Lecture 06 - OpenFOAM

Finite-volume CFD workflows and solver execution

Dr. Lambert Theisen & Dr. Georgii Oblapenko

1 Objectives

This lecture introduces OpenFOAM from the perspective of a simulation software workflow rather than from a full CFD theory derivation.

1. Understand what an OpenFOAM case directory contains.
2. Run a minimal solver workflow from the command line.
3. Inspect logs, residuals, and generated time directories.
4. Modify one numerical or physical parameter and rerun the case reproducibly.
5. Connect OpenFOAM's finite-volume abstractions to the discretization ideas from the previous lectures.

> **Execution note**
>
> The code cells are intentionally written as shell-style execution examples. They require an OpenFOAM environment to be sourced before use, for example through the installation's `bashrc`/`zshrc` script or a container image. They are not evaluated during Quarto rendering by default.

2 What is OpenFOAM?

OpenFOAM is a C++ finite-volume toolbox for continuum mechanics, with a strong focus on CFD. A typical workflow is not "write one monolithic program" but:

- choose a solver executable whose equations match the model,
- prepare a case directory with mesh, fields, numerical schemes, and solver controls,
- run mesh generation / preprocessing utilities,
- execute the solver,
- postprocess fields, forces, residuals, and derived quantities.

The important software-development lesson is that OpenFOAM exposes many modeling decisions as text files under version control. That makes case setup reviewable, scriptable, and reproducible.

3 What is OpenFOAM actually solving?

Most OpenFOAM solvers are built around conservation laws. For a flow solver, the unknowns are usually fields such as velocity `U`, pressure `p`, temperature `T`, or turbulence quantities. In the incompressible cavity example used below, `icoFoam` solves the unsteady incompressible Navier-Stokes equations:

$$\nabla \cdot \mathbf{u} = 0,$$

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) - \nabla \cdot (\nu \nabla \mathbf{u}) = -\nabla p.$$

Here `U` is the cell-centered velocity field, `p` is the kinematic pressure, and `nu` is the kinematic viscosity from `constant/transportProperties`. The boundary conditions in `0/U` and `0/p` close the initial-boundary value problem.

The practical interpretation is:

- continuity says that incompressible velocity has zero net volume flux imbalance,
- momentum says that velocity changes due to transient storage, convection, diffusion/viscosity, and pressure forces,
- the pressure equation is not an independent material law; it is introduced by the pressure-velocity coupling algorithm to enforce incompressibility.

4 Finite-volume idea

The finite-volume method starts from the integral form of a conservation law on each mesh cell. For a scalar quantity ϕ, a typical transport equation can be written as

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\mathbf{u}\phi) - \nabla \cdot (\Gamma \nabla \phi) = S.$$

Integrating over one control volume V_i gives

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{V_i} \phi \, \mathrm{d}V + \int_{\partial V_i} \mathbf{u}\phi \cdot \mathbf{n} \, \mathrm{d}A - \int_{\partial V_i} \Gamma \nabla \phi \cdot \mathbf{n} \, \mathrm{d}A = \int_{V_i} S \, \mathrm{d}V.$$

The divergence terms become fluxes through cell faces by Gauss' theorem. This is the core finite-volume move: instead of asking only what happens at points, ask how much quantity enters and leaves each cell. Neighboring cells share a face, so the flux leaving one cell is the flux entering the next cell. This gives local conservation by construction.

OpenFOAM stores most primary unknowns at cell centers. Face values and face gradients are reconstructed by interpolation schemes from `system/fvSchemes`. The assembled result is an algebraic equation for every cell value:

$$a_P \phi_P = \sum_N a_N \phi_N + b_P,$$

where P is the current cell, N are neighboring cells, and the coefficients contain time-stepping, convection, diffusion, source terms, and boundary-condition contributions. The linear solvers and tolerances for these systems are configured in `system/fvSolution`.

5 What do the schemes mean?

`system/fvSchemes` tells OpenFOAM how to approximate the differential operators in the finite-volume equations. It does not usually define the physical model itself; it defines the numerical translation from integrals over cells and faces to algebraic coefficients.

A typical file is organized into dictionaries such as:

```
ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    div(phi,U)      Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}
```

Read entries from left to right. For example, `Gauss linear corrected` means: use Gauss' theorem to convert a volume divergence into face fluxes, use linear interpolation to obtain face values, and apply a correction for non-orthogonal mesh geometry.

5.1 `ddtSchemes`: time derivative

`ddtSchemes` discretizes terms such as

$$\frac{\partial \phi}{\partial t}.$$

Common choices:

- `steadyState`: no transient term; used for steady solvers or pseudo-steady equations.
- `Euler`: first-order implicit Euler; robust and damping, but only first-order accurate in time.
- `backward`: second-order backward differentiation; more accurate for smooth transients, but uses more time history.
- `CrankNicolson`: second-order centered-in-time blend; less diffusive, but can oscillate unless blended with Euler, for example `CrankNicolson 0.9`.

For a first live demo, `Euler` is a good default because it is predictable. For accuracy studies, compare `Euler` and `backward` while reducing the time step.

5.2 `gradSchemes`: gradients at cells and faces

`gradSchemes` approximates $\nabla\phi$. Gradients appear directly in diffusion terms, stress models, limiters, and postprocessing quantities.

Common choices:

- `Gauss linear`: compute gradients from face values obtained by linear interpolation; common and efficient.
- `leastSquares`: reconstruct gradients by fitting neighboring cell values; often useful on irregular unstructured meshes.
- `cellLimited Gauss linear 1`: limit the gradient to avoid creating new extrema; more bounded, less sharp.

The main tradeoff is accuracy versus robustness. Unrestricted gradients can be more accurate on smooth fields, while limited gradients are safer near sharp fronts, bad cells, or strongly convective regions.

5.3 `divSchemes`: convection and flux divergence

`divSchemes` is often the most important part for flow simulations because it controls convection terms such as

$$\nabla \cdot (\mathbf{u}\phi).$$

In OpenFOAM this often appears as `div(phi,U)`, where `phi` is the face flux and `U` is the transported field. The central question is: what value of `U` should be used on a face between two cells?

Common choices:

- `Gauss upwind`: use the upstream cell value at each face. Very robust and bounded, but first-order and numerically diffusive.
- `Gauss linear`: central interpolation between neighboring cells. Second-order on regular meshes, but can oscillate for convection-dominated problems.
- `Gauss linearUpwind grad(U)`: upwind-biased second-order reconstruction using a gradient. Less diffusive than upwind, usually more stable than pure central differencing.
- `Gauss limitedLinear 1`: total-variation-diminishing style limited linear scheme. It blends high-order behavior in smooth regions with limiting near steep gradients.
- `bounded Gauss ...`: adds boundedness treatment for convective terms, often useful for steady incompressible solvers.

For teaching, this is a useful stability experiment: `upwind` usually runs but smears features; `linear` is sharper but can fail or oscillate at high Peclet/Courant numbers; limited schemes are the practical compromise.

5.4 `laplacianSchemes`: diffusion terms

`laplacianSchemes` discretizes terms such as

$$\nabla \cdot (\Gamma \nabla \phi),$$

which represent viscosity, thermal diffusion, or other diffusive transport.

A common entry is:

```
laplacian(nu,U)  Gauss linear corrected;
```

The parts mean:

- `Gauss`: convert the volume integral of the divergence into face fluxes.
- `linear`: interpolate the diffusion coefficient or field values to faces.
- `corrected`: account for non-orthogonal meshes, where the line between cell centers is not aligned with the face normal.

Geometry matters strongly here. On an orthogonal structured mesh, the basic face-normal gradient is simple. On skewed or non-orthogonal meshes, OpenFOAM needs correction terms. Common variants are `orthogonal`, `uncorrected`, `corrected`, and `limited corrected`. More correction improves consistency on poor meshes, but can make the matrix less benign; limited correction is a robustness compromise.

5.5 `interpolationSchemes` and `snGradSchemes`

`interpolationSchemes` controls how cell-centered values are moved to faces when an operator needs a face value. The common default is `linear`, meaning a central interpolation between neighboring cell centers.

`snGradSchemes` controls surface-normal gradients, for example

$$\nabla \phi \cdot \mathbf{n}$$

on a cell face. These gradients are central for diffusion and pressure equations. As with `laplacianSchemes`, `corrected` and `limited corrected` are about handling mesh non-orthogonality.

A good mental model is:

- interpolation schemes answer: what value lives on this face?
- surface-normal gradient schemes answer: what derivative crosses this face?
- divergence schemes answer: what net flux leaves this control volume?

5.6 Practical scheme-selection heuristics

For a first OpenFOAM case, avoid treating schemes as cosmetic settings. They decide the balance between accuracy, boundedness, and nonlinear robustness.

- Start robust: `Euler`, `Gauss upwind`, `Gauss linear corrected`.
- Then improve accuracy: try `backward` in time and `linearUpwind` or limited convection schemes.
- Watch diagnostics: residuals, Courant number, continuity errors, boundedness of fields, and mesh quality.
- Do not compare physical conclusions across schemes until time-step, mesh, and scheme sensitivity are understood.

The solver in `fvSolution` decides how to solve the assembled algebraic equations. The schemes in `fvSchemes` decide what algebraic equations are assembled in the first place.

6 How this appears in OpenFOAM code

OpenFOAM's C++ equation syntax is deliberately close to the mathematical operator notation. A momentum equation in a solver can look schematically like this:

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
);
```

The prefix matters:

- `fvm::...` means an implicit finite-volume operator that contributes to the matrix,
- `fvc::...` means an explicit finite-volume calculation that is evaluated from known fields,
- `phi` is usually the face flux field, not the scalar unknown from the abstract equation above.

This is the main conceptual bridge: OpenFOAM solvers are C++ programs, but the solver code reads like a discretized conservation law assembled over mesh cells and faces.

7 Case directory anatomy

A basic OpenFOAM case usually has this structure:

```
case/
|-- 0/                  # initial and boundary fields
|   |-- U
|   `-- p
|-- constant/           # physical properties and mesh
|   |-- transportProperties
|   `-- polyMesh/
`-- system/             # numerics and run control
    |-- blockMeshDict
```

```
    |-- controlDict
    |-- fvSchemes
    `-- fvSolution
```

In lecture, keep returning to the separation of concerns:

- `0/`: what is the initial-boundary value problem?
- `constant/`: what material/model parameters and mesh are used?
- `system/`: how is the discrete problem advanced and solved?

8 Environment check

On macOS, use a container unless OpenFOAM is already available through a VM, cluster module, or native installation. The examples below use the multi-architecture Docker image `microfluidica/openfoam:13`, which works on Apple Silicon and exposes OpenFOAM 13 commands such as `blockMesh` and `icoFoam`.

If OpenFOAM is installed directly on the host, the same solver commands can be run without the `docker run ...` prefix.

```python
import subprocess
from pathlib import Path

OPENFOAM_IMAGE = "microfluidica/openfoam:13"
PROJECT_DIR = Path.cwd().resolve()


def foam_cmd(*args, workdir="/work"):
    return [
        "docker",
        "run",
        "--rm",
        "-v",
        f"{PROJECT_DIR}:/work",
        "-w",
        workdir,
        OPENFOAM_IMAGE,
        *args,
    ]


subprocess.run(foam_cmd("blockMesh", "-help"), check=True)
```

```
Usage: blockMesh [OPTIONS]
options:
  -blockTopology    write block edges and centres as .obj files
  -case <dir>       specify alternate case directory, default is the cwd
  -dict <file>      read control dictionary from specified location
  -fileHandler <handler>
```

```
                    override the fileHandler
  -libs '("lib1.so" ... "libN.so")'
                    pre-load libraries
  -mesh <name>      specify alternative mesh
  -noClean          keep the existing files in the polyMesh
  -region <name>    specify alternative mesh region
  -srcDoc           display source code in browser
  -doc              display application documentation in browser
  -help             print the usage

Block description

  For a given block, the correspondence between the ordering of
  vertex labels and face labels is shown below.
  For vertex numbering in the sequence 0 to 7 (block, centre):
    faces 0 (f0) and 1 are left and right, respectively;
    faces 2 and 3 are front and back;
    and faces 4 and 5 are bottom and top::

                7 ---- 6
          f5    |\     |\   f3
          |     | 4 ---- 5   \
          |     3 |--- 2 |    \
          |      \|     \|    f2
          f4      0 ---- 1

       Z          f0 ----- f1
       |  Y
       | /
       0 --- X


Using: OpenFOAM-13 (see https://openfoam.org)
Build: 13-58ed5c2046ef


CompletedProcess(args=['docker', 'run', '--rm', '-v', '/Users/lamberttheisen/Desktop/gits/acom
w', '/work', 'microfluidica/openfoam:13', 'blockMesh', '-help'], returncode=0)
```

9 Example 1: run the cavity tutorial

The classical first OpenFOAM case is the transient lid-driven cavity solved by `icoFoam`. It is small enough for live execution and useful for explaining the case directory.

The workflow is:

1. Copy the tutorial case into a scratch directory.
2. Generate the mesh with `blockMesh`.
3. Run `icoFoam` and store the log.
4. Inspect the created time directories and residual output.

```python
work = Path("openfoam-work")
case = work / "cavity"

work.mkdir(exist_ok=True)

script = """
set -euo pipefail
rm -rf openfoam-work/cavity
cp -r "$FOAM_TUTORIALS/legacy/incompressible/icoFoam/cavity/cavity" openfoam-work/cavity
blockMesh -case openfoam-work/cavity
icoFoam -case openfoam-work/cavity > openfoam-work/cavity/log.icoFoam 2>&1
"""
subprocess.run(foam_cmd("bash", "-lc", script), check=True)

print("Generated time directories:")
print([p.name for p in case.iterdir() if p.is_dir() and p.name.replace(".", "", 1).isdigit()][
```

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  13
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
Build  : 13-58ed5c2046ef
Exec   : blockMesh -case openfoam-work/cavity
Date   : May 13 2026
Time   : 08:33:50
Host   : "4d2148ac0623"
PID    : 197
I/O    : uncollated
Case   : openfoam-work/cavity
nProcs : 1
sigFpe : Floating point exception trapping - not supported on this platform
fileModificationChecking : Monitoring run-time modified files using timeStampMaster (fileModif
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Reading "blockMeshDict"

Creating block mesh from
    "system/blockMeshDict"
No non-linear block edges defined
No non-planar block faces defined
Creating topology blocks
Creating topology patches

Creating block mesh topology
```

```
Basic statistics
    Number of internal faces : 0
    Number of boundary faces : 6
    Number of defined boundary faces : 6
    Number of undefined boundary faces : 0

Checking topology
Creating block offsets
Creating merge list .

Creating polyMesh from blockMesh
Creating points with scale 0.1
    Block 0 cell size :
        i : 0.005 .. 0.005
        j : 0.005 .. 0.005
        k : 0.01
Creating cells
Creating patches

No patch pairs to merge

Writing polyMesh
----------------
Mesh Information
----------------
  boundingBox: (0 0 0) (0.1 0.1 0.01)
  nPoints: 882
  nCells: 400
  nFaces: 1640
  nInternalFaces: 760
----------------
Patches
----------------
  patch 0 (start: 760 size: 20) name: movingWall
  patch 1 (start: 780 size: 60) name: fixedWalls
  patch 2 (start: 840 size: 800) name: frontAndBack

End

Generated time directories:
['0.3', '0.4', '0', '0.5', '0.2', '0.1']
```

10 Inspect the solver log

OpenFOAM logs are part of the numerical evidence. For a first pass, look for:

- selected solver and preconditioner for each equation,
- initial and final residuals,
- time-step progression,
- continuity errors,

- execution time.

```
log_text = (case / "log.icoFoam").read_text(errors="replace")

interesting = [
    line
    for line in log_text.splitlines()
    if "Solving for" in line or "Courant Number" in line or "ExecutionTime" in line
]

print("\n".join(interesting[:30]))
```

```
Courant Number mean: 0 max: 0
smoothSolver:  Solving for Ux, Initial residual = 1, Final residual = 8.90511e-
06, No Iterations 19
smoothSolver:  Solving for Uy, Initial residual = 0, Final residual = 0, No Iterations 0
DICPCG:  Solving for p, Initial residual = 1, Final residual = 0.0492854, No Iterations 12
DICPCG:  Solving for p, Initial residual = 0.590864, Final residual = 2.65225e-
07, No Iterations 35
ExecutionTime = 0.00299 s  ClockTime = 0 s
Courant Number mean: 0.0976825 max: 0.585607
smoothSolver:  Solving for Ux, Initial residual = 0.160686, Final residual = 6.83031e-
06, No Iterations 19
smoothSolver:  Solving for Uy, Initial residual = 0.260828, Final residual = 9.65939e-
06, No Iterations 18
DICPCG:  Solving for p, Initial residual = 0.428925, Final residual = 0.0103739, No Iterations
DICPCG:  Solving for p, Initial residual = 0.30209, Final residual = 5.26569e-
07, No Iterations 33
ExecutionTime = 0.003706 s  ClockTime = 0 s
Courant Number mean: 0.144686 max: 0.758934
smoothSolver:  Solving for Ux, Initial residual = 0.0447632, Final residual = 9.12473e-
06, No Iterations 15
smoothSolver:  Solving for Uy, Initial residual = 0.0817804, Final residual = 6.96014e-
06, No Iterations 17
DICPCG:  Solving for p, Initial residual = 0.131584, Final residual = 0.00445878, No Iteration
DICPCG:  Solving for p, Initial residual = 0.0973392, Final residual = 9.15339e-
07, No Iterations 31
ExecutionTime = 0.004231 s  ClockTime = 0 s
Courant Number mean: 0.167375 max: 0.800501
smoothSolver:  Solving for Ux, Initial residual = 0.0255033, Final residual = 6.64964e-
06, No Iterations 15
smoothSolver:  Solving for Uy, Initial residual = 0.0467692, Final residual = 7.05331e-
06, No Iterations 16
DICPCG:  Solving for p, Initial residual = 0.0699325, Final residual = 0.00118408, No Iteratio
DICPCG:  Solving for p, Initial residual = 0.0550257, Final residual = 9.8463e-
07, No Iterations 30
ExecutionTime = 0.004821 s  ClockTime = 0 s
Courant Number mean: 0.182124 max: 0.819978
smoothSolver:  Solving for Ux, Initial residual = 0.014788, Final residual = 6.26013e-
06, No Iterations 14
```

```
smoothSolver:  Solving for Uy, Initial residual = 0.0306892, Final residual = 8.61293e-
06, No Iterations 15
DICPCG:  Solving for p, Initial residual = 0.0366985, Final residual = 0.000863614, No Iterati
DICPCG:  Solving for p, Initial residual = 0.0280626, Final residual = 6.16271e-
07, No Iterations 30
ExecutionTime = 0.005342 s  ClockTime = 0 s
```

11 Example 2: change runtime controls

The main simulation loop is configured in `system/controlDict`. For live teaching, change one setting at a time and rerun from a clean case.

Useful first edits:

- reduce `endTime` for a fast test run,
- change `writeInterval` to control output volume,
- switch `writeFormat` between `ascii` and `binary`,
- set `purgeWrite` to avoid filling the working directory during experiments.

```python
control_dict = case / "system" / "controlDict"
print(control_dict.read_text())

# Classroom exercise:
# 1. Edit endTime and writeInterval in system/controlDict.
# 2. Delete generated time directories except 0.
# 3. Rerun icoFoam.
# 4. Compare the new log and output directory list.
```

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  13
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

startFrom       startTime;

startTime       0;

stopAt          endTime;
```

```
endTime         0.5;

deltaT          0.005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

// ************************************************************************* //
```

12 Example 3: solver controls

Linear-solver and pressure-velocity coupling controls live in `system/fvSolution`. This is the OpenFOAM analogue of the PETSc discussion in Lecture 04: the PDE discretization leads to algebraic systems, and solver settings decide how those systems are solved.

For the cavity case, inspect `fvSolution` and identify:

- which equations are solved,
- which linear solvers are selected,
- the tolerances and relative tolerances,
- the pressure-velocity coupling loop.

```python
fv_solution = case / "system" / "fvSolution"
print(fv_solution.read_text())
```

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  13
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
```

```
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}


// ************************************************************************* //
```

13 Example 4: mesh generation with `blockMesh`

`blockMesh` is enough for simple hexahedral meshes and a useful entry point for the OpenFOAM dictionary format. Read `system/blockMeshDict` before running the mesher.

Discussion prompts:

- Which vertices define the computational domain?
- Which patches become boundary conditions in the field files under `0/`?

- How does increasing the cell count change runtime and output size?
- Where would an external mesh generator enter this workflow?

```
block_mesh_dict = case / "system" / "blockMeshDict"
print(block_mesh_dict.read_text())

subprocess.run(foam_cmd("checkMesh", "-case", str(case)), check=True)
```

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  13
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
```

```
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);


// ************************************************************************* //

/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  13
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
Build  : 13-58ed5c2046ef
Exec   : checkMesh -case openfoam-work/cavity
Date   : May 13 2026
Time   : 08:33:50
Host   : "d6cf3d27a156"
PID    : 1
I/O    : uncollated
Case   : openfoam-work/cavity
nProcs : 1
sigFpe : Floating point exception trapping - not supported on this platform
fileModificationChecking : Monitoring run-time modified files using timeStampMaster (fileModif
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0

Time = 0s
```

```
Mesh stats
    points:           882
    internal points:  0
    faces:            1640
    internal faces:   760
    cells:            400
    faces per cell:   6
    boundary patches: 3
    point zones:      0
    face zones:       0
    cell zones:       0

Overall number of cells of each type:
    hexahedra:     400
    prisms:        0
    wedges:        0
    pyramids:      0
    tet wedges:    0
    tetrahedra:    0
    polyhedra:     0

Checking topology...
    Boundary definition OK.
    Cell to face addressing OK.
    Point usage OK.
    Upper triangular ordering OK.
    Face vertices OK.
    Number of regions: 1 (OK).

Checking patch topology for multiply connected surfaces...
    Patch               Faces    Points   Surface topology
    movingWall          20       42       ok (non-closed singly connected)
    fixedWalls          60       122      ok (non-closed singly connected)
    frontAndBack        800      882      ok (non-closed singly connected)

Checking geometry...
    Overall domain bounding box (0 0 0) (0.1 0.1 0.01)
    Mesh has 2 geometric (non-empty/wedge) directions (1 1 0)
    Mesh has 2 solution (non-empty) directions (1 1 0)
    All edges aligned with or perpendicular to non-empty directions.
    Max cell openness = 1.35525e-16 OK.
    Max aspect ratio = 1 OK.
    Minimum face area = 2.5e-05. Maximum face area = 5e-05.  Face area magnitudes OK.
    Min volume = 2.5e-07. Max volume = 2.5e-07.  Total volume = 0.0001.  Cell volumes OK.
    Mesh non-orthogonality Max: 0 average: 0
    Non-orthogonality check OK.
    Face pyramids OK.
    Max skewness = 1.81581e-14 OK.
    Coupled point location match (average 0) OK.
```

```
Mesh OK.

End
```

```
CompletedProcess(args=['docker', 'run', '--rm', '-v', '/Users/lamberttheisen/Desktop/gits/acom
w', '/work', 'microfluidica/openfoam:13', 'checkMesh', '-case', 'openfoam-work/cavity'], retur
```

14 Postprocessing hooks

For a live session, use ParaView or `paraFoam` when available. For scripted workflows, prefer utilities that create text output suitable for versioned comparison.

Possible follow-ups:

- add function objects to `controlDict`,
- sample along a line or on a plane,
- compute forces for an external-flow case,
- compare residual histories between two solver settings.

```
# Interactive visualization: open the generated case with ParaView on the host,
# or run ParaView/paraFoam in a GUI-capable OpenFOAM environment.

# Lightweight file-system inspection for a scripted notebook workflow:
for path in sorted(case.glob("*/U"))[:10]:
    print(path)
```

```
openfoam-work/cavity/0/U
openfoam-work/cavity/0.1/U
openfoam-work/cavity/0.2/U
openfoam-work/cavity/0.3/U
openfoam-work/cavity/0.4/U
openfoam-work/cavity/0.5/U
```

15 Suggested in-class exercise

Use the cavity case as a reproducibility exercise.

1. Commit the untouched tutorial case.
2. Change exactly one setting in `system/controlDict` or `system/fvSolution`.
3. Rerun the workflow from mesh generation to solver execution.
4. Compare `git diff`, generated time directories, and selected log lines.
5. Decide whether the change affected the model, the discretization, the algebraic solve, or only output management.

16 Placeholders for lecture expansion

- Add one slide comparing OpenFOAM dictionaries to FEniCSx/PETSc Python options.
- Add one case with a steady solver, for example `simpleFoam` on a simple incompressible tutorial.
- Add one failure mode: missing boundary condition, bad mesh, unstable time step, or too-loose tolerance.
- Add one containerized execution example if the teaching environment does not provide a native OpenFOAM installation.