Report on advanced mesh generation using BlockMesh

Table of Contents

1. BlockMesh

1.1. General information

BlockMesh is a tool for mesh generation. Generated mesh is ready to be used in OpenFoam without further modifications. However, some simulation parameters, like AMI, GGI, or FSI, requires additional modifications of a mesh. BlockMesh runs by typing *blockMesh* in a terminal window, and all mesh parameters are taken from blockMeshDict file, located at *system* folder for OpenFoam.org version or *constant/polyMesh* folder for OpenFoam.com version.

Coming from the name, BlockMesh generates mesh by creating and subdividing blocks. Each block is defined by exactly 8 vertices. Each edge could be straight, arc, or spline lines. Despite it is possible to create a block from less than 8 vertices, these types of blocks would not be discussed in this report due to problems with accuracy and stability. An example of block could be seen in figure 1. This block is defined by x_1, x_2, and x_3 coordinates, which are not always align with x, y, and z coordinates of a domain. Also, each vertex has its ID, starting with 0 and ending with 7. These vertices should be specified in the same order as their ID. The IDs were given by the right-hand rule, where x_1 defines the edge from vertex 0 to vertex 1, x_2 defines edge from vertex 1 to vertex 2, and x_3 defines direction of the right-hand rule (from plane 0123 to 4567). The following two points should be mentioned:

1. Vertex 0 and vertex 4 should be connected by the same edge.
2. Axes x_1, x_2, and x_3 are not the same axes, as x, y, and z. This means that block defining axes can face any direction.

Then, each edge should be subdivided required number of times to create a mesh. If two blocks have common vertices, these vertices could be defined only once and could be used by both blocks. In this case, number of subdivisions should match between two blocks. In cases, when it is impossible to have matching number of subdivisions, two edges or faces could be merged by *stitchMesh* command. Both methods would be described further in the report.

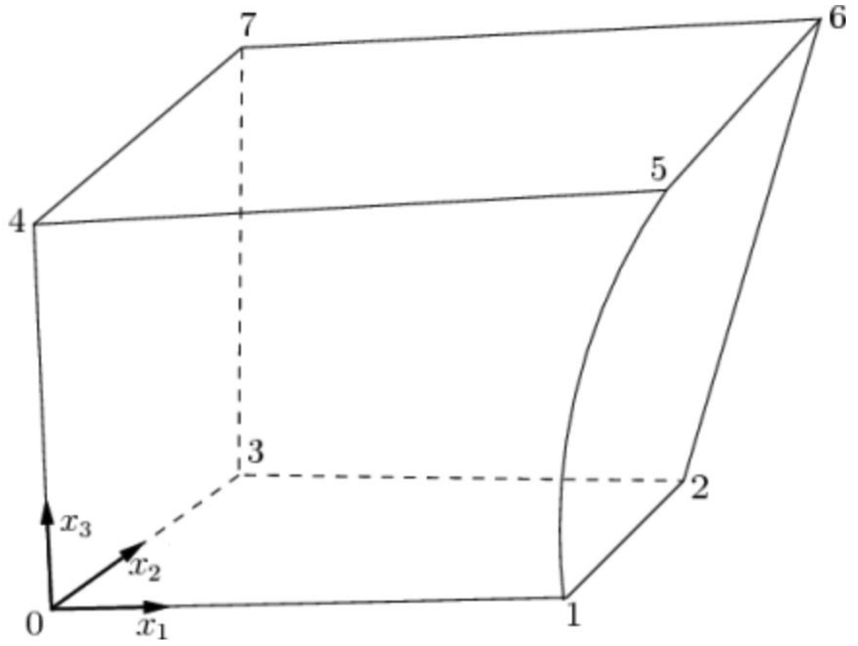

Figure 1. Defining block's coordinates

More information could be found at the official user guide from OpenFoam, available at the following link: https://cfd.direct/openfoam/user-guide/v6-blockmesh/

1.2. Vertices

The first thig to define in a *blockMeshDict* file is vertices section. In this section, each vertex is defined line by line, where the first vertex has ID 0, second vertex has ID 1 etc.

1.3. Blocks

The blocks are defined by the following line:

$$hex\ (0\ 1\ 2\ 3\ 4\ 5\ 6\ 7)\ zoneName\ (10\ 10\ 10)\ simpleGrading\ (1\ 1\ 1)$$

The *hex* shows that this line defines a block. Each block defining line will start with *hex*. The next 8 numbers are the vertex IDs. The *zoneName* is a name of a zone. This could be used to rotate a region by a zone name. Then, the next 3 numbers define number of subdivisions along x_1, x_2, and x_3 axes. The final part of the line is grading type and amount of grading along x_1, x_2, and x_3 axes. The grading defines how close subdivisions would be located to one of the axes.

1.4. Edges

This section is not mandatory. By default, all edges are assumed to be straight. However, it is possible to create arcs and BSpline curves. Following lines define arc and BSpline edges respectively:

$$arc\ 0\ 1\ (0\ 0\ 0)$$

$$BSpline\ 0\ 1\ \big((0\ 0\ 0)\ (1\ 1\ 1)\big)$$

The first word in each line defines the type of edge. The next two numbers are vertex IDs, and the last item are coordinates, through which line should go.

1.5. Boundary

Almost every face should be named because merging faces and setting boundary conditions are relying on face names. Common boundary is defined by following line:

$$WallName\ \{type\ wall;faces\ ((0\ 1\ 2\ 3)\ (4\ 5\ 6\ 7))\}$$

The first word is a name of a patch, then, the type of the patch should be identified, and lastly, all faces should be listed inside *faces*.

2. Implementation for a wind turbine

2.1. Preparations

The first step is to extract airfoils. The extracted airfoils from NREL 5MW wind turbine could be seen in figure 2. Later, each airfoil would be used to extract coordinates for block defining vertices and edges.

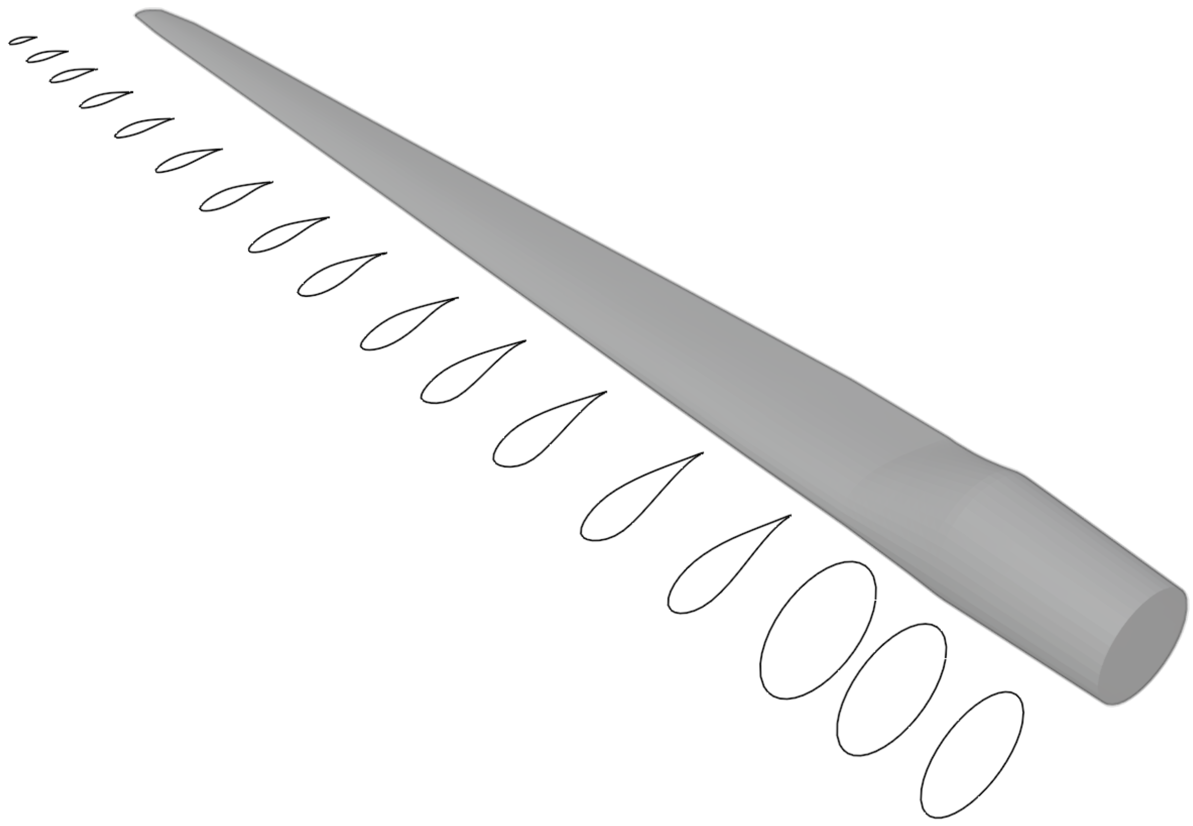

Figure 2. Extracted airfoils

2.2. Blade

There are two common airfoil meshing types: c-mesh and o-mesh. The c-mesh would be used in this report due to higher mesh quality. The example of a c-mesh could be viewed in figure 3. This mesh is created by 6 blocks (figure 4). Moreover, figure 4 shows that internal edges should be turned to BSplines in order to create airfoil profile. The creation of blade follows these steps:

$$Write\ vertices \rightarrow create\ blocks\ using\ vertices\ ID \rightarrow$$
$$define\ BSpline\ curves \rightarrow name\ boundaryes$$

To mesh other blades, created vertex coordinates should be duplicated and rotated by 120 degrees. Because all vertices are rotated in 2D plane, the following formulas could be used to calculate new vertices coordinates:

$$x' = xcos\theta - ysin\theta$$

$$y' = ycos\theta - xsin\theta$$

Thus, we can take existing coordinates and duplicate them through the equation above. As a result, new coordinates with new IDs are obtained. Then, new block would be created with new vertex IDs.

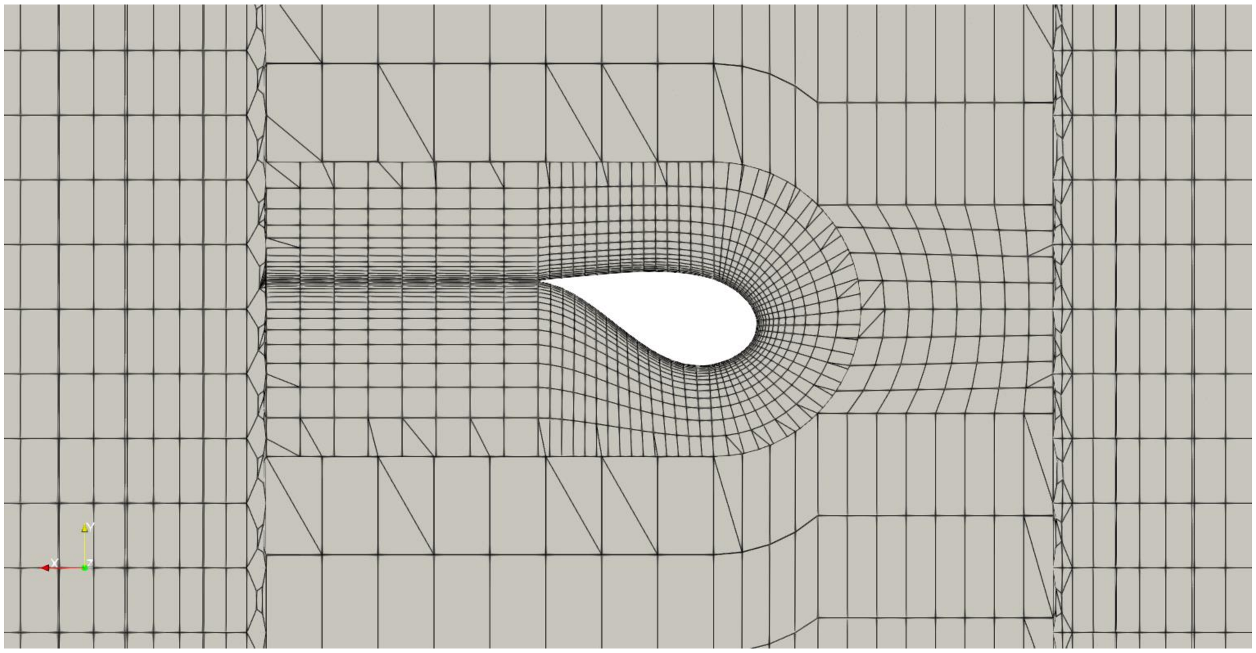

Figure 3. Example of a C-mesh

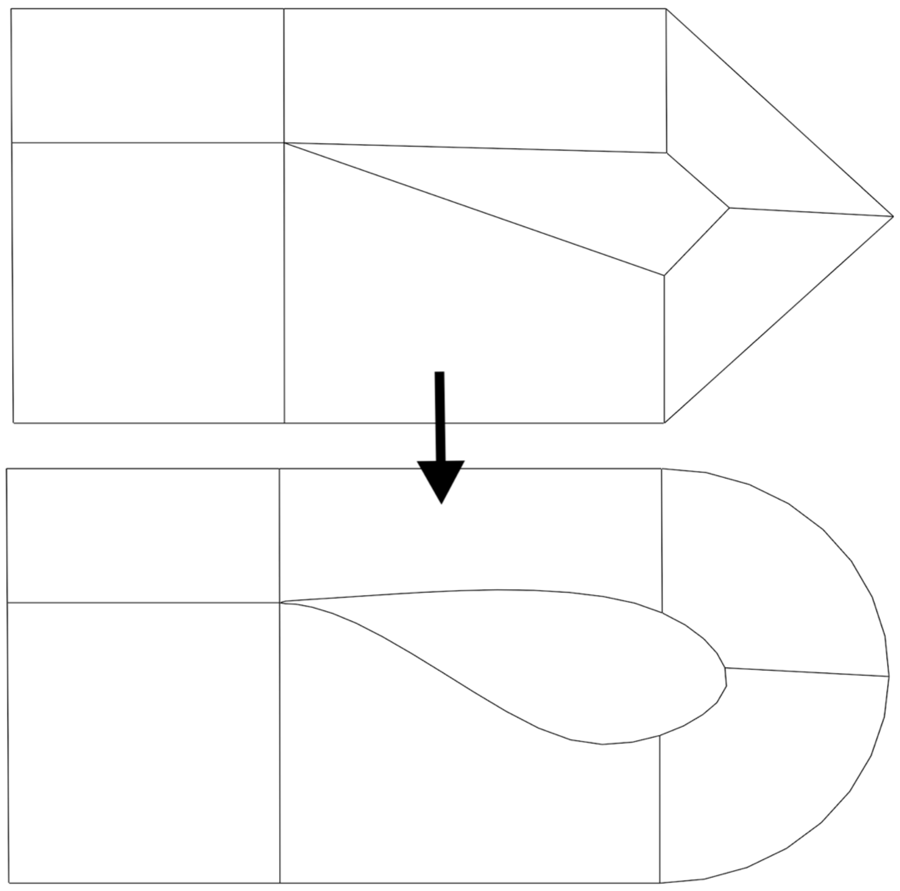
Figure 4. Core blade blocks

2.3. Hub

After blade meshes were created, they should be connected to a hub. In the figure 5, the connection of three blades to the hub could be seen. All blades were merged with the hub by using *stitchMesh* command. The stitched faces could be clearly seen in the figure below.

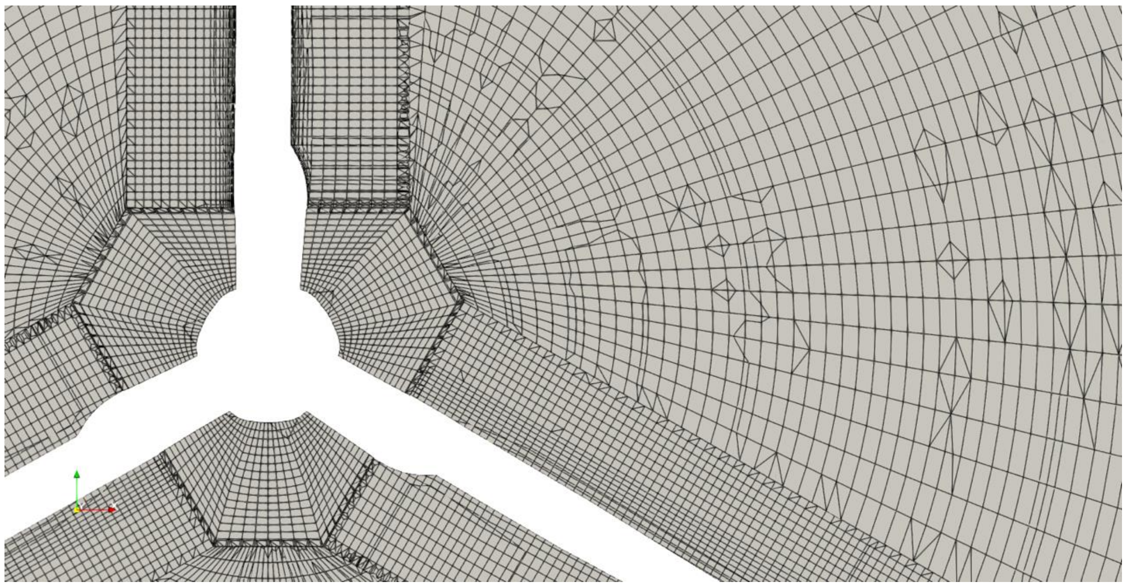
Figure 5. Blade to hub connection

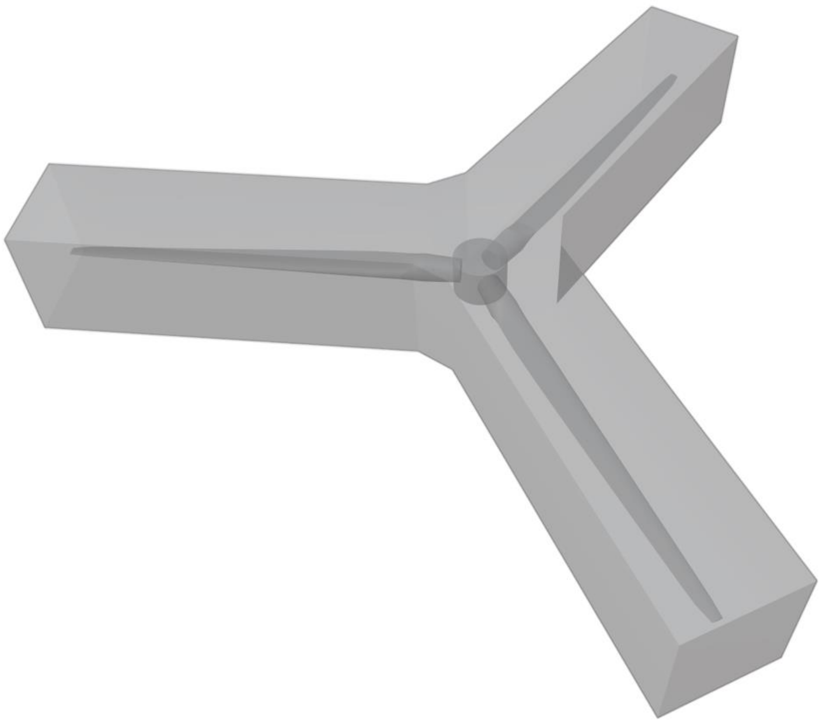
Figure 6. Blade and Hub blocks

2.4. Fillers

The last part is to create fillers between blades. An example of a filler is highlighted by orange lines in figure 7. This filler consists of a single block and could be compared to the real mesh at figure 8. The outer edges of the filler then should be curved to create a cylindrical shape. This procedure is repeated required amount of time.

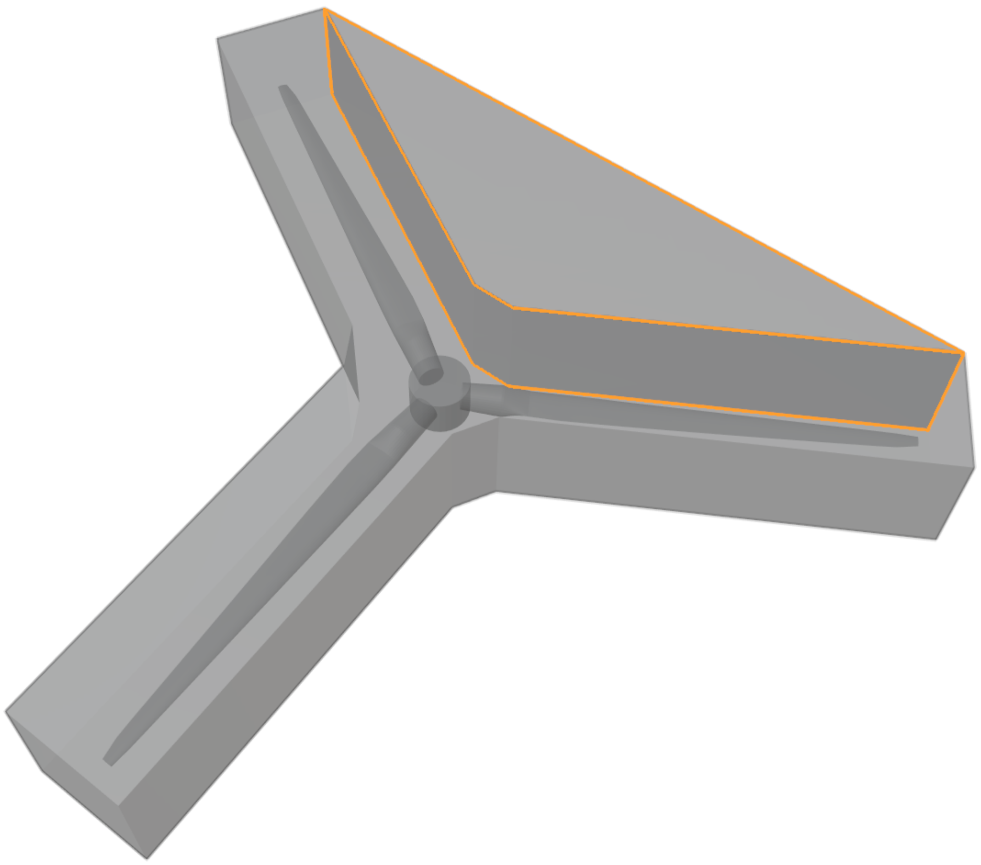

Figure 7. Mesh filler

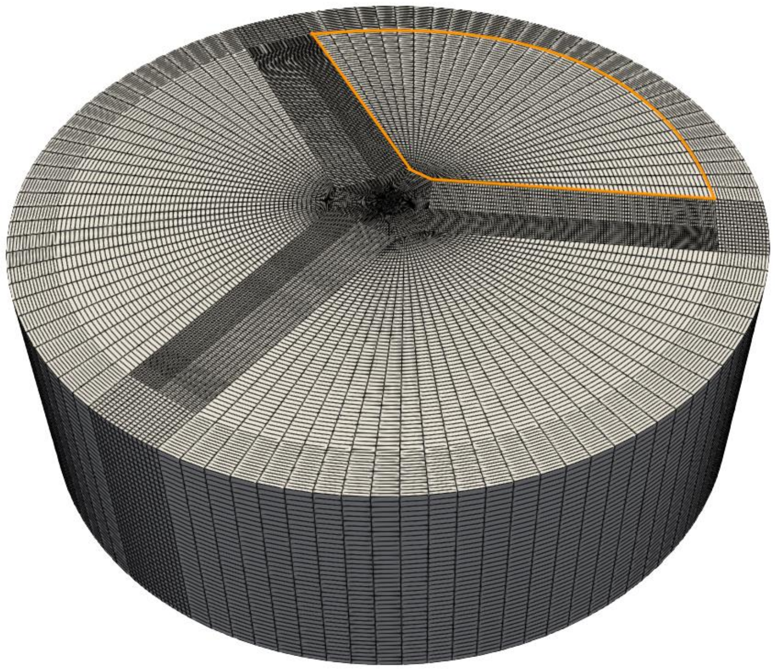

Figure 8. Real mesh

2.5. Code

All codes, input and output files could be found at the following link:
https://github.com/di2325/bladeToBlockMesh

3. Creating final mesh

3.1. CFD mesh

To create the final mesh, the rotating mesh (figure 8) and stationary mesh (figure 9) should be combined into one mesh. Initially, rotating mesh and stationary mesh are located in different case folders: *inner_mesh* and *outer_mesh*. To combine them into one mesh, the following command should be used:

```
mergeMesh inner_mesh outer_mesh -overwrite
```

Additionally, the boundary between rotating and stationary mesh should be replaced with GGI. This should be done by changing *constant/polyMesh/boundary* file, where required boundaries should be changed according to the following example, where *frontIn* and *frontOut* are two GGI patches:

```
frontIn
{
    type            ggi;
    nFaces          228;
    startFace       2983474;
    shadowPatch     frontOut;
    zone            frontInZone;
    bridgeOverlap   true;
}
frontOut
{
    type            ggi;
    nFaces          17286;
    startFace       3033044;
    shadowPatch     frontIn;
    zone            frontOutZone;
    bridgeOverlap   true;
}
```

From above example, GGI requires zones to be defined. These zones could be created by the following commands

```
setSet -batch setBatchGgi
setsToZones -noFlipMap
```

Where *setBatchGgi* file has following content:

```
faceSet frontInZone new patchToFace frontIn
faceSet frontOutZone new patchToFace frontOut
```

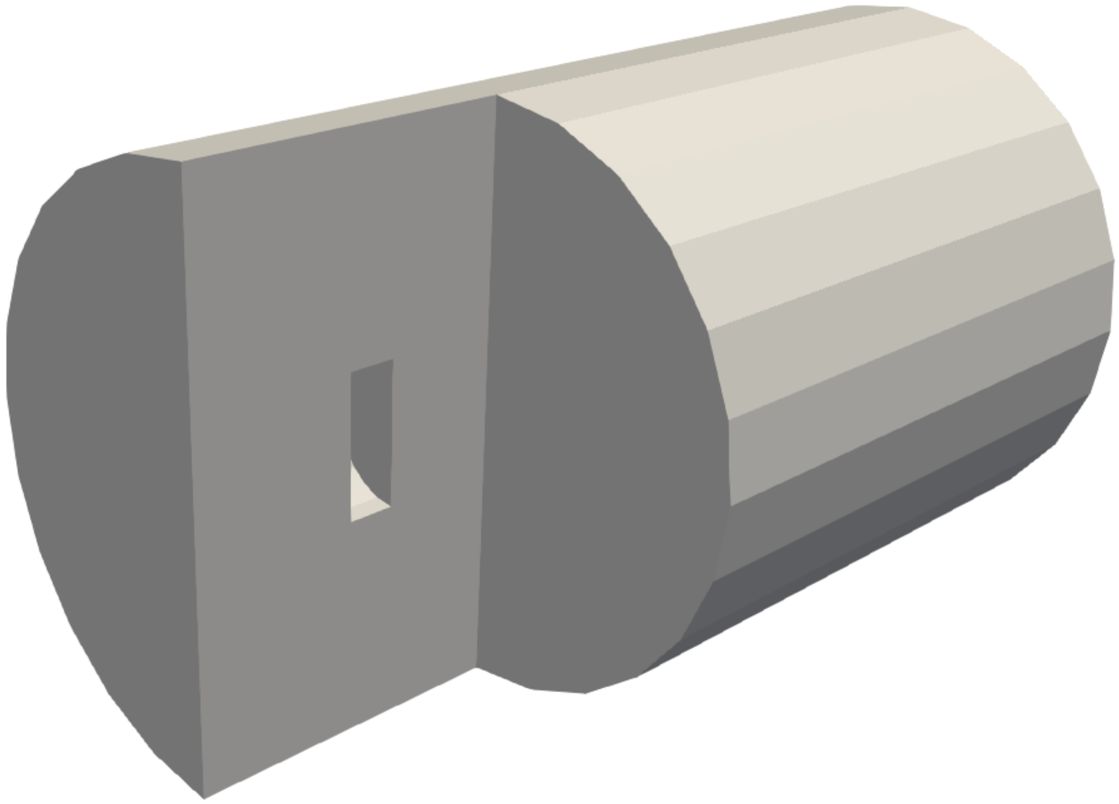

Figure 9. Outer mesh

3.2. FSI mesh

To create FSI mesh, the CFD mesh should be placed into *constant/fluid/polyMesh* folder, and solid mesh into *constant/solid/polyMesh* folder. The interpolation between fluid and solid meshes are controlled through *fsiProperties* file, located at the *constant* folder and containing names of fluid and solid patches. This file has following content:

```
fluidSolidInterface    Aitken;

AitkenCoeffs
{
        solidPatch                 blade;
        fluidPatch                 blade;
        relaxationFactor           0.5;
        outerCorrTolerance         1e-6;
        nOuterCorr                 30;
        coupled                    yes;
        interfaceTransferMethod    GGI;
}
```