first cell size are filled in the snappyHexMesh dictionary.

Solid mesh

The input file for GMSH is written by Recup_Aile.py to mesh the solid part. The type of elements used in the solid mesh can be selected in the Python script. The default is second order elements but first order elements can be chosen. For fluid–structure simulations it is desirable to collocate the fluid and solid mesh nodes at the fluid–solid interfaces to avoid interpolation errors. To ensure this, the Python script extracts the fluid nodes at the interface to generate the solid nodes. Without manually adding refinement options to the GMSH file, refinement is constrained by the fluid mesh, but a GMSH user can easily add refinements in the process [10].

As the tool is initially designed for fluid–structure simulations with OpenFOAM and CalculiX, a Python script ("RewriteINP.py") converts the mesh to the CalculiX format. Three files are written, "All.msh", "fix.nam" and "surface.nam". These files are input files for CalculiX [7].

2.2.3. Special options

Here we present some functionalities that are useful for specific cases. As already mentioned, pyMeshFOAM simplifies the treatment of symmetrical NACA profiles thanks to the use of analytical equations. The wing tip geometry can be simply flat or circular as shown in Fig. 4. The trailing edge of the wing can also be automatically smoothed to avoid any sharp edge in the geometry (Fig. 4b). This often results in a poor-quality mesh. The smoothing is included in the modification of the profile equation. Finally, for 2D meshes with a sharp edge, such as a standard trailing edge, the mesh will look like Fig. 4a. There is an option to break the propagation of the boundary layer mesh up to the outflow boundary. This results in a local unstructured area in the 2D mesh instead of the Cartesian propagation of the boundary layer mesh, but is still achieved by blockMesh. This option is not enabled by default as it requires more complex parameterisation and may result in meshes of lower quality.

3. Illustrative examples

Fig. 2 shows the regular meshes of two basic shapes, a 2D triangle (target of 40 cells per side) and a 3D non-symmetrical

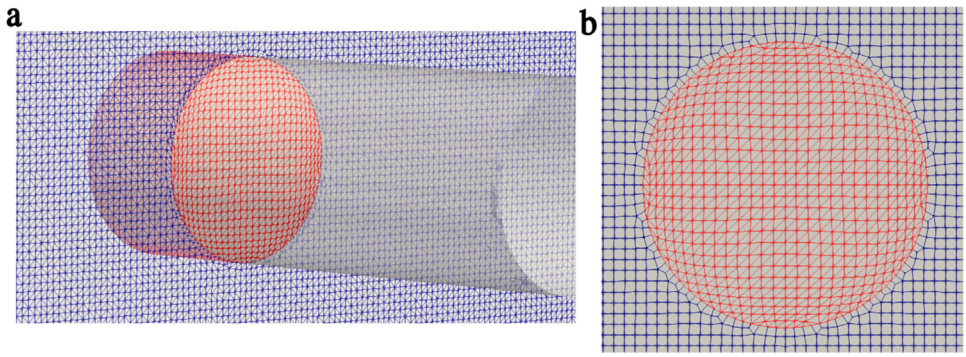

Fig. 3. (a) Fluid mesh (blue) around a cylinder and inner solid mesh (red). (b) Zoom on the solid cylinder mesh with collocated interface nodes. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

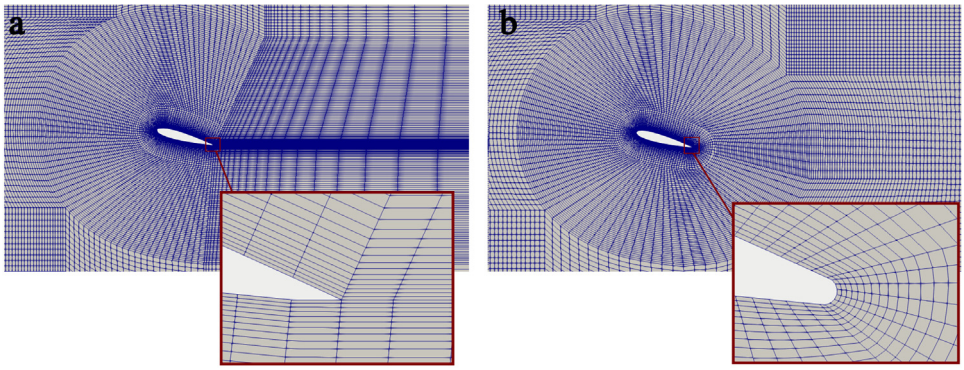

Fig. 4. 2D mesh of the fluid around a NACA0015 profile, (a) with a sharp trailing edge, (b) with a smoothed trailing edge.

H-shape (target of 18 cells for the height). To generate them, the input file contained the three points of the triangle and the twelve points of the 2D H-shape respectively. Fig. 3 shows the mesh of a cylinder (target of 25 cells along the diameter) based on the two hundred points that make up the initial circle. Fig. 3b focuses on the solid cylinder and shows that the solid boundary nodes are collocated with the fluid boundary nodes.

Fig. 4 shows two resulting meshes of a fluid around a NACA0015 airfoil. Fig. 4a shows the mesh produced using the first method described in Section 2.2.1. Fig. 4b shows the mesh produced using the second method. The trailing edge is automatically smoothed (can be disabled, see Section 2.2.3), which allows the use of good quality cells surrounding the wing. Both meshes have a high refinement in the boundary layer (30 layers with an exponential ratio of 1.15). There are 30 cells along the chord for the two meshes.

The 3D mesh of the same airfoil is shown in Fig. 5. The mesh is composed of areas with different refinements (user parameters in the cfMesh dictionary). The refinement levels produce cells twice the size of the next finer level. In the finest zone around the wing, there are 100 cells along the chord. The boundary layer has also been thoroughly meshed, as is often done in fluid simulations. The exponential ratio is equal to 0.15 and there 17 layers in the boundary layer. A circular wing tip (option presented in Section 2.2.3) has been chosen for this illustration.

We finally show an example of a very irregular geometry in Fig. 6. An arterial system is meshed with 160,000 cells based on the CAD file of this geometry. This mesh is only achieved by cfMesh as no boundary layer has been demanded.

4. Impact

In numerical simulations, the mesh of the geometry is of paramount importance as all results depend on its quality. However, creating a mesh, especially a refined one for CFD simulations, can be very tedious and time-consuming. The functionality of free tools is limited and sometimes defective. To overcome these problems, pyMeshFOAM allows the user to create parametric meshes in an easy and fast way. In particular, the automatic meshing of boundary layers, trailing edges and wing tips are key features of the tool. For complex geometries, the mesh of boundary layers can remain difficult. This limitation is directly correlated with that of snappyHexMesh in 3D. This software code will help CFD users who want or need to use free meshing software for their work. The automation of mesh generation also allows iterative testing of meshes based on different parameters, a cumbersome problem faced by CFD users looking for a suitable mesh for their problems.

The tool is designed to be intuitive and flexible. The Python scripts can be easily modified by any Python user and can also be used independently. For example, the circular wing tip shown in Fig. 5 is an option that can be replaced or enriched with other end shapes depending on the case. Local refinements can also be applied using the software dictionary files. For now, pyMeshFOAM is strictly limited to CAD files and extruded 2D shapes, but could be extended to include different extrusion rules (e.g. swept, curved, twisted or tapered wings).

Although pyMeshFOAM is primarily intended for OpenFOAM users, it can be used for other CFD software by using available mesh conversion tools. The tool developed is also dedicated to fluid–structure simulations. It allows the generation of two

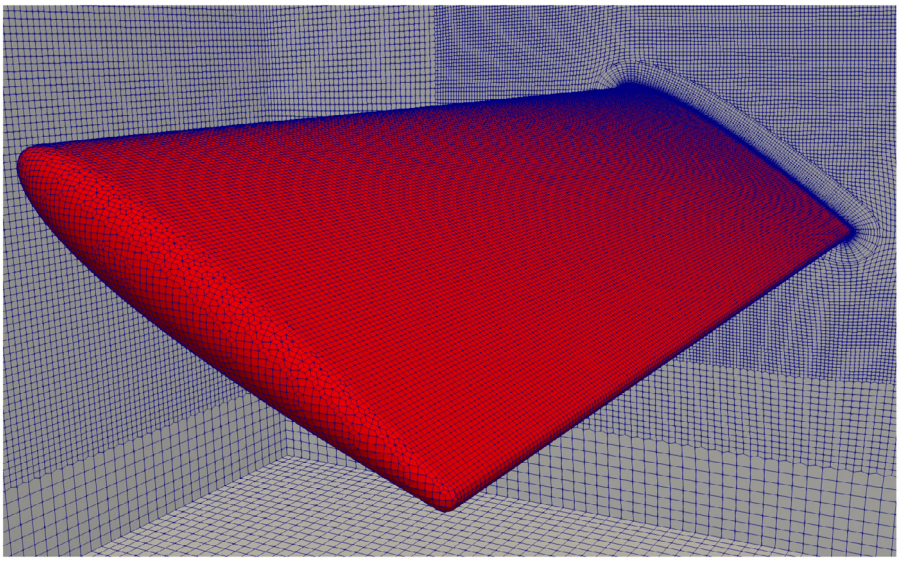

Fig. 5. Mesh of a 3D NACA0015 airfoil profile with a circular wing tip.

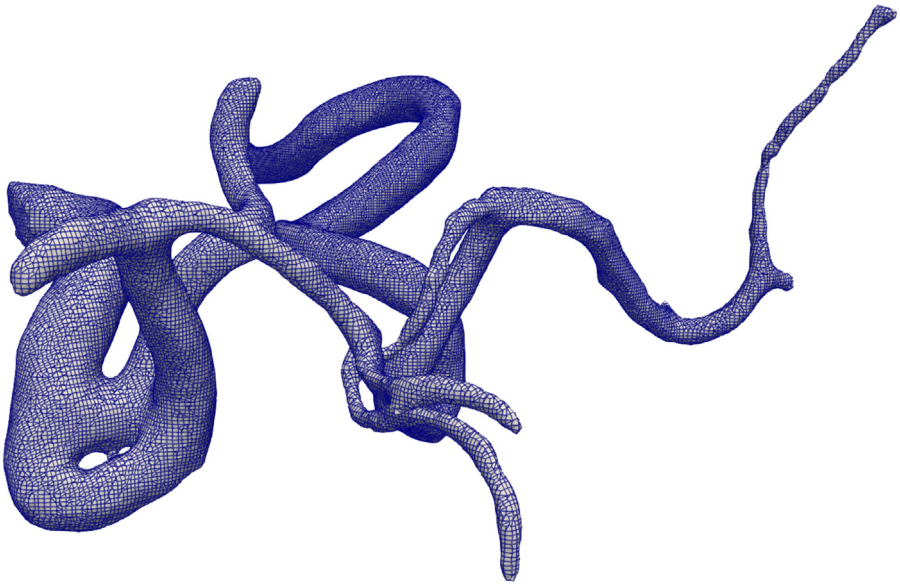

Fig. 6. Mesh of an arterial system.

meshes, one for the fluid and one for the solid. To avoid interpolations in fluid–structure simulations, pyMeshFOAM automatically generates two meshes with the same boundary nodes. Primarily intended for OpenFOAM-CalculiX simulations, pyMeshFOAM calls GMSH to generate CalculiX compatible finite element meshes. Note that any other format supported by GMSH is practicable.

5. Conclusions

pyMeshFOAM is a Python application that generates meshes based on existing free or open source meshing tools. It basically writes input files for blockMesh, cfMesh, snappyHexMesh and GMSH depending on user parameters. Among other things, pyMeshFOAM manages the meshing process of boundary layers, a crucial issue in most CFD simulations. The developed tool also handles solid meshes for fluid–structure simulations.

Several types of shapes can be meshed based on pyMeshFOAM. We have introduced three basic shapes: triangle, cylinder and non-symmetric H-shape. Two 2D fluid meshes around a NACA0015 profile have also been treated. The 3D mesh of a cantilever wing is also successfully managed by pyMeshFOAM by extruding an airfoil profile and terminating it with a wing tip. The tool developed was also able to generate the mesh of an arterial system, a very irregular geometry.

In an ongoing work, this tool allows us to perform fluid–structure simulations of a NACA0015 airfoil surrounded by a water flow.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program, project HOMER: Holistic Optical Metrology for Aero-Elastic Research (grant agreement No 648161).

References

[1] OpenFOAM, https://openfoam.org/.
[2] Jasak H. OpenFOAM: Open source CFD in research and industry. Int J Naval Archit Ocean Eng 2009;1(2):89–94.
[3] Higuera P, Lara JL, Losada IJ. Realistic wave generation and active wave absorption for Navier–Stokes models, application to OpenFOAM®. Coast Eng 2013;71:102–18.
[4] Pendar M-R, Esmaeilifar E, Roohi E. LES study of unsteady cavitation characteristics of a 3-D hydrofoil with wavy leading edge. Int J Multiph Flow 2020;132:103415.
[5] Gärtner JW, Kronenburg A, Martin T. Efficient WENO library for OpenFOAM. SoftwareX 2020;12:100611.
[6] Spitzenberger A, Neumann S, Heinrich M, Schwarze R. Particle detection in VOF-simulations with OpenFOAM. SoftwareX 2020;11:100382.
[7] G. Dhondt, CalculiX, http://www.calculix.de/.
[8] Bungartz H-J, Lindner F, Gatzhammer B, Mehl M, Scheufele K, Shukaev A, et al. Precice - a fully parallel library for multi-physics surface coupling. Comput & Fluids 2016;141:250–8.
[9] cfMesh. 2021, http://cfmesh.com.
[10] Geuzaine C, Remacle J-F. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Internat J Numer Methods Engrg 2009;79(11):1309–31.
[11] Salmon F, Chatellier L. 3D fluid–structure interaction simulation of an hydrofoil at low Reynolds number. J Fluids Struct 2022;111:103573.