6. Fluid dynamics of a Very Light Aircraft

6.0.18 Description of the case

The last chapter of the guide describes a completely different case to be solved. While up to now only two-dimensional fluid mechanics problems have been solved, the current chapter encompasses the simulation of a real 3D body in a free stream. More specifically, this body is a very light aircraft flying at sea level conditions.

Unlike previous chapters, the mesh of the current case is going to be created using an external surface generated with a 3D CAD design software. Nevertheless, the user can follow the tutorial using a geometry created by himself. As the main steps are common for generic geometries, it will only be necessary to take into consideration particular instructions which may depend on the specifications of each case. It implies that standard bodies in a free stream (a sphere, an automobile, etc.) will be suitable to be simulated according to the instructions and methods shown in Chapter 6.

6.0.19 Hypotheses

- Incompressible flow
- Turbulent flow
- Newtonian flow
- Negligible gravitatory effects
- Sea level conditions
- RAS turbulence modelling with wall functions
- Non-static components of the aircraft are not included (as for instance the propeller), as well as the landing gear

- Aircraft flying at $\alpha = 0$

6.0.20 Physics of the problem

The problem encompasses a very light aircraft flying at a speed of $V = 45$ m/s at sea level. As the medium is air ($\nu = 1.5 \times 10^{-5}\ m^2$/s), the Reynolds number is

$$Re = \frac{Vc}{\nu} = \frac{45 \cdot 1.276}{1.5 \times 10^{-5}} = 38.28 \times 10^5$$

where c is the mean aerodynamic chord of the wings and is equal to 1.276 m. The mean aerodynamic chord is the chord of a rectangular wing which has the same area, aerodynamic force and position of the pressure center for a given angle of attack as the original. The very light aircraft that it is going to be used for the simulation does not belong to any real aeronautics company; it was designed by the author of the guide during a course taught at the university (and thus it might contain flaws).

The aircraft is shown at Figure 6.1.

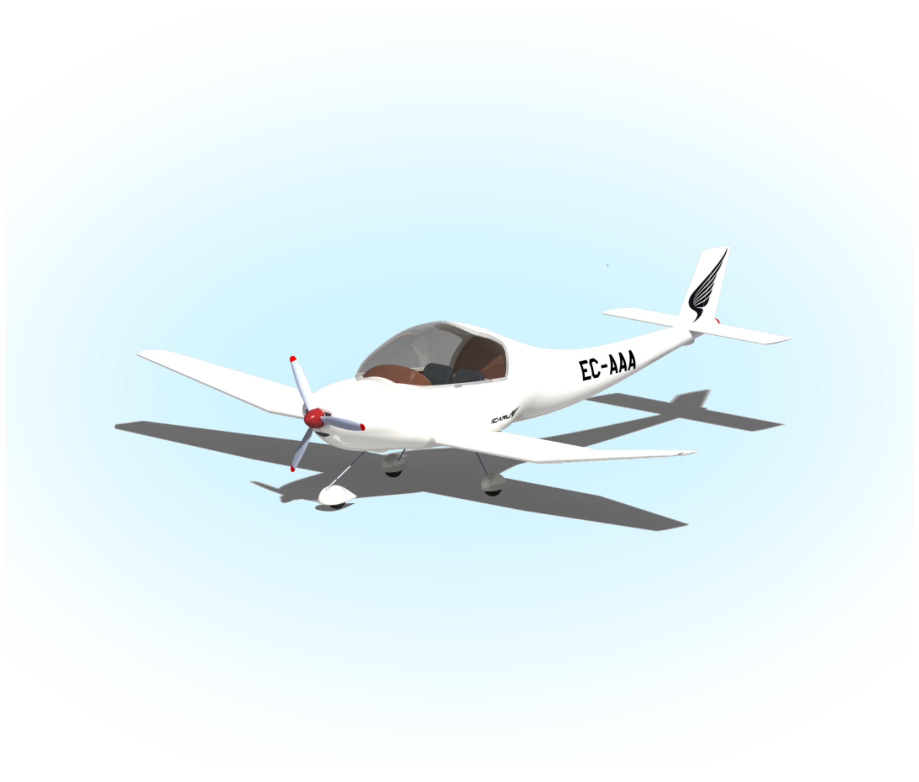


Figure 6.1: Aircraft used in the simulation of the *Very Light Aircraft* case

The aircraft measures and geometrical properties are:

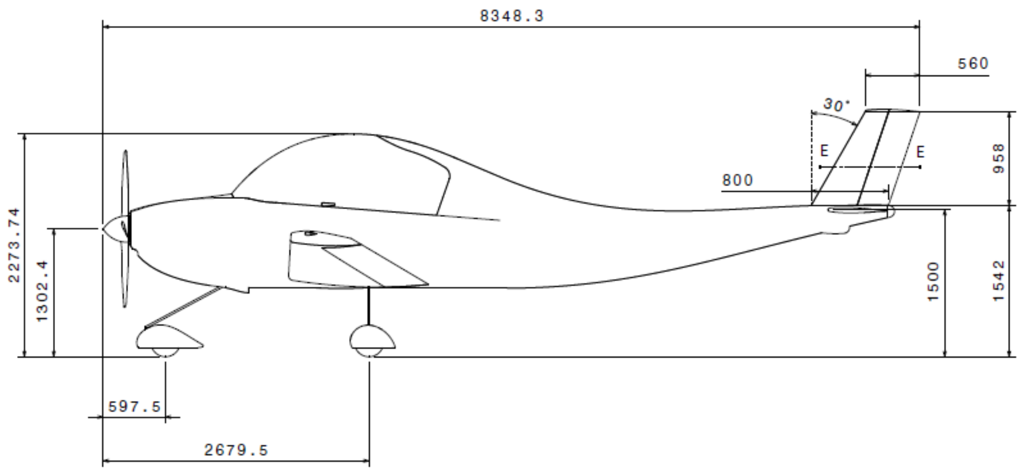


Figure 6.2: Measures of the aircraft used in the simulation of the *Very Light Aircraft* case

The problem statement is shown at Figure 6.3.

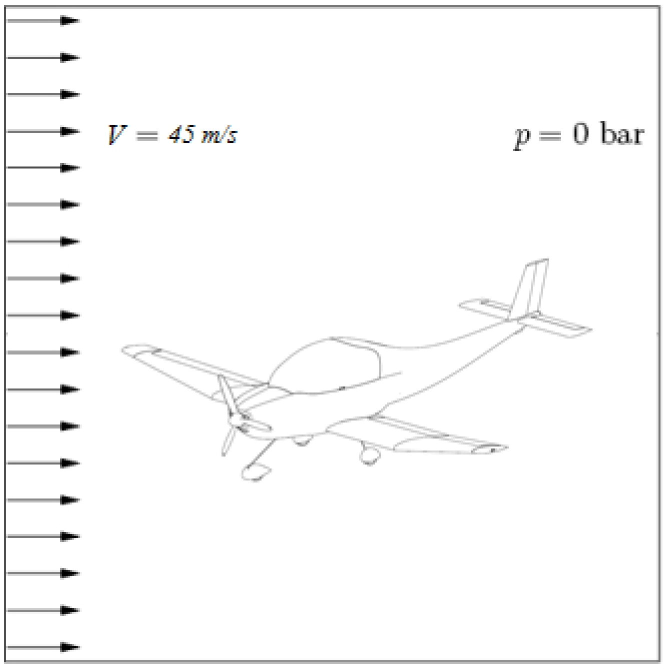


Figure 6.3: Very light aircraft flying at 45 m/s and ambient pressure

In the current chapter there is no easy analytical solution to describe the behaviour of the fluid. However, it is necessary to keep in mind the main equations involved in the problem:

The continuity equation,

$$\nabla \cdot \mathbf{U} = 0 \tag{6.1}$$

The momentum equation,

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 \mathbf{U} \tag{6.2}$$

As it can be seen at Figure 6.2, the aircraft presents a length of 8.3 m. Unlike previous cases, it cannot be directly meshed and treated because of its large size (it would be necessary to create an enormous number of cells). This is the reason why the geometry will be rescaled 100 times. It can be understood as if the analysis would be done to a prototype introduced into a wind tunnel.

Consequently, it is necessary to do a dimensional analysis to find out the dimensionless numbers involved in the problem. It ensures that the flow conditions are consistent to guarantee that the results of the 100 times rescaled aircraft are the same as if the real aircraft would be flying in the sky.

The dimensionless variables of Equations 6.1 and 6.2 are:

$$\tilde{\mathbf{x}} = \frac{\mathbf{x}}{c}$$
$$\tilde{\mathbf{U}} = \frac{\mathbf{U}}{V}$$
$$\tilde{p} = \frac{p}{\rho V^2}$$
$$\tilde{t} = \frac{t}{c/V}$$

By introducing them, the dimensionless equations of mass and momentum are:

$$\tilde{\nabla} \cdot \tilde{\mathbf{U}} = 0 \tag{6.3}$$

$$\frac{\partial \tilde{\mathbf{U}}}{\partial \tilde{t}} + \tilde{\mathbf{U}} \cdot \tilde{\nabla} \tilde{\mathbf{U}} = -\tilde{\nabla}\tilde{p} + \frac{\mu}{\rho c V}\tilde{\nabla}^2 \tilde{\mathbf{U}}$$

which can be rewritten as:

$$\frac{\partial \tilde{\mathbf{U}}}{\partial \tilde{t}} + \tilde{\mathbf{U}} \cdot \tilde{\nabla} \tilde{\mathbf{U}} = -\tilde{\nabla}\tilde{p} + \frac{1}{Re}\tilde{\nabla}^2 \tilde{\mathbf{U}} \tag{6.4}$$

It can be seen that the only dimensionless number involved in the problem is the

Reynolds number. Therefore, maintaining *Re* (comparing the real aircraft flying in the sky with the prototype analysed in the wind tunnel), the behaviour of the flow around them may be comparable and the case preserves its coherence. As a consequence, if the mean aerodynamic chord has been reduced 100 times, the kinematic viscosity will be also reduced this quantity:

$$Re = \frac{Vc}{\nu} = \frac{45 \cdot 0.01276}{1.5 \times 10^{-7}} = 38.28 \times 10^5$$

6.0.21 Pre-processing

The following codes contain the information to simulate the case with $Re = 38.28 \times 10^5$ using simpleFoam and the SST k-w turbulence model. The main differences with previous cases come from the fact that the *constant* and *system* directories will include new features. Therein it will be contained the dictionaries and files required to treat and mesh the geometry.

It is highly recommendable to follow the current tutorial bearing in mind the structure of directories that the case is going to contain (shown at the end of Section 6.0.21.6), as the case setting changes significantly. The scheme of directories is similar as the one used in the official *OpenFOAM*® motorBike case (located within *incompressible/simpleFoam*). It may help the user to figure up the case resolution, before and whilst running it.

The case directory is named aircraft and will be located within FoamCases.

6.0.21.1 Mesh generation

Handling surface meshes

First of all, create empty *constant* and *system* directories within aircraft. In *constant*, there will be two new directories, *polyMesh* and *triSurface*, this second containing the case geometry saved with the .stl extension. In the current tutorial, the file with the geometry is going to be named aircraft.stl.

STL (STereoLithography) is a file format native to the stereolithography CAD software created by 3D Systems. This file format is supported by many other software packages; it is widely used for rapid prototyping and computer-aided manufacturing. STL files describe only the surface geometry of a three-dimensional object without any representation of color, texture or other common CAD model attributes.

Next, as it was done in Chapter 5, a *dummy* *controlDict* file is needed to be included.

Caution:

> For coherence with further explanations, set deltaT and writeInterval within *controlDict* equal to 1

Now the user has to include a new file: *surfaceFeatureExtractDict*. It is contained within *system* and is used to extract feature edges from tri-surfaces. The file includes:

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.0                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     object      surfaceFeatureExtractDict;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16 
17 aircraft.stl
18 {
19     // How to obtain raw features (extractFromFile || extractFromSurface)
20     extractionMethod    extractFromSurface;
21 
22     extractFromSurfaceCoeffs
23     {
24         // Mark edges whose adjacent surface normals are at an angle less
25         // than includedAngle as features
26         // - 0  : selects no edges
27         // - 180: selects all edges
28         includedAngle   120;
29     }
30 
31     // Write options
32 
33         // Write features to obj format for postprocessing
34         writeObj                yes;
35 }
36 // ************************************************************************* //
```

Advice:

> At line 17 the name has to match the one used for the file located within *constant/triSurface*

Next, the user has to execute the dictionary to extract the features. Type within the case:

```
surfaceFeatureExtract
```

Note that a new directory and a new file have been generated within *constant* and *constant/triSurface* respectively.

At this moment the user can launch ParaView to view the aircraft shape, opening the .stl file contained in *constant/triSurface*.

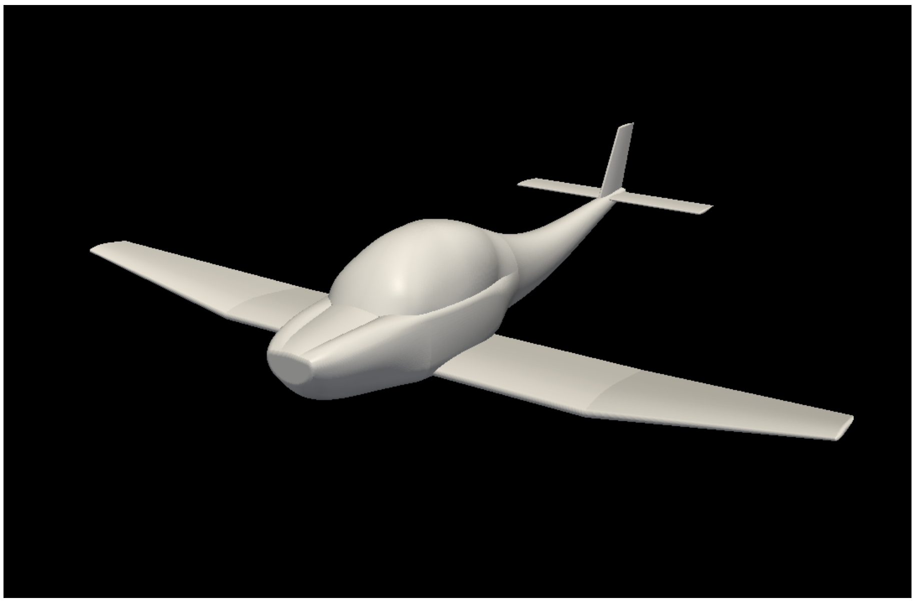


Figure 6.4: Aircraft' STL surface used in the simulation of the *Very Light Aircraft* case

Generating a background mesh

Before meshing the aircraft itself, it is first necessary to create a background mesh. It can be simply done using blockMesh and *blockMeshDict*. The aim is to create a big rectangular prism encompassing the whole aircraft. This prism will become the fluid domain of the case. The inside of the aircraft will be removed using further instructions transforming the aircraft into a real obstacle for the flow.

Caution:

> As it is necessary to analyze the flow downstream more than the flow upstream, the distance from the inlet patch of the prism to the aircraft's cabin will be shorter than the distance between the outlet patch and the aircraft's tail

Advice:

> Try to generate the cells of the background mesh with an aspect ratio as close to 1 as possible

So, to create an adequate mesh for the aircraft, copy the following *blockMeshDict* code within *constant/polyMesh* and type:

blockMesh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.01;

vertices
(
    (-30 -8 -10)
    ( 10 -8 -10)
    ( 10  8 -10)
    (-30  8 -10)
    (-30 -8  10)
    ( 10 -8  10)
    ( 10  8  10)
    (-30  8  10)

);

blocks
(
    hex (0 1 2 3 4 5 6 7) (32 13 16) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    inlet
    {
            type patch;
    faces
    (
        (1 2 6 5)
    );
      }

      outlet
      {
            type patch;
    faces
    (
        (0 4 7 3)
    );
      }

      bottom
      {
            type slip;
    faces
    (
        (0 3 2 1)
    );
      }

      top
      {
            type slip;
    faces
    (
              (4 5 6 7)
    );
      }

      frontAndBack
      {
            type slip;
    faces
    (
              (0 1 5 4)
              (2 3 7 6)
    );
      }

);

mergePatchPairs
(
);

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

Figure 6.5 shows the result of combining the mesh generated with the initial STL surface:

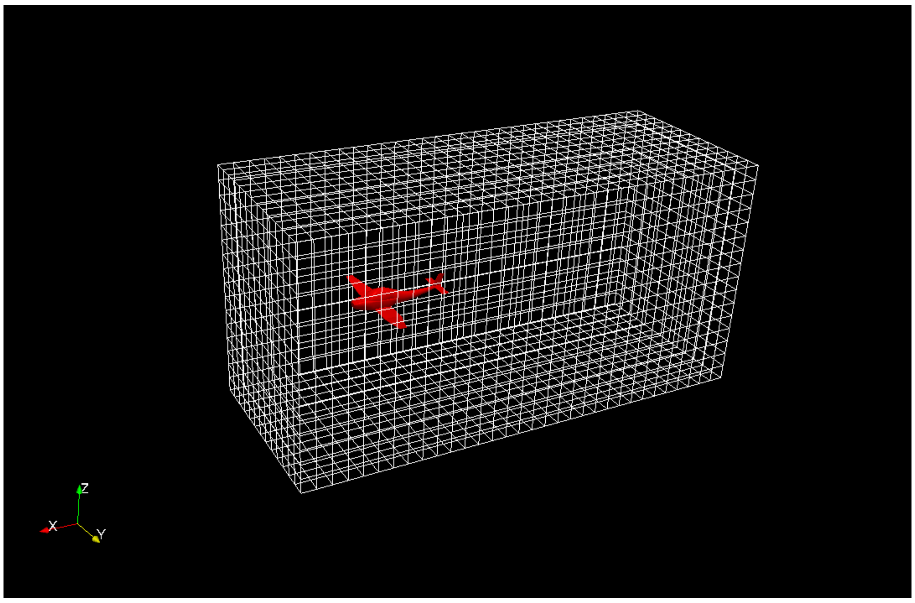


Figure 6.5: Aircraft' STL surface contained within the mesh generated with blockMesh in the *Very Light Aircraft* case

Advice:

> To obtain the results shown in Figure 6.5 it is necessary to include *dummy* files of *fvScheme* and *fvSolution*

Using snappyHexMesh

To mesh, the snappyHexMesh application is going to be used, being controlled by the *snappyHexMeshDict* file located in *system*. The main parts of the dictionary are:

- 3 switches to control the individual meshing processes
- A geometry subdictionary for the surface geometry used in the meshing
- 3 subdictionaries, one for each meshing process
- A subdictionary for the control of the quality criteria

The *snappyHexMeshDict* file that it is going to be used for the meshing of the aircraft is:

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|   \\      /    O peration        | Version:   2.2.0                                        |
|    \\   /      A nd              | Web:       www.OpenFOAM.org                             |
|     \\/        M anipulation     |                                                         |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Which of the steps to run
castellatedMesh true;
snap            false;
addLayers       false;

// Geometry. Definition of all surfaces. All surfaces are of class
// searchableSurface.
// Surfaces are used
// - to specify refinement for any mesh cell intersecting it
// - to specify refinement for any mesh cell inside/outside/near
// - to 'snap' the mesh boundary to the surface
geometry
{
    aircraft.stl //STL filename where all the regions are added
    {
        type triSurfaceMesh;

        regions
  {
    patch0              //Named region in the STL file
    {
      name aircraftPatch;   //User-defined patch name. If not provided will be
          <name>_<region>
    }
  }
    }

    refinementBox //Geometry to refine. Entities: Box, Cylinder, Sphere, Plane
    {
        type searchableBox;
        min (-0.3 -0.06 -0.04);
        max (0 0.06 0.04);
    }
};

// Settings for the castellatedMesh generation.
castellatedMeshControls
{

    // Refinement parameters
    // ~~~~~~~~~~~~~~~~~~~~~

    // If local number of cells is >= maxLocalCells on any processor
    // switches from from refinement followed by balancing
```

```
    // (current method) to (weighted) balancing before refinement.
    maxLocalCells 1500000;

    // Overall cell limit (approximately). Refinement will stop immediately
    // upon reaching this number so a refinement level might not complete.
    // Note that this is the number of cells before removing the part which
    // is not 'visible' from the keepPoint. The final number of cells might
    // actually be a lot less.
    maxGlobalCells 2000000;

    // The surface refinement loop might spend lots of iterations refining just a
    // few cells. This setting will cause refinement to stop if <= minimumRefine
    // are selected for refinement. Note: it will at least do one iteration
    // (unless the number of cells to refine is 0)
    minRefinementCells 0;

    // Allow a certain level of imbalance during refining
    // (since balancing is quite expensive)
    // Expressed as fraction of perfect balance (= overall number of cells /
    // nProcs). 0=balance always.
    maxLoadUnbalance 0.1;

    // Number of buffer layers between different levels.
    // 1 means normal 2:1 refinement restriction, larger means slower
    // refinement.
    nCellsBetweenLevels 3;


    // Explicit feature edge refinement
    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    // Specifies a level for any cell intersected by explicitly provided
    // edges.
    // This is a featureEdgeMesh, read from constant/triSurface for now.
    // Specify 'levels' in the same way as the 'distance' mode in the
    // refinementRegions (see below). The old specification
    //      level   2;
    // is equivalent to
    //      levels  ((0 2));

    features //Disabled because the refinement will be surface-based (as interal
        aircraft cells are removed)

    (
        //{
        //    file "aircraft.eMesh";
        //    levels ((4 4));
        //}
    );

    // Surface based refinement
    // ~~~~~~~~~~~~~~~~~~~~~~~~

    // Specifies two levels for every surface. The first is the minimum level,
    // every cell intersecting a surface gets refined up to the minimum level.
    // The second level is the maximum level. Cells that 'see' multiple
    // intersections where the intersections make an
```

```
    // angle > resolveFeatureAngle get refined up to the maximum level.

    refinementSurfaces
    {

      aircraft.stl     //STL filename where all the regions are added
        {
    level (6 6);
    regions
    {
      /*zone0 //Named region in the STL file
      {
                  // Surface-wise min and max refinement level
        level (2 2);
                        // Optional specification of patch type (default is wall). No
                        // constraint types (cyclic, symmetry) etc. are allowed.
                  patchInfo
                  {
                        type patch;
                        inGroups (meshedPatches);
                        }
      }*/
    }
  }
    }

    // Feature angle:
    // - used if min and max refinement level of a surface differ
    // - used if feature snapping (see snapControls below) is used
    resolveFeatureAngle 30;


    // Region-wise refinement
    // ~~~~~~~~~~~~~~~~~~~~~~

    // Specifies refinement level for cells in relation to a surface. One of
    // three modes
    // - distance. 'levels' specifies per distance to the surface the
    //   wanted refinement level. The distances need to be specified in
    //   increasing order.
    // - inside. 'levels' is only one entry and only the level is used. All
    //   cells inside the surface get refined up to the level. The surface
    //   needs to be closed for this to be possible.
    // - outside. Same but cells outside.

    refinementRegions
    {
        refinementBox
        {
            mode inside;
            levels ((1E15 3)); //(1E15) not relevant.
        }

    //aircraft.stl
        //{
        //    mode distance;
        //    levels (0.008 5);
```

```
        //}
    }

    // Mesh selection
    // ~~~~~~~~~~~~~~

    // After refinement patches get added for all refinementSurfaces and
    // all cells intersecting the surfaces get put into these patches. The
    // section reachable from the locationInMesh is kept.
    // NOTE: This point should never be on a face, always inside a cell, even
    // after refinement.
    locationInMesh (-0.051 -0.04 -0.008);

    // Whether any faceZones (as specified in the refinementSurfaces)
    // are only on the boundary of corresponding cellZones or also allow
    // free-standing zone faces. Not used if there are no faceZones.
    allowFreeStandingZoneFaces true;
}

// Settings for the snapping.
snapControls
{
    // Number of patch smoothing iterations before finding correspondence
    // to surface
    nSmoothPatch 3;

    // Maximum relative distance for points to be attracted by surface.
    // True distance is this factor times local maximum edge length.
    // Note: changed(corrected) w.r.t 17x! (17x used 2* tolerance)
    tolerance 1.0;

    // Number of mesh displacement relaxation iterations.
    nSolveIter 30;

    // Maximum number of snapping relaxation iterations. Should stop
    // before upon reaching a correct mesh.
    nRelaxIter 5;

    // Feature snapping

        // Number of feature edge snapping iterations.
        // Leave out altogether to disable.
        nFeatureSnapIter 10;

        // Detect (geometric only) features by sampling the surface
        // (default=false).
        implicitFeatureSnap false;

        // Use castellatedMeshControls::features (default = true)
        explicitFeatureSnap true;

        // Detect features between multiple surfaces
        // (only for explicitFeatureSnap, default = false)
        multiRegionFeatureSnap false;
}

// Settings for the layer addition.
```

```
addLayersControls
{
    // Are the thickness parameters below relative to the undistorted
    // size of the refined cell outside layer (true) or absolute sizes (false).
    relativeSizes true;

    // Layer thickness specification. This can be specified in one of four ways
    // - expansionRatio and finalLayerThickness (cell nearest internal mesh)
    // - expansionRatio and firstLayerThickness (cell on surface)
    // - overall thickness and firstLayerThickness
    // - overall thickness and finalLayerThickness

        // Expansion factor for layer mesh
        expansionRatio 2;

        // Wanted thickness of the layer furthest away from the wall.
        // If relativeSizes this is relative to undistorted size of cell
        // outside layer.
        finalLayerThickness 0.4;

        // Wanted thickness of the layer next to the wall.
        // If relativeSizes this is relative to undistorted size of cell
        // outside layer.
        //firstLayerThickness 0.3;

        // Wanted overall thickness of layers.
        // If relativeSizes this is relative to undistorted size of cell
        // outside layer.
        //thickness 0.5


    // Minimum overall thickness of total layers. If for any reason layer
    // cannot be above minThickness do not add layer.
    // If relativeSizes this is relative to undistorted size of cell
    // outside layer..
    minThickness 0.2;


    // Per final patch (so not geometry!) the layer information
    // Note: This behaviour changed after 21x. Any non-mentioned patches
    //       now slide unless:
    //          - nSurfaceLayers is explicitly mentioned to be 0.
    //          - angle to nearest surface < slipFeatureAngle (see below)
    layers
    {
        aircraftPatch
        {
            nSurfaceLayers 2;

        }
        maxY
        {
            nSurfaceLayers 2;
            // Per patch layer data
            expansionRatio      2;
            finalLayerThickness 0.4;
            minThickness        0.2;
        }

        // Disable any mesh shrinking and layer addition on any point of
        // a patch by setting nSurfaceLayers to 0
        frozenPatches
        {
            nSurfaceLayers 0;
        }
    }

    // If points get not extruded do nGrow layers of connected faces that are
    // also not grown. This helps convergence of the layer addition process
    // close to features.
    // Note: changed(corrected) w.r.t 17x! (didn't do anything in 17x)
    nGrow 0;

    // Advanced settings

    // When not to extrude surface. 0 is flat surface, 90 is when two faces
    // are perpendicular
    featureAngle 60;

    // At non-patched sides allow mesh to slip if extrusion direction makes
    // angle larger than slipFeatureAngle.
    slipFeatureAngle 30;

    // Maximum number of snapping relaxation iterations. Should stop
    // before upon reaching a correct mesh.
    nRelaxIter 5;

    // Number of smoothing iterations of surface normals
    nSmoothSurfaceNormals 1;

    // Number of smoothing iterations of interior mesh movement direction
    nSmoothNormals 3;

    // Smooth layer thickness over surface patches
    nSmoothThickness 10;

    // Stop layer growth on highly warped cells
    maxFaceThicknessRatio 0.5;

    // Reduce layer growth where ratio thickness to medial
    // distance is large
    maxThicknessToMedialRatio 0.3;

    // Angle used to pick up medial axis points
    // Note: changed(corrected) w.r.t 17x! 90 degrees corresponds to 130 in 17x.
    minMedianAxisAngle 90;

    // Create buffer region for new layer terminations
    nBufferCellsNoExtrude 0;

    // Overall max number of layer addition iterations. The mesher will exit
    // if it reaches this number of iterations; possibly with an illegal
    // mesh.
    nLayerIter 50;
```

```

    // Max number of iterations after which relaxed meshQuality controls
    // get used. Up to nRelaxIter it uses the settings in meshQualityControls,
    // after nRelaxIter it uses the values in meshQualityControls::relaxed.
    nRelaxedIter 20;

    // Additional reporting: if there are just a few faces where there
    // are mesh errors (after adding the layers) print their face centres.
    // This helps in tracking down problematic mesh areas.
    //additionalReporting true;
}

// Generic mesh quality settings. At any undoable phase these determine
// where to undo.
meshQualityControls
{
    // Maximum non-orthogonality allowed. Set to 180 to disable.
    maxNonOrtho 45;

    // Max skewness allowed. Set to <0 to disable.
    maxBoundarySkewness 20;
    maxInternalSkewness 4;

    // Max concaveness allowed. Is angle (in degrees) below which concavity
    // is allowed. 0 is straight face, <0 would be convex face.
    // Set to 180 to disable.
    maxConcave 80;

    // Minimum pyramid volume. Is absolute volume of cell pyramid.
    // Set to a sensible fraction of the smallest cell volume expected.
    // Set to very negative number (e.g. -1E30) to disable.
    minVol 1e-13;

    // Minimum quality of the tet formed by the face-centre
    // and variable base point minimum decomposition triangles and
    // the cell centre. This has to be a positive number for tracking
    // to work. Set to very negative number (e.g. -1E30) to
    // disable.
    //     <0 = inside out tet,
    //      0 = flat tet
    //      1 = regular tet
    minTetQuality 1e-9;

    // Minimum face area. Set to <0 to disable.
    minArea -1;

    // Minimum face twist. Set to <-1 to disable. dot product of face normal
    // and face centre triangles normal
    minTwist 0.05;

    // minimum normalised cell determinant
    // 1 = hex, <= 0 = folded or flattened illegal cell
    minDeterminant 0.001;

    // minFaceWeight (0 -> 0.5)
    minFaceWeight 0.05;

    // minVolRatio (0 -> 1)
    minVolRatio 0.01;

    // must be >0 for Fluent compatibility
    minTriangleTwist -1;

    //- if >0 : preserve single cells with all points on the surface if the
    //  resulting volume after snapping (by approximation) is larger than
    //  minVolCollapseRatio times old volume (i.e. not collapsed to flat cell).
    //  If <0 : delete always.
    //minVolCollapseRatio 0.5;

    // Advanced

    // Number of error distribution iterations
    nSmoothScale 4;
    // amount to scale back displacement at error points
    errorReduction 0.75;

    // Optional : some meshing phases allow usage of relaxed rules.
    // See e.g. addLayersControls::nRelaxedIter.
    relaxed
    {
        //- Maximum non-orthogonality allowed. Set to 180 to disable.
        maxNonOrtho 45;
    }
}

// Advanced

// Flags for optional output
// 0 : only write final meshes
// 1 : write intermediate meshes
// 2 : write volScalarField with cellLevel for postprocessing
// 4 : write current intersections as .obj files
debug 0;

// Merge tolerance. Is fraction of overall bounding box of initial mesh.
// Note: the write tolerance needs to be higher than this.
mergeTolerance 1e-6;

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

To start meshing, make sure that the first switch is set to true while the others are set to false (lines 18 through 20). Then, within the case directory, type:

```
snappyHexMesh
```

The first switch, controlling the castellatedMeshControls uses the cells of the background mesh to divide the domain in small cubes according to the instructions specified in the dictionary. A new directory has appeared (named *1* if deltaT and writeInterval have been previously set to 1). It contains the information of the first meshing process of snappyHexMesh.

The results are shown in the following figure:

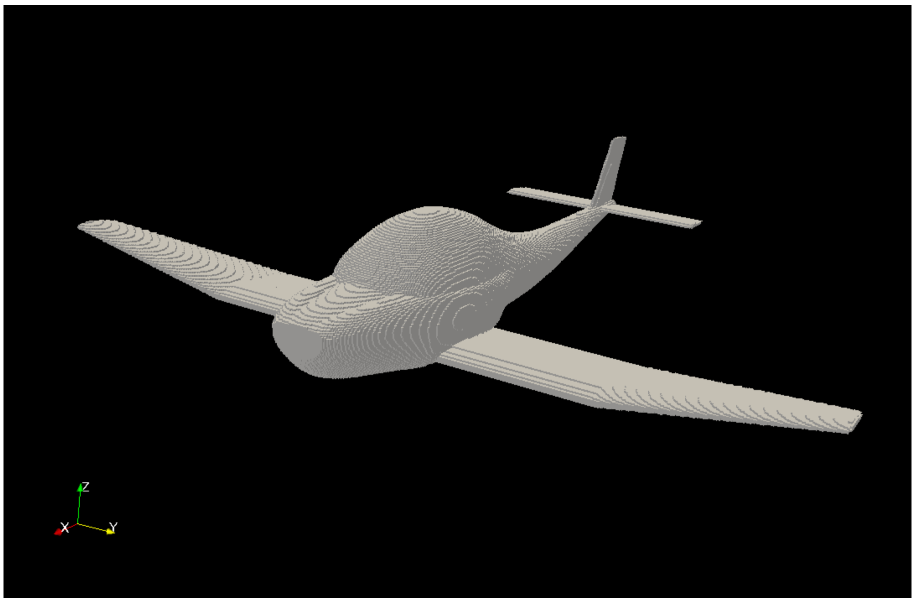


Figure 6.6: Shape of the aircraft at the first step of the meshing process of snappy-HexMesh

To obtain the results shown in Figure 6.6, launch ParaView, go to time interval 1 and select `aircraftPatch` located in Mesh Parts.

As it was previously said, the internal cells of the aircraft are removed. This characteristic, as well as the cell refinement distribution along the domain can be observed in the following figures:

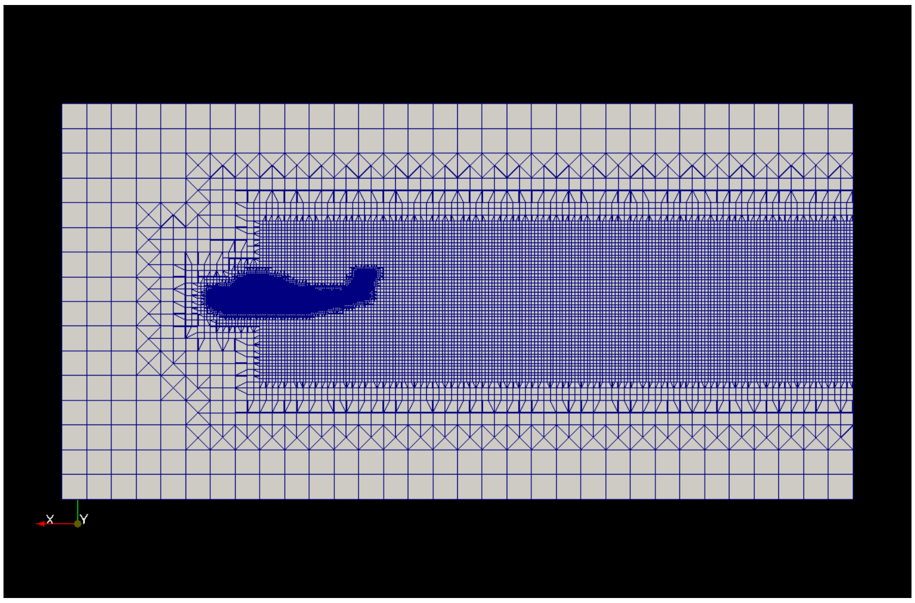


Figure 6.7: Mesh of the domain at the first step of the meshing process of snappy-HexMesh

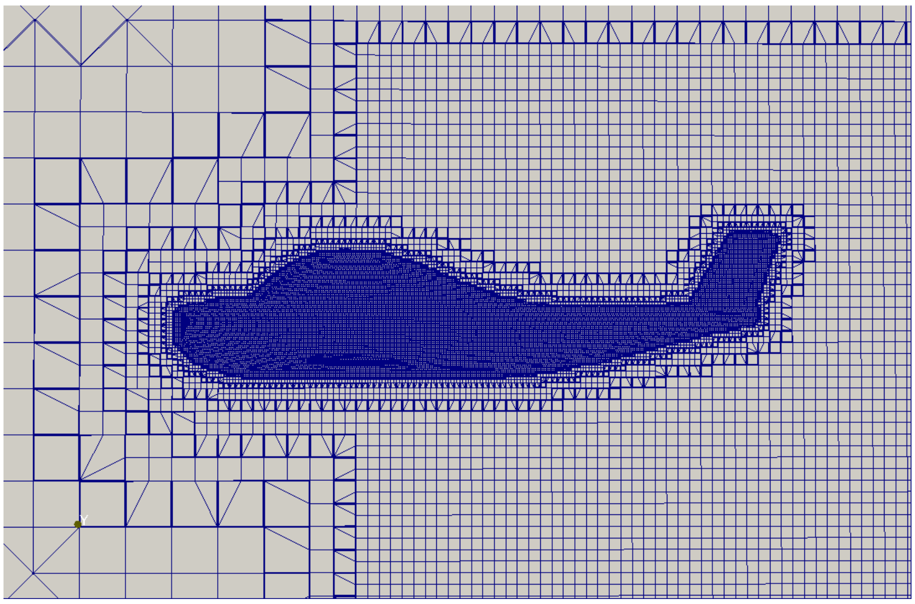


Figure 6.8: Detail of the mesh at the first step of the meshing process of snappy-HexMesh

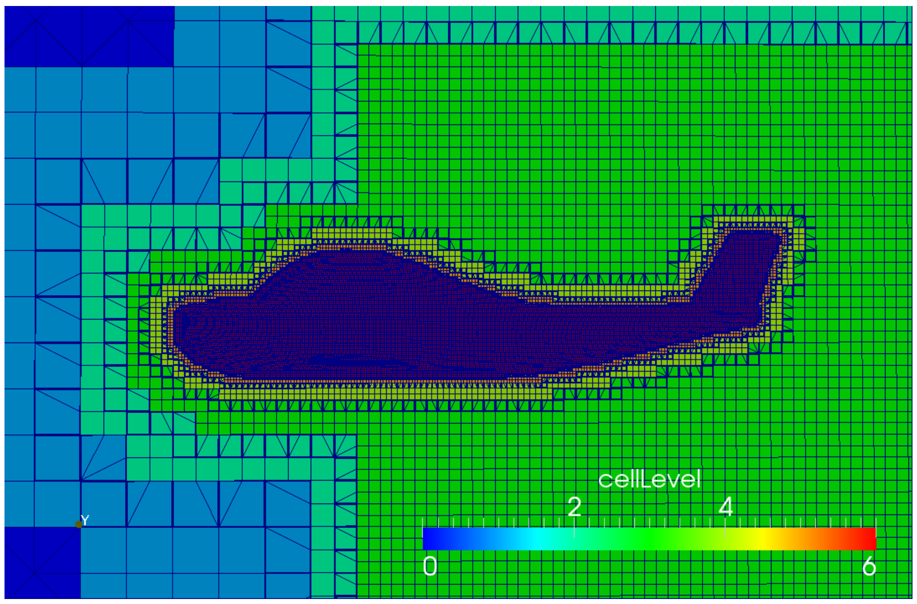


Figure 6.9: Detail of the mesh with a representation of the cell refinement at the first step of the meshing process of snappyHexMesh

For the representation of the previous Figures, `internalMesh` has to be selected instead of `aircraftPatch`. Moreover, as the aircraft shape is internal, a cut using the Clip icon has to be used. To show the cell refinement as a field, click the Volume Fields box located within Properties. Then, select the cellLevel option in the first drop-down menu at the top of ParaView's screen.

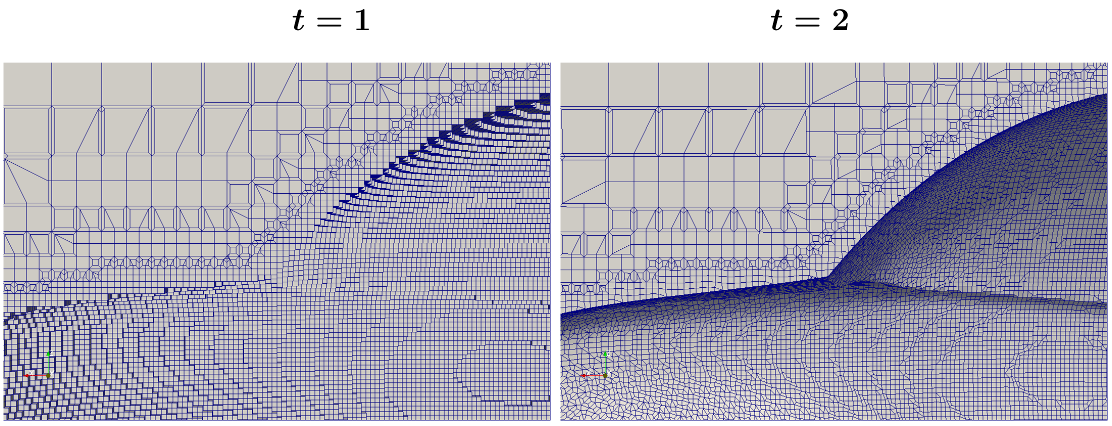
As it can be seen in previous figures, the surface of the aircraft is irregular and cube-based. The second switch of snappyHexMesh controls the snapControls, involving the displacement of boundary vertices to conform to surface. All vertex displacements are reversible to ensure mesh quality.

To proceed to the next step, set the first switch to false, the second to true and type:

```
snappyHexMesh
```

Once it has been run, the user can observe the *2* directory within the case containing the information generated. During this second process, castellated mesh boundary and internal mesh have been smoothed. Furthermore, mesh boundary is *snapped* to geometry surface.

Now, in ParaView, the user can use the time control buttons to view the background mesh ($t = 0$), the castellated mesh ($t = 1$) and the mesh generated in the second step of snappyHexMesh ($t = 2$). In the following figures it is possible to observe the differences between directories *1* and *2*:



Now, the mesh of the *Very Light Aircraft* case is ready.

It exists a third switch in snappyHexMesh which introduces cell layers at the patches of the case. When doing it, surface points are identified and are not displaced. In the current tutorial, this third process of snappyHexMesh is not going to be used.

Finally, it is important to mention that the *snappyHexMeshDict* file is complex to fully understand; it includes a lot of instructions and only with time and experience it is possible to reach a broad domain of its contents. Nevertheless, the user can identify some basic instructions that are relevant when preparing the meshing process:

- **Name of the geometry used for the meshing process**: line 30
- **Definition of cell refinement level at the patches of the domain**: lines 118 through 140
- **Specification of a point contained in the volume which will be meshed**: line 184

Regarding the last sentence, as in the current case the point specified is located outside the aircraft, its internal cells have been removed while the background mesh has become the fluid domain.

Advice:

> If the user wants to remesh the case (changing or not the instructions of *snappyHexMeshDict*) it is first necessary to remove *1* and *2* directories. Then, after setting the switches properly, snappyHexMesh can be run again

6.0.21.2 Boundary and initial conditions

Besides the *p* and *U* files, as the case is set turbulent, three new files are necessary to be created. Their names are *k*, *nut* and *omega*. These dictionaries contain the boundary conditions of the parameters used to implement the SST k-w turbulence model. Before discussing their calculation, the boundary conditions for p and $\mathbf{U}$ are:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            freestreamPressure;
    }

    outlet
    {
        type            freestreamPressure;
    }

    aircraftPatch
    {
        type            zeroGradient;
    }

    top
    {
        type            slip;
    }

    bottom
    {
        type            slip;
    }

    frontAndBack
    {
        type            slip;
    }

}

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

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (-45 0 0);

boundaryField
{
    inlet
    {
        type            freestream;
        freestreamValue uniform (-45 0 0);
    }

    outlet
    {
        type            freestream;
        freestreamValue uniform (-45 0 0);
    }

    aircraftPatch
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    top
    {
        type            slip;
    }

    bottom
    {
        type                slip;
    }

    frontAndBack
    {
        type                slip;
    }
}

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

Caution:

> As in the current case there is not *0* directory, the dictionaries of *p*, *U*, *k*, *nut* and *omega* are included within the last directory generated with snappyHexMesh

The dictionaries of *k*, *nut* and *omega* are:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      binary;
    class       volScalarField;
    location    "2";
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 1.215;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1.215;
    }

    outlet
    {
        type            inletOutlet;
        inletValue      uniform 1.215;
        value           uniform 1.215;
    }

    aircraftPatch
    {
        type            kqRWallFunction;
        value           uniform 1.215;
    }

    top
    {
        type            slip;
    }

    bottom
    {
        type            slip;
    }

    frontAndBack
    {
        type            slip;
    }
}

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

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      binary;
    class       volScalarField;
    location    "2";
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }

    outlet
    {
        type            calculated;
        value           uniform 0;
    }

    aircraftPatch
    {
        type            nutkWallFunction;
        value           uniform 0;
    }

    top
    {
        type            calculated;
        value           uniform 0;
    }

    bottom
    {
        type            calculated;
        value           uniform 0;
    }

    frontAndBack
    {
        type            calculated;
        value           uniform 0;
    }
}

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

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      binary;
    class       volScalarField;
    location    "2";
    object      omega;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 -1 0 0 0 0];

internalField   uniform 2147.745;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 2147.745;
    }

    outlet
    {
        type            inletOutlet;
        inletValue      uniform 2147.745;
        value           uniform 2147.745;
    }

    aircraftPatch
    {
        type            omegaWallFunction;
        value           uniform 2147.745;
    }

    top
    {
        type            slip;
    }

    bottom
    {
        type            slip;
    }

    frontAndBack
    {
        type            slip;
    }
}

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

k, ν_t and ω are parameters required to simulate the case with a turbulence model. Mathematically, they are determined as:

$$k = \frac{3}{2}(|\mathbf{U}|I)^2 \tag{6.5}$$

$$\omega = \frac{C_\mu^{-0.25} k^{0.5}}{l} \tag{6.6}$$

$$C_\mu = 0.09k \tag{6.7}$$

Where $|\mathbf{U}|$ is the mean flow velocity, I is the turbulence intensity and l the turbulent length scale. They can be computed as:

$$I \approx 2\%$$

$$l \approx 0.07 \cdot c$$

With these assumptions, $k = 1.215$, $C_\mu = 0.109$ and $\omega = 2147.745$.

6.0.21.3 Physical properties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

nu              nu [0 2 -1 0 0 0 0] 1.5e-07;

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

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      RASProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

RASModel            kOmegaSST;

turbulence          on;

```

```
printCoeffs          on;

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

Remember that although the medium is air, $\nu = 1.5 \times 10^{-7}$ to maintain the Reynolds number as the dimensional analysis showed.

6.0.21.4 Control

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     simpleFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         300;

deltaT          1;

writeControl    timeStep;

writeInterval   1;

purgeWrite      0;

writeFormat     binary;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

```
46
47  functions
48  {
49      #include "readFields"
50      #include "cuttingPlane"
51      #include "forceCoeffs"
52  }
53
54  // ************************************************************************* //
```

As it can be seen, at lines 47 through 52 the program is using external functions to develop specific tasks. While the case is running, these external functions are called using its dictionaries located within *system*. It facilitates the treatment of the code. For instance, it is not necessary to include all the instructions to calculate the force coefficients directly in *controlDict*. The dictionaries are shown in Section 6.0.21.6.

6.0.21.5 Discretization and linear-solver settings

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    default         Gauss linear;
    grad(U)         cellLimited Gauss linear 1;
}

divSchemes
{
    default         none;
    div(phi,U)      bounded Gauss linearUpwindV grad(U);
    div(phi,k)      bounded Gauss upwind;
    div(phi,omega)  bounded Gauss upwind;
    div((nuEff*dev(T(grad(U))))) Gauss linear;
}
```

```

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p;
}

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

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-7;
        relTol          0.01;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration on;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels     1;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-8;
        relTol          0.1;
        nSweeps         1;
    }

    k
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-8;
        relTol          0.1;
        nSweeps         1;
    }

    omega
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-8;
        relTol          0.1;
        nSweeps         1;
    }
}

SIMPLE
{
    nCorrectors                1;
    nNonOrthogonalCorrectors   2;
    pRefCell                   0;
    pRefValue                  0;
}

potentialFlow
{
    nNonOrthogonalCorrectors   10;
}

relaxationFactors
{
    fields
    {
        p               0.3;
    }
    equations
    {
        U               0.5;
        k               0.5;
        omega           0.5;
    }
}

cache
{
    grad(U);
}

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

6.0.21.6 External functions

This section contains the dictionaries used by the functions of *controlDict*. They must be copied into files named *readFiles*, *forceCoeffs* and *cuttingPlane* respectively. They are located within *system*.

readFields

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

// Make sure all fields for functionObjects are loaded. Prevents any
// problems running with execFlowFunctionObjects.
readFields
{
    // Where to load it from (if not already in solver)
    functionObjectLibs ("libfieldFunctionObjects.so");

    type            readFields;
    fields          (p U k);
}


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

forceCoeffs

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

forces
{
        type                forces;
        functionObjectLibs  ( "libforces.so" ); //Lib to load
        outputControl       timeStep;
        outputInterval      1;
        patches             (aircraftPatch); //Patch name over forces will be
            calculated
        pName               p;
        UName               U;
        rhoName             rhoInf; //Reference density
        log                 true;
        rhoInf              1.225; //Air density
        CofR                (0 0 0); //Origin for moment calculations
}

forceCoeffs
{
type forceCoeffs;
functionObjectLibs ("libforces.so");
patches (aircraftPatch);
// pName p;
// UName U;
rhoName rhoInf;
rhoInf 1.225;
CofR (0 0 0);
liftDir (0 1 0);
dragDir (-1 0 0);
pitchAxis (0 0 1);
magUInf 45; // Free stream velocity
lRef 0.01276; // Mean Chord
Aref 0.001218; // Ref. Area
outputControl timeStep;
outputInterval 1;
}

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

cuttingPlane

The *cuttingPlane* function generates a VTK plane cutting the fluid domain in a specified direction. It can be easily open from the *postProcessing* directory once the case has been run and allows a practial and rapid view of the p and $\mathbf{U}$ fields.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

cuttingPlane
{
    type                surfaces;
    functionObjectLibs ("libsampling.so");
    outputControl     outputTime;

    surfaceFormat   vtk;
    fields          ( p U );

    interpolationScheme cellPoint;

    surfaces
    (
        yNormal
        {
            type            cuttingPlane;
            planeType       pointAndNormal;
            pointAndNormalDict
            {
                basePoint       (0 0 0);
                normalVector    (0 1 0);
            }
            interpolate     true;
        }
    );
}

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

At the end of the pre-processing, the structure of directories, subdirectories and files within aircraft should be as follows:

- aircraft
  - 0
    - p
    - U
    - k
    - nut
    - omega
  - constant
    - polyMesh
    - triSurface
    - extendedFeatureEdgeMesh
    - transportProperties
    - RASProperties
  - system
    - controlDict
    - fvSchemes
    - fvSolutions
    - surfaceFeatureExtractDict
    - snappyHexMeshDict
    - readFields
    - cuttingPlane
    - forceCoeffs

6.0.22 Post-processing

6.0.22.1 Results of the simulation

The pressure field around the aircraft:

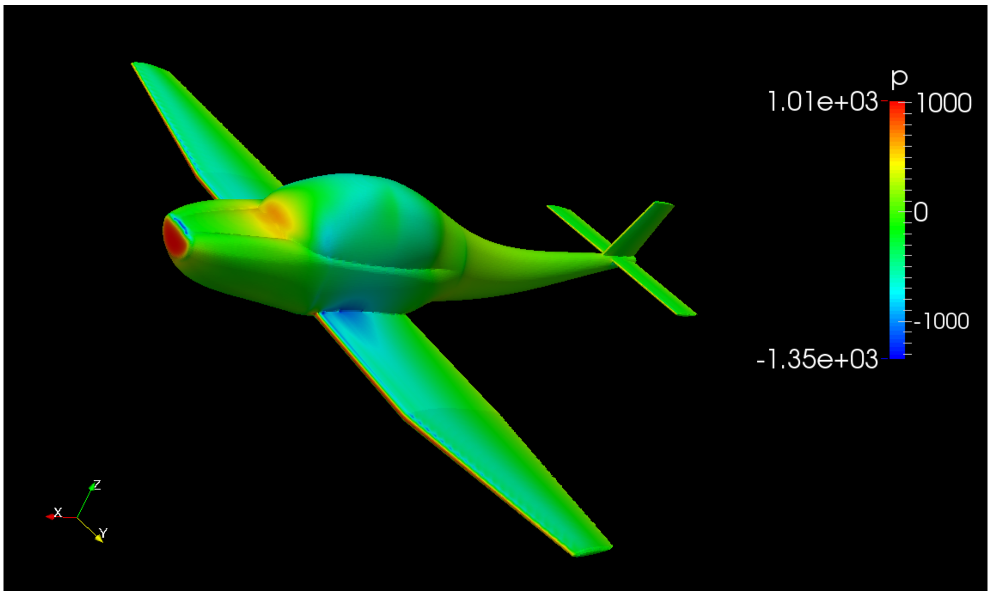


Figure 6.10: Pressure field around the aircraft (m^2/s^2)

The velocity field in the whole domain:

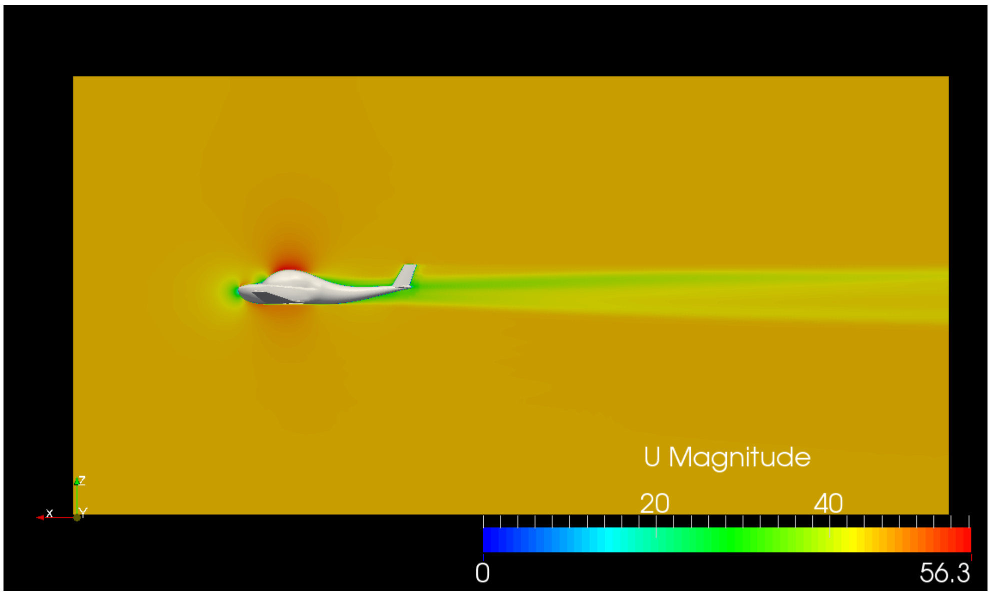


Figure 6.11: Velocity field in the domain of the aircraft case (m/s)

The velocity field around the aircraft:

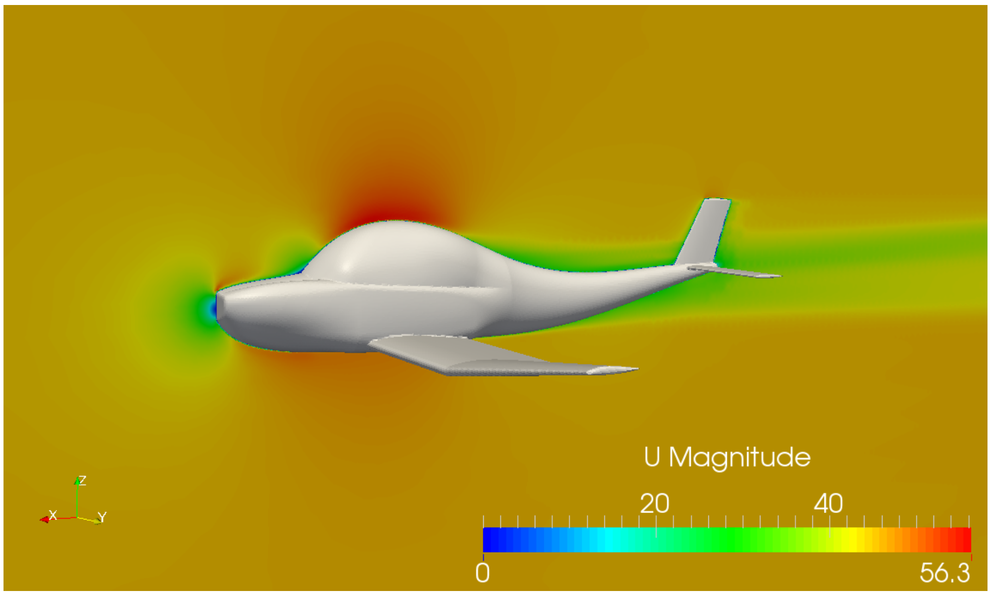


Figure 6.12: Velocity field around the aircraft (m/s)

At the end of the simulation, the values of drag coefficient and lift coefficient stabilized at:

- $C_D = 0.0276$
- $C_L = 0.005$

Although the value of the drag coefficient matches with other real very light aircrafts (as for instance the Cessna 172), the lift coefficient is very low, and therefore the aerodynamic efficiency ($E = L/D$) is far from the current commercial aircrafts. As it was explained, the aircraft geometry used in this chapter was not designed according to a realistic aerodynamic study (as it would have been CFD or wind tunnel experiments).

Another consideration that must be done is the fact that although the case is set turbulent, the wake of the aircraft shows a continuous appearance. It is because a RAS turbulence model has been used, based on average flow conditions. If other turbulence models had been used (as for instance LES or DNS), it would have been possible to observe turbulence in the wake.

For Figures 6.11 and 6.12, the VTK plane obtained from the *postProcessing* directory has been used.