

e3prepToFoam: a mesh generator for OpenFOAM

Mechanical Engineering Technical Report 2015/04

Ingo JAHN; Kan QIN

School of Mechanical and Mining Engineering
The University of Queensland.

May 1, 2015

Abstract

`e3prepToFoam.py` is a utility to convert structured body-conforming multi-block meshes from the `e3prep`/Eilmer format to the OpenFOAM `foam` format. This report is a user-guide for the mesh conversion tool and provides several examples to help with the conversion. The tool can perform the following tasks:

- Convert 2-D Eilmer mesh into 1 cell deep 2-D OpenFOAM `foam` mesh
- Convert 2-D axisymmetric Eilmer mesh into 1 layer thick wedge shaped OpenFOAM `foam` mesh
- Convert 3-D Eilmer mesh into 3-D OpenFOAM `foam` mesh.

1 Introduction

As part of their code collection the *CFCFD Group* at the University of Queensland distributes the multi-block structured mesh generator `e3prep`. The mesh generator allows the generation of body conforming grids using simple easily scripted python front end. To utilise these capabilities with OpenFOAM, the grid conversion tool `e3prepToFoam` has been created. The tool allows the conversion of structured multi-block `e3prep` meshes into the OpenFOAM `foam` format. The tool also supports the generation of grouped boundary patches to simplify the boundary condition definition in OpenFOAM.

This report acts as a user guide and theory guide for `e3prepToFoam`. It is to be read in conjunction with the Eilmer user guide [1], which describes the mesh generation using `e3prep`.

1.1 Compatibility

`e3prepToFoam` uses functions from the *CFCFD Group* code collection (Eilmer3), the OpenFOAM distribution [2], python, and C++. The following dependencies exist:

Eilmer3 `e3prepToFoam` has been included as part of Eilmer code distribution from November 2014 onwards.

OpenFOAM The utility has been tested with OpenFOAM Vers. 2.3 and OpenFOAM-extended Vers. 3.1.

However it should be compatible with earlier releases also.

python The code has a number of python and C++ dependencies. It is recommended to install the dependencies list from the CFCFD webpage <http://cfcfd.mechmining.uq.edu.au/getting-started.html>

1.2 Citing this tool

When using the tool in simulations that lead to published works, it is requested that the following works are cited:

- This report to cover the `e3prepToFoam.py` mesh conversion tool.
Ingo Jahn, Kan Qin (2015), "e3prepToFoam: a mesh generator for OpenFOAM", Mechanical Engineering Technical Report 2015/04, pp 1-66, The University of Queensland
- The following report which covers `e3prep.py` the underlying code used to generate the mesh.
PA Jacobs, RJ Gollan, DF Potter (2014), "The Eilmer3 code: user guide and example book", Mechanical Engineering Technical Report 2014/04, pp 1-447, The University of Queensland

2 Distribution and Installation

`e3prepToFoam.py` is distributed as part of the code collection maintained by the *CFCFD Group* at the University of Queensland [3]. This collection is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version. This program collection is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details <http://www.gnu.org/licenses/>.

The code will be automatically installed during a typical build of Eilmer3. Download and build instructions are available from the CFCFD webpage <http://cfcfd.mechmining.uq.edu.au/>.

2.1 Modifying the code

The working version of `e3prepToFoam.py` is installed in the `$HOME/e3bin` directory. If you perform modifications or improvements to the code please submit an updated version together with a short description of the changes to the authors. Once reviewed the changes will be included in future versions of the code.

3 Using the Tool

3.1 5-minute version for experienced python, e3prep, and OpenFOAM Users

If you have used `e3prep.py` and OpenFOAM before, this for you.

3.1.1 Creating the directory structure

`e3prepToFoam.py` requires the correct OpenFOAM directory structure to function. Either copy an existing directory structure from an OpenFOAM example or manually create a directory structure. Once created add a folder titled `e3prep` to the root (`/case`) directory of your simulation. The resulting directory and file structure, required for `e3prepToFoam` to run correctly is:

```
case/  
  0/  
    constant/  
      constant/polyMesh  
    system/  
      system/ controlDict ← file required  
      system/ fvSchemes ← file required  
      system/ fvSolution ← file required  
    e3prep/ ← added for e3prepToFoam
```

All files and directories are required, to ensure correct operation. The directories can be empty, apart from the 3 files in the `/system` directory listed above.

3.1.2 Creating your job.py file for e3prep

Within the `/case/e3prep/` directory, create your typical `job.py` file as used for `e3prep.py`. See examples in section 5 for more details. The key differences to typical Eilmer meshing are:

- set `gdata.dimensions = X` with `X = 2` or `3`
- set `gdata.axisymmetric_flag = X` with `X = 0` or `1`
- setting of gas model is not required. Actual gas model is defined in `case/constant/...`
- define blocks as usual. setting of fill condition is not required, as this is defined in `case/0/...`
- include `identify_block_connections()`
- use following command to label external block faces: `blk0.bc_list[EAST] = ExtrapolateOutBC(label="NAME"` where `NAME` is one of the following `OF_inlet_nn`, `OF_outlet_nn`, `OF_wall_nn`, `OF_symmetry_nn`. In a 2-D mesh only the north, south, west and east faces need to be labelled. See note below for more details.
- include `sketch.prefer_bc_labels_on_faces()` to show face labels in svg sketch.

- For 2-D meshes keep code to draw .svg file

Note about boundary conditions:

`e3prepToFoam` does not define boundary conditions. Instead by labelling the block faces using the names listed above, where `nn` can be the numbers `00`, `01`, ... `10`, these corresponding block faces are grouped into a single patch consisting of all the faces for OpenFOAM. The correct boundary conditions, corresponding to a given patch are then set in OpenFOAM in the `case/0` directory.

Optionally: Once the `job.py` file has been generated, to run the following sequence of commands to view the mesh in paraview.

```
$ e3prep.py --job=job.py --do-svg --openfoam
$ e3post.py --job=job.py --vtk-xml
$ paraview
```

3.1.3 Running `e3prepToFoam.py`

The mesh generation and conversion is performed in two steps using the following commands:

```
$ e3prep.py --job=job.py --do-svg --openfoam ← creates mesh
$ e3prepToFoam.py --job=job.py [--create_0] ← converts mesh to foam
```

Running the second command, generates the foam mesh, overwriting any mesh that already exists within the `case/constant/polymesh` directory. Adding the `--create_0` option additionally writes files `case/0/U` and `case/0/p` corresponding to the boundary face labels defined in the `job.py` file. Any existing `0/U` and `0/p` files are copied to `/U.bak` and `/p.bak`. Other initial and boundary condition files required by the selected solver (e.g. `0/T`) need to be generated manually.

At this stage the foam mesh can be viewed using the command:

```
$ parafoam
```

Perform the following checks:

- `$ checkMesh` This provides a quick overview of the quality of the mesh and whether error occurred in the conversion.
- If patches with names `t0000`, `b0000`, `n0000`, `s0000`, `w0000` or `e0000` (where `0000` can be any 4-digit number) are listed, this indicates that the t-top (b-bottom, n-north, s-south, w-west, e-east) face of the corresponding block was not labelled in the `job.py` file.
- Open mesh in paraview (`$ paraFoam`).
 - If paraview crashes when loading mesh, this typically indicates that not all boundary conditions/initial conditions have been set. (e.g. when performing compressible and turbulent simulation, files `/0/T`, `/0/nut`, etc need to exist for parafoam to work correctly.)
Otherwise de-select all `Volume Fields` and the mesh alone should load without errors.
 - In paraview, check mesh quality, check that external faces have been correctly grouped.

3.1.4 Adjusting the initial fill condition, boundary conditions, gas models, and other items for OpenFOAM

After the mesh conversion the OpenFOAM simulation is initialised, set-up and started using the normal OpenFOAM procedure.

- Dimensions, initial conditions, and boundary condition for each simulated variable are defined in respective file in the `case/0/` directory. Boundary conditions for each grouped face name defined in the `job.py` file (e.g. `OF_wall_00`) must have a corresponding definition.
- Gas and transport properties are set in the `case/constant/` directory.
- Simulation properties are set in the `case/system/` directory.

3.2 Detailed Instructions

Read this section if you are new to python, Eilmer3, or OpenFOAM.

3.2.1 Creating the Directory Structure

`e3prepToFoam.py` requires the correct OpenFOAM File directory structure to function. The easiest way approach to generate this is to find an existing OpenFOAM example (or tutorial) that is similar to the simulation you want to run and to copy the directories. For example to perform an incompressible simulation using `icoFoam` you could do the following:

```
$ of230 ← load the OpenFOAM commands
$ tut ← change to tutorial directory
$ cd incompressible/icoFoam ← change to directory containing the icoFoam example
$ cp -r cavityClipped/. $FOAM_RUN/cavityClipped ← copy the icoFoam example to your
run directory
$ run ← change to run directory
$ cd cavityClipped ← change to your simulation working directory
$ mkdir e3prep ← create the empty e3prep directory
```

At the end of this you should have the following directory and file structure. There may be some extra files in there, but these don't matter

```
case/
  0/
  constant/
  constant/polyMesh
  system/
  system/ controlDict ← file required
  system/ fvSchemes ← file required
  system/ fvSolution ← file required
  e3prep/ ← added for e3prepToFoam
```

3.2.2 Creating your `job.py` file for `e3prep`

Now move to the `e3prep` directory, where you will create your `e3prep` mesh.

```
$ cd e3prep
```

At this point you can either copy an existing `job.py` (advised) and modify this or create your own file. Detailed instructions for the creation of a `job.py` file and examples are available in the Eilmer user guide [1]. The `job.py` file should contain the following parts:

File Header

The file header must include the following lines of code to define the mesh type:

```
gdata.dimensions =X ← X=2 2-D mesh or X=3 for 3-D mesh
gdata.axisymmetric_flag =X ← X=0 for planar 2-D or 3-D or X=1 for 2-D axi-symmetric
```

Block Definition

The core part of the `job.py` file is the definition paths, which then can be turned into a 2-D or 3-D block. The Eilmer user guide provides extensive examples for the generation of points and path segments.

The definition of a typical 2-D block, consisting of north, east, south, west edges defined by the respective paths (e.g. `north_path`) is:

```
blk0 = Block2D( make_patch(north_path, east_path, south_path, west_path),
nni=2, nnj=2, cf_list=[None,]*4)
```

Do not include definition of boundaries at this point.

After completing the block definitions, include the command `identify_block_connections()`

This ensures that the blocks are joint correctly at internal faces.

External Boundary Definition

`e3prep` or `e3prepToFoam` does not set boundary conditions for OpenFOAM. Instead it allows multiple block faces, labelled with one of a list of pre-defined names in `job.py` to be grouped into a single boundary patch. The OpenFOAM boundary conditions are then set for each patch in the `case/0` directory after the generation of the `foam` mesh. Currently the following pre-defined labels are recognised by `e3prepToFoam`.

- `OF_inlet_00, OF_inlet_01, ... OF_inlet_10`
Suitable for inlets
- `OF_outlet_00, OF_outlet_01, ... OF_outlet_10`
Suitable for outlets.
- `OF_wall_00, OF_wall_01, ... OF_wall_10`
Suitable for walls.
- `OF_symmetry_00, OF_symmetry_01, ... OF_symmetry_10`
Suitable for symmetry planes.

To label faces use the command

`blk0.bc_list[EAST] = ExtrapolateOutBC(label="NAME")`, where `EAST` can be any side of the block and `NAME` any of the labels from above. Possible block sides are: `NORTH, EAST, SOUTH, WEST` for 2-D with the addition of `TOP` and `BOTTOM` for 3-D. To group multiple faces, simply give them the same name.

For example add both the north and east face of the block above to the group `OF_wall_03`, use the following lines of code:

```
blk0.bc_list[NORTH] = ExtrapolateOutBC(label="OF_wall_03")
blk0.bc_list[EAST] = ExtrapolateOutBC(label="OF_wall_03")
```

3.2.3 Running `e3prep.py` and `e3prepToFoam.py`

The grid generation and conversion is a 2-stage process.

Step 1: Grid Generation

To generate the mesh run:

```
$ e3prep.py --job=job.py --do=svg --openfoam (from within the e3prep directory)
```

This will create an `e3prep` mesh, stored in `case/e3prep/grid`. If error messages arise in this stage, fix these before proceeding to the next stage.

Optionally to view the mesh, run

```
$ e3post.py --job=job.py --vtk-xml
$ paraview
```

This allows checking of the mesh quality before proceeding to the next step.

Step 2: Grid Conversion

To convert the mesh to the foam format run:

```
$ e3prepToFoam.py --job=job.py (from within the case/e3prep or case directory)
or $ e3prepToFoam.py --job=job.py --create_0 to auto-generate template boundary conditions for p and U.
```

WARNING: the `--create_0` option replaces existing `p` and `U` files and copies the old files to `p.bak` and `U.bak`.

Should running `e3prepToFoam` fail, a range of on-screen error messages with suggested solutions are provided. Or use the details below:

- Error with `mergeMeshes`. Try running `of230` to load OpenFOAM module
This is typically caused if the OpenFOAM environment hasn't been loaded and thus OpenFOAM commands are not recognised. Run `$ of230` or equivalent command to load the OpenFOAM environment.
- WARNING: Not all external boundaries were defined in `e3prep`
The list of block faces (e.g. `b0001` is bottom face of block 1) indicate faces on the mesh external boundaries that have not been given names as described in section 3.2.2.
- WARNING: labels used to define boundary faces do not follow standard `OF_names`
External boundaries were labelled with names that do not match the predefined list. Check for spelling mistakes and/or change names.
- WARNING: Problem during execution of `renumberMesh`.
Fixing this error is optional. This error arises if the files in the `case/0` directory are missing or if the patch labels do not match the labels of the the newly generated mesh. The easiest fix is to re-run with the `--create_0` option.

3.2.4 Checking mesh and boundary faces

To check the mesh and the boundary conditions, execute the `$ checkMesh` command from the `case` directory.

The on screen output provides an overview of the mesh. The list of faces should reflect the external mesh boundaries defined in the `job.py` file. If there are faces listed with names consisting of a the letters `t, b, n, s, w, e` followed by a 4 digit number, this indicates that the corresponding face of the block identified by the number was not labelled using one of the OpenFOAM names. If the faces are internal to the mesh, check that `identify_block_connections()` was included in `job.py`.

Further mesh checking is possible using `paraview` by running the `$ paraFoam` command from the `case` directory.

Before loading the mesh in the `paraview` GUI, de-select all `Volume Fields` (e.g. `p` and `U`). Then

by selecting specific mesh features (E.g. `OF_wall00`) the corresponding faces that form this patch can be visualised.

3.2.5 Setting boundary conditions

Boundary conditions for the OpenFOAM simulation are set in the `case/0/` directory. In most cases it is possible to copy file templates from existing OpenFOAM examples.

Incompressible, laminar (e.g. `icoFoam`)

Running `e3prepToFoam` with the `--create_0` option creates the `p` or `U` file required for an incompressible laminar flow solver, such as `icoFoam`. The resulting files contain template entries for all the patches (group of external faces) that were defined using the `OF_name_nn` labels. To set up the simulation, simply change the boundary condition to the correct type (e.g. change `zeroGradient` to `FixedValue`) and set the correct boundary values as required.

Other solvers, using more than `p` and `U` variable

When using more complex solvers from the OpenFOAM collection initial condition and boundary conditions for additional variables have to be defined. Either modify existing files in `case/0` or create new files for each variable. The files must `boundaryField` definitions for all the external patches grouped using the `OF_name_nn` names (and `FrontBack`, `Front`, `Back`, `Centreline` if present). The `p` or `U` files created using `--create_0` can be used as templates.

3.2.6 Adjusting gas models and other items for OpenFOAM

After the mesh conversion the OpenFOAM simulation is set-up and started using the normal OpenFOAM procedure.

- Gas and transport properties are set in the `case/constant/` directory.
- Simulation properties are set in the `case/system/` directory.

3.2.7 Running the simulation

At this point you should be ready to run you OpenFOAM simulation.

4 Theory behind code

The following sections describe in more detail the theory and steps of the code. As an overview, the mesh conversion by `e3prpToFoam` is performed by the following steps:

1. check that suitable directory structure exists
2. execute `e3post.py` to write individual foam meshes, corresponding to each block generated by `e3prep`. Different approaches are used for 3-D, 2-D and axi-symmetric meshes. See 4.1 for details.
3. use OpenFOAM `mergeMesh` utility to combine individual blocks into single mesh
4. (optional) For axisymmetric meshes, remove zero area faces along centreline
5. use OpenFOAM `stichMesh` utility to link blocks and remove internal faces
6. (optional) For axisymmetric meshes, automatically group all faces that fall on Centreline
7. (optional) For 2-D meshes automatically group top and bottom faces in group `FrontBack` with type `empty`
8. (optional) For axisymmetric meshes, automatically group top faces in group `Front` and bottom faces in group `Back` with type `wedge` and faces along x-axis in group `Centreline` with type `empty`.
9. group external faces into patches according to the labels: `OF_inlet_nn`, `OF_outlet_nn`, `OF_wall_nn`, `OF_symmetry_nn`, where `nn` can be numbers 00, 01, ... 10. See 4.2 for details.
10. (optional) if `--create_0` option is used, create `/0/p` and `/0/p` files. See 4.3 for details.
11. use OpenFOAM `renumberMesh` utility to reorder faces and cells for numerical efficiency.

4.1 e3prep → foam block conversion

The conversion of individual `e3prep` blocks to corresponding foam meshes is carried out by invoking the `--OpenFOAM` option of `e3post.py`. The corresponding code is shown in section 7.1. Depending on mesh type the following procedures are applied to convert the mesh.

4.1.1 3-D meshes

Eilmer uses a body-fitting structured mesh. A simple $3 \times 1 \times 2$ grid is shown in Figure 1, which results in 24 vertices (labelled from 0 to 23) and 6 cells (labelled from 0 to 5). This figure is used to explain how the Eilmer mesh is converted to OpenFOAM format. The OpenFOAM foam mesh contains 5 files, namely, `points`, `faces`, `owner`, `neighbour`, and `boundary`, which are defined as:

- `points`

A list of vectors describing the cell vertices, where the first vector in the list represents vertex 0, the second vector represents vertex 1, etc. Since a structured mesh is used in Eilmer, the `points` file is created by sequentially going through the mesh, first in the i direction, next the j direction and then k direction and writing a corresponding file.

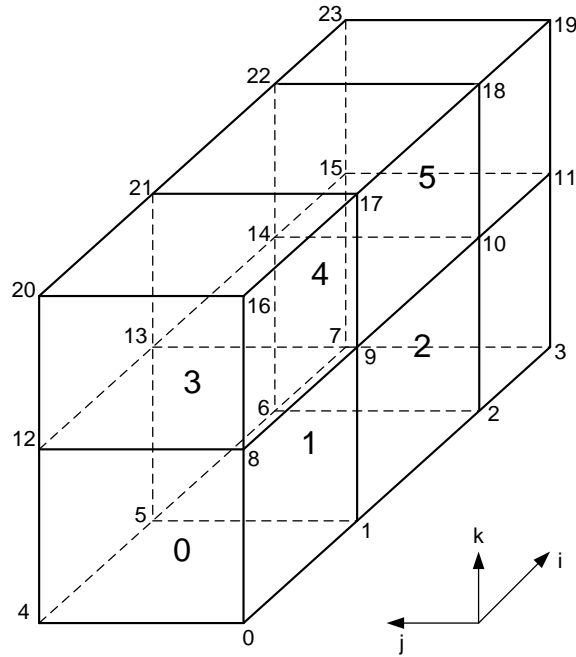


Figure 1: Simple structured mesh in Eilmer3.

- **faces**

A face is an ordered list of points, where a point is referred to by its label. The ordering of point labels in a face is such that each two neighbouring points are connected by an edge. Faces are compiled into a list and each face is referred to by its label, representing its position in the list. The direction of the face normal vector is defined by the right-hand rule. There are two types of faces within the `foam` format which must be treated differently

1. **Internal face:** All faces that connect two cells (and it can never be more than two). The order of points is selected such, that each face normal points in the positive i , j or k direction. For example, for the internal face between block 0 and 1 (created by the four points: 1, 5, 13 and 9) is defined as (1 5 13 9) to ensure that the face normal vector points in the positive i direction.
2. **Boundary face:** All faces that belonging to one cell only, since they coincide with the boundary of the domain. A boundary face is therefore addressed by one cell (only) and a boundary patch. The ordering of the point labels is such that the face normal points outside of the computational domain. For example, for the boundary face which is created by the points: 0 1 9 8, to make sure that the normal vector of this boundary face points outside of computational domain, the order of points in the label is (0 1 9 8).

Using the above convention all the face labels are written to the `faces` file.

- **owner and neighbour**

owner and neighbour files define which cell owns and neighbours each face, respectively.

From the definition, owners exist for both internal and boundary faces, but neighbours only exist for internal face. In Eilmer, cells are numbered in order of i direction first, j direction next and then k direction. The owner of the specific internal face is defined as the starting cell based on the right hand rule. For example, for the internal face of (1 5 13 9), its normal vector points from cell 0 to cell 1, in this case, cell 0 is the owner while cell 1 neighbour and corresponding entries are written to the `owner` and `neighbour` files. For the boundary face of (0 1 9 8), cell 0 is the owner, and there is no neighbour. Here an entry is only added to the `owner` file.

- **boundary**

A list of patches, containing a dictionary entry for each patch. The number of faces and the starting face for this boundary is provided. Corresponding entries are created for the six sides of the structured mesh.

These are the typical steps for 3-D mesh conversion from Eilmer3 to OpenFOAM, however, what if 2-D mesh is generated in Eilmer, how do we convert it into OpenFOAM format since only 3-D mesh is accepted in OpenFOAM, this will be discussed below.

4.1.2 2-D meshes

Eilmer 2-D meshes are created in the west, east, north, south plane (corresponding to the x-y plane). To allow conversion of 2-D meshes for simulations in OpenFOAM, which requires a 3-D mesh, the mesh is first extruded in the downwards direction by 1×10^{-3} m to form a 1 cell deep 3-D mesh. This mesh is then converted as outlined above.

4.1.3 2-D axi-symmetric meshes

Eilmer 2-D axi-symmetric meshes are created in the west, east, north, south plane (corresponding to the x-y plane). To generate 3-D axi-symmetric mesh as required by OpenFOAM, the Eilmer mesh is then rotated by ± 0.04 rad (2.3°) about the x-axis, to create a wedge shaped, 1 cell wide mesh centred on the x-y plane. This mesh is then converted as outlined above.

4.2 Grouping of external faces

`e3prep` or `e3prepToFoam` does not set boundary conditions. Instead it allows multiple block faces, labelled with one of a list of pre-defined names in `job.py` to be grouped into a single boundary patch. In OpenFOAM boundary conditions are then set for each patch. Currently the following pre-defined labels are recognised by `e3prepToFoam`. Depending on type, different patch types are set.

- `OF_inlet_00, OF_inlet_01, ... OF_inlet_10`
Patch is defined with type `patch`.
- `OF_outlet_00, OF_outlet_01, ... OF_outlet_10`
Patch is defined with type `patch`.
- `OF_wall_00, OF_wall_01, ... OF_wall_10`
Patch is defined with type `wall`.
- `OF_symmetry_00, OF_symmetry_01, ... OF_symmetry_10`
Patch is defined with type `symmetry`.

The type of the individual patches can be changed retrospectively by editing the `case/constan/polymesh/boundary` file. The names of the individual patches can be changed by editing `case/constan/polymesh/boundary` and the respective `boundaryField` names in the files within the `case/0/` directory.

4.3 Creation of boundary conditions (`--create_0` option)

If the `$ e3prepToFoam.py --job=job.py --create_0` is executed, in addition to converting the mesh, the initial and boundary condition files for pressure (`p`) and velocity (`U`) are created in the `case/0` directory. These files contain dimensions, initial conditions, pre-populated boundary condition entries for all the labeled external faces, and boundary condition entries for empty front and rear faces in 2-D meshes.

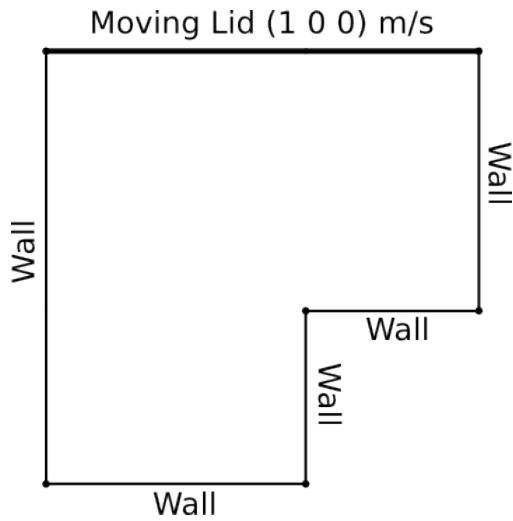
The `p` file is initialised as:

- `dimensions` `[0 2 -2 0 0 0 0]` ($\text{m}^2 \text{s}^{-2}$) ← needs to be altered for compressible
- `internalField` `uniform 0` ← needs to be altered for compressible
- `boundaryField` template generated based on label type
 - `OF_inlet, OF_outlet, OF_wall` → `zeroGradient`
 - `OF_symmetry` → `symmetry`
 - `FrontBack` (automatically set for 2-D) → `empty`
 - `Front, Back` (automatically set for axi-symmetric) → `wedge`
 - `Centreline` (automatically set for axi-symmetric) → `empty`

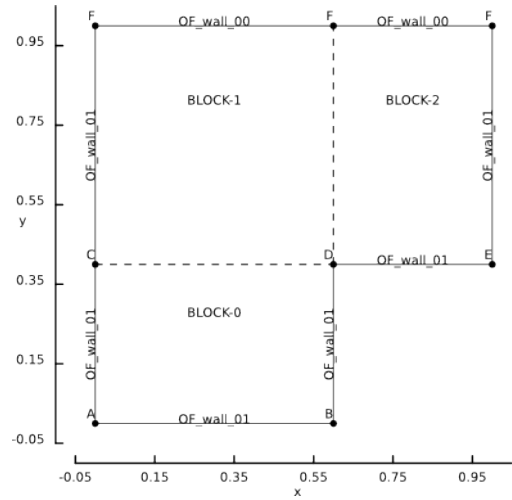
The `U` file is initialised as:

- `dimensions` `[0 1 -1 0 0 0 0]` (m s^{-1})
- `internalField` `uniform (0 0 0)`
- `boundaryField` template generated based on label type
 - `OF_inlet , OF_wall` `\rightarrow \verbfixedValue'`
 - `OF_outlet'` → `zeroGradient`
 - `OF_symmetry` → `symmetry`
 - `FrontBack` (automatically set for 2-D) → `empty`
 - `Front, Back` (automatically set for axi-symmetric) → `wedge`
 - `Centreline` (automatically set for axi-symmetric) → `empty`

If the extra variables are solved, additional files need to be created in the `case/0/` directory. The `p` and `U` files can be used as templates.



(a) Domain used for 2-D example and corresponding boundary conditions.



(b) Blocking strategy for mesh and labels as applied by e3prep.

Figure 2: Domain, boundary conditions, blocking strategy, and boundary labels for 2-D example.

5 Examples

5.1 2-D mesh

This example is based on the *clippedCavity* example (Section 2.1.9 and 2.1.10) from the OpenFOAM Manual [2]. The first step is to copy the existing OpenFOAM `cavityClipped` example and to create the `e3prep` directory (see section 3.2.1).

Meshing

The domain is a rectangle with a quarter missing as shown in Figure 2(a). All lower walls are stationary and the top wall is moving to the right with a velocity of 1 m/s. For meshing using `e3prep` the mesh is split into 3 blocks as shown in Figure 2(b). The external walls are split into 2 groups:

- OF_wall_00 for the moving lid
- OF_wall_01 for the remaining walls.

The corresponding grid generation file `cavityClipped.py` is

```

1 # cavityClipped.py
2 # Simple job-specification file for e3prep.py and e3prepToFoam.py
3 # IJ, 27-Apr-2015
4
5 job_title = "cavityClipped example for e3prepToFoam."
6 print job_title
7
8 # We can set individual attributes of the global data object.
9 gdata.dimensions = 2
10 gdata.title = job_title
11 gdata.axisymmetric_flag = 0
12
13 # Set up 3 rectangles in the (x,y)-plane by first defining

```

```

14 # the corner nodes, then the lines between those corners.
15 a = Node(0.0, 0.0, label="A")
16 b = Node(0.6, 0.0, label="B")
17 c = Node(0.0, 0.4, label="C")
18 d = Node(0.6, 0.4, label="D")
19 e = Node(1.0, 0.4, label="E")
20 f = Node(0.0, 1.0, label="F")
21 g = Node(0.6, 1.0, label="F")
22 h = Node(1.0, 1.0, label="F")
23
24 # Define Lines connecting blocks
25 ab = Line(a, b) # horizontal lines (lowest level)
26 cd = Line(c, d); de = Line(d, e) # horizontal lines (mid level)
27 fg = Line(f, g); gh = Line(g, h) # horizontal lines (top level)
28 ac = Line(a, c); cf = Line(c, f) # vertical lines (left)
29 bd = Line(b, d); dg = Line(d, g) # vertical lines (mid)
30 eh = Line(e, h) # vertical lines (right)
31
32 # Define the blocks, with particular discretisation.
33 nx0 = 12; nx1 = 8; ny0 = 8; ny1 = 12
34 blk_0 = Block2D(make_patch(cd, bd, ab, ac), nni=nx0, nnj=ny0,
35 label="BLOCK-0")
36 blk_1 = Block2D(make_patch(fg, dg, cd, cf), nni=nx0, nnj=ny1,
37 label="BLOCK-1")
38 blk_2 = Block2D(make_patch(gh, eh, de, dg), nni=nx1, nnj=ny1,
39 label="BLOCK-2")
40
41 # Command to identify internal face connections
42 identify_block_connections()
43
44 # Set boundary conditions.
45 blk_1.bc_list [WEST] = ExtrapolateOutBC(label="OF_wall_01") # labelling wall B/C
46 blk_0.bc_list [WEST] = ExtrapolateOutBC(label="OF_wall_01")
47 blk_0.bc_list [SOUTH] = ExtrapolateOutBC(label="OF_wall_01")
48 blk_0.bc_list [EAST] = ExtrapolateOutBC(label="OF_wall_01")
49 blk_2.bc_list [SOUTH] = ExtrapolateOutBC(label="OF_wall_01")
50 blk_2.bc_list [EAST] = ExtrapolateOutBC(label="OF_wall_01")
51 blk_1.bc_list [NORTH] = ExtrapolateOutBC(label="OF_wall_00")
52 blk_2.bc_list [NORTH] = ExtrapolateOutBC(label="OF_wall_00")
53
54 # command to write BC labels
55 sketch.prefer_bc_labels_on_faces()
56
57 # plot .svg
58 sketch.xaxis(-0.05, 1.05, 0.2, -0.05)
59 sketch.yaxis(-0.05, 1.05, 0.2, -0.05)
60 sketch.window(-0.05, -0.05, 1.05, 1.05, 0.05, 0.05, 0.17, 0.17)

```

The grid generation and grid conversion is performed using the commands (from the `e3prep` directory):

```
e3prep.py --job=cavity3.py --do=svg --openfoam
```

```
e3prepToFoam.py --job=cavity3.py --create_0
```

Running `checkMesh` provides following summary of the grid:

```
Create polyMesh for time = 0
```

```
Time = 0
```

```
Mesh stats
```

```

points:          754
internal points: 0
faces:          1384
internal faces: 632
cells:          336
faces per cell: 6
boundary patches: 3
point zones:    0
face zones:     2
cell zones:     0

```

Overall number of cells of each type:

```

hexahedra:      336
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
FrontBack	672	754	ok (non-closed singly connected)
OF_wall_00	20	42	ok (non-closed singly connected)
OF_wall_01	60	122	ok (non-closed singly connected)

Checking geometry...

```

Overall domain bounding box (0 0 0) (1 1 0.001)
Mesh (non-empty, non-wedge) directions (1 1 0)
Mesh (non-empty) directions (1 1 0)
All edges aligned with or perpendicular to non-empty directions.
Boundary openness (1.36813e-19 -2.01195e-19 -3.88871e-17) OK.
Max cell openness = 8.67362e-17 OK.
Max aspect ratio = 1 OK.
Minimum face area = 5e-05. Maximum face area = 0.0025. Face area magnitudes
OK.
Min volume = 2.5e-06. Max volume = 2.5e-06. Total volume = 0.00084. Cell
volumes OK.
Mesh non-orthogonality Max: 0 average: 0
Non-orthogonality check OK.
Face pyramids OK.
Max skewness = 1e-09 OK.
Coupled point location match (average 0) OK.

```

Mesh OK.

End

As can be seen the external faces of the mesh are defined by 3 patches:

FrontBack → front and back face of single cell deep 3-D mesh used by OpenFOAM for 2-D simulations

OF_wall_00 → moving lid

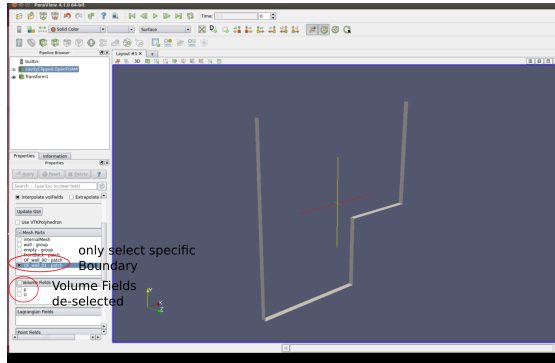


Figure 3: Parts of the mesh visualised in Paraview.

OF_wall_01 → stationary walls at left, bottom, right.

Figure 3 shows visualisation of various mesh components in paraview.

Boundary Conditions

Next the boundary conditions are set in the p and U files. Templates with the correct patch names have already been create by running the `--create_0` option. For the p file all faces are set to `zeroGradient`. For the U file the moving lid is set to `(1 0 0)` and the stationary walls are set to `(0 0 0)`. The corresponding files are:

```

1 /*-----* C++ -*-----*\
2 |=====|
3 | \ \ \ \ \ | F i e l d | OpenFOAM: The Open Source CFD Toolbox
4 | / / / / / | O p e r a t i o n | Version: 2.3.0
5 | / / / / / | A n d | Web: www.OpenFOAM.org
6 | \ \ \ \ \ | M a n i p u l a t i o n |
7 \*-----*\
8 FoamFile
9 {
10     version      2.0;
11     format       ascii;
12     class        volScalarField;
13     location     "0";
14     object       p;
15 }
16 // * * * * *
17
18 dimensions      [0 2 -2 0 0 0 0];
19
20 internalField   uniform 0;
21
22 boundaryField
23 {
24     FrontBack
25     {
26         type      empty;
27     }
28     OF_wall_00
29     {
30         type      zeroGradient;
31     }

```

```

32     OF_wall_01
33     {
34         type          zeroGradient;
35     }
36 }
37
38
39 // *****

```

```

1 /*----- C++ -----*/
2 |
3 | //        F i e l d       | OpenFOAM: The Open Source CFD Toolbox
4 | //        O p e r a t i o n   | Version:   2.3.0
5 | //        A n d               | Web:      www.OpenFOAM.org
6 | //        M a n i p u l a t i o n |
7 */
8 FoamFile
9 {
10    version     2.0;
11    format      ascii;
12    class       volVectorField;
13    location    "0";
14    object      U;
15 }
16 // *****

```

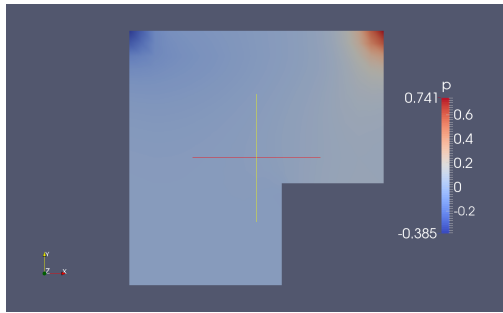
```

17
18 dimensions          [0 1 -1 0 0 0 0];
19
20 internalField       uniform (0 0 0);
21
22 boundaryField
23 {
24     FrontBack
25     {
26         type          empty;
27     }
28     OF_wall_00
29     {
30         type          fixedValue;
31         value         uniform (1 0 0);
32     }
33     OF_wall_01
34     {
35         type          fixedValue;
36         value         uniform (0 0 0);
37     }
38 }
39
40
41 // *****

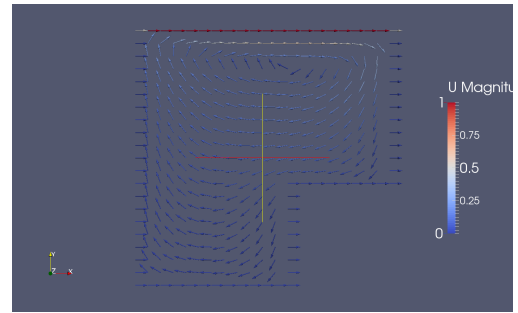
```

Running the simulation

The simulation can now be run using the `$ icoFoam` command. The pressure and velocity field obtained from the simulation are shown in Figure 4



(a) Pressure Field.



(b) Velocity Vectors coloured by magnitude.

Figure 4: Solution of the clippedCavity 2-D example.

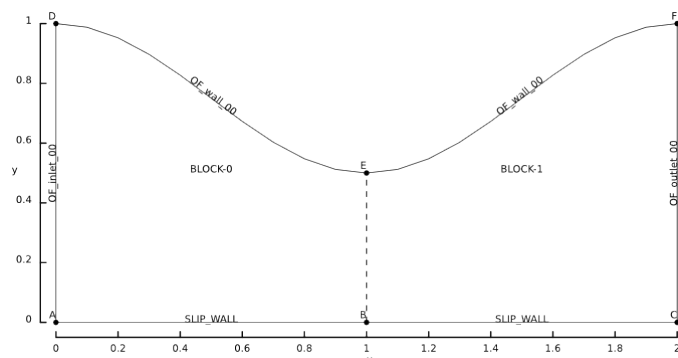


Figure 5: Fluid domain and block structure for 2-D axi-symmetric convergent-divergent nozzle

5.2 2-D axi-symmetric mesh

This example is based on a convergent divergent nozzle with sinusoidal shape as shown in Figure 5. The nozzle has a length of $2 \times L_0 = 2$ m and starting radius of $R_0 = 1$ m. The nozzle contour is defined as $R = R_0 (0.75 + 0.25 \cos(x\pi))$

Meshing

The external walls are split into 3 groups:

OF_wall_00 for the outer radius

OF_inlet_00 for inlet (left end)

OF_outlet_00 for inlet (right end)

The corresponding grid generation file `condiv.py` is

```

1 # condiv.py
2 # Simple job-specification file for e3prep.py and e3prepToFoam.py
3 # IJ, 27-Apr-2015
4
5 job_title = "condiv a Convergent-Divergent Nozzle example for e3prepToFoam."
6 print job_title
7
8 # We can set individual attributes of the global data object.
9 gdata.dimensions = 2
10 gdata.title = job_title
11 gdata.axisymmetric_flag = 1 # set equal 1 as axi-symmetric
12
13 # Define variables for parametric geometry definition
14 R0 = 1.
15 L0 = 1.
16 # Set up the corner nodes, then the lines between those corners.
17 a = Node(0.0 , 0.0, label="A")
18 b = Node(L0 , 0.0, label="B")
19 c = Node(2. *L0 , 0.0, label="C")
20 d = Node(0.0 , R0, label="D")
21 e = Node(L0 , 0.5*R0, label="E")
22 f = Node(2.0*L0 , R0, label="F")
23
24 # Define straight Lines
25 ab = Line(a, b); bc = Line(b, c) # Centreline
26 ad = Line(a, d); be = Line(b, e); cf = Line(c, f) # vertical lines
27

```

```

28 # use PyFunctionPath to define nozzle contour
29 import numpy as np
30 def path0(t):
31     global R0
32     global L0
33     x = t * L0
34     y = R0 * (0.75 + 0.25 * np.cos(t * np.pi))
35     return x,y,0.
36
37 def path1(t):
38     global R0
39     global L0
40     x = (1+t) * L0
41     y = R0 * (0.75 + 0.25 * np.cos((t+1.) * np.pi))
42     return x,y,0.
43
44 # Define Nozzle Contours
45 de = PyFunctionPath(path0); ef = PyFunctionPath(path1)
46
47 # Define the blocks, with particular discretisation.
48 nx0 = 10; nx1 = 10; ny0 = 10
49 blk_0 = Block2D(make_patch(de, be, ab, ad), nni=nx0, nnj=ny0,
50                 fill_condition = initial, label="BLOCK-0")
51 blk_1 = Block2D(make_patch(ef, cf, bc, be), nni=nx1, nnj=ny0,
52                 fill_condition = initial, label="BLOCK-1")
53
54 # Command to identify internal face connections
55 identify_block_connections()
56
57 # Set boundary conditions.
58 blk_0.bc_list [WEST] = ExtrapolateOutBC(label="OF_inlet_00") # labelling wall B/C
59 blk_0.bc_list [NORTH] = ExtrapolateOutBC(label="OF_wall_00")
60 blk_1.bc_list [NORTH] = ExtrapolateOutBC(label="OF_wall_00")
61 blk_1.bc_list [EAST] = ExtrapolateOutBC(label="OF_outlet_00")
62
63 # command to write BC labels
64 sketch.prefer_bc_labels_on_faces()
65
66 # plot .svg
67 sketch.xaxis(0.0, 2.0, 0.2, -0.05)
68 sketch.yaxis(0.0, 1.0, 0.2, -0.05)
69 sketch.window(-0.05, -0.05, 1.05, 2.05, 0.05, 0.05, 0.17, 0.27)

```

The grid generation and grid conversion is performed using the commands (from the `e3prep` directory):

```

e3prep.py --job=condiv.py --do=svg
e3prepToFoam.py --job=condiv.py --create_0

```

Running `checkMesh` provides following summary of the grid faces/patches:

```

Checking patch topology for multiply connected surfaces...
Patch      Faces    Points  Surface topology
Back       200      231     ok (non-closed singly connected)
Front      200      231     ok (non-closed singly connected)
OF_inlet_00  10      21      ok (non-closed singly connected)
OF_outlet_00  10      21      ok (non-closed singly connected)
OF_wall_00  20      42      ok (non-closed singly connected)

```

As can be seen from the output, a `Front` and `Back` patch has been added.

Boundary Conditions

Next the boundary conditions are set in the `p` and `U` files. Templates with the correct patch names have already been create by running the `--create_0` option. For the `p` file all faces are set to `zeroGradient`. For the `U` file the inlet is set to `uniform (1 0 0)`, the stationary walls are set to `uniform (0 0 0)` and the outlet is set to `zeroGradient`. The corresponding files are:

```

/*-----* C++ *-----*\
|=====|           | OpenFOAM: The Open Source CFD Toolbox |
| \\\   / F i e l d   | Version: 2.3.0                       |
| \\\   / O p e r a t i o n   | Web: www.OpenFOAM.org          |
| \\\   / A n d           |                                     |
| \\\   / M a n i p u l a t i o n   |                                     |
|=====|           |                                     |
\*-----*

FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       p;
}
// *****

dimensions      [0 2 -2 0 0 0 0];

internalField    uniform 0;

boundaryField
{
    Back
    {
        type      wedge;
    }
    Front
    {
        type      wedge;
    }
    OF_inlet_00
    {
        type      zeroGradient;
    }
    OF_outlet_00
    {
        type      zeroGradient;
    }
    OF_wall_00
    {
        type      zeroGradient;
    }
}

// *****

/*-----* C++ *-----*\
|=====|           | OpenFOAM: The Open Source CFD Toolbox |
| \\\   / F i e l d   |                                     |

```

```

|  \ \  /  /  O peration   |  Version:  2.3.0   |
|  \ \  /  /  A nd         |  Web:       www.OpenFOAM.org  |
|  \ \  /  /  M anipulation |  |
|*-----*|
FoamFile
{
  version      2.0;
  format       ascii;
  class        volVectorField;
  location     "0";
  object       U;
}
// *****

dimensions    [0 1 -1 0 0 0];
internalField uniform (0 0 0);

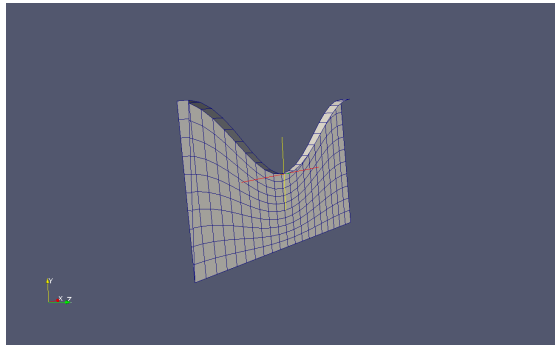
boundaryField
{
  Back
  {
    type       wedge;
  }
  Front
  {
    type       wedge;
  }
  OF_inlet_00
  {
    type       fixedValue;
    value      uniform (1 0 0);
  }
  OF_outlet_00
  {
    type       zeroGradient;
  }
  OF_wall_00
  {
    type       fixedValue;
    value      uniform (0 0 0);
  }
}

// *****

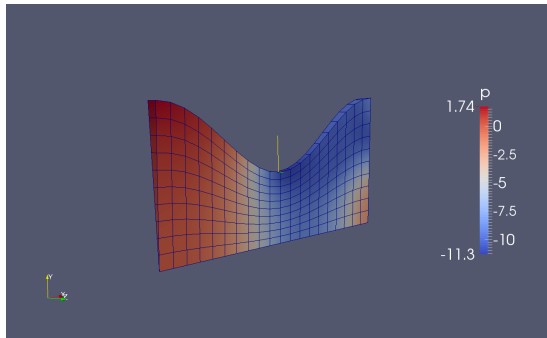
```

Mesh and Results

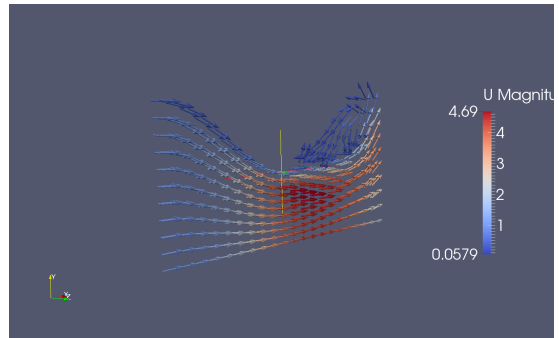
The resulting mesh and solution, as viewed in paraview is shown in Figure 6.



(a) Mesh showing inlet, back wedge face, and outer wall.



(b) Pressure Field.



(c) Velocity Vectors coloured by magnitude.

Figure 6: Mesh and resulting Flow Field of the Convergent-Divergent Nozzle 2-D Axisymmetric example.

5.3 3-D mesh

This example is the heat conduction analysis through a thrust disk. This disk has a inner radius of 25.4mm and an outer radius of 50.8mm, and the fixed temperature is set to west and east boundary patch, while others are regarded as adiabatic wall.

5.3.1 Meshing

The external wall boundaries are splited into 3 groups:

OF_wall_00 for the adiabatic wall

OF_wall_01 for the west boundary patch

OF_wall_02 for the east boundary patch

The corresponding grid generation file `rotor.py` is

```
# Mesh generation of rotor
# for OpenFOAM simulation

from math import pi, sin, cos, sqrt
import numpy

# Control Parameter
gdata.dimensions = 3
gdata.title = "Foil thrust bearing—Rotor"

# Define Geometry
r_omega = 2*pi*21000.0/60.0
theta0 = 0.0
theta1 = 15.0/180.0*pi
theta2 = 60.0/180.0*pi

r1 = 0.0510/2.0    # inner radius
r2 = 0.1016/2.0    # outer radius
h1 = 0.0
h2 = 3.0e-3

# Define parametric volume
def makeSimpleBox(ini_angular1, ini_angular2, ini_h1, ini_h2, r_1, r_2):

    inih1 = ini_h1
    inih2 = ini_h2
    ini1 = ini_angular1
    ini2 = ini_angular2
    center_b = Node(0.0, 0.0, inih1)
    center_t = Node(0.0, 0.0, inih2)

    up0 = Vector(r_1*cos(ini1), r_1*sin(ini1), inih1)
    up1 = Vector(r_2*cos(ini1), r_2*sin(ini1), inih1)
    up2 = Vector(r_2*cos(ini2), r_2*sin(ini2), inih1)
    up3 = Vector(r_1*cos(ini2), r_1*sin(ini2), inih1)
    up4 = Vector(r_1*cos(ini1), r_1*sin(ini1), inih2)
    up5 = Vector(r_2*cos(ini1), r_2*sin(ini1), inih2)
    up6 = Vector(r_2*cos(ini2), r_2*sin(ini2), inih2)
    up7 = Vector(r_1*cos(ini2), r_1*sin(ini2), inih2)

    up01 = Line(up0, up1)
    up12 = Arc(up1, up2, center_b)
    up32 = Line(up3, up2)
    up03 = Arc(up0, up3, center_b)
    up45 = Line(up4, up5)
```

```

up56 = Arc(up5, up6, center_t)
up76 = Line(up7, up6)
up47 = Arc(up4, up7, center_t)
up04 = Line(up0, up4)
up15 = Line(up1, up5)
up26 = Line(up2, up6)
up37 = Line(up3, up7)

return WireFrameVolume(up01, up12, up32, up03, up45, up56,
                        up76, up47, up04, up15, up26, up37)

# set cluster functions
c_x = RobertsClusterFunction(1,1,1.0)
c_y = RobertsClusterFunction(1,1,1.0)
c_z = RobertsClusterFunction(1,1,1.0)

## block 0
pvolume0 = makeSimpleBox(theta0, theta2, h1, h2, r1, r2)
cflist0 = [c_x, c_y, c_x, c_y, c_x, c_y, c_x, c_y, c_z, c_z, c_z, c_z];
nx0 = 48 ; ny0= 48; nz0= 10;
blk0= Block3D(label="rotor-0", nni=nx0, nnj=ny0, nnk=nz0,
              parametric_volume=pvolume0,
              cf_list=cflist0)

# label Boundary Conditions
blk0.bc_list[NORTH] = ExtrapolateOutBC(label='OF_wall_00')
blk0.bc_list[EAST] = ExtrapolateOutBC(label='OF_wall_01')
blk0.bc_list[SOUTH] = ExtrapolateOutBC(label='OF_wall_00')
blk0.bc_list[WEST] = ExtrapolateOutBC(label='OF_wall_02')
blk0.bc_list[TOP] = ExtrapolateOutBC(label='OF_wall_00')
blk0.bc_list[BOTTOM] = ExtrapolateOutBC(label='OF_wall_00')

sketch.prefer_bc_labels_on_faces()

identify_block_connections()

```

The command generating mesh file is shown in `prep-simulation.sh`, please note that this is a 3-D mesh case, the option of `--do-svg` is only working for 2-D using `e3prep.py`. Also, this 3-D mesh is for the heat conduction analysis, the temperature boundary field is only needed in this case, so the option of `--create_0` using `e3prepToFoam.py` is ignored here.

```

#!/bin/bash -l

e3prep.py --job=rotor --openfoam

e3prepToFoam.py --job=rotor

```

5.3.2 Boundary Conditions

Next the boundary conditions are set in the T file. Since the `--create_0` option can only generate p and U for pressure and velocity, respectively, the temperature T file needs to be added manually, which is shown below. For this T file, the adiabatic wall are set to `zeroGradient`, the west boundary patch is set to a fixed temperature of 300 K, while the east boundary patch is set to a fixed temperature of 340 K.

```

/*-----* C++ *-----*/
|=====|
| \ \ \ \ | F i e l d | OpenFOAM: The Open Source CFD Toolbox
| \ \ \ \ | O p e r a t i o n | Version: 2.2.2
| \ \ \ \ | A n d | Web: www.OpenFOAM.org
| \ \ \ \ | M a n i p u l a t i o n |
|=====|
FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       T;
}
// *****

dimensions      [0 0 0 1 0 0 0];

internalField   uniform 300;

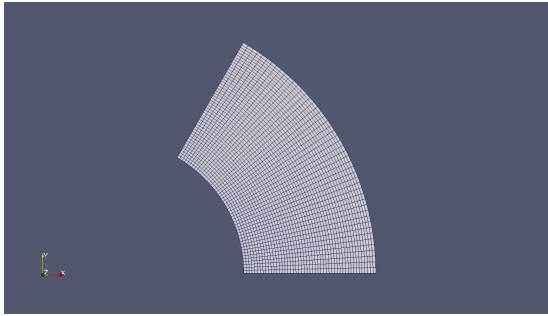
boundaryField
{
    OF_wall_00
    {
        type      zeroGradient;
    }
    OF_wall_01
    {
        type      fixedValue;
        value     uniform 340;
    }
    OF_wall_02
    {
        type      fixedValue;
        value     uniform 300;
    }
}

// *****

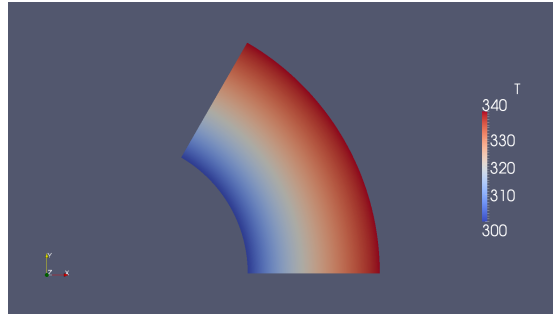
```

5.3.3 Mesh and Results

For the heat conduction analysis of this thrust disk, the solver called `laplacianFoam` is used, and the resulting mesh and solution, as viewed in paraview is shown in Figure 7.



(a) Mesh for thrust disk.



(b) Temperature field for thrust disk.

Figure 7: Mesh and resulting temperature field of the thrust disk: 3-D example.

6 References

References

- [1] P.A. Jacobs, R.J. Gollan, D.F. Potter, 2014, *The Eilmer3 Code: User Guide and Example-Book*, Mechanical Engineering Report 2014/05, The University of Queensland
- [2] OpenFOAM The Open Source CFD Toolbox, *Userguide*, Version 2.3.1, 3r December 2014, www.foam.sourceforge.net/docs/Guides-a4/UserGuide.pdf OpenFOAM Foundation
- [3] CFCFD, *The Compressible Flow Project* <http://cfcfd.mechmining.uq.edu.au> The University of Queensland

7 Appendix

7.1 Addition to e3post.py

Following function has been added to `e3post.py` to enable conversion of `e3prep` blocks to corresponding individual foam meshes.

Additions to `e3_post.py`. This part calls the `write_OpenFOAM_files()` function to perform the mesh conversion.

```
if uoDict.has_key("—OpenFOAM"):  
    configFileName = rootName + ".config"  
    cp = ConfigParser.ConfigParser()  
    cp.read(configFileName)  
    axisymmetric_flag = cp.get("global_data", "axisymmetric_flag")  
    if verbosity_level > 0:  
        print "writing OpenFOAM grid, 2-D or 3-D"  
        if axisymmetric_flag == 1:  
            print "creating axisymmetric OpenFOAM grid"  
            grid, flow, dimensions = read_all_blocks(rootName, nblock,  
            tindx, zipFiles, movingGrid)  
            add_auxiliary_variables(nblock, flow, uoDict, omegaz,  
            aux_var_names, compute_vars)  
            write_OpenFOAM_files(rootName, nblock, grid, flow,  
            axisymmetric_flag)
```

Additions to `e3_flow.py`. This part converts the `e3prep` block into an unstructured foam mesh and writes the corresponding mesh files.

```
#
```

```
# OpenFOAM-related functions by Jason (Kan) Qin, July 2014.
```

```
def write_general_OpenFOAM_header(fp):  
    fp.write("/*-----* C++  
    *-----*\\n")
```

```

fp.write(" | ===== |
                                         |\n")
fp.write(" | \\      / F ield      | OpenFOAM: The Open Source CFD
  Toolbox      |\n")
fp.write(" | \\      / O peration   | Version:  2.2.2
                                         |\n")
fp.write(" |  \\ /   A nd           | Web:      www.OpenFOAM.org
                                         |\n")
fp.write(" |  \\/    M anipulation  | This file generated by e3post.
  py          |\n")
fp.write("
  \*-----*\n")
fp.write("FoamFile\n")
fp.write("{\n")
fp.write("  version      2.0;\n")
fp.write("  format       ascii;\n")
return

def write_general_OpenFOAM_bottom(fp):
fp.write(")\n")
fp.write(")\n")
fp.write("//
  *****
  //\n")
return

def write_OpenFOAM_unstructured_file(fp0, fp1, fp2, fp3, fp4, jb, grid,
flow, axi_flag):
"""
Write the OpenFOAM format data from a single block
as an unstructured grid of finite-volume cells.
Since OpenFOAM only accept 3D grid, this tool can be operated in the
following 3 modes:
a) 3D grid from Eilmer —> 3D foam grid
b) 2D grid from Eilmer —> 3D foam grid with width 0.001 meter
c) 2D axi-symmetric grid from Eilmer —> 3D foam grid with angle
+/-0.04 radians

:param fp0: reference to the file object: points
:param fp1: reference to the file object: faces
:param fp2: reference to the file object: owner
:param fp3: reference to the file object: neighbour
:param fp4: reference to the file object: boundary
:param grid: single-block grid of vertices
:param flow: single-block of cell-centre flow data
:param axi_flag: integr
"""
nio = grid.ni; njo = grid.nj; nko = grid.nk
nif = flow.ni; njf = flow.nj; nkf = flow.nk
two_D = (nko == 1)
if two_D:
    nko = 2

```

```

        z_set = [0, 0.001]
SumOfPoints = nio * njo * nko
SumOfCells = nif * njf * nkf
#
# cells
cells_number = 0
cells_id = {}
for k in range(nko-1):
    for j in range(njo-1):
        for i in range(nio-1):
            cells_id [(i,j,k)] = cells_number
            cells_number += 1
# points
vtxs_number = 0
vtxs_id = {}
for k in range(nko):
    for j in range(njo):
        for i in range(nio):
            vtxs_id [(i,j,k)] = vtxs_number
            vtxs_number += 1
# faces
face_number = 0
# searching internal faces at k-direction
if two_D == False:
    for k in range(1,nko-1):
        for j in range(njo-1):
            for i in range(nio-1):
                face_number += 1
# searching internal faces at j-direction
for j in range(1,njo-1):
    for k in range(nko-1):
        for i in range(nio-1):
            face_number += 1
# searching internal faces at i-direction
for i in range(1,nio-1):
    for k in range(nko-1):
        for j in range(njo-1):
            face_number += 1

SumOfInternalFaces = face_number
# searching boundary faces at NORTH faces
NF_start = face_number
for i in range(nio-1):
    for k in range(nko-1):
        face_number += 1
NF_end = face_number
SumOfNF = NF_end - NF_start
# searching boundary faces at WEST faces
WF_start = face_number
for k in range(nko-1):
    for j in range(njo-1):
        face_number += 1
WF_end = face_number

```



```

# searching internal faces at i-direction
for i in range(1, nio-1):
    for j in range(njo-1):
        for k in range(nko-1):
            fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i, j, k)], vtxs_id[(i, j+1, k)], vtxs_id[(i, j+1, k+1)], vtxs_id[(i, j, k+1)]))
            owner_id = cells_id[(i-1, j, k)]
            neighbour_id = cells_id[(i, j, k)]
            fp2.write("%d\n" % (owner_id))
            fp3.write("%d\n" % (neighbour_id))
# searching internal faces at j-direction
for j in range(1, njo-1):
    for i in range(nio-1):
        for k in range(nko-1):
            fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i, j, k)], vtxs_id[(i, j, k+1)], vtxs_id[(i+1, j, k+1)], vtxs_id[(i+1, j, k)]))
            owner_id = cells_id[(i, j-1, k)]
            neighbour_id = cells_id[(i, j, k)]
            fp2.write("%d\n" % (owner_id))
            fp3.write("%d\n" % (neighbour_id))
# searching internal faces at k-direction
if two_D == False:
    for k in range(1, nko-1):
        for i in range(nio-1):
            for j in range(njo-1):
                fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i, j, k)], vtxs_id[(i+1, j, k)], vtxs_id[(i+1, j+1, k)], vtxs_id[(i, j+1, k)]))
                owner_id = cells_id[(i, j, k-1)]
                neighbour_id = cells_id[(i, j, k)]
                fp2.write("%d\n" % (owner_id))
                fp3.write("%d\n" % (neighbour_id))
#
# searching boundary faces at NORTH
j = njo-1
for i in range(nio-1):
    for k in range(nko-1):
        fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i, j, k)], vtxs_id[(i, j, k+1)], vtxs_id[(i+1, j, k+1)], vtxs_id[(i+1, j, k)]))
        owner_id = cells_id[(i, j-1, k)]
        fp2.write("%d\n" % (owner_id))
# searching boundary faces at WEST faces
i = 0
for k in range(nko-1):
    for j in range(njo-1):
        fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i, j, k)], vtxs_id[(i, j, k+1)], vtxs_id[(i, j+1, k+1)], vtxs_id[(i, j+1, k)]))
        owner_id = cells_id[(i, j, k)]
        fp2.write("%d\n" % (owner_id))
# searching boundary faces at EAST faces
i = nio-1
for k in range(nko-1):
    for j in range(njo-1):

```

```

        fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i,j,k)],vtxs_id[(i,j
            +1,k)],vtxs_id[(i,j+1,k+1)],vtxs_id[(i,j,k+1)]))
        owner_id = cells_id[(i-1,j,k)]
        fp2.write("%d\n" % (owner_id))
# searching boundary faces at SOUTH faces
j = 0
for i in range(nio-1):
    for k in range(nko-1):
        fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i,j,k)],vtxs_id[(i+1,j
            ,k)],vtxs_id[(i+1,j,k+1)],vtxs_id[(i,j,k+1)]))
        owner_id = cells_id[(i,j,k)]
        fp2.write("%d\n" % (owner_id))
# searching boundary faces at BOTTOM faces
k = 0
for i in range(nio-1):
    for j in range(njo-1):
        fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i,j,k)],vtxs_id[(i,j
            +1,k)],vtxs_id[(i+1,j+1,k)],vtxs_id[(i+1,j,k)]))
        owner_id = cells_id[(i,j,k)]
        fp2.write("%d\n" % (owner_id))
# searching boundary faces at TOP faces
k = nko-1
for i in range(nio-1):
    for j in range(njo-1):
        fp1.write("4(%d %d %d %d)\n" % (vtxs_id[(i,j,k)],vtxs_id[(i+1,j
            ,k)],vtxs_id[(i+1,j+1,k)],vtxs_id[(i,j+1,k)]))
        owner_id = cells_id[(i,j,k-1)]
        fp2.write("%d\n" % (owner_id))

#
# Boundaries
fp4.write("    class        polyBoundaryMesh;\n")
fp4.write("    location     \"constant/polyMesh\";\n")
fp4.write("    object          boundary;\n")
fp4.write("}\n")
fp4.write("// * * * * * //\n")
fp4.write("\n")
fp4.write("\n")
fp4.write("6\n")
fp4.write("\n")
# North boundary
fp4.write("    n%04d\n" % (jb))
fp4.write("{\n")
fp4.write("        type          wall;\n")
fp4.write("        nFaces         %d;\n" % (SumOfNF))
fp4.write("        startFace      %d;\n" % (NF_start))
fp4.write("    }\n")
# West boundary
fp4.write("    w%04d\n" % (jb))
fp4.write("{\n")
fp4.write("        type          wall;\n")
fp4.write("        nFaces         %d;\n" % (SumOfWF))
fp4.write("        startFace      %d;\n" % (WF_start))

```

```

fp4.write("    }\n")
# East boundary
fp4.write("    e%04d\n" % (jb))
fp4.write("    {\n")
fp4.write("        type            wall;\n")
fp4.write("        nFaces            %d;\n" % (SumOfEF))
fp4.write("        startFace        %d;\n" % (EF_start))
fp4.write("    }\n")
# South boundary
fp4.write("    s%04d\n" % (jb))
fp4.write("    {\n")
fp4.write("        type            wall;\n")
fp4.write("        nFaces            %d;\n" % (SumOfSF))
fp4.write("        startFace        %d;\n" % (SF_start))
fp4.write("    }\n")
# Bottom boundary
fp4.write("    b%04d\n" % (jb))
fp4.write("    {\n")
fp4.write("        type            wall;\n")
fp4.write("        nFaces            %d;\n" % (SumOfBF))
fp4.write("        startFace        %d;\n" % (BF_start))
fp4.write("    }\n")
# Top boundary
fp4.write("    t%04d\n" % (jb))
fp4.write("    {\n")
fp4.write("        type            wall;\n")
fp4.write("        nFaces            %d;\n" % (SumOfTF))
fp4.write("        startFace        %d;\n" % (TF_start))
fp4.write("    }\n")
return

```

```

def write_OpenFOAM_files(rootName, nblock, grid, flow, axi_flag):
    """
    Writes the grid files for OpenFOAM, support 2-D and 3-D cases.

    :param rootName: specific file names are built by adding bits to this
        name
    :param nblock: integer
    :param grid: list of StructuredGrid objects
    :param flow: list of StructuredGridFlow objects
    :param axi_flag: integer
    """
    plotPath = "foam"
    if not os.access(plotPath, os.F_OK):
        os.makedirs(plotPath)
    for jb in range(nblock):
        subplotPath = plotPath+("/b%04d" % (jb))
        if not os.access(subplotPath, os.F_OK):
            os.makedirs(subplotPath)
        #
        fileName0 = "points"
        fileName0 = os.path.join(subplotPath, fileName0)
        OFFFile0 = open(fileName0, "wb")

```

```

fileName1 = "faces"
fileName1 = os.path.join(subplotPath, fileName1)
OFFile1 = open(fileName1, "wb")
fileName2 = "owner"
fileName2 = os.path.join(subplotPath, fileName2)
OFFile2 = open(fileName2, "wb")
fileName3 = "neighbour"
fileName3 = os.path.join(subplotPath, fileName3)
OFFile3 = open(fileName3, "wb")
fileName4 = "boundary"
fileName4 = os.path.join(subplotPath, fileName4)
OFFile4 = open(fileName4, "wb")
#
write_general_OpenFOAM_header(OFFile0)
write_general_OpenFOAM_header(OFFile1)
write_general_OpenFOAM_header(OFFile2)
write_general_OpenFOAM_header(OFFile3)
write_general_OpenFOAM_header(OFFile4)
#
write_OpenFOAM_unstructured_file(OFFile0, OFFile1, OFFile2, OFFile3
    , OFFile4, jb, grid[jb], flow[jb], axi_flag)
#
write_general_OpenFOAM_bottom(OFFile0)
write_general_OpenFOAM_bottom(OFFile1)
write_general_OpenFOAM_bottom(OFFile2)
write_general_OpenFOAM_bottom(OFFile3)
write_general_OpenFOAM_bottom(OFFile4)
#
OFFile0.close()
OFFile1.close()
OFFile2.close()
OFFile3.close()
OFFile4.close()
return

```

7.2 Source Code e3prepToFoam.py

```

1 #!/usr/bin/env python
2 """
3 Function to automatically convert e3prep output to OpenFoam mesh.
4 Can perform following 3 mesh conversions:
5     - 2D Eilmer to equivalent 2D OpenFoam mesh
6     - 2D Eilmer axisymmetric, to equivalent OpenFoam axisymmetric mesh
7     - 3D Eilmer to equivalent 3D OpenFoam mesh
8
9 The script performs the following tasks:
10 1) check appropriate OpenFoam File Structure exists
11 2) execute e3post.py --openFoam to generate individual unstructured OF
    meshes for each Eilmer block. These meshes are stored in /foam/bxxxx
    corresponding to respective blocks

```

```

12 3) combines the individual blocks into a single unstructured OpenFoam mesh
13 4) stitch internal faces
14 5) for 2D combine front and back faces of mesh (Top and Bottom in the
    e3prep blocks) to form "empty" or "wedge" type patches
15 6) group boundaries defined in job.py based on the respective names. The
    following boundary names are recognised (replace XX by 01, 02, 03, 04,
    05,06, 07, 08, 09, 10):
16     - OF_inlet_XX
17     - OF_outlet_XX
18     - OF_wall_XX
19     - OF_symmetry_XX
20     - Anything else will retain its name and be set as "patch". Duplicate
      names may cause errors.
21 7) optionally a /0/p and /0/U file containing pressure and velocity
    boundary conditions is created.
22
23 Author: Ingo Jahn 03/02/2015
24 """
25
26 import os as os
27 import numpy as np
28 import shutil as sh
29 from getopt import getopt
30 import sys as sys
31
32 shortOptions = ""
33 longOptions = ["help", "job=", "create_0"]
34
35 def printUsage():
36     print """
37     print "Usage: e3prepToFoam.py [--help] [--job=<jobFileName>] [--
      create_0]"
38     return
39
40
41 def get_folders():
42     print 'Obtaining directory from which code is executed'
43     start_dir = os.getcwd()
44     str2 = start_dir.split('/')
45     n = len(str2)
46     str3 = str2[n-1]
47     if str3 == 'e3prep':
48         case_dir = os.path.dirname(start_dir)
49         root_dir = os.path.dirname(case_dir)
50         case_name = str2[n-2]
51     else:
52         case_dir = start_dir
53         root_dir = os.path.dirname(case_dir)
54         case_name = str2[n-1]
55     return root_dir, case_dir, start_dir, case_name
56
57 def check_case_structure(case_dir, root_dir):
58     print 'Checking if correct OpenFOAM case structure exists \n'

```

```

59     flag = 0
60     if os.path.exists(case_dir+'/'+'0') is not True:
61         flag = 1
62         print "Missing /0 directory"
63     if os.path.exists(case_dir+'/'+'system') is not True:
64         flag = 1
65         print "Missing /system directory"
66     if os.path.exists(case_dir+'/'+'constant') is not True:
67         flag = 1
68         print "Missing /constant directory"
69
70     if os.path.exists(case_dir+'/'+'constant/polyMesh') is not True:
71         flag = 1
72         print "Missing /constant/polyMesh directory"
73     if os.path.exists(root_dir+'/'+'slave_mesh') is True:
74         flag = 1
75         print "Folder ../slave_mesh/ already exists \n Delete this folder
76             and try again"
77     print '\n'
78     return flag
79
80 def face_index_to_string(ind):
81     if ind == 0:
82         return 'n'
83     elif ind == 1:
84         return 'e'
85     elif ind == 2:
86         return 's'
87     elif ind == 3:
88         return 'w'
89     elif ind == 4:
90         return 't'
91     elif ind == 5:
92         return 'b'
93     else:
94         print 'Error'
95     return
96
97 def get_job_config_data(job):
98     print 'Extracting required variables from job.config'
99     f = open((job + '.config'), 'r')
100    # find dimensions
101    for line in f:
102        if "dimensions" in line:
103            temp = line.split()
104            dimensions = int(temp[2])
105            break
106    # differentiate between axisymmetric and 2-D cases
107    for line in f:
108        if "axisymmetric_flag" in line:
109            temp = line.split()
110            axisymmetric_flag = int(temp[2])

```

```

111         break
112     # find number of blocks
113     for line in f:
114         if "nblock" in line:
115             temp = line.split()
116             nblock = int(temp[2])
117             break
118
119     other_block = np.zeros((nblock,6))
120     other_face = np.zeros((nblock,6))
121     # find connecting blocks
122     f.seek(0,0)
123     block = 0
124     face = 0
125     for line in f:
126         if "other_block" in line:
127             temp = line.split()
128             other_block[block, face] = int(temp[2])
129             face = face + 1
130             if dimensions == 2:
131                 if face == 4:
132                     other_block[block, 4] = -1
133                     other_block[block, 5] = -1
134                     face = 0
135                     block = block + 1
136                 elif dimensions == 3:
137                     if face == 6:
138                         face = 0
139                         block = block + 1
140     # find connecting faces
141     f.seek(0,0)
142     block = 0
143     face = 0
144     for line in f:
145         if "other_face" in line:
146             temp = line.split()
147             if temp[2] == "none":
148                 other_face[block, face] = -1
149             elif temp[2] == "north":
150                 other_face[block, face] = 0
151             elif temp[2] == "east":
152                 other_face[block, face] = 1
153             elif temp[2] == "south":
154                 other_face[block, face] = 2
155             elif temp[2] == "west":
156                 other_face[block, face] = 3
157             elif temp[2] == "top":
158                 other_face[block, face] = 4
159             elif temp[2] == "bottom":
160                 other_face[block, face] = 5
161             else:
162                 print "Error"
163             face = face + 1

```



```

164         if dimensions == 2:
165             if face == 4:
166                 other_face[block,4] = -1
167                 other_face[block,5] = -1
168                 face = 0
169                 block = block + 1
170         elif dimensions == 3:
171             if face == 6:
172                 face = 0
173                 block = block + 1
174     # find boundary labels
175     f.seek(0,0)
176     Label = [[None for i in range(6)] for i in range(nblock)]
177     for block in range(nblock):
178         if dimensions == 2:
179             for face in range(4):
180                 temp = find_boundary_info(f, block, face, "label")
181                 temp = temp.split()
182                 if len(temp) == 3:
183                     Label[block][face] = temp[2]
184                     # print "I was here", block, face, temp[2]
185                 else:
186                     Label[block][face] = "EMPTY"
187             Label[block][4] = "EMPTY"
188             Label[block][5] = "EMPTY"
189         if dimensions == 3:
190             for face in range(6):
191                 temp = find_boundary_info(f, block, face, "label")
192                 temp = temp.split()
193                 if len(temp) == 3:
194                     Label[block][face] = temp[2]
195                     # print "I was here", block, face, temp[2]
196                 else:
197                     Label[block][face] = "EMPTY"
198     f.close()
199     return (nblock, dimensions, axisymmetric_flag, other_block, other_face,
200           Label)
201
202 def find_boundary_info(fp, block, face, lookup):
203     if face == 0:
204         phrase = (" [block/]+str(block)+"/face/north]")
205     elif face == 1:
206         phrase = (" [block/]+str(block)+"/face/east]")
207     elif face == 2:
208         phrase = (" [block/]+str(block)+"/face/south]")
209     elif face == 3:
210         phrase = (" [block/]+str(block)+"/face/west]")
211     elif face == 4:
212         phrase = (" [block/]+str(block)+"/face/top]")
213     elif face == 5:
214         phrase = (" [block/]+str(block)+"/face/bottom]")
215     else:

```

```

216         print "wrong face_index"
217     fp.seek(0,0)
218     for num, line in enumerate(fp, 1):
219         if phrase in line:
220             break
221     for line in fp:
222         if lookup in line:
223             break
224     return line
225
226
227 def write_general_OpenFoam_header(fp):
228     fp.write("/*----- C++
229         /*-----*\n")
230     fp.write(" | ===== |
231         | \\\ / Field | OpenFOAM: The Open Source CFD
232         Toolbox | \n")
233     fp.write(" | \\\ / O peration | Version: 2.2.2
234         | \n")
235     fp.write(" | \\\ / A nd | Web: www.OpenFOAM.org
236         | \n")
237     fp.write(" | \\\ / M anipulation | This file generated by e3post.
238         py | \n")
239     fp.write("
240         \n")
241     fp.write("FoamFile\n")
242     fp.write("{\n")
243     fp.write("    version 2.0;\n")
244     fp.write("    format ascii;\n")
245     return
246
247 def write_general_OpenFoam_bottom_round(fp):
248     fp.write(");\n")
249     fp.write("\n")
250     fp.write("//
251         *****
252         //\n")
253     return
254
255 def write_general_OpenFoam_bottom_curly(fp):
256     fp.write("};\n")
257     fp.write("\n")
258     fp.write("//
259         *****
260         //\n")
261     return
262
263 def write_createPatch_header(fp):
264     #
265     # ----- writing files now -----
266     # points

```



```

357     fp.write("    {\n")
358     if btype == "zeroGradient":
359         fp.write("        type    zeroGradient; \n")
360     elif btype == "empty":
361         fp.write("        type    empty; \n")
362     elif btype == "wedge":
363         fp.write("        type    wedge; \n")
364     elif btype == "symmetry":
365         fp.write("        type    symmetry; \n")
366     elif btype == "fixedValue":
367         fp.write("        type    fixedValue; \n")
368         fp.write("        value    uniform (0 0 0); \n")
369     else:
370         print ("Boundary type, " + btype + " not recognised. Setting empty"
371              )
372     fp.write("    }\n")
373     return
374 def combine_faces(case_dir , start_dir , patch_str , patch_name , patch_type):
375     file_createPatchDict = "createPatchDict"
376     file_createPatchDict = os.path.join((case_dir+ '/system'),
377         file_createPatchDict)
378     OFFile0 = open(file_createPatchDict , "wb")
379     write_general_OpenFoam_header(OFFile0)
380     write_createPatch_header(OFFile0)
381     write_patches(OFFile0 , patch_str , patch_name , patch_type)
382     write_general_OpenFoam_bottom_round(OFFile0)
383     OFFile0.close()
384     print "createPatchDict has been written. \n"
385     # execute createPatch
386     os.chdir(case_dir)
387     flag = os.system('createPatch -overwrite')
388     # move back to starting_directory
389     os.chdir(start_dir)
390     if flag == 0:
391         print ("The following boundaries" + patch_str + " have been combined
392              to form Patch: " + patch_name + " with the type: " + patch_type)
393     else:
394         raise MyError("Problem during execution of createPaatch.")
395     return
396 def check_for_undefined_faces(case_dir , nblock):
397     file_name = "boundary"
398     file_name = os.path.join((case_dir+ '/constant/polyMesh/'), file_name)
399     File = open(file_name , "r")
400     String = []
401     for n in range(nblock):
402         for line in File:
403             if ('n'+ '%04d' % n) in line:
404                 String.append(line + ';' )
405             if ('e'+ '%04d' % n) in line:
406                 String.append(line + ';' )

```

```

407         if ('s'+'%04d' % n) in line:
408             String.append(line + '; ')
409         if ('w'+'%04d' % n) in line:
410             String.append(line + '; ')
411     File.seek(0,0)
412     File.close()
413     return String
414
415
416 def check_for_undefined_labels(patch_Label):
417     A = [item for sublist in patch_Label for item in sublist]
418     A = set(A)
419     String = ['EMPTY', 'Centreline']
420     for i in range(10):
421         String.append("OF_inlet_"+'%02d' % i)
422         String.append("OF_outlet_"+'%02d' % i)
423         String.append("OF_wall_"+'%02d' % i)
424         String.append("OF_symmetry_"+'%02d' % i)
425     String = set(String)
426
427     return list(A.difference(String))
428
429
430 def collapse_faces(case_dir, start_dir):
431     fn = "collapseDict"
432     fn = os.path.join((case_dir+ '/system'), fn)
433     OFFile0 = open(fn, "wb")
434
435     write_general_OpenFoam_header(OFFile0)
436     write_collapseDict_header(OFFile0)
437     write_general_OpenFoam_bottom_curly(OFFile0)
438     OFFile0.close()
439     print "collapseDict has been written. \n"
440     # execute createPatch
441     os.chdir(case_dir)
442     flag = os.system('collapseEdges -overwrite')
443     # move back to starting_directory
444     os.chdir(start_dir)
445     if flag == 0:
446         print ("Aligned edges have been collapsed")
447     else:
448         raise MyError("Problem during execution of collapseEdges.")
449
450     return flag
451
452 class MyError(Exception):
453     def __init__(self, value):
454         self.value = value
455     def __str__(self):
456         return repr(self.value)
457
458
459 def main(uoDict):

```

```

460 # create string to collect warning messages
461 warn_str = "\n"
462
463 # main file to be executed
464 jobName = uoDict.get("--job", "test")
465
466 # strip .py extension form jobName
467 jobName = jobName.split('.')
468 jobName = jobName[0]
469
470 # establish case, root, and start directory
471 root_dir, case_dir, start_dir, case_name = get_folders()
472
473 # check that correct directory structure exists
474 dir_flag = check_case_structure(case_dir, root_dir)
475 if dir_flag == 1:
476     raise MyError('ERROR: Incorrect Directory Structure. e3preToFoam
        must be run inside an OpenFoam case with appropriate sub-
        directories. \nSee error message above and create missing
        folders or copy from existing case. \nOnce folders have been
        created, re-run.')
477
478 # change into e3prep directory
479 os.chdir((case_dir+'e3prep'))
480
481 # get data from job.config
482 nblock, dimensions, axisymmetric_flag, other_block, other_face,
    patch_Label = get_job_config_data(jobName)
483
484 # check that combination of diemnsions and axi-symmetric flag is
    appropriate
485 if not (((dimensions == 2 or dimensions == 3) and axisymmetric_flag ==
    0) or (dimensions == 2 and axisymmetric_flag == 1)):
486     raise MyError('ERROR: Combination of dimensions and
        axisymmetric_flag is not supported')
487 # run e3post to generate /foam folder containing meshes for respective
    block
488 os.system(("e3post.py --job=" + jobName + " --OpenFoam"))
489
490 print 'e3post has been executed and individual foam meshes have been
    generated for each block \n \n '
491
492 # merging individual blocks
493 print ('Working on Case = '+case_name)
494
495 ## move data currently in /polMesh
496 #sh.move('polyMesh', 'polyMesh_old')
497 sh.rmtree(case_dir + '/constant/polyMesh')
498
499 # create case file for slave_mesh
500 sh.copytree(case_dir, (root_dir+'slave_mesh'))
501
502 # copy Master mesh data into required folder

```

```

503 sh.copytree((case_dir+'/e3prep/foam/b0000/'),case_dir+'/constant/
      polyMesh')
504
505 for block in range(nblock-1):
506     # copy correct slave_mesh into slave_mesh case
507     sh.copytree((case_dir+'/e3prep/foam/b'+ '%04d' % (block+1) + '/'),
      root_dir+'/slave_mesh/constant/polyMesh')
508
509     # execute mergeMeshes command
510     os.chdir(root_dir)
511     flag = os.system('mergeMeshes -overwrite ' + case_name + '
      slave_mesh')
512     if flag == 0:
513         print ('Block ' + '%04d' % block + ' and ' + '%04d' % (block+1)
      + ' have been merged.')
514     else:
515         sh.rmtree(root_dir+'/slave_mesh') # removing slave_mesh
      directory before exiting
516         os.chdir(start_dir)
517         raise MyError('Error with mergeMeshes. \n Try running of230 to
      load OpenFOAM module')
518
519     # remove polyMesh from slave_mesh
520     sh.rmtree(root_dir+'/slave_mesh/constant/polyMesh')
521
522     # remove slave_mesh
523     sh.rmtree(root_dir+'/slave_mesh')
524     # move back to starting_directory
525     os.chdir(start_dir)
526
527     print "Merging of meshes complete. \n \n "
528
529     # Remove faces with zero area, positioned along centreline
530     if axisymmetric_flag == 1:
531         print "Removing zero Area faces along centreline. \n"
532         flag = collapse_faces(case_dir, start_dir)
533
534     #identify number of block connections
535     interfaces = len(other_block[np.where(other_block != -1)]) # counts 2 x
      internal connections, as seen by other blocks
536     if interfaces > 0:
537
538         # move /0 directory
539         sh.move((case_dir + '/0'), case_dir + '/temp')
540
541         while True:
542             (block, face) = np.where(other_block != -1)
543             if len(block) == 0:
544                 break
545
546             # print (block, face)
547             o_block = other_block[block[0], face[0]]
548             o_face = other_face[block[0], face[0]]

```



```

549
550     current_facename = (face_index_to_string(face[0]) + '%04d' %
551         block[0])
552     other_facename = (face_index_to_string(o_face) + '%04d' %
553         o_block)
554
555     # print (current_facename, other_facename)
556
557     # overwrite matching face in other block
558     other_block[o_block, o_face] = -1
559     other_face[o_block, o_face] = -1
560     other_block[block[0], face[0]] = -1
561
562     # execute stitchMesh command
563     os.chdir(case_dir)
564     flag = os.system('stitchMesh -overwrite -perfect ' +
565         current_facename + ' ' + other_facename)
566     # move back to starting directory
567     os.chdir(start_dir)
568     if flag == 0:
569         print ('Face ' + current_facename + ' and ' +
570             other_facename + ' have been stitched.')
571     else:
572         raise MyError('Error with stitchMesh.')
573
574     # move /0 directory back
575     sh.move((case_dir + '/temp'), case_dir + '/0')
576
577 print "Stitching of internal Faces complete. \n \n"
578
579 # Group all boundaries with Centreline label as corresponding patch
580 if axisymmetric_flag == 1:
581     name = "Centreline"
582     cent_str = ""
583     for block in range(nblock):
584         L_block = patch_Label[block]
585         #print L_block
586         ind = [n for n, s in enumerate(L_block) if name in s]
587
588         if ind != []:
589             for n in ind:
590                 if n == 0:
591                     cent_str = (cent_str + ' n' + '%04d' % block)
592                 if n == 1:
593                     cent_str = (cent_str + ' e' + '%04d' % block)
594                 if n == 2:
595                     cent_str = (cent_str + ' s' + '%04d' % block)
596                 if n == 3:
597                     cent_str = (cent_str + ' w' + '%04d' % block)
598                 if n == 4:
599                     cent_str = (cent_str + ' t' + '%04d' % block)
600                 if n == 5:
601                     cent_str = (cent_str + ' b' + '%04d' % block)

```

```

598     print cent_str
599     if cent_str != "":
600         combine_faces(case_dir , start_dir , cent_str , name, 'empty')
601
602     # do automatic patch combination
603     # top and bottom faces
604     if dimensions == 2:
605         if axisymmetric_flag == 0:
606             # create empty FrontBack patch
607             patch_str = ' '
608             for i in range(nblock):
609                 patch_str = (patch_str+' b'+ '%04d' % i + ' t' + '%04d' % i
610                             )
611                 patch_name = 'FrontBack'
612                 patch_type = 'empty'
613                 combine_faces(case_dir , start_dir , patch_str , patch_name ,
614                             patch_type)
615             elif axisymmetric_flag == 1:
616                 # create pair of wedge patches
617                 patch_str = ' '
618                 for i in range(nblock):
619                     patch_str = (patch_str+' b'+ '%04d' % i)
620                     patch_name = 'Back'
621                     patch_type = 'wedge'
622                     combine_faces(case_dir , start_dir , patch_str , patch_name ,
623                                 patch_type)
624                     patch_str = ' '
625                     for i in range(nblock):
626                         patch_str = (patch_str+' t'+ '%04d' % i)
627                         patch_name = 'Front'
628                         patch_type = 'wedge'
629                         combine_faces(case_dir , start_dir , patch_str , patch_name ,
630                                     patch_type)
631
632     # combine patches , based on block label.
633     # Following labels are supported:
634     # OF_inlet_00 , OF_inlet_01 , OF_inlet_02 (up to 09)
635     # OF_outlet_00 , OF_outlet_01 , OF_outlet_02 (up to 09)
636     # OF_wall_00 , OF_wall_01 , OF_wall_02 (up to 09)
637     # OF_symmetry_00 , OF_symmetry_01 , OF_symmetry_02 (up to 09)
638
639     N_list_in = []
640     N_list_out = []
641     N_list_wall = []
642     N_list_sym = []
643
644     for i in range(10):
645         in_n = ("OF_inlet_" + '%02d' % i)
646         out_n = ("OF_outlet_" + '%02d' % i)
647         wall_n = ("OF_wall_" + '%02d' % i)
648         sym_n = ("OF_symmetry_" + '%02d' % i)
649
650     inlet_str = ""

```

```

647 outlet_str = ""
648 wall_str = ""
649 sym_str = ""
650 for block in range(nblock):
651     L_block = patch_Label[block]
652     #print L_block
653     i_ind = [n for n, s in enumerate(L_block) if in_n in s]
654     o_ind = [n for n, s in enumerate(L_block) if out_n in s]
655     w_ind = [n for n, s in enumerate(L_block) if wall_n in s]
656     s_ind = [n for n, s in enumerate(L_block) if sym_n in s]
657     #print i_ind != []
658     #print o_ind != []
659
660     if i_ind != []:
661         for n in i_ind:
662             if n == 0:
663                 inlet_str = (inlet_str + ' n' + '%04d' % block)
664             if n == 1:
665                 inlet_str = (inlet_str + ' e' + '%04d' % block)
666             if n == 2:
667                 inlet_str = (inlet_str + ' s' + '%04d' % block)
668             if n == 3:
669                 inlet_str = (inlet_str + ' w' + '%04d' % block)
670             if n == 4:
671                 inlet_str = (inlet_str + ' t' + '%04d' % block)
672             if n == 5:
673                 inlet_str = (inlet_str + ' b' + '%04d' % block)
674
675     if o_ind != []:
676         for n in o_ind:
677             if n == 0:
678                 outlet_str = (outlet_str + ' n' + '%04d' % block)
679             if n == 1:
680                 outlet_str = (outlet_str + ' e' + '%04d' % block)
681             if n == 2:
682                 outlet_str = (outlet_str + ' s' + '%04d' % block)
683             if n == 3:
684                 outlet_str = (outlet_str + ' w' + '%04d' % block)
685             if n == 4:
686                 outlet_str = (outlet_str + ' t' + '%04d' % block)
687             if n == 5:
688                 outlet_str = (outlet_str + ' b' + '%04d' % block)
689
690     if w_ind != []:
691         for n in w_ind:
692             if n == 0:
693                 wall_str = (wall_str + ' n' + '%04d' % block)
694             if n == 1:
695                 wall_str = (wall_str + ' e' + '%04d' % block)
696             if n == 2:
697                 wall_str = (wall_str + ' s' + '%04d' % block)
698             if n == 3:
699                 wall_str = (wall_str + ' w' + '%04d' % block)
700             if n == 4:
701                 wall_str = (wall_str + ' t' + '%04d' % block)
702             if n == 5:
703                 wall_str = (wall_str + ' b' + '%04d' % block)

```

```

700         if n == 5:
701             wall_str = (wall_str + ' b' + '%04d' % block)
702     if s_ind != []:
703         for n in s_ind:
704             if n == 0:
705                 sym_str = (sym_str + ' n' + '%04d' % block)
706             if n == 1:
707                 sym_str = (sym_str + ' e' + '%04d' % block)
708             if n == 2:
709                 sym_str = (sym_str + ' s' + '%04d' % block)
710             if n == 3:
711                 sym_str = (sym_str + ' w' + '%04d' % block)
712             if n == 4:
713                 sym_str = (sym_str + ' t' + '%04d' % block)
714             if n == 5:
715                 sym_str = (sym_str + ' b' + '%04d' % block)
716
717
718     print inlet_str, outlet_str, wall_str, sym_str
719     if inlet_str != "":
720         combine_faces(case_dir, start_dir, inlet_str, in_n, 'patch')
721         N_list_in.append(in_n)
722     if outlet_str != "":
723         combine_faces(case_dir, start_dir, outlet_str, out_n, 'patch')
724         N_list_out.append(out_n)
725     if wall_str != "":
726         combine_faces(case_dir, start_dir, wall_str, wall_n, 'wall')
727         N_list_wall.append(wall_n)
728     if sym_str != "":
729         combine_faces(case_dir, start_dir, sym_str, sym_n, 'symmetry')
730         N_list_sym.append(sym_n)
731
732     # check if there are patches remaining that havent been defined.
733     String1 = check_for_undefined_faces(case_dir, nblock)
734     String2 = check_for_undefined_labels(patch.Label)
735
736     if not(String1 == []):
737         warn_str = warn_str + 'WARNING: Not all external boundaries were
738             defined in e3prep \n' + 'Check these faces: ' + String1 + '\n'
739
740     if not(String2 == []):
741         warn_str = warn_str + 'WARNING: labels used to define boundary
742             faces do not follow standard OF_names \n' + 'Check these labels
743             : ' + String2 + '\n'
744
745     # Option to create template entries for /0.
746     if uoDict.has_key("--create_0"):
747
748         # check if /0/p file exists
749         if os.path.isfile(case_dir+'/0/'+p) == 1:
750             sh.copyfile(case_dir+'/0/'+p, case_dir+'/0/'+p.bak)
751             warn_str = warn_str + "WARNING: Existing copy of /0/p has been
752                 copied to /0/p.bak \n"

```

```

749 # check if /0/U file exists
750 if os.path.isfile(case_dir+'/'+'U') == 1:
751     sh.copyfile(case_dir+'/'+'U', case_dir+'/'+'U.bak')
752     warn_str = warn_str + "WARNING: Existing copy of /0/U has been
        copied to /0/U.bak \n"
753
754 # U and p template are created. The others can be duplicated form
        these
755 file_name = "p"
756 file_name = os.path.join((case_dir+'/'+''), file_name)
757 OFFile0 = open(file_name, "wb")
758
759 write_general_OpenFoam_header(OFFile0)
760 write_p_header(OFFile0)
761
762 for n in range(len(N_list_in)):
763     write_p_Boundary(OFFile0, N_list_in[n], 'zeroGradient')
764 for n in range(len(N_list_out)):
765     write_p_Boundary(OFFile0, N_list_out[n], 'zeroGradient')
766 for n in range(len(N_list_wall)):
767     write_p_Boundary(OFFile0, N_list_wall[n], 'zeroGradient')
768 for n in range(len(N_list_sym)):
769     write_p_Boundary(OFFile0, N_list_sym[n], 'symmetry')
770 if dimensions == 2:
771     if axisymmetric_flag == 0:
772         write_p_Boundary(OFFile0, 'FrontBack', 'empty')
773     else:
774         write_p_Boundary(OFFile0, 'Front', 'wedge')
775         write_p_Boundary(OFFile0, 'Back', 'wedge')
776         write_p_Boundary(OFFile0, 'Centreline', 'empty')
777
778 write_general_OpenFoam_bottom_curly(OFFile0)
779 OFFile0.close()
780 print "/0/p has been written. \n"
781
782 file_name = "U"
783 file_name = os.path.join((case_dir+'/'+''), file_name)
784 OFFile0 = open(file_name, "wb")
785
786 write_general_OpenFoam_header(OFFile0)
787 write_U_header(OFFile0)
788 for n in range(len(N_list_in)):
789     write_U_Boundary(OFFile0, N_list_in[n], 'fixedValue')
790 for n in range(len(N_list_out)):
791     write_U_Boundary(OFFile0, N_list_out[n], 'zeroGradient')
792 for n in range(len(N_list_wall)):
793     write_U_Boundary(OFFile0, N_list_wall[n], 'zeroGradient')
794 for n in range(len(N_list_sym)):
795     write_U_Boundary(OFFile0, N_list_sym[n], 'symmetry')
796 if dimensions == 2:
797     if axisymmetric_flag == 0:
798         write_U_Boundary(OFFile0, 'FrontBack', 'empty')
799     else:

```

```

800         write_U_Boundary(OFFile0, 'Front', 'wedge')
801         write_U_Boundary(OFFile0, 'Back', 'wedge')
802         write_U_Boundary(OFFile0, 'Centreline', 'empty')
803
804     write_general_OpenFoam_bottom_curly(OFFile0)
805     OFFile0.close()
806     print "/0/U has been written. \n"
807
808
809     # Re-order numbering of faces/cells for numerical efficiency
810     # execute renumberMesh
811     os.chdir(case_dir)
812     flag = os.system('renumberMesh -overwrite')
813     # move back to starting_directory
814     os.chdir(start_dir)
815     if flag != 0:
816         raise MyError("Problem during execution of renumberMesh.")
817
818     print warn_str
819
820 if __name__ == "__main__":
821     userOptions = getopt(sys.argv[1:], shortOptions, longOptions)
822     uoDict = dict(userOptions[0])
823
824     if len(userOptions[0]) == 0 or uoDict.has_key("--help"):
825         printUsage()
826         sys.exit(1)
827
828     try:
829         main(uoDict)
830         print "\n \n"
831         print "SUCESS: The multi-block mesh created by e3prep.py has been
            converted into a single Polymesh for use with OpenFoam."
832         print "\n \n"
833     except MyError as e:
834         print "This run of e3prepToFoam.py has gone bad."
835         print e.value
836         sys.exit(1)

```