

From imaging to simulation: a workflow for converting CT/SEM data into OpenFOAM meshes

Jan Bohacek^{1*}, Jiri Hvozda¹, Miroslav Raudensky¹, Alexander Vakhrushev², Ebrahim Karimi-Sibaki², and Mayken Espinoza-Andaluz³

¹Heat Transfer and Fluid Flow Laboratory, Faculty of Mechanical Engineering, Brno University of Technology, Antonínska 548/1, 602 00 Brno, Czechia

²Christian-Doppler Laboratory for Metallurgical Applications of Magnetohydrodynamics, Dept. of Metallurgy, Montanuniversitaet Leoben, Franz-Joseph Strasse 18, 8700 Leoben, Austria.

³Centro de Energías Renovables y Alternativas, Facultad de Ingeniería Mecánica y Ciencias de la Producción, Escuela Superior Politécnica del Litoral, Guayaquil, Ecuador

Abstract. CT and SEM imaging often produce complex geometries, such as porous or irregular structures, which pose significant challenges for numerical simulations. In contrast to meshless methods like LBM or SPH, traditional FVM and FEM solvers require a high-quality computational mesh with consistent connectivity — a task that becomes particularly demanding for image-based data. This work presents a practical, step-by-step workflow for converting CT or SEM datasets into simulation-ready meshes for OpenFOAM. Two alternative meshing strategies are introduced, each discussed in terms of workflow complexity, mesh quality, and computational efficiency. The study aims to simplify the integration of experimental imaging and CFD modeling, bridging the gap between real microstructures and numerical analysis.

1 Introduction

In most research articles presenting computational results, the focus naturally lies on the governing equations used to describe the underlying physical phenomena, accompanied by the corresponding initial and boundary conditions. Computational details such as discretization schemes, linear solvers, and convergence criteria are typically summarized in a brief paragraph, often together with general information about the mesh—such as the number and type of elements or a comment on mesh quality. This brevity is understandable, as meshes are commonly generated using well-established commercial tools such as ANSYS Meshing, ANSA, or ICEM CFD, and are not themselves the central subject of the study. The primary goal usually concerns the simulation and analysis of physical processes rather than the meshing workflow. Nevertheless, it is important to recall the principle often summarized by the acronym GIGO—Garbage In, Garbage Out, originally coined by computer scientist George Fuchs in the 1950s. In computational modeling, this adage serves as a reminder that even the most sophisticated numerical methods cannot compensate for poor-quality input data or inadequate mesh generation.

It is worth noting that numerical methods beyond the classical finite element (FEM) and finite volume (FVM) formulations have been in use for quite some time. Approaches such as Smoothed Particle Hydrodynamics

(SPH) [1] and the Lattice Boltzmann Method (LBM), implemented for example in Palabos [2], OpenLB [3] do not require a computational mesh in the conventional sense. Instead, these meshless methods rely directly on the geometry—typically provided as an STL file—to define boundary surfaces, while the discretization of the volume is handled intrinsically by the method itself.

In contrast, complex multiphysics problems are still most often addressed using traditional FVM or FEM frameworks, nowadays frequently implemented in widely used open-source platforms such as OpenFOAM [4] or SU2 [5]. In these cases, a computational mesh is indispensable, and researchers must rely on either built-in meshing utilities or external mesh generators, which may be commercial or open-source.

In the author's experience, simpler geometries can be effectively meshed using open-source tools such as SALOME [6], whereas more complex or sensitive geometries are more reliably handled with robust commercial software such as ANSA. Nevertheless, even in commercial tools, manual intervention in geometry cleanup and meshing parameters is often required to achieve a mesh of sufficient quality. When the complexity of the geometry increases further—such as in cases where a three-dimensional image obtained from computed tomography (CT) represents an irregular, surface-rich porous medium—the meshing task becomes extremely demanding.

In this work, the authors intentionally employ a fully open-source workflow based on OpenFOAM and

* Corresponding author: jan.bohacek@vut.cz

ParaView [7], and present a straightforward method for converting image data into a simulation-ready computational mesh.

In [8], the authors used OpenFOAM to simulate flow through a 400^3 -voxel micro-CT image of Bentheimer sandstone ($5\ \mu\text{m}$ resolution). The image was initially meshed with a 200^3 Cartesian grid, and voxels with intermediate porosity ($0.01 \leq \epsilon_f \leq 0.99$) were locally refined, yielding a two-level mesh of 32 million cells with $5\ \mu\text{m}$ resolution near the fluid–solid interface. While the refinement strategy was outlined, details of the overall meshing workflow were not fully discussed.

In [9], a voxel-based workflow converted micro-CT stacks into a CFD-ready mesh. VoxTex [10] generated the voxel model, imported into Siemens Simcenter 3D [11], then exported as an STL for Simcenter STAR-CCM+ to create a polyhedral volume mesh. As noted, “the creation of a volume mesh can take more computational time than the simulation itself, it is important to study the mesh settings especially for a new type of geometry” [12]. VoxTex is not publicly available and its licensing conditions are unclear, while the other tools are commercial.

In [13], Tube2FEM provides a fully automated pipeline to convert 3D imaging data of tubular structures into a tetrahedral volume mesh for flow simulations. The workflow integrates image processing, graph-based network extraction, CAD reconstruction, and mesh generation using TetGen, which is free for research and educational use but requires permission for commercial purposes. The workflow also requires a commercial MATLAB installation, along with Python and ParaView, and its utilization is apparently not trivial. The approach minimizes manual intervention but has been demonstrated mainly on tubular geometries.

In [14], the authors propose an open-source workflow for generating CFD-ready volume meshes from μCT data, employing exclusively freely available software. The workflow comprises four key steps: (1) post-processing of image stacks in ImageJ (including median filtering, binning, and binarization), (2) surface reconstruction and STL generation in ParaView using the Contour filter, (3) file size reduction and surface smoothing in Blender through decimate collapse and simple smoothing algorithms, and (4) volumetric mesh generation with snappyHexMesh. This approach specifically targets boundary-aligned meshes by sequentially engaging the Contour filter in ParaView and snappyHexMesh in OpenFOAM. Notably, [14] represents the study most closely aligned with the methodology presented in the present work.

The objective of the present work is to provide relatively inexperienced OpenFOAM users with a ready-to-use solution for converting CT images into simulation-ready meshes without requiring the installation of additional software. In the following, we first discuss how image data can be efficiently processed in ParaView. Subsequently, three distinct approaches are presented for transforming the processed data from ParaView into high-quality computational meshes. The study is complemented with several illustrative examples of numerical heat transfer simulations.

2 Methodology

The objective of the proposed workflow is to enable the user to remain within the familiar environment of ParaView and OpenFOAM, without the need to rely on external meshing or preprocessing software.

2.1 ParaView: Image to Scalar Field

The simplest scenario involves 2D images in standard formats such as PNG, TIFF, or JPG, which are automatically recognized by ParaView through the appropriate image reader. Handling CT data, however, is slightly more involved, as these datasets are not necessarily identified automatically, requiring the user to manually select the correct image reader. In the present study, the CT data were provided as volumetric/voxel data in binary `.vol` format. The minimum setup then requires specifying the resolution, encoding style, and integer type. This information is typically available in a separate header file (e.g., from VGStudio MAX) or, in the case of binary files, can be determined using software such as ImageJ, myVGL, or other compatible tools.

In ParaView, the image data are always attributed as Point Data. In the next step, the image data are mapped onto an OpenFOAM mesh using the *Resample With Dataset* filter, where the source is set to the mesh and the input to the image dataset. Since OpenFOAM fields are stored as cell-centered values, the *Point Data to Cell Data* filter is subsequently applied to convert point values to cell-centered quantities associated with the corresponding cellIDs of the mesh. The resulting data can then be exported, for example, as a `.csv` table for further use.

In many cases, particularly for two-dimensional tests, a 2D image serves as input, whereas the OpenFOAM mesh is a 3D layer extruded from a 2D surface. Since the *Resample With Dataset* filter requires a 3D image, a simple workaround is to duplicate and slightly offset the 2D image using the *Transform* filter, and then merge both with the *Group Datasets* filter to create a thin volumetric dataset suitable for resampling.

2.2 Approach 1: setSet

So far, nothing has been modified in the original mesh generated, for example, by the utility `blockMesh`, and in some cases, this is not even necessary. Instead, the geometry can be divided into *cell zones*, within which different properties or physical models may be assigned. To achieve this, a scalar field (e.g., *imageData*) is first created in the `0/` folder by copying the previously stored CSV values into the `internalField` entry of *imageData*. Then, using the utility *setSet*, cell sets can be generated by thresholding the field, for example:

```
cellSet c0 new fieldToCell imageData  
1 255
```

Subsequently, the cell sets are converted into cell zones using:

```
cellZoneSet c0Zone new setToCellZone  
c0.
```

Approach 1 is demonstrated on a 2D steady-state heat diffusion case through a porous oxide, as shown in Fig. 1.

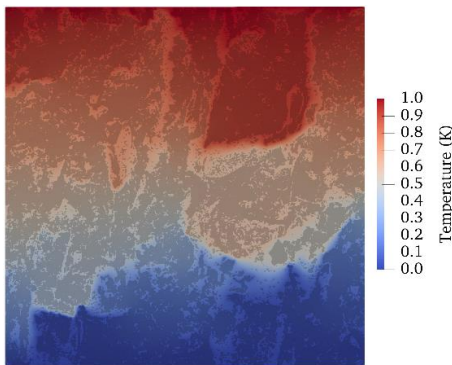


Fig. 1. Steady-state heat diffusion through porous iron oxide, characterized by solid (oxide) and gas (pores) cell zones defined using the *setSet* utility with the *fieldToCell* source.

2.3 Approach 2: *splitMeshRegions*

Unlike in the previous subsection, where neither the geometry nor the mesh was modified and both pores and oxides were included in the heat diffusion modeling, here the pores are assumed unnecessary and can be omitted (Fig. 2). The OpenFOAM utility *splitMeshRegions* is recommended, typically with the *cellZonesOnly* flag. If the retained cell zone contains multiple unconnected regions, the *-cellZones* flag should be used, as it splits cell zones and applies the walking algorithm, allowing a single cell zone to be subdivided into multiple regions if unconnected. Once the desired region is identified, the remaining regions can be removed from the *0/*, *constant/*, and *system/* folders. Alternatively, the *subsetMesh* utility can retain only the desired portion of the mesh, and both utilities can be combined sequentially to achieve the intended result.

Practical note: During this process, the original mesh may be split into thousands of regions. Typically, the largest region corresponds to the desired geometry, and *splitMeshRegions* can be safely aborted before full completion without affecting the usable region. The computational cost of Approach 2 is higher than that of Approach 1 and strongly depends on the number of regions generated.

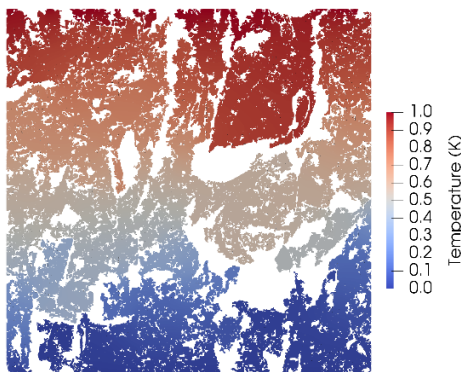


Fig. 2. Steady-state heat diffusion through porous iron oxide, with the gas (pores) removed from the computational mesh using the *splitMeshRegions* utility.

2.4 Approach 3: *snappyHexMesh*

Among the three approaches presented, this is by far the most computationally demanding, although it produces a boundary-aligned mesh. Its computational cost becomes firstly apparent in ParaView when using the *Contour* filter (based on the Marching Cubes algorithm) to generate smooth surface meshes (.stl) representing interfaces between two or more distinct regions. The threshold value for contouring can, for instance, be chosen to preserve the total volume of a region, measurable beforehand. The resulting STL file often contains multiple disconnected triangular meshes, which can be conveniently separated using the Python library *trimesh*. To avoid excessively long geometry definitions in *snappyHexMesh*, selected STL files can also be concatenated using *trimesh*.

In Fig. 3, a cut-out of the iron oxide on a steel sample is thresholded, where *surface_0.stl* represents the interface between the free surface and the oxide, and *surface_1.stl* corresponds to the interface between the oxide and either pores or the steel substrate.

In the previous subsection, the staircase mesh was generated, which is also the first step performed by *snappyHexMesh*—commonly referred to as the castellated mesh. Boundary alignment of mesh elements is achieved in the second step through snapping to the STL surfaces. Snapping may be followed by the addition of boundary layers, which is beyond the scope of this work. Nevertheless, the authors note that the *addLayers* option may lead to poor-quality meshes, particularly for complex geometries. In contrast, snapping can be well controlled to achieve high mesh quality, with the mesh shown in Fig. 4 exhibiting a non-orthogonality less than 50.

Only the critical parts of the *snappyHexMeshDict* are shown here, beginning with the geometry settings. STL surfaces are defined individually, for example:

```
geometry { s0 { type triSurfaceMesh; file "surface_0.stl"; } }. To enable snapping to interior STL surfaces, they must also be defined within refinementSurfaces along with parameters such as face zones, cell zones, and the corresponding cell zone orientation, for example: 

```
castellatedMeshControls { refinementSurfaces { s0 { level (0 0); faceZone fName; cellZone cName; cellZoneInside inside; } } }
```


```

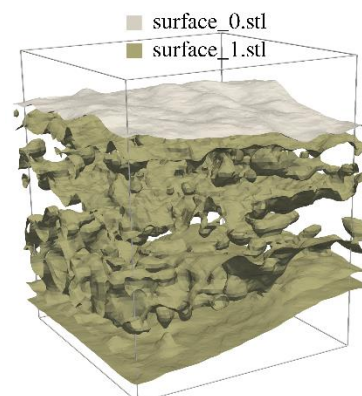


Fig. 3. Generation of suitable STLs for *snappyHexMesh*.

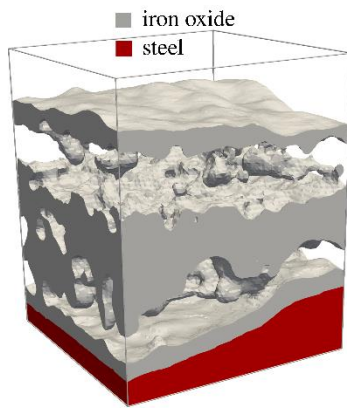


Fig. 4. Meshed geometry after *snappyHexMesh* and *splitMeshRegions* of iron oxide and steel substrate.

3 Summary and Conclusions

This work presented a practical and fully open-source workflow for converting CT and SEM imaging data into simulation-ready computational meshes for OpenFOAM. The primary objective was to provide relatively inexperienced users with a clear, step-by-step methodology that avoids reliance on commercial preprocessing or meshing software. A key element of the proposed workflow is the use of the *Resample With Dataset* filter in ParaView, which enables direct mapping of voxel-based image data onto an existing OpenFOAM mesh and thus represents the crucial link between experimental imaging and finite volume discretization. Based on this mapping, three alternative strategies for mesh handling were presented. The first approach employed the *setSet* utility to define cell zones directly from image-based scalar fields, enabling multiphase or multi-material simulations without modifying the original mesh. The second approach used *splitMeshRegions* to remove undesired regions and retain only the relevant computational domain. The third and most computationally demanding approach combined ParaView-based surface extraction with *snappyHexMesh* to generate boundary-aligned meshes. The three methods were demonstrated on representative heat transfer problems, highlighting their respective advantages in terms of simplicity, flexibility, and mesh quality. Overall, the proposed workflow bridges experimental imaging and CFD modeling and offers OpenFOAM users a transparent and reproducible path from image data to numerical simulation.

This research was supported by the INTER-EXCELLENCE II program of Ministry of Education, Youth and Sports of the Czech Republic, under project LUAUS24006. The paper presented has been supported by the internal grant of the Brno University of Technology focused on specific research and development No. FSI-S-23-8254. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic. Computational resources were provided by the ELIXIR-CZ project (ID:90255), part of the international ELIXIR infrastructure.

Data will be made available on request.

References

1. A. J. C. Crespo, J. M. Domínguez, B. D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, and O. García-Feal, *DualSPHysics: Open-Source Parallel CFD Solver Based on Smoothed Particle Hydrodynamics (SPH)*, *Computer Physics Communications*, **187**, 204 (2015) <https://doi.org/10.1016/j.cpc.2014.10.004>
2. J. Latt, O. Malaspinas, D. Kontaxakis, A. Parmigiani, D. Lagrava, F. Brogi, M. B. Belgacem, Y. Thorimbert, S. Leclaire, S. Li, F. Marson, J. Lemus, C. Kotsalos, R. Conradin, C. Coreixas, R. Petkantchin, F. Raynaud, J. Beny, and B. Chopard, *Palabos: Parallel Lattice Boltzmann Solver*, *Computers & Mathematics with Applications*, **81**, 334 (2021) <https://doi.org/10.1016/j.camwa.2020.03.022>
3. M. J. Krause, A. Kummerländer, S. J. Avis, H. Kusumaatmaja, D. Dapelo, F. Klemens, M. Gaedtke, N. Hafen, A. Mink, R. Trunk, J. E. Marquardt, M.-L. Maier, M. Haussmann, and S. Simonis, *OpenLB—Open Source Lattice Boltzmann Code*, *Computers & Mathematics with Applications*, **81**, 258 (2021) <https://doi.org/10.1016/j.camwa.2020.04.033>
4. H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, *A Tensorial Approach to Computational Continuum Mechanics Using Object-Oriented Techniques*, *Computer in Physics*, **12**, 620 (1998) <https://doi.org/10.1063/1.168744>
5. F. Palacios, T. D. Economon, A. Aranake, S. R. Copeland, A. K. Lonkar, T. W. Lukaczyk, D. E. Manosalvas, K. R. Naik, S. Padron, B. Tracey, A. Variyar, and J. J. Alonso, in *Stanford University Unstructured (SU2): Analysis and Design Technology for Turbulent Flows*, 52nd Aerospace Sciences Meeting (American Institute of Aeronautics and Astronautics, 2014)
6. A. Ribes and C. Caremoli, in *Salomé Platform Component Model for Numerical Simulation*, 31st Annual International Computer Software and Applications Conference (COMPSAC 2007) (2007), pp. 553–564
7. J. Ahrens, B. Geveci, and C. Law, in *ParaView: An End-User Tool for Large-Data Visualization* (Elsevier, 2005), pp. 717–731
8. J. Maes and H. P. Menke, *GeoChemFoam: Direct Modelling of Flow and Heat Transfer in Micro-CT Images of Porous Media*, *Heat Mass Transfer*, **58**, 1937 (2022) <https://doi.org/10.1007/s00231-022-03221-2>
9. R. Otto, U. Soellner, C. Kiener, S. Boschert, R. Wüchner, and K. Sørby, *Voxel-Based 3D Reconstruction of Additively Manufactured Open Porous Structures for CFD Simulation*, *Int J Adv*

- Manuf Technol, **134**, 841 (2024)
<https://doi.org/10.1007/s00170-024-14169-4>
10. I. Straumit, S. V. Lomov, and M. Wevers, Quantification of the Internal Structure and Automatic Generation of Voxel Models of Textile Composites from X-Ray Computed Tomography Data, *Composites Part A: Applied Science and Manufacturing*, **69**, 150 (2015)
<https://doi.org/10.1016/j.compositesa.2014.11.016>
 11. O. Shishkina, M. Wevers, S. V. Lomov, and L. Farkas, in *Micro-CT-Based Numerical Validation of the Local Permeability Map for the B-Pillar Infusion Simulation*, Proceedings of the 20th European Conference on Composite Materials - Composites Meet Sustainability (Lausanne, Switzerland, 2022), pp. 1334–1341
 12. J. H. Ferziger, M. Perić, and R. L. Street, *Computational Methods for Fluid Dynamics* (Springer International Publishing, Cham, 2020)
 13. H. Cheikh Sleiman, K. M. Moerman, D. C De Oliveira, J. Jacob, N. Mogulkoc, B. R. Davidson, S. Walker-Samuel, and R. J. Shipley, Tube2FEM: A General-Purpose Highly Automated Pipeline for Flow-Related Processes in (Embedded) Tubular Objects, *R Soc Open Sci*, **12**, 242025 (2025)
<https://doi.org/10.1098/rsos.242025>
 14. K. Kuhlmann, C. Sinn, J. M. U. Siebert, G. Wehinger, J. Thöming, and G. R. Pesch, From μ CT Data to CFD: An Open-Source Workflow for Engineering Applications, *Engineering Applications of Computational Fluid Mechanics*, **16**, 1706 (2022)
<https://doi.org/10.1080/19942060.2022.2109758>