User Guide

Document version: 1.1

Principal developer and document author:
Dr. Franjo Juretić, M. Eng., Assist. Prof.
Managing Director and Founding Partner
Creative Fields, Ltd.

Zagreb, May 2015
## Contents

1. Introduction .................................................................................................................................................. 2

2. Available meshing workflows ..................................................................................................................... 2

   2.1. Cartesian ................................................................................................................................................ 4

   2.2. 2D Cartesian ........................................................................................................................................... 4

   2.3. Tetrahedral ............................................................................................................................................ 5

   2.4. Polyhedral ............................................................................................................................................. 6

3. Input geometry .............................................................................................................................................. 7

4. Required dictionaries and available settings .................................................................................................. 9

   4.1. Dictionaries ........................................................................................................................................... 9

   4.2. Mandatory settings in meshDict ............................................................................................................ 9

   4.3. Refinement settings in meshDict ........................................................................................................... 9

   4.4. Keeping/removing of cells in user-defined regions ............................................................................... 14

   4.5. Boundary layers ................................................................................................................................. 16

   4.6. Anisotropic meshing ............................................................................................................................ 18

   4.7. Workflow controls ............................................................................................................................... 19

   4.8. Renaming patches .............................................................................................................................. 20

   4.9. Enforcing geometry constraints ........................................................................................................ 22

5. Utilities ......................................................................................................................................................... 23

Appendix A: Installation instructions .................................................................................................................. 24
1. Introduction

cfMesh is a cross-platform library for automatic mesh generation that is built on top of OpenFOAM®. It is licensed under GPL, and compatible with all recent versions of OpenFOAM® and foam-extend.

cfMesh supports various 3D and 2D workflows, built by using components from the main library, which are extensible and can be combined into various meshing workflows. The core library utilises the concept of mesh modifiers, which allows for efficient parallelisation using both shared memory parallelisation (SMP) and distributed memory parallelisation using MPI. In addition, special care has been taken on memory usage, which is kept low by implementing data containers (lists, graphs, etc.) that do not require many dynamic memory allocation operations during the meshing process.

2. Available meshing workflows

Meshing workflows implemented in cfMesh require input geometry in a form of surface triangulation, shown in Figure 1, and the user-specified settings, which will be explained in detail in subsequent sections. All workflows are parallelised for shared memory machines and use all available computer cores while running. The number of utilised cores can be controlled by the OMP_NUM_THREADS environment variable, which can be set to the desired number of cores.

![Surface mesh](image1)

*Figure 1 – Surface mesh*

Currently available meshing workflows start the meshing process by creating a so-called mesh template from the input geometry and the user-specified settings. The template is later on adjusted to match the input geometry. The process of fitting the template to the input geometry is designed to be tolerant to poor quality input data, which does not need to be watertight. The available workflows differ by the type of cells generated in the template.

---

1 None of the OpenFOAM® related offering by Creative Fields, Ltd., including cfMesh, is approved or endorsed by OpenCFD, Ltd. (ESI Group), producer of the OpenFOAM® software. OpenFOAM® and OpenCFD® are registered trade marks of ESI Group.
2.1. Cartesian
Cartesian workflow generates 3D meshes consisting of predominantly hexahedral cells with polyhedra in the transition regions between the cells of different size. It is started by typing `cartesianMesh` in a shell window. By default, it generates one boundary layer, which can be further refined on user request. In addition, this workflow can be run using MPI parallelisation, which is intended for generation of large meshes, which do not fit into the memory of a single available computer. An example of the mesh generated by `cartesianMesh` is shown in Figure 2 and Figure 3.

Figure 2 – Volume mesh generated by cartesianMesh

Figure 3 – Detail of the mesh generated by cartesianMesh
2.2. 2D Cartesian

It generates 2D meshes and is started by typing `cartesian2DMesh` in the console. By default it generates one boundary layer which can be further refined. The geometry is given in a form of a ribbon in the x-y plane and extruded in the z direction, shown in Figure 4. An example of final mesh is given in Figure 5.

![Figure 4 – Input geometry for the cartesian2DMesh](image)

![Figure 5 – Mesh generated by cartesian2DMesh](image)
2.3. Tetrahedral
Tetrahedral workflow generates meshes consisting of tetrahedral cells and is started by typing `tetMesh` in a console. By default, it does not generate any boundary layers, and they can be added and refined on user request. An example of a tetrahedral mesh generated by this workflow is given in Figure 6 and Figure 7.

![Mesh generated by tetMesh](image1)

*Figure 6 – Mesh generated by tetMesh*

![Detail of the mesh generated by tetMesh](image2)

*Figure 7 – Detail of the mesh generated by tetMesh*
2.4. Polyhedral

Polyhedral workflow is a new concept for generating meshes consisting of arbitrary polyhedral cells. The meshing process starts by typing `pMesh` in a console. It is applicable to dirty geometries and is optimised to memory usage. An example of a polyhedral mesh generated by this workflow is given in Figure 8 and Figure 9.

![Figure 8 – Mesh generated by pMesh](image1)

![Figure 9 - Detail of the mesh generated by pMesh](image2)
3. Input geometry

cfMesh requires geometries in the form of a surface triangulation. For 2D cases, the geometry is given in a form of a ribbon of triangles with boundary edges in the x-y plane (other orientations are not supported).

The geometry consists of the following entities:

- **List of points** – contains all points in the surface triangulation.
- **List of triangles** – contains all triangles in the surface mesh.
- **Patches** are entities that are transferred onto the volume mesh in the meshing process. Every triangle in the surface is assigned to a single patch, and cannot be assigned to more than one patch. Each patch is identified by its name and type. By default, all patch names and types are transferred on the volume mesh, and are readily available for definition of boundary conditions for the simulation.
- **Facet subsets** are entities, which are not transferred onto the volume mesh in the meshing process. They are used for definition of meshing settings. Each face subset contains indices of triangles in the surface mesh. Please note that a triangle in the surface mesh can be contained in more than one subset.
- **Feature edges** are treated as constraints in the meshing process. Surface points where three or more feature edges meet are treated as corners. Feature edges can be generated by the `surfaceFeatureEdges` utility.

The user must define all sharp features transferred by the cfMesh prior to the meshing process. The edges at the border between the two patches and the feature edges are handled as sharp features in the meshing process. Other edges in the triangulation are not constrained. Figure 10 shows a surface mesh with a patch highlighted in green, a facet subset coloured in blue and user-selected feature edges coloured in red.

*Figure 10 – Geometry with entities*
The file formats suggested for meshing are: fms, ftr, and stl. In addition, the geometry can be imported in all formats supported by the surfaceConvert utility, which comes with OpenFOAM®. In addition, cfMesh also provides additional utilities for conversion of surface mesh into fms and from fms to other formats. However, the three suggested formats support definition of patches, which are transferred onto the volume mesh by default. Other formats can also be used for meshing but they do not support definition of patches in the input geometry and all faces at the boundary of the resulting volume mesh end up in a single patch.

The preferred format for cfMesh is fms, designed to hold all relevant information for setting up a meshing job. It stores patches, subsets, and feature edges in a single file. In addition, it is the only format, which can store all geometric entities into a single file, and the users are strongly encouraged to use it. An example of the surface mesh written in fms is depicted in Figure 11.

```plaintext
// patch names and types
6
(
  bottom
  patch
top
  wall
fwd
  patch
back
  patch
  left
  patch
right
  wall
)

// coordinates of surface points
8((0 0 0) (1 0 0) (1 1 0) (0 1 0) (0 0 1) (1 0 1) (1 1 1) (0 1 1))

// list of triangles
12
(
  ((0 1 2) 0) ((0 3 2) 0) ((5 4 7) 1) ((5 6 7) 1) ((0 1 5) 2) ((0 4 5) 2)
  ((2 6 7) 3) ((2 3 7) 3) ((0 3 7) 4) ((0 4 7) 4) ((2 1 5) 5) ((2 6 5) 5)
)

// list of feature edges
2((1 2) (3 4))

// list of point subsets
1
  pointSubset0 4 4(1 3 4 6)
)

// list of facet subsets
1
  facetSubset1 2 3(0 4 10)
)

// subsets of feature edges
0()
```

_Figure 11 – Structure of the fms geometry file format_
4. Required dictionaries and available settings

4.1. Dictionaries

The meshing process is steered by the settings provided in a `meshDict` dictionary located in the `system` directory of the case. For parallel meshing using MPI, a `decomposeParDict` located in the `system` directory of a case is required, and the number of nodes used for the parallel run must match the `numberOfSubdomains` entry in `decomposeParDict`. Other entries in `decomposeParDict` are not required.

The resulting volume mesh is written in the `polyMesh` directory inside the `constant` directory. The settings available in `meshDict` will be handled in the remainder of this section.

4.2. Mandatory settings in meshDict

`cfMesh` requires only two mandatory settings (see Figure 12) to start a meshing process:

- `surfaceFile` points to a geometry file. The path to the geometry file is relative to the path of the case directory.
- `maxCellSize` represents the default cell size used for the meshing job. It is the maximum cell size generated in the domain.

```
// path to the surface mesh
// relative from case or absolute
surfaceFile "surfacesMeshes/surf.fms";

// maximum cell size in the mesh (mandatory)
maxCellSize 3; // [m]
```

Figure 12 – Mandatory settings

4.3. Refinement settings in meshDict

Quite often a uniform cell size is not satisfactory, and there are many options for local refinement sources in `cfMesh`. The `boundaryCellSize` option is used for refinement of cells at the boundary. It is a global option and the requested cell size is applied everywhere at the boundary. An example is given in Figure 13. The `boundaryCellSizeRefinementThickness` specifies the distance from the boundary at which the `boundaryCellSize` is still applied. The `minCellSize` is a global option which activates automatic refinement of the mesh template. This option performs refinement in regions where the cells are larger than the estimated feature size. The scalar value provided with this setting specifies the smallest cell size allowed by this procedure, see Figure 13. This option is useful for quick simulation because it can generate meshes in complex geometry with low user effort. However, if high mesh quality is required, it provides hints where some mesh refinement is needed.

```
// size of the cells at the boundary (optional)
boundaryCellSize 1.5; // [m]

// distance from the boundary at which
// boundary cell size shall be used (optional)
boundaryCellSizeRefinementThickness 4.5; // [m]

// minimum cell size allowed in the automatic refinement procedure (optional)
minCellSize 0.375; // [m]
```

Figure 13 – boundaryCellSize and minCellSize
localRefinement allows for local refinement regions at the boundary. It is a dictionary of dictionaries and each dictionary inside the main localRefinement dictionary is named by a patch or facet subset in the geometry that is used for refinement, see Figure 14. The requested cell size for an entity is controlled by the cellSize keyword and a scalar value, or by specifying additionalRefinementLevels keyword and the desired number of refinements relative to the maximum cell size. It is possible to specify patches via regular expressions. The thickness of the refinement zone can be specified by the refinementThickness option.

```python
// refinement zones at the surface
// of the mesh (optional)
localRefinement
{
  // patch name
  "patch15.*" // accepts regex
  {
    // additional refinement levels
    // to the maxCellSize
    additionalRefinementLevels 1;

    // thickness of the refinement region
    // away from the patch
    refinementThickness 4.5;
  }

  // subset name
  subset1
  {
    cellSize 1.5; // [m]
  }
}
```

Figure 14 – localRefinement

objectRefinement is used for specifying refinement zones inside the volume. The supported objects that can be used for refinement are: lines, spheres, boxes, truncated cones, and hollow cones (annulus). It is specified as a dictionary of dictionaries, where each dictionary inside the objectRefinement dictionary represents the name of the object used for refinement. refinementThickness option can be used to specify the thickness of the refinement zone away from the object. cellSize specifies the requested cell size inside the volume. Alternatively, additionalRefinementLevels specifies the number of additional refinement levels compared to maxCellSize, applicable inside the volume. Available primitives and the associated settings are:

- **box** is defined by its centre and the size in x, y, and z directions. An example is show in Figure 15.

```python
// refinement zones inside the mesh
// based on primitive geometric objects (optional)
objectRefinements
{
  boxExample
  {
    type box; // determined by the centre and th size in each coordinate
    cellSize 7.51;
    centre (500 500 150);
    lengthX 100;
    lengthY 150;
    lengthZ 200;
  }
}
```

Figure 15 - Box refinement
• *cone* is intended for defining cylinders, truncated cones and cones. It is specified by two points on the axis, and the radius at each point, see Figure 16.

```c
// refinement zones inside the mesh
// based on primitive geometric objects (optional)
objectRefinements
{
  coneExample // name of the object
  {
    type cone; // determined by two points and the radius at each point
    cellSize 7.5; // [m] cell size
    p0 (-100 1873 -320);
    p1 (-560 1400 0);
    radius0 200;
    radius1 200;
  }
}
```

Figure 16 - Cone refinement

• *hollowCone* (annulus) is defined by two points on the axis, and two radiuses for each point. An example is available below, Figure 17.

```c
// refinement zones inside the mesh
// based on primitive geometric objects (optional)
objectRefinements
{
  hollowConeExample
  {
    // determined by two points and two radiuses at each point
    type hollowCone;
    additionalRefinementLevels 2; // additional refinement relative to maxCellSize
    p0 (-100 1873 -320);
    p1 (-560 1400 0);
    radius0_Inner 200;
    radius0_Outer 300;
    radius1_Inner 200;
    radius1_Outer 300;
  }
}
```

Figure 17 - Hollow cone refinement (annulus)

• *sphere* is defined by its centre and the radius, Figure 18.
**Figure 18 - Sphere refinement**

- *line* is defined by the two points, see Figure 19.

```plaintext
// refinement zones inside the mesh
// based on primitive geometric objects (optional)
objectRefinements
{
    sphereExample
    {
        type    sphere; // determined by the centre and the radius
        cellSize 7.51;
        centre  (0 700 0);
        radius  50; // [m] radius
        refinementThickness 50; // [m] distance from the sphere
    }
}
```

**Figure 19 - Line refinement**

```plaintext
objectRefinements
{
    lineExample
    {
        type    line; // line is determined by two endpoints
        cellSize 7.51;
        p0      (-750 1000 450);
        p1      (-750 1500 450);
        refinementThickness 40; // [m] distance from the line
    }
}
```

`surfaceMeshRefinement` allows for using surface meshes as refinement zones in the mesh. It is specified as a dictionary of dictionary where each refinement zone is a sub-dictionary inside the `surfaceMeshRefinement` dictionary. Surface mesh used for the refinement zone is provided with the `surfaceFile` keyword, and the requested cell size for an entity is controlled by the `cellSize` keyword and a scalar value, or by specifying `additionalRefinementLevels` keyword and the desired number of refinements relative to the maximum cell size. Currently, the setting is used to refine the mesh in the region intersected by the surface mesh. It is possible to control the thickness of the refinement zone via `refinementThickness` keyword. An example of the available settings is given in Figure 20.
edgeMeshRefinement allows for using edge meshes as refinement zones in the mesh. It is specified as a dictionary of dictionary where each refinement zone is a sub-dictionary inside the edgeMeshRefinement dictionary. Edge mesh used for the refinement zone is provided with the edgeFile keyword, and the requested cell size for an entity is controlled by the cellSize keyword and a scalar value, or by specifying additionalRefinementLevels keyword and the desired number of refinements relative to the maximum cell size. Currently, the setting is used to refine the mesh in the region intersected by the edge mesh. It is possible to control the thickness of the refinement zone via refinementThickness keyword. An example of the available settings is given in Figure 21.
4.4. Keeping/removing of cells in user-defined regions

Meshing workflows implemented in cfMesh are based on the inside-out meshing, and the meshing process starts by generating the so-called mesh template based on the user-specified cell size. However, if the cell size is locally larger than the geometry feature size it may result with gaps in the geometry being filled by the mesh. On the contrary, the mesh in thin parts of the geometry can be lost if the specified cell size is larger than the local feature size.

**keepCellsIntersectingBoundary** is a global option which ensures that all cells in the template which are intersected by the boundary remain part of the template. By default, all meshing workflows keep only cells in the template which are completely inside the geometry. **keepCellsIntersectingBoundary** keyword must be followed by either 1 (active) or 0 (inactive), see Figure 22. Activation of this option can cause locally connected mesh over a gap, and the problem can be remedied by the **checkForGluedMesh** option which also must be followed by either 1 (active) or 0 (inactive).

**Figure 22 – keepCellsIntersectingBoundary and checkForGluedMesh**

**keepCellsIntersectingPatches** is an option which preserves cells in the template in the regions specified by the user. It is a dictionary of dictionaries, and each dictionary inside the main **keepCellsIntersectingPatches** dictionary is named by patch or facet subset, see Figure 23. This option is not active when the **keepCellsIntersectingBoundary** option is switched on. Patches can be specified using regular expressions.

**Figure 23 – keepCellsIntersectingPatches**
Figure 23 – keepCellsIntersectingPatches subdictionary

- `removeCellsIntersectingPatches` is an option which removes cells from the template in the regions specified by the user. It is a dictionary of dictionaries, and each dictionary inside the main `removeCellsIntersectingPatches` dictionary is named by a patch or a facet subset as shown in Figure 24. The option is active when the `keepCellsIntersectingBoundary` option is switched on. Patches can be specified using regular expressions.

Figure 24 – removeCellsIntersectingPatches subdictionary
4.5. Boundary layers

Boundary layers in cfMesh are extruded from the boundary faces of the volume mesh towards the interior, and cannot be extruded prior to the meshing process. In addition, their thickness is controlled by the cell size specified at the boundary and the mesher tends to produce layers of similar thickness to the cell size. Layers in cfMesh can span over multiple patches if they share concave edges or corners with valence greater than three. All boundary layer settings are provided inside a boundaryLayers dictionary, shown in Figure 25. The options are:

- **nLayers** specifies the number of layers which will be generated in the mesh. It is not mandatory. In case it is not specified the meshing workflow generates the default number of layers, which is either one or zero.
- **thicknessRatio** is a ratio between the thickness of the two successive layer. It is not mandatory. The ratio must be larger than 1. The default value is 1.
- **maxFirstLayerThickness** ensures that the thickness of the first boundary layer never exceeds the specified value. It is not mandatory.

patchBoundaryLayers setting is a dictionary which is used for specifying local properties of boundary layers for individual patches. It is possible to specify nLayers, thicknessRatio and maxFirstLayerThickness for each patch individually within a dictionary with the name equal to the patch name. By default, the number of layers generated at a patch is governed by the global number of layers, or the maximum number of layers specified at any of the patches which form a continuous layer together with the existing patch. allowDiscontinuity option ensures that the number of layers required for a patch shall not spread to other patches in the same layer. Patches can be specified using regular expressions.

```plaintext
// settings for boundary layers
boundaryLayers
{
  // global number of layers (optional)
  nLayers 3;

  // thickness ratio (optional)
  thicknessRatio 1.2;

  // max thickness of the first layer (optional)
  maxFirstLayerThickness 0.5; // [m]

  // local settings for individual patches
  patchBoundaryLayers
  {
    // patch name
    "patch7.*" // accepts regex
    {
      // number of layers (optional)
      nLayers 4;

      // thickness ratio (optional)
      thicknessRatio 1.1;

      // max thickness of the first layer
      // (optional)
      maxFirstLayerThickness 0.5; // [m]

      // active 1 or inactive 0
      allowDiscontinuity 0;
    }
  }
}
```

*Figure 25 – boundaryLayers subdictionary*

cfMesh provides additional controls of boundary layer quality, intended for situations when the large number of layers is required, and when the thickness shall vary smoothly. Activation of the feature is controlled via the
The optimiseLayer keyword. In addition, the parameters of the quality control procedure can also be controlled via additional parameters given inside the optimisationParameters dictionary located inside the boundaryLayers dictionary. The parameters available inside the optimisationParameters dictionary are:

- **nSmoothNormals** is the number of iterations in the procedure for smoothing normal vectors in the boundary layer. It is not mandatory, and the default value is five.
- **maxNumIterations** is the number of iterations in the smoothing procedure. The settings is not mandatory and its default value is five.
- **featureSizeFactor** is the ratio between the maximum allowed layer thickness and the estimated feature size. It is used to limit layer thickness in the regions dominated by curvature. It is not mandatory, and its valid range is between zero and one. The default value is 0.3.
- **reCalculateNormals** calculates the surface normal vectors, and aligns boundary-layer edges to point in the normal direction. The setting is not mandatory, and is active by default.
- **relThicknessTol** controls the maximum difference of the layer thickness between the two neighbouring points, divided by the distance between the points. It is not mandatory, and the valid range is between zero and infinity. Lower values reduce layer thickness thin and enforce uniform thickness distribution.

The above procedure does not guarantee inexistence of any negative volume cells in the boundary layer, and by default, cfMesh performs smoothing and untangling of the layer cells until their volume becomes positive. This procedure is activated by default, and it is possible to deactivate it using untangleLayers setting in the boundaryLayers dictionary.

An example of settings related to boundary layer optimisation is shown in Figure 26.

```
boundaryLayers
{
    // activates smoothing of boundary layers (optional)
    // deactivate by default
    optimiseLayer 1;

    // deactivate untangling of boundary layers
    // (optional) activated by default
    untangleLayers 0;

    optimisationParameters
    {
        // number of iterations in the procedure
        // for redcing normal variation (optional)
        nSmoothNormals 3;

        // maximum number of iterations
        // of the whole procedure (optional)
        maxNumIterations 5;

        // ratio between the maximum layer thickness
        // and the estimated feature size (optional)
        featureSizeFactor 0.4;

        // activate 1 or deactivate 0 calculation of normal
        // (optional)
        reCalculateNormals 1;

        // maximum allowed thickness variation of thickness
        // between two neighbouring points, divided by
        // the distance between the points (optional)
        relThicknessTol 0.1;
    }
}
```

Figure 26 - Boundary layer optimisation
4.6. Anisotropic meshing

Quite often, the physics of the solution requires anisotropic cells, in order to generate accurate solutions with a minimum number of cells. The requirement for generating anisotropic cells is enabled via primitive objects, used for specifying regions with anisotropic cells. The settings for anisotropic meshing are given in `anisotropicSources` dictionary.

Primitive objects applicable are:

- **Box** primitive defines the range of coordinates, and generates anisotropic cells within the whole range of x, y, and z coordinates. The settings applicable to a box are:
  - `centre` are the coordinates of the centre.
  - `lengthX` is the length of the box in the x-direction.
  - `lengthY` is the length of the box in the y-direction.
  - `lengthZ` is the length of the box in the z-direction.
  - `scaleX` is a scaling factor applied in the x-direction. The valid range is between zero and infinity. Values smaller than 1 make the cells smaller in the x-direction, and value greater than 1 make the cells larger, respectively.
  - `scaleY` is a scaling factor applied in the y-direction.
  - `scaleZ` is a scaling factor applied in the z-direction.

- **Plane** primitive, defined by its origin and normal, is intended for situations when the anisotropic cells are not aligned with the main axes of the coordinate system. It generates anisotropic cells at the positive side of the plane, within the distance specified by the user. The settings applicable to a plane are:
  - `origin` specifies the origin of the plane.
  - `normal` specified the normal vector of the plane.
  - `scalingDistance` specifies the distance from the plane in the positive direction of surface normal. The mesher generates anisotropic cells in the in the region between the plane and plane translated by a scaling distance.
  - `scalingFactor` is applied in the normal direction. The valid range is between zero and infinity. Values smaller than one make the cells smaller in the direction of the normal vector, and value greater than one make the cells larger, respectively.

An example of `anisotropicSources` is given in Figure 27.
It is possible to use more than one anisotropic source in a single meshing job, with a restriction that their zone of influence must not overlap. In case their zones of influence overlap, the mesher stops and reports the error.

4.7. Workflow controls

This feature enables users to save the mesh and stop the meshing process after every step in the meshing workflow. In addition, it is possible to restart from the latest step, in case the mesh meets the expectations. Settings are provided in a workflowControls dictionary, and the locations to save the mesh and stop the meshing process are given after the stopAfter keyword. Possible locations in the workflow are:

1. **templateGeneration** — stops the meshing process after the initial template has been generated. It is useful for checking whether the template mesh is fine enough to resolve required geometry features.
2. **surfaceTopology** — stops after the surface of the mesh has been cleansed from invalid connections of surface faces. It is useful to stop here when the settings are coarse for the geometry under consideration.
3. **surfaceProjection** — instructs the mesher to stop after projection of volume mesh’s surface on the input geometry. It is useful to stop here and check whether the mesh has been projected to the correct locations at the input geometry.
4. **patchAssignment** — stops the meshing process after each boundary face is assigned a boundary patch it belong to. It is useful to stop here and check whether feature edges are resolved correctly.
5. **edgeExtraction** — stops the meshing process after the input geometry has been preserved in the meshing process. It is useful for checking the quality of mesh surface, and fin locations where modifications are needed to generate the desired mesh.

6. **boundaryLayerGeneration** — stops the meshing process after the boundary layers have been generated. It is useful for checking whether the topology of the layer meets the required criteria.

7. **meshOptimisation** — stops after the thorough optimisation has been performed. It is useful for checking the quality of the resulting mesh.

8. **boundaryLayerRefinement** - stops after the boundary layer generation process is completed. It is useful for checking the quality of the boundary layers.

In addition, it is possible to read the mesh saved on disk and restart the workflow from the latest step by using the `restartFromLatestStep` keyword in the `workflowControls` dictionary.

An example is presented in Figure 28

```plaintext
workflowControls
{
  // stops the meshing process after the initial mesh has been generated
  // available steps are:
  // 1. templateGeneration
  // 2. surfaceTopology
  // 3. surfaceProjection
  // 4. patchAssignment
  // 5. edgeExtraction
  // 6. boundaryLayerGeneration
  // 7. meshOptimisation
  // 8. boundaryLayerRefinement
  stopAfter templateGeneration;

  // reads the mesh from disk and
  // restarts the meshing process after the latest step
  // please use binary instead of ascii
  restartFromLatestStep 1;
}
```

Figure 28 - Workflow controls

### 4.8. Renaming patches

The settings presented in this section are used for changing of patch names and patch types during the meshing process. The settings are provided inside a `renameBoundary` dictionary with the following options:

- **newPatchNames** is a dictionary inside the `renameBoundary` dictionary. It contains dictionaries with names of patches which shall be renamed. For each patch it is possible to specify the new name or the new patch type with the settings:
  - **newName** keyword is followed by the new name for the given patch. The setting is not mandatory.
  - **type** keyword is followed by the new type for the given patch. The setting is not mandatory.
- **defaultName** is a new name for all patches except the ones specified in `newPatchNames` dictionary. The setting is not mandatory.
- **defaultType** sets the new type for all patches except the ones specified in `newPatchNames` directory. The setting is not mandatory.

Patch names can be specified using regular expressions. An example is given in Figure 29.
Figure 29 – renameBoundary subdictionary
4.9. **Enforcing geometry constraints**

Sometimes it is not possible to capture all geometric features during the meshing process and the acceptance criteria are not met. `enforceGeometryConstraints` option stops the meshing process when it is not possible to capture all features of the input geometry. When active, it stops the meshing process and writes a subset of points that had to be moved away from the geometry in order to yield a valid mesh. An example is given in Figure 30.

```plaintext
// stops the meshing process
// when it is not possible to capture all
// geometric features (optional)
enforceGeometryConstraints 1; // 1 active or 0 inactive
```

*Figure 30 – enforceGeometryConstraints*
5. Utilities

**checkSurfaceMesh**
Performs basic topology and geometric checks on the inout surface mesh, and reports the kind of problems it could cause in the meshing process.

**copySurfaceParts**
It copies surface facets in a specified facet subset into a new surface mesh.

**extrudeEdgesInto2DSurface**
This utility extrudes edges written as feature edges in the geometry into a ribbon of triangles required for 2D mesh generation. Generated triangles are stored in a single patch.

**FLMAToSurface**
It converts geometry from AVL’s flma format into the format readable by cfMesh. Cell selections defined in the input file are transferred as facet subsets.

**FMSToSurface**
The utility converts the data in a fms file into several files which can be imported into ParaView.

**FMSToVTK**
Converts fms file into ParaView’s vtm file, containing all data available in an fms file.

**generateBoundaryLayers**
Generates boundary layer in the volume mesh based on the settings available in meshDict.

**importSurfaceAsSubset**
The utility matches the triangles of the surface mesh used for meshing with the other surface, and create a facet subset in the original surface.

**improveMeshQuality**
Reads the mesh from disk applies the smoother in order to improve mesh quality. The number of iterations is controllable via optional parameters.

**improveSymmetryPlanes**
OpenFOAM is very sensitive to the position of points in the symmetry plane. This utility calculates the regression plane for all vertices belonging to a symmetry plane, and ensures that all vertices lie in a plane.

**mergeSurfacePatches**
The utility allow the user to specify the patches in the surface mesh which shall be merge together.

**meshToFPMA**
This utility converts the mesh into the AVL’s fpma format.
patchesToSubsets
It converts patches in the geometry into facet subsets.

preparePar
It creates processor directories required for MPI parallelisation. The number of processor directories is dependent on the `numberOfSubdomains` specified in `decomposeParDict`.

removeSurfaceFacets
It is a utility for removing facets from the surface mesh. The facets that shall be removed are given by a patch name or a facet subset.

scaleMesh
Scales the mesh by a given factor.

scaleSurfaceMesh
Scale the surface mesh by a given factor. Operates on fms files.

subsetToPatch
It creates a patch in the surface mesh consisting of facets in the given facet subset.

surfaceFeatureEdges
It is used for generating feature edges in the geometry. In case the output is a fms file, generated edges are stored as feature edges. Otherwise it generates patches bounded by the selected feature edges.

surfaceGenerateBoundingBox
It generates a box around the geometry. It does not resolve self-intersections in case the box intersects with the rest of the geometry.

surfaceToFMS
A converter from common surface triangulation formats into fms.

Python utilities

extractFeatureEdges.py
A script to extract feature edges from geometries created in Salome.

salomeTriSurf.py
A script for extracting a fms file from the geometry created in Salome.

Appendix A: Installation instructions
The simplest way to install cfMesh is to download the installer for OpenFOAM® and cfMesh from the company website www.c-fields.com.
If you want to build it from source, please download the source of cfMesh from www.c-fields.com. Open the archive by typing `tar xzf cfMesh-v1.1.tgz` in a Linux shell, and check that you have the OpenFOAM® environment set up. The build is started by the `Allwmake` script within the cfMesh directory.