ICHE 2014, Hamburg - Lehfeldt & Kopmann (eds) - © 2014 Bundesanstalt für Wasserbau ISBN 978-3-939230-32-8

Generation of a 3D Mesh Using snappyHexMesh Featuring Anisotropic Refinement and Near-wall Layers

D. Gisen
Federal Waterways Engineering and Research Institute, Karlsruhe, Germany

ABSTRACT: A workflow for 3D mesh generation in hydraulic engineering using the tool snappyHexMesh is delineated in this article. snappyHexMesh is the standard mesh tool of the free CFD software package OpenFOAM®. 3D hydraulic engineering problems typically include scales ranging from the order of decimeters to hundreds of meters, curved structure geometries, and a free surface, making it necessary to apply multiple levels of local refinement in several regions. As a real-life example for the process, a hydro power dam tailrace mesh is chosen. Its dimensions are about 133 m x 105 m x 12 m and it is needed to investigate the flow field within the tailrace, supporting the projection of a fishway. Beginning with geometry generation and completing with quality control, the workflow is split into seven steps that are explained separately, including practice guidelines based on experience. Made decisions in the example model in general as well as pros and cons of anisotropic refinement and layer insertion in particular are discussed.

Keywords: Mesh, Anisotropic, snappyHexMesh, OpenFOAM

1 INTRODUCTION

The standard tool for generation of computational meshes (also: grids) in the free, open source software package OpenFOAM is called *snappyHexMesh*. Third-party mesh formats like e.g. FLUENT (.msh) or I-DEAS (.unv) can be used as well. However, they need to be converted, which can produce compatibility problems. The major disadvantage of *snappyHexMesh* is its non-intuitive operation by control files and command line and the patchy documentation, which is replaced in practice by a combination of Internet research, exchange with colleagues and troubleshooting using trial-and-error method. The present contribution aims to mitigate these problems, providing detailed documentation of problems occurring through the mesh generation process and thereby enabling solution transfer. As an example from the field of hydraulic engineering, the tail water of a hydropower plant including draft tubes is employed. Its dimensions are about 133 m x 105 m x 12 m and it is needed to investigate the flow field within the tailrace, supporting the projection of a fishway. Knowledge of application and structure of both OpenFOAM and visualization tool ParaView is premised.

The workflow is split into seven steps including two optional ones (*refineMesh* and *addLayers*). Names of files, paths, (sub-)dictionaries, variables and program options are written in *italics*. Terminal commands within the text are printed in `Courier`. File excerpts are `bordered` in addition.

2 METHODS

For computations, OpenFOAM 2.2.2 (Weller et al., 1998) including the tools *snappyHexMesh* and *refineMesh* is applied on a Linux cluster. For 3D visualization, ParaView 4.0.1 is run locally on openSUSE 12.3.

Some filters in ParaView cannot handle arbitrary polyeders. Hence, every mesh loaded is split by default into tetrahedra and pyramids (checkbox *Decompose polyhedra* in ParaView). That causes deceptive

visual output and an overstated number of volume elements. Since deactivating the split function can lead to crashes (segmentation faults), it should be applied only for screenshot creation and with a previous backup of the current view settings using *Save State*. The option *Crinkle clip*, which moves a given cutting plane upon next local cell borders, is recommended for viewing *Clips* and *Slices*.

Figures shown in the following are exported in a width of 1890 pixels and their correspondent height. This equals a printing width of 6.3 inch = 16 cm at 300 dpi (dots per inch).

3 MESH GENERATION WORKFLOW

3.1 *General settings*

Standard OpenFOAM folder structure is used for mesh generation. For clarity, the mesh is generated in a separate directory and copied into the actual case directory at last. The following settings are written in *system/controlDict/* to save every grid generation step in sub-folders numbered in ascending order:

```
application     snappyHexMesh;          endTime         30;
startFrom       latestTime;             deltaT          1;
startTime       0;                      writeControl    timeStep;
stopAt          endTime;                writeInterval   1;
```

Because the process is always based on the latest sub-folder present in the directory, this and every subsequent sub-folder have to be removed before iterating a step using `rm -r FOLDER_NAME .`

3.2 *Step 1: CAD drawings and export*

To start with, the geometry is drawn using a CAD (computer aided design) program. For large scale hydraulic engineering models (dimensions in order of the river width or smaller), structures, banks, and bottom are typically fixed. The position of flow boundaries such as upstream and downstream, atmosphere ("lid") and possible longitudinal in-stream boundaries, in contrast, needs to be chosen. It has proven to be useful to draw the geometry always up to maximum positions of the river margins, since a reduction is possible very easily by reducing the basic mesh (in the following step), whereas a later enlargement, however, makes necessary re-work with the CAD program.

The model is drawn and later calculated in local (relative) coordinates. Shifting the model to e.g. 8-digit global coordinates would facilitate the comparison with field data, but increase either inaccuracy or computational requirements massively. The local point of origin (circle in Figure 1a) is set at the intersection between the left (in flow direction) bank and a virtual plane at the end of the three draft tubes of the hydropower plant. This makes it easy to find in situ and to read distances from the hydropower plant on the longitudinal axis. As a convention, the x-axis is pointing in the main flow direction, and the z-axis is pointing opposite to the direction of gravity. Hence the y-axis yields across the river. Z-coordinates are initially left in sea level in order to simplify drawing based on the construction plans.

After finishing the drawing, geometry parts with identical properties are being exported together into files in STL (STereoLithography) format. Typical criteria for sameness are type of boundary condition, roughness, or wall layers. In the present example, turbine inlet 1–3, draft tube 1–3, weir inlet 1, sheetpile walls with roughness, and other walls without roughness are exported to one file each. Binary is chosen as data format, since ASCII tends to cause files significantly larger when dealing with complex geometries. The bottom boundary is created on the base of echo sounding data using Janet 2.12.2. Finally, the STLs are copied into the sub-folder *constant/triSurface/*.

3.3 *Step 2: blockMesh*

Using the OpenFOAM tool `blockmesh`, a basic mesh in the form of a rectangular box covering the entire simulation domain is created. Its settings are stored and changed in the dictionary *constant/polyMesh/blockMeshDict/*. As general refinement method, OpenFOAM applies uniform edge bisection in the three spatial directions. By choosing the basic edge length to $\Delta = 1.60$ m in the sub-dictionary *blocks*, a total of five refinement stages (up to $(1.60 \text{ m})/2^5 = 0.05$ m) with an even centimeter digit are available, which eases mental arithmetics during mesh generation. Cubes are chosen as cell form, since they represent the optimum geometrical shape for node movements in a later step (OpenFOAM Founda-

tion, 2014) and for turbulence modeling with a Large Eddy Simulation (LES) in the simulation itself (Spalart, 2001).

A total of six boundaries of the basic mesh are declared as *xmin, xmax, ymin, ymax, zmin* and *zmax* in the sub-dictionary *boundary*. If they reside within the borders of the STL geometry, they are recognized as simulation boundaries in the next step automatically. If they do reside outside of the STL geometry, they are cut and discarded. In the present case, height of the atmosphere (*zmax*) and position of the outflow boundaries (*xmax* and *ymin*) can be varied without changing the geometry.

The result of *blockMesh* (Figure 1b) is stored under *constant/polyMesh/* differing from the results of the next steps and can be viewed in ParaView as *Time 0*.

3.4 *Step 3: castellatedMesh*

Following the preparing steps, *snappyHexMesh* is called for the first time. The tool reads its settings from *system/snappyHexMeshDict/* (a template can be copied from the tutorial *mesh/snappyHexMesh/ motorBike/*). The dictionary contains parameters for the three steps, *castellatedMesh*, *snap*, and *addLayers*, in three correspondent sub-dictionaries. For space limitations only the most important parameters are discussed here. Besides descriptive comments in the dict, the presentation of Jackson (2012) is adjuvant for understanding. For calling the first step only, it is set:

```
castellatedMesh true; snap false; addLayers false;
```

After executing `snappyHexMesh | tee log1`, the result is stored into the folder *1/polyMesh/*. The prompt output is sent towards the Linux system tool *tee* by the vertical bar ("pipe"), which stores the output for possible later use into the text file *log1*.

castellatedMesh performs three actions on the mesh: (a) dividing all cells cut by STL geometries, (b) refining cells within defined local domains, and (c) discarding cells outside of the STL borders. The processes are explained in the following sections.

(a) The sub-dictionary *refinementSurface* (code excerpt below) contains all boundaries of the model. The minimum and maximum number of bisection respectively refinement steps is given by two entries behind *level*. Maximum refinement is applied to the cells at every bend in the STL file, whose *featureAngle* $\varphi \geq$ *resolveFeatureAngle* φ_{rFA} (in degree). φ is the angle between two face normals sharing a common edge. It is computed through the dot product of the unity normal vectors and resides always within the interval [0°, 180°]. In other words, φ_{rFA} is exactly the angle an virtual face may bend towards in positive or negative direction without causing maximum refinement at the resulting *featureEdge* (Figure 2). With the code given in the excerpt, every cell cut by the STL *walls* is split into $2^{3\cdot2} = 64$ new cells minimum. Only if the STL bends more than $\varphi_{rFA} = 30°$ within a certain cell, it is split into $2^{3\cdot3} = 512$ new cells.

Entered values φ_{rFA} outside of [0°, 180°] are automatically set to *–1.0e+15* (*refinementParameters.C*), resulting in maximum refinement for every bend.

```
  nCellsBetweenLevels 2;      // decrease cell count
  refinementSurfaces
      {   walls               { level (2 3); }
          tube_1              { level (4 4); }
          ...
      }
  resolveFeatureAngle 30;
```

(b) Isotropic refinement inside of the boxes defined in the sub-dictionary *geometry* is possible in the sub-directionary *refinementRegions*. With *mode inside*, as chosen in the tutorial template, the first entry of *levels* (not to confuse with *level* from above) is treated as a dummy value and the number of cell splits is determined by the second entry. Another option is *mode distance*, which refines up to the distance from the domain given in the first entry by the number given in the second entry.

It is useful for the definition of the box borders to create a *castellatedMesh* without refinement beforehand and to load it in ParaView along with the STLs. Employing the filter *slice*, borders in *x*, *y*, and *z* directions can be visualized and moved until they cut the exact cell needed. Cells are cut if a part of their volume lies inside a refinement box. The respective spatial coordinate may then be read and copied from ParaView to *snappyHexMesh*.

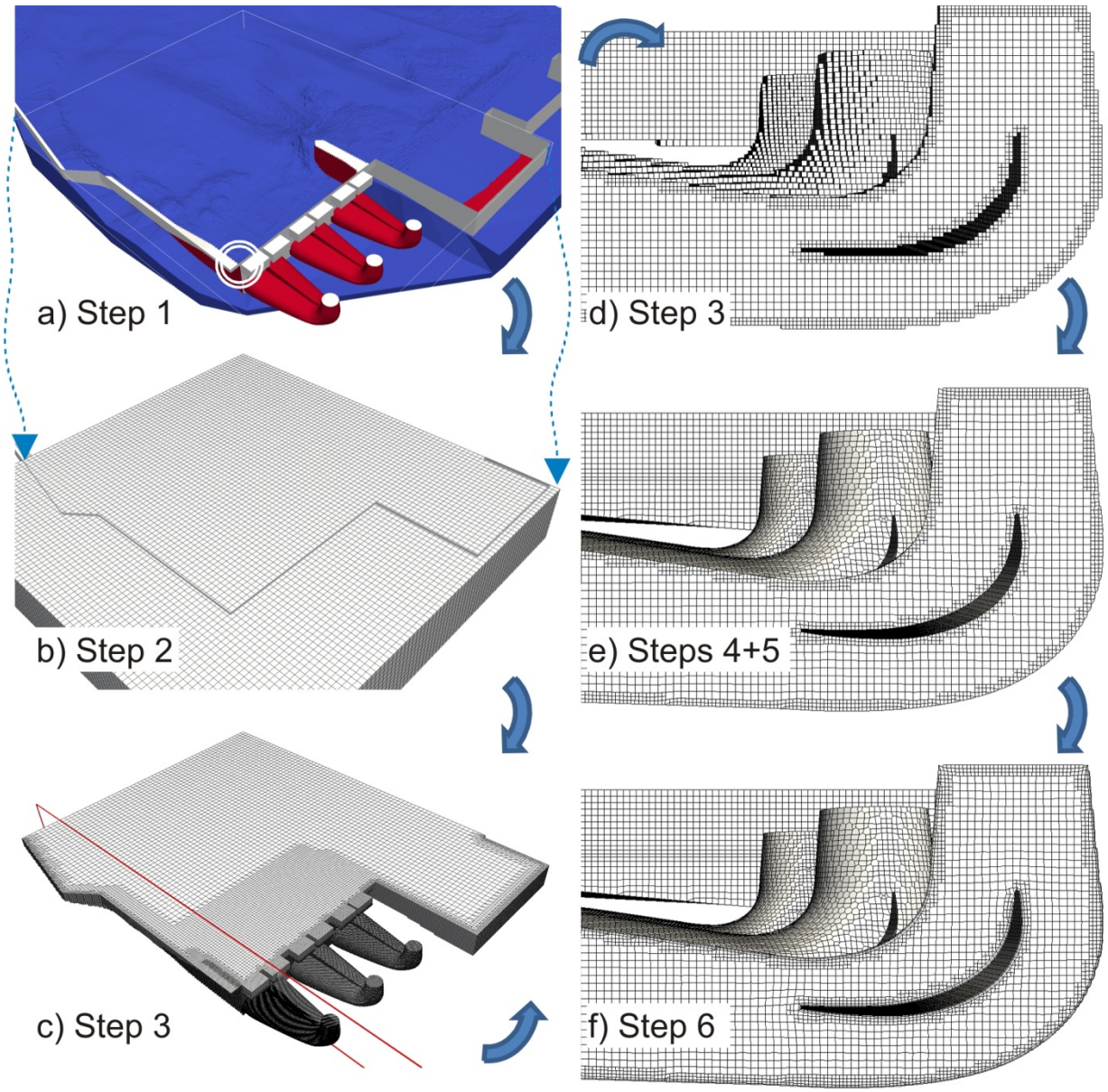

Figure 1: Steps of mesh generation: a) CAD drawings, b) blockMesh, c/d) *castellatedMesh* view and section, e) *refineMesh* and *snap*, f) *addLayers*. Flow direction from right to left.

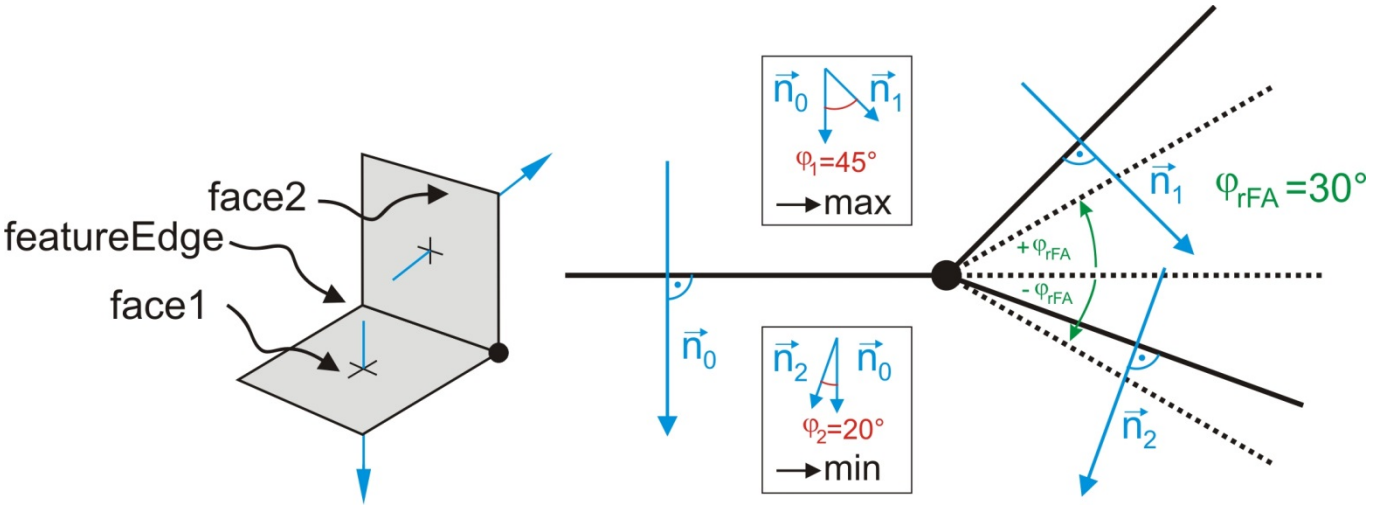

Figure 2: Principal sketch of *featureEdge*, face normals, and angles φ and φ_{rFA}.

(c) All dispensable cells that are part of *blockMesh*, but not inside the computational volume, are discarded. They are not connected to the point *locationInMesh*, as one or more STL surfaces are blocking the way. An occasional problem appears when some outer cells are still connected to this point through tiny gaps between two STLs and therefore are not removed in the step (so-called "creeping" since the mesh "creeps" through the gaps). The following countermeasures can be applied against creeping:

1. To make sure that all STLs needed are present in the folder, console output is showing no errors, and all boundaries are defined in *snappyHexMeshDict*'s sub-dictionaries *geometry* and *refinementSurfaces*.
2. To check if the point *locationInMesh* is inside the intended mesh, but not at an edge. The latter can be ensured by adding arbitrary digits, e.g.:

```
locationInMesh (10.49 -10.01 83.003);
```

3. At concave surface joints: Extension of the STLs in outside pointing direction to allow them to not only touch but intersect each other in the shared edge. This is employed here at the edges between the bottom and the vertical walls.
4. Visual search for holes with ParaView. The mesh is opened, colored by *vtkCompositeIndex* or *cellLevel*, and *Surface with Edges* is activated. It is viewed from the inside by zooming and re-picking the camera center (*Pick Center*). The searching domain can be downsized systematically using *blockMesh*. Typical problems are at curved edges, because they cannot be represented exactly by the triangle based STL mesh. If a hole is found, a local refinement box (see b) around the problematic element can help. In case it is caused by two STLs not fitting perfectly together, they should be re-exported from the CAD program with a higher resolution.
5. In some cases, creeping can be fixed by moving the basic mesh few centimeters in an arbitrary direction, whereby the STLs cut the cells at slightly other positions. However, success of this solution is random, hence it should be used as last resort only.

Following a visual check using ParaView, meshing is continued with anisotropic refinement.

3.5 *Step 4: refineMesh (anisotropic, optional)*

Inside every cell layer possibly containing a water/air interface during the simulation, sudden changes to the z resolution should be avoided. So far, only cells inside the isotropic refinement box at the turbines (see Figure 1c) have the uniform height of $\Delta z = 20$ cm. For adjustment of all further cells close to the water level, anisotropic refinement with *refineMesh* is deployed. It substitutes every affected cell just by 2^1 and not by 2^3 new cells, making the computation faster. However, depending on the mesh volume and number of simulations, it can be timesaving to refine just isotropic where needed, since the step *refineMesh* requires a high effort.

At sudden resolution jumps, numerical-induced currents may arise from the Volume of Fluid method used in the simulation. When the mesh resolution at the water/air interface increases by factor 2 as shown in Figure 3, the three cells depicted receive three differing filling states: water ($\alpha = 1$), air ($\alpha = 0$), and a mixture of both ($\alpha = 0.5$). Due to gravity acceleration g, the fluids tend to order themselves according to their density (here proportional to α), inducing balancing currents (arrows) without real influence.

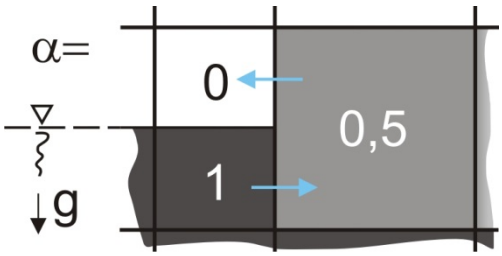

Figure 3: Creation of balancing currents between cells filled up to different states.

refineMesh is a separate tool contained in OpenFOAM and not part of *snappyHexMesh*. Its application makes most sense following *castellatedMesh*, as the cell splitting results are easier to predict for orthogonal than for non-orthogonal cells, e.g. resulting from *snap* (step 5).

The tutorial *multiphase/cavitatingFoam/les/throttle3D/* contains files *Allrun*, *refineMeshDict*, and *topoSetDict.X*, which may be used as templates. The following settings are applied to *refineMeshDict*:

```
directions (tan3); useHexTopology yes; geometricCut no; writeMesh no;
```

Multiple refinement boxes diminishing in *z* direction are defined in the *topoSetDict.X* files. It is handy to choose all point coordinates lying definitely outside the geometry either very large or small, respectively, (e.g. -999 or 999) to make sure they don't have to be adapted if the geometry. Boxes can be assembled in *topoSetDict.X* from multiple *cellSets cX* by writing:

```
{     name c0;
      type cellSet;
      action add;
      source cellToCell;
      sourceInfo {set c1;}
}
```

By using `cellSet`, it is also possible to select only cells of e.g. a certain volume.

With the code contained in `Allrun`, the files *topoSetDict.X* are renamed before execution of *refineMesh*, as only the file *topoSetDict* (without appended digit) can be called.

The coarser ranges get adapted to the existing finer range step by step (Figure 4). From experience, many iterations are necessary to define every border of the refinement boxes satisfactory.

Generated results may be loaded in ParaView by pressing *Refresh* without restarting the application.

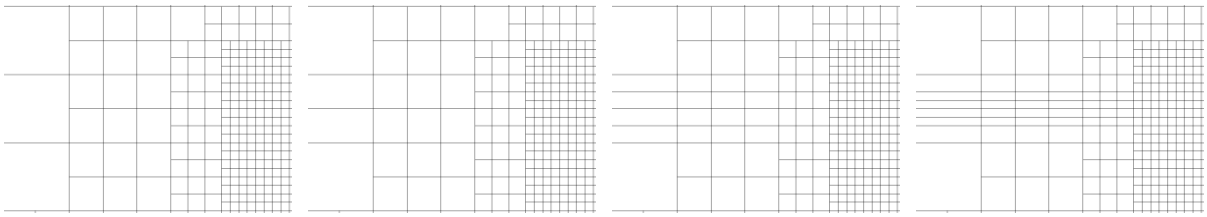

Figure 4: Stepwise anisotropic refinement.

3.6 *Step 5: snap*

After refinement, boundary cells may now be adapted to the geometry. Therefore, *castellatedMesh false* and *snap true* are set in *snappyHexMeshDict* before `snappyHexMesh` is executed. Settings differing from the template in the sub-dictionary *snapControls* are:

```
tolerance 2.0;              // Smaller skew of the inner mesh cells
nSolveIter 30;              // Improved fit to STLs
nFeatureSnapIter 10;        // Exact edge fit, optical improvement
implicitFeatureSnap true;   // Exact edge fit, optical improvement
```

Exact edge fit (*nFeatureSnapIter*) is not needed here for solution accuracy, but allows an optically improved post-processing. *implicitFeatureSnap* applies the *resolveFeatureAngle* defined above. The result is shown in Figure 1e.

3.7 *Step 6: addLayers (optional)*

The addition of anisotropic wall layers (also: prismatic layers) along certain boundaries enables the calculation of variables such as velocity very close to the wall with comparatively little additional grid cells. This is relevant, for example, when separation points or wall shear stresses are to be determined, e.g. by means of low-Re modeling (Menter, 2009). In the present example a high-Re modeling approach (with wall functions) is used. Layers are inserted only to ensure a smooth flow pattern in wall proximity in the draft tubes. For this, the following settings are changed in *snappyHexMeshDict* contrary to the template:

```
relativeSizes false;
layers { tube_1_patch0
           { nSurfaceLayers 2;
             finalLayerThickness 0.05;
             expansionRatio 2.0;
           }
       }
minThickness 0.001; featureAngle 360; nSmoothThickness 5;
maxThicknessToMedialRatio 1; nLayerIter 20;
```

Precedent calling *snappyHexMesh*, *snap false* and *addLayers true* are set. By setting *relativeSizes false*, required length units may be set directly and are easier to envision. Edge length in the mesh far from the wall amounts to 0.10 m in *tube_1*, to half of that (0.05 m) in the coarsest layer (*finalLayerThickness*), and to 0.025 m in the layer closest to the wall (Figure 1f). *minThickness* is practically switched off by the small value 0.001 m. For parameter *featureAngle* φ_{fA}, a high value is preferred contrary to *resolveFeatureAngle* φ_{rfA} (see *castellatedMeshControls*), because layers are only inserted if $\varphi < \varphi_{fA}$. Despite the equal name, *featureAngle* φ_{fA} must not be confused with *featureAngle* φ between the normal vectors of two planes. According to own tests, in contrast to φ_{rfA} also values of φ_{fA} above 180° do influence the result, therefore maximum value 360° is set. An explanation of this behavior has not been found yet.

nSmoothThickness and *nLayerIter* are both reduced to cut the mesh generation time. *maxThicknessToMedialRatio* is set to maximum value *1* to minimize thinning of two wall layers approaching each other in a pointed angle between two walls.

A common mistake is to use the STL names defined in *geometry* in *addLayers*, too. In fact needed are the names from *5/polyMesh/boundary/* that often have additions like *_OBJECT*, *_patch0*, or *_CADfix* from the first and last line of the STL. If ASCII STLs are being used, they can be removed optionally in the STL file.

Especially when applied to multi-curved surfaces such as the draft tubes, *addLayers* with default settings often generates not any or faulty layers. As a first counter-measure, near-wall isotropic refinement should be increased in *refinementSurfaces*. If nevertheless, even with the settings given above, no continuous layers arise, all entries in *meshQualityControls* may be deactivated by setting them to maximum respective negative values for debugging purposes. Afterwards, they are activated again line-by-line under observation of the results to find the cause of the problem. A lasting change of these parameters is discouraged, as only symptoms, but no reasons for mesh problems can be fought through it and mesh quality decreases.

In general, layers can be omitted in hydraulic engineering problems with little or no influence on the bulk flow from boundary layer effects or roughness. Due to insertion of the layers, cell deformations may form inside the computational domain depending on number and thickness of the layers, since additional cells demanding space are created. As, in this case, the flow between two cells does not point through the center of their shared face, interpolation gets necessary during the simulation causing inaccuracy. Other mesh generators like e.g. that of StarCCM+ avoid this downside by reserving a volume close to the walls for layers preceding to the generation process (CD-Adapco, 2013). This requires an integrated workflow preventing selective re-iteration of single steps.

3.8 *Step 7: checkMesh*

After optional layer insertion, for a final check of the mesh, the command is executed:

```
checkMesh -latestTime | tee log.cm.6
```

From the output, cell count and agreement with predefined quality criteria are gained. Cells with quality problems are stored into single *sets*, for instance in *skewFaces* for high skew cells (conceivable as geometrical deviation from the ideal form, e.g. deviation from a rectangular cuboid in the case of a general hexahedron). Using

```
foamToVTK -faceSet skewFaces -latestTime
```

a VTK file is generated, which can be opened with ParaView to determine the positions of the bad cells in the mesh (view as *Wireframe*). It can be tried to improve the quality by global and local refinements. If the cells reside in areas irrelevant for the task and/or are few by number, they can be tolerated as well.

After completion of the seven work steps, the final mesh is copied from *6/polyMesh/* in the mesh folder to *constant/polyMesh/* in the computation folder. From own experience, boundary conditions using free water surface are more stable if this is close to $z = 0$, since hydrostatic pressure gets very small. Therefore, the mesh, created in natural height, is translated at last in the computation folder by an even amount in z direction using the command

```
transformPoints -translate '(0 0 -80)'
```

REFERENCES

CD-Adapco (2013). User Guide Star-CCM+ Version 8.02.

Jackson A. (2012). http://openfoamwiki.net/images/f/f0/Final-AndrewJacksonSlides OFW7.pdf (called 17.04.2014).

Menter F.R. (2009). Review of the shear-stress transport turbulence model experience from an industrial perspective. Int. J. Comput. Fluid Dyn. 23, No. 4, pp. 305 – 316.

Spalart P.R. (2001). Young-Person's Guide to Detached-Eddy Simulation Grids. NASA Technical Report NASA/CR-2001-211032.

OpenFOAM Foundation (2014). http://www.openfoam.org/docs/user/ (called 17.04.2014).

Weller H.G., Tabor G., Jasak H. & Fureby C. (1998). A Tensorial Approach to CFD using Object Orientated Techniques, Comput. Phys., Vol. 12, No. 6, pp. 620 – 631.