

## 6. Fluid dynamics of a Very Light Aircraft

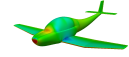
### 6.0.18 Description of the case

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<sup>2</sup>/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:

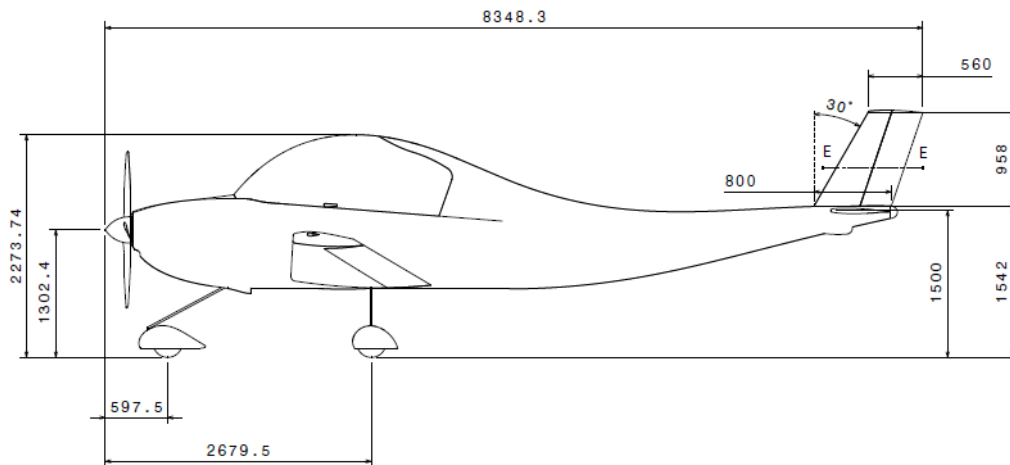


Figure 6.2: Measures of the aircraft used in the simulation of the *Very Light Aircraft* case

The problem statement is shown at Figure 6.3.

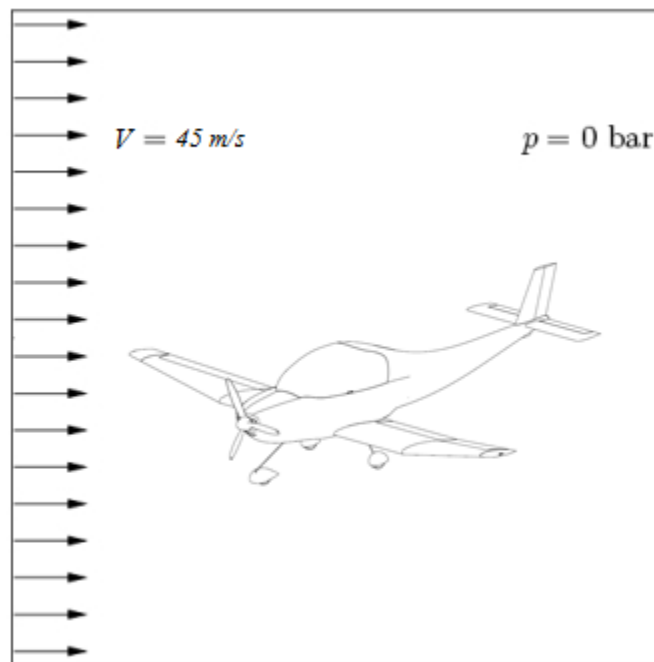
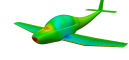


Figure 6.3: Very light aircraft flying at 45 m/s and ambient pressure

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,

$$\nabla \cdot \mathbf{U} = 0 \quad (6.1)$$

The momentum equation,

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

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:

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

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

$$\tilde{\nabla} \cdot \tilde{\mathbf{U}} = 0 \quad (6.3)$$

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

which can be rewritten as:

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

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- $\omega$  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`.

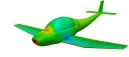
### 6.0.21.1 Mesh generation

#### 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 `writelnInterval` 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 i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \   /  O p e r a t i o n | Version:  2.2.0                |
5  |  \ \   /  A n d              | Web:      www.OpenFOAM.org        |
6  |  \ \   /  M a n i p u l a t i o n |                               |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class          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*.

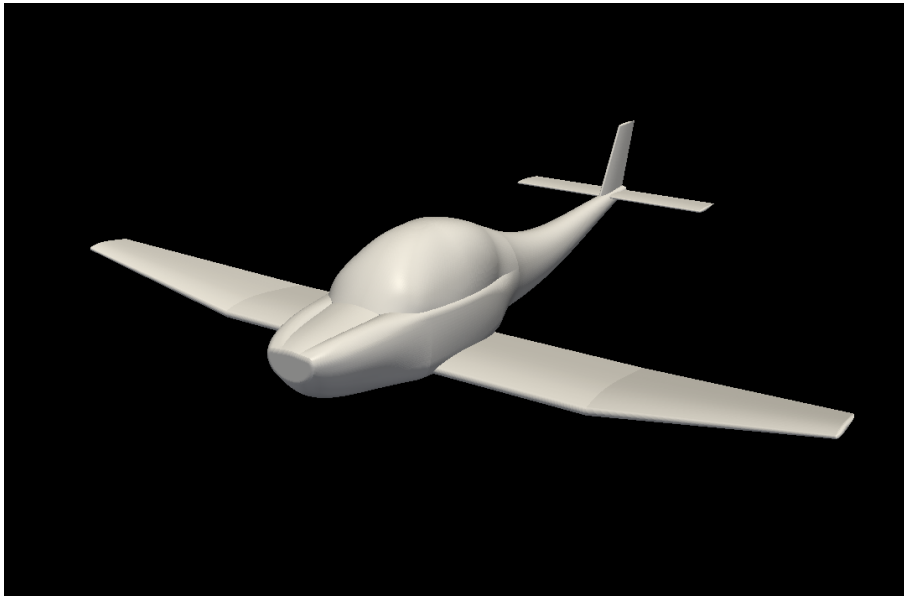


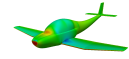
Figure 6.4: Aircraft' STL surface used in the simulation of the *Very Light Aircraft* case

## 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

```

1  /*----- C++ -----*/
2  |=====|
3  | \ \ / F i e l d | OpenFOAM: The Open Source CFD Toolbox |
4  | \ \ / O p e r a t i o n | Version: 2.2.0 |
5  | \ \ / A n d | Web: www.OpenFOAM.org |
6  | \ \ / M a n i p u l a t i o n | |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        blockMeshDict;
14 }
15 // ***** //
16
17 convertToMeters 0.01;
18
19 vertices
20 (
21     (-30 -8 -10)
22     ( 10 -8 -10)
23     ( 10 8 -10)
24     (-30 8 -10)
25     (-30 -8 10)
26     ( 10 -8 10)
27     ( 10 8 10)
28     (-30 8 10)
29 );
30
31 blocks
32 (
33     hex (0 1 2 3 4 5 6 7) (32 13 16) simpleGrading (1 1 1)
34 );
35
36 edges
37 (
38 );
39
40 boundary
41 (
42     inlet
43     {

```



```

45         type patch;
46     faces
47     (
48         (1 2 6 5)
49     );
50     }
51
52     outlet
53     {
54         type patch;
55     faces
56     (
57         (0 4 7 3)
58     );
59     }
60
61     bottom
62     {
63         type slip;
64     faces
65     (
66         (0 3 2 1)
67     );
68     }
69
70     top
71     {
72         type slip;
73     faces
74     (
75         (4 5 6 7)
76     );
77     }
78
79     frontAndBack
80     {
81         type slip;
82     faces
83     (
84         (0 1 5 4)
85         (2 3 7 6)
86     );
87     }
88
89 );
90
91 mergePatchPairs
92 (
93 );
94
95 // ***** //
```

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

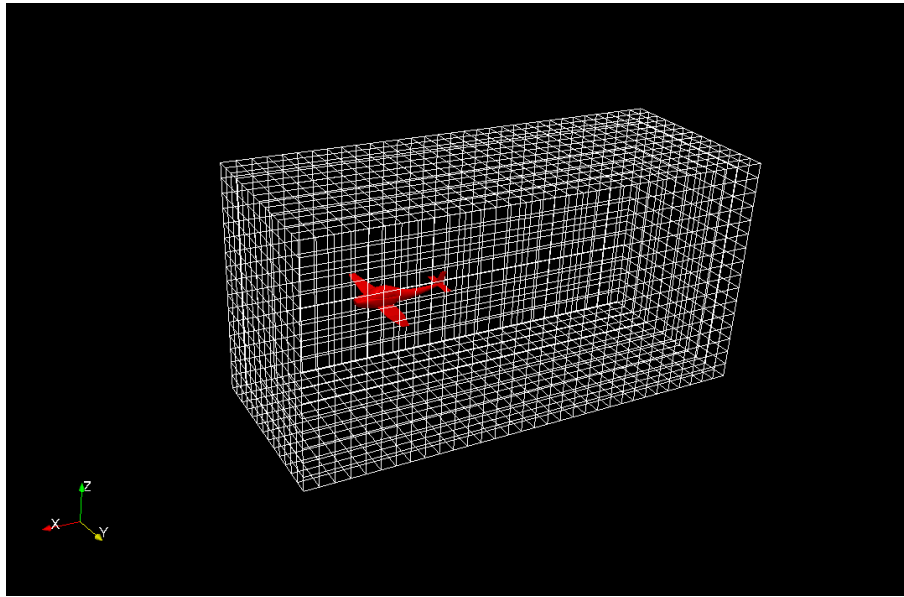
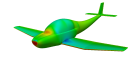


Figure 6.5: Aircraft' STL surface contained within the mesh generated with `blockMesh` in the *Very Light Aircraft* case

*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:

```

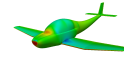
1  /*-----* C++ *-----*/
2  |=====|
3  | \\      / F i e l d      | OpenFOAM: The Open Source CFD Toolbox |

```

```

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       snappyHexMeshDict;
14 }
15 // * * * * *
16
17 // Which of the steps to run
18 castellatedMesh true;
19 snap            false;
20 addLayers       false;
21
22 // Geometry. Definition of all surfaces. All surfaces are of class
23 // searchableSurface.
24 // Surfaces are used
25 // - to specify refinement for any mesh cell intersecting it
26 // - to specify refinement for any mesh cell inside/outside/near
27 // - to 'snap' the mesh boundary to the surface
28 geometry
29 {
30     aircraft.stl //STL filename where all the regions are added
31     {
32         type triSurfaceMesh;
33
34         regions
35         {
36             patch0 //Named region in the STL file
37             {
38                 name aircraftPatch; //User-defined patch name. If not provided will be
39                                     <name>_<region>
40             }
41         }
42
43         refinementBox //Geometry to refine. Entities: Box, Cylinder, Sphere, Plane
44         {
45             type searchableBox;
46             min (-0.3 -0.06 -0.04);
47             max (0 0.06 0.04);
48         }
49     };
50
51 // Settings for the castellatedMesh generation.
52 castellatedMeshControls
53 {
54
55     // Refinement parameters
56     // ~~~~~
57
58     // If local number of cells is >= maxLocalCells on any processor
59     // switches from from refinement followed by balancing

```




---

```

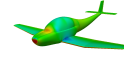
60      // (current method) to (weighted) balancing before refinement.
61      maxLocalCells 1500000;
62
63      // Overall cell limit (approximately). Refinement will stop immediately
64      // upon reaching this number so a refinement level might not complete.
65      // Note that this is the number of cells before removing the part which
66      // is not 'visible' from the keepPoint. The final number of cells might
67      // actually be a lot less.
68      maxGlobalCells 2000000;
69
70      // The surface refinement loop might spend lots of iterations refining just a
71      // few cells. This setting will cause refinement to stop if <= minimumRefine
72      // are selected for refinement. Note: it will at least do one iteration
73      // (unless the number of cells to refine is 0)
74      minRefinementCells 0;
75
76      // Allow a certain level of imbalance during refining
77      // (since balancing is quite expensive)
78      // Expressed as fraction of perfect balance (= overall number of cells /
79      // nProcs). 0=balance always.
80      maxLoadUnbalance 0.1;
81
82      // Number of buffer layers between different levels.
83      // 1 means normal 2:1 refinement restriction, larger means slower
84      // refinement.
85      nCellsBetweenLevels 3;
86
87
88      // Explicit feature edge refinement
89      // ~~~~~
90
91      // Specifies a level for any cell intersected by explicitly provided
92      // edges.
93      // This is a featureEdgeMesh, read from constant/triSurface for now.
94      // Specify 'levels' in the same way as the 'distance' mode in the
95      // refinementRegions (see below). The old specification
96      //      level 2;
97      // is equivalent to
98      //      levels ((0 2));
99
100     features //Disabled because the refinement will be surface-based (as internal
101             aircraft cells are removed)
102
103     (
104         //{
105         //      file "aircraft.eMesh";
106         //      levels ((4 4));
107         //}
108     );
109
110     // Surface based refinement
111     // ~~~~~
112
113     // Specifies two levels for every surface. The first is the minimum level,
114     // every cell intersecting a surface gets refined up to the minimum level.
115     // The second level is the maximum level. Cells that 'see' multiple
116     // intersections where the intersections make an

```

```

116 // angle > resolveFeatureAngle get refined up to the maximum level.
117
118 refinementSurfaces
119 {
120
121     aircraft.stl //STL filename where all the regions are added
122     {
123         level (6 6);
124         regions
125         {
126             /*zone0 //Named region in the STL file
127             {
128                 // Surface-wise min and max refinement level
129                 level (2 2);
130                 // Optional specification of patch type (default is wall). No
131                 // constraint types (cyclic, symmetry) etc. are allowed.
132                 patchInfo
133                 {
134                     type patch;
135                     inGroups (meshedPatches);
136                 }
137             }*/
138         }
139     }
140 }
141
142 // Feature angle:
143 // - used if min and max refinement level of a surface differ
144 // - used if feature snapping (see snapControls below) is used
145 resolveFeatureAngle 30;
146
147
148 // Region-wise refinement
149 // ~~~~~
150
151 // Specifies refinement level for cells in relation to a surface. One of
152 // three modes
153 // - distance. 'levels' specifies per distance to the surface the
154 //   wanted refinement level. The distances need to be specified in
155 //   increasing order.
156 // - inside. 'levels' is only one entry and only the level is used. All
157 //   cells inside the surface get refined up to the level. The surface
158 //   needs to be closed for this to be possible.
159 // - outside. Same but cells outside.
160
161 refinementRegions
162 {
163     refinementBox
164     {
165         mode inside;
166         levels ((1E15 3)); //(1E15) not relevant.
167     }
168
169     //aircraft.stl
170     //{
171     //    mode distance;
172     //    levels (0.008 5);

```



```

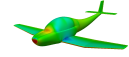
173         //}
174     }
175
176     // Mesh selection
177     // ~~~~~
178
179     // After refinement patches get added for all refinementSurfaces and
180     // all cells intersecting the surfaces get put into these patches. The
181     // section reachable from the locationInMesh is kept.
182     // NOTE: This point should never be on a face, always inside a cell, even
183     // after refinement.
184     locationInMesh (-0.051 -0.04 -0.008);
185
186     // Whether any faceZones (as specified in the refinementSurfaces)
187     // are only on the boundary of corresponding cellZones or also allow
188     // free-standing zone faces. Not used if there are no faceZones.
189     allowFreeStandingZoneFaces true;
190 }
191
192 // Settings for the snapping.
193 snapControls
194 {
195     // Number of patch smoothing iterations before finding correspondence
196     // to surface
197     nSmoothPatch 3;
198
199     // Maximum relative distance for points to be attracted by surface.
200     // True distance is this factor times local maximum edge length.
201     // Note: changed(corrected) w.r.t 17x! (17x used 2* tolerance)
202     tolerance 1.0;
203
204     // Number of mesh displacement relaxation iterations.
205     nSolveIter 30;
206
207     // Maximum number of snapping relaxation iterations. Should stop
208     // before upon reaching a correct mesh.
209     nRelaxIter 5;
210
211     // Feature snapping
212
213     // Number of feature edge snapping iterations.
214     // Leave out altogether to disable.
215     nFeatureSnapIter 10;
216
217     // Detect (geometric only) features by sampling the surface
218     // (default=false).
219     implicitFeatureSnap false;
220
221     // Use castellatedMeshControls::features (default = true)
222     explicitFeatureSnap true;
223
224     // Detect features between multiple surfaces
225     // (only for explicitFeatureSnap, default = false)
226     multiRegionFeatureSnap false;
227 }
228
229 // Settings for the layer addition.

```

```

230  addLayersControls
231  {
232      // Are the thickness parameters below relative to the undistorted
233      // size of the refined cell outside layer (true) or absolute sizes (false).
234      relativeSizes true;
235
236      // Layer thickness specification. This can be specified in one of four ways
237      // - expansionRatio and finalLayerThickness (cell nearest internal mesh)
238      // - expansionRatio and firstLayerThickness (cell on surface)
239      // - overall thickness and firstLayerThickness
240      // - overall thickness and finalLayerThickness
241
242      // Expansion factor for layer mesh
243      expansionRatio 2;
244
245      // Wanted thickness of the layer furthest away from the wall.
246      // If relativeSizes this is relative to undistorted size of cell
247      // outside layer.
248      finalLayerThickness 0.4;
249
250      // Wanted thickness of the layer next to the wall.
251      // If relativeSizes this is relative to undistorted size of cell
252      // outside layer.
253      //firstLayerThickness 0.3;
254
255      // Wanted overall thickness of layers.
256      // If relativeSizes this is relative to undistorted size of cell
257      // outside layer.
258      //thickness 0.5
259
260
261      // Minimum overall thickness of total layers. If for any reason layer
262      // cannot be above minThickness do not add layer.
263      // If relativeSizes this is relative to undistorted size of cell
264      // outside layer..
265      minThickness 0.2;
266
267
268      // Per final patch (so not geometry!) the layer information
269      // Note: This behaviour changed after 21x. Any non-mentioned patches
270      // now slide unless:
271      // - nSurfaceLayers is explicitly mentioned to be 0.
272      // - angle to nearest surface < slipFeatureAngle (see below)
273      layers
274      {
275          aircraftPatch
276          {
277              nSurfaceLayers 2;
278          }
279      }
280      maxY
281      {
282          nSurfaceLayers 2;
283          // Per patch layer data
284          expansionRatio 2;
285          finalLayerThickness 0.4;
286          minThickness 0.2;

```



```

287     }
288
289     // Disable any mesh shrinking and layer addition on any point of
290     // a patch by setting nSurfaceLayers to 0
291     frozenPatches
292     {
293         nSurfaceLayers 0;
294     }
295 }
296
297 // If points get not extruded do nGrow layers of connected faces that are
298 // also not grown. This helps convergence of the layer addition process
299 // close to features.
300 // Note: changed(corrected) w.r.t 17x! (didn't do anything in 17x)
301 nGrow 0;
302
303 // Advanced settings
304
305 // When not to extrude surface. 0 is flat surface, 90 is when two faces
306 // are perpendicular
307 featureAngle 60;
308
309 // At non-patched sides allow mesh to slip if extrusion direction makes
310 // angle larger than slipFeatureAngle.
311 slipFeatureAngle 30;
312
313 // Maximum number of snapping relaxation iterations. Should stop
314 // before upon reaching a correct mesh.
315 nRelaxIter 5;
316
317 // Number of smoothing iterations of surface normals
318 nSmoothSurfaceNormals 1;
319
320 // Number of smoothing iterations of interior mesh movement direction
321 nSmoothNormals 3;
322
323 // Smooth layer thickness over surface patches
324 nSmoothThickness 10;
325
326 // Stop layer growth on highly warped cells
327 maxFaceThicknessRatio 0.5;
328
329 // Reduce layer growth where ratio thickness to medial
330 // distance is large
331 maxThicknessToMedialRatio 0.3;
332
333 // Angle used to pick up medial axis points
334 // Note: changed(corrected) w.r.t 17x! 90 degrees corresponds to 130 in 17x.
335 minMedianAxisAngle 90;
336
337 // Create buffer region for new layer terminations
338 nBufferCellsNoExtrude 0;
339
340 // Overall max number of layer addition iterations. The mesher will exit
341 // if it reaches this number of iterations; possibly with an illegal
342 // mesh.
343 nLayerIter 50;

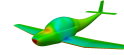
```



```

344
345 // Max number of iterations after which relaxed meshQuality controls
346 // get used. Up to nRelaxIter it uses the settings in meshQualityControls,
347 // after nRelaxIter it uses the values in meshQualityControls::relaxed.
348 nRelaxedIter 20;
349
350 // Additional reporting: if there are just a few faces where there
351 // are mesh errors (after adding the layers) print their face centres.
352 // This helps in tracking down problematic mesh areas.
353 //additionalReporting true;
354 }
355
356 // Generic mesh quality settings. At any undoable phase these determine
357 // where to undo.
358 meshQualityControls
359 {
360 // Maximum non-orthogonality allowed. Set to 180 to disable.
361 maxNonOrtho 45;
362
363 // Max skewness allowed. Set to <0 to disable.
364 maxBoundarySkewness 20;
365 maxInternalSkewness 4;
366
367 // Max concaveness allowed. Is angle (in degrees) below which concavity
368 // is allowed. 0 is straight face, <0 would be convex face.
369 // Set to 180 to disable.
370 maxConcave 80;
371
372 // Minimum pyramid volume. Is absolute volume of cell pyramid.
373 // Set to a sensible fraction of the smallest cell volume expected.
374 // Set to very negative number (e.g. -1E30) to disable.
375 minVol 1e-13;
376
377 // Minimum quality of the tet formed by the face-centre
378 // and variable base point minimum decomposition triangles and
379 // the cell centre. This has to be a positive number for tracking
380 // to work. Set to very negative number (e.g. -1E30) to
381 // disable.
382 // <0 = inside out tet,
383 // 0 = flat tet
384 // 1 = regular tet
385 minTetQuality 1e-9;
386
387 // Minimum face area. Set to <0 to disable.
388 minArea -1;
389
390 // Minimum face twist. Set to <-1 to disable. dot product of face normal
391 // and face centre triangles normal
392 minTwist 0.05;
393
394 // minimum normalised cell determinant
395 // 1 = hex, <= 0 = folded or flattened illegal cell
396 minDeterminant 0.001;
397
398 // minFaceWeight (0 -> 0.5)
399 minFaceWeight 0.05;
400

```



```

401 // minVolRatio (0 -> 1)
402 minVolRatio 0.01;
403
404 // must be >0 for Fluent compatibility
405 minTriangleTwist -1;
406
407 // - if >0 : preserve single cells with all points on the surface if the
408 // resulting volume after snapping (by approximation) is larger than
409 // minVolCollapseRatio times old volume (i.e. not collapsed to flat cell).
410 // If <0 : delete always.
411 //minVolCollapseRatio 0.5;
412
413 // Advanced
414
415 // Number of error distribution iterations
416 nSmoothScale 4;
417 // amount to scale back displacement at error points
418 errorReduction 0.75;
419
420 // Optional : some meshing phases allow usage of relaxed rules.
421 // See e.g. addLayersControls::nRelaxedIter.
422 relaxed
423 {
424     // - Maximum non-orthogonality allowed. Set to 180 to disable.
425     maxNonOrtho 45;
426 }
427 }
428
429 // Advanced
430
431 // Flags for optional output
432 // 0 : only write final meshes
433 // 1 : write intermediate meshes
434 // 2 : write volScalarField with cellLevel for postprocessing
435 // 4 : write current intersections as .obj files
436 debug 0;
437
438 // Merge tolerance. Is fraction of overall bounding box of initial mesh.
439 // Note: the write tolerance needs to be higher than this.
440 mergeTolerance 1e-6;
441
442 // ***** //

```

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:

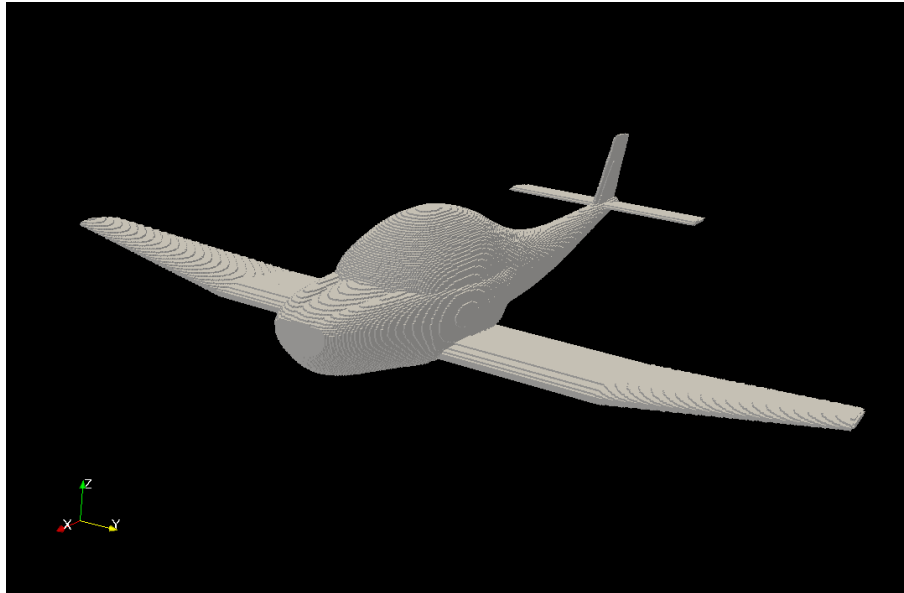


Figure 6.6: Shape of the aircraft at the first step of the meshing process of `snappyHexMesh`

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:

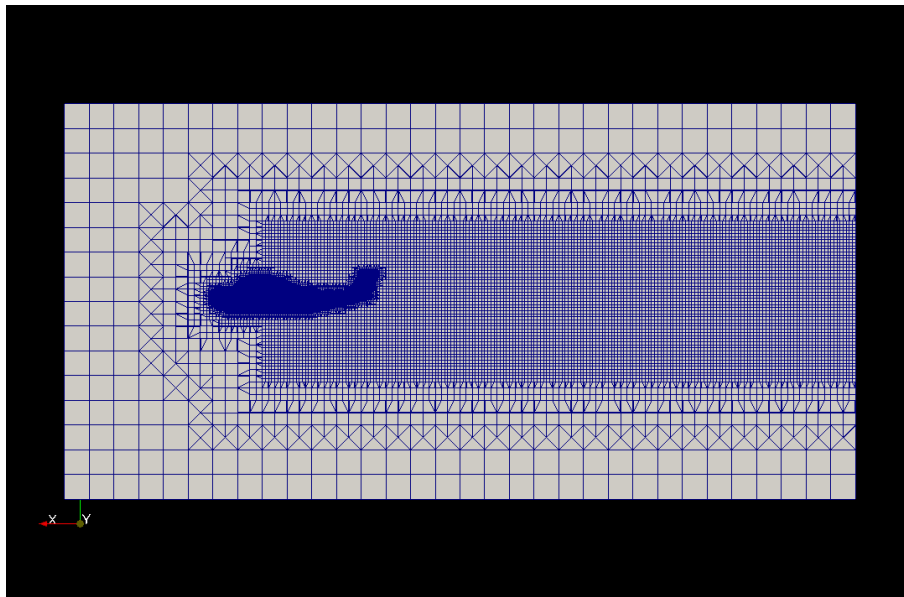
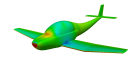


Figure 6.7: Mesh of the domain at the first step of the meshing process of **snappy-HexMesh**

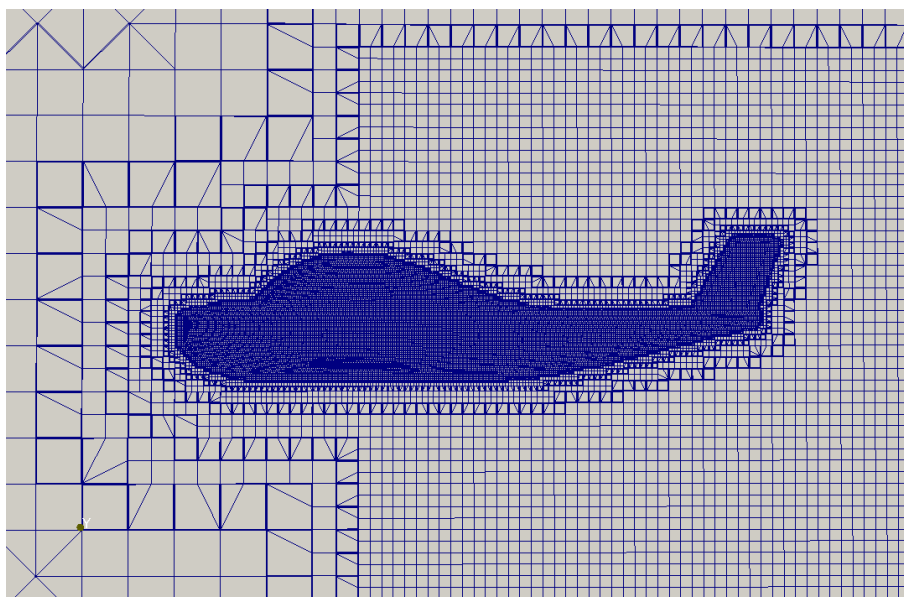


Figure 6.8: Detail of the mesh at the first step of the meshing process of **snappy-HexMesh**

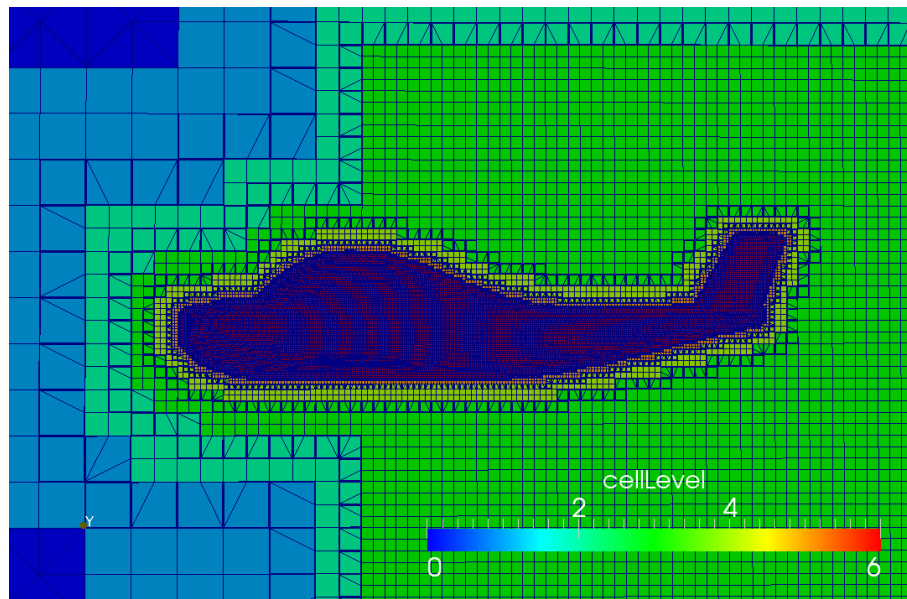


Figure 6.9: Detail of the mesh with a representation of the cell refinement at the first step of the meshing process of **snappyHexMesh**

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.

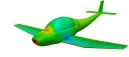
As it can be seen in previous figures, the surface of the aircraft is irregular and cube-based. The second switch of **snappyHexMesh** controls the **snapControls**, involving the displacement of boundary vertices to conform to surface. All vertex displacements are reversible to ensure mesh quality.

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

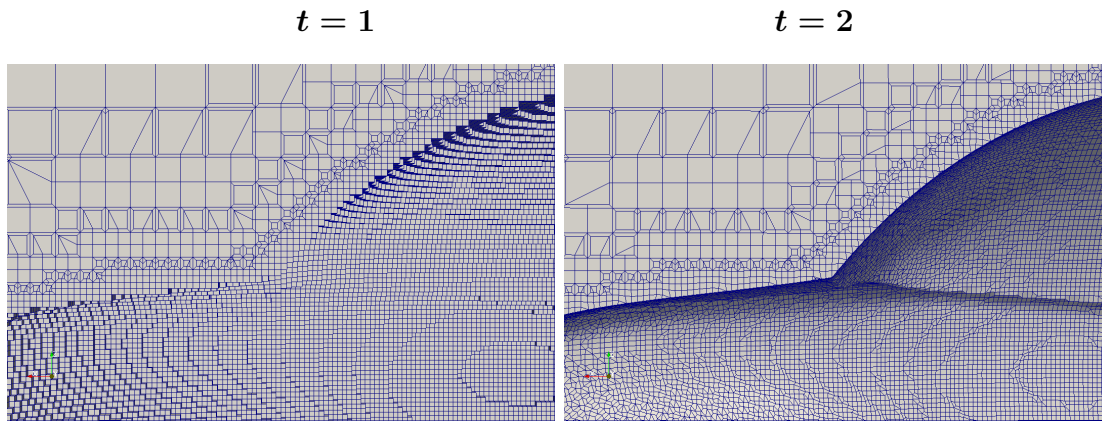
**snappyHexMesh**

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

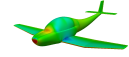
### 6.0.21.2 Boundary and initial conditions

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

```

1  /*----- C++ -----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version: 2.2.1 |
5  |  \ \  /  A n d      | Web: www.OpenFOAM.org |
6  |  \ \  /  M a n i p u l a t i o n |
7  \*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;
13     object        p;
14  }
15  // *****
16
17  dimensions      [0 2 -2 0 0 0 0];
18
19  internalField    uniform 0;
20
21  boundaryField
22  {
23     inlet
24     {
25         type      freestreamPressure;
26     }
27
28     outlet
29     {
30         type      freestreamPressure;
31     }
32
33     aircraftPatch
34     {
35         type      zeroGradient;
36     }
37
38     top
39     {
40         type      slip;
41     }
42
43     bottom
44     {
45         type      slip;
46     }
47

```



```

48     frontAndBack
49     {
50         type                slip;
51     }
52
53 }
54
55 // *****

1  /*----- C++ -----*\
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version: 2.2.1 |
5  |  \ \  /  A n d           | Web:      www.OpenFOAM.org |
6  |  \ \  /  M a n i p u l a t i o n |
7  \*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        volVectorField;
13     object       U;
14 }
15 // *****

16
17 dimensions      [0 1 -1 0 0 0 0];
18
19 internalField    uniform (-45 0 0);
20
21 boundaryField
22 {
23     inlet
24     {
25         type                freestream;
26         freestreamValue     uniform (-45 0 0);
27     }
28
29     outlet
30     {
31         type                freestream;
32         freestreamValue     uniform (-45 0 0);
33     }
34
35     aircraftPatch
36     {
37         type                fixedValue;
38         value                uniform (0 0 0);
39     }
40
41     top
42     {
43         type                slip;
44     }
45
46     bottom
47     {

```



```

48         type            slip;
49     }
50
51     frontAndBack
52     {
53         type            slip;
54     }
55 }
56
57 // *****

```

### Caution:

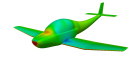
As in the current case there is not *0* directory, the dictionaries of *p*, *U*, *k*, *nut* and *omega* are included within the last directory generated with *snappyHexMesh*

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

```

1  /*-----* C++ *-----*\
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version:  2.2.1                    |
5  |  \ \  /  A n d              | Web:      www.OpenFOAM.org         |
6  |  \ \  /  M a n i p u l a t i o n |                               |
7  \*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        binary;
12     class         volScalarField;
13     location      "2";
14     object        k;
15 }
16 // *****
17
18 dimensions      [0 2 -2 0 0 0 0];
19
20 internalField    uniform 1.215;
21
22 boundaryField
23 {
24     inlet
25     {
26         type      fixedValue;
27         value      uniform 1.215;
28     }
29
30     outlet
31     {
32         type      inletOutlet;
33         inletValue uniform 1.215;
34         value      uniform 1.215;

```



```

35     }
36
37     aircraftPatch
38     {
39         type            kqRWallFunction;
40         value            uniform 1.215;
41     }
42
43     top
44     {
45         type            slip;
46     }
47
48     bottom
49     {
50         type            slip;
51     }
52
53     frontAndBack
54     {
55         type            slip;
56     }
57 }
58
59 // ***** //

1  /*----- C++ -----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version: 2.2.1 |
5  |  \ \  /  A n d      | Web: www.OpenFOAM.org |
6  |  \ \  /  M a n i p u l a t i o n |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        binary;
12     class         volScalarField;
13     location      "2";
14     object        nut;
15 }
16 // ***** //
17
18 dimensions      [0 2 -1 0 0 0 0];
19
20 internalField    uniform 0;
21
22 boundaryField
23 {
24     inlet
25     {
26         type      calculated;
27         value      uniform 0;
28     }
29
30     outlet

```

```

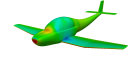
31     {
32         type            calculated;
33         value            uniform 0;
34     }
35
36     aircraftPatch
37     {
38         type            nutkWallFunction;
39         value            uniform 0;
40     }
41
42     top
43     {
44         type            calculated;
45         value            uniform 0;
46     }
47
48     bottom
49     {
50         type            calculated;
51         value            uniform 0;
52     }
53
54     frontAndBack
55     {
56         type            calculated;
57         value            uniform 0;
58     }
59 }
60
61 // *****

```

```

1  /*----- C++ -----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n  | Version:  2.2.1                    |
5  |  \ \  /  A n d      | Web:      www.OpenFOAM.org                |
6  |  \ \ /  M a n i p u l a t i o n  |                               |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        binary;
12     class         volScalarField;
13     location      "2";
14     object        omega;
15 }
16 // *****
17
18 dimensions      [0 0 -1 0 0 0 0];
19
20 internalField    uniform 2147.745;
21
22 boundaryField
23 {
24     inlet

```



```

25     {
26         type          fixedValue ;
27         value          uniform 2147.745;
28     }
29
30     outlet
31     {
32         type          inletOutlet ;
33         inletValue    uniform 2147.745;
34         value          uniform 2147.745;
35     }
36
37     aircraftPatch
38     {
39         type          omegaWallFunction ;
40         value          uniform 2147.745;
41     }
42
43     top
44     {
45         type          slip ;
46     }
47
48     bottom
49     {
50         type          slip ;
51     }
52
53     frontAndBack
54     {
55         type          slip ;
56     }
57 }
58
59 // ***** //
```

$k$ ,  $\nu_t$  and  $\omega$  are parameters required to simulate the case with a turbulence model. Mathematically, they are determined as:

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

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

$$C_\mu = 0.09k \quad (6.7)$$

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

```

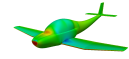
1  /*----- C++ -----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version:  2.2.1                |
5  |  \ \  /  A n d           | Web:      www.OpenFOAM.org        |
6  |  \ \ /  M a n i p u l a t i o n |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     object       transportProperties;
14 }
15 // *****
16
17 transportModel  Newtonian;
18
19 nu              nu [0 2 -1 0 0 0 0] 1.5e-07;
20
21 // *****

```

```

1  /*----- C++ -----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version:  2.2.1                |
5  |  \ \  /  A n d           | Web:      www.OpenFOAM.org        |
6  |  \ \ /  M a n i p u l a t i o n |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     object       RASProperties;
14 }
15 // *****
16
17 RASModel        kOmegaSST;
18
19 turbulence      on;
20

```



```

21 printCoeffs          on;
22
23 // *****

```

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

```

1  /*----- C++ -----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version:  2.2.1                      |
5  |  \ \  /  A n d              | Web:      www.OpenFOAM.org          |
6  |  \ \  /  M a n i p u l a t i o n |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        controlDict;
14 }
15 // *****
16
17 application      simpleFoam;
18
19 startFrom         latestTime;
20
21 startTime         0;
22
23 stopAt            endTime;
24
25 endTime           300;
26
27 deltaT            1;
28
29 writeControl       timeStep;
30
31 writeInterval      1;
32
33 purgeWrite         0;
34
35 writeFormat        binary;
36
37 writePrecision     6;
38
39 writeCompression   uncompressed;
40
41 timeFormat         general;
42
43 timePrecision      6;
44
45 runtimeModifiable true;

```

```

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

```

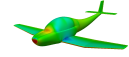
As it can be seen, at lines 47 through 52 the program is using external functions to develop specific tasks. While the case is running, these external functions are called using its dictionaries located within *system*. It facilitates the treatment of the code. For instance, it is not necessary to include 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

```

1  /*----- C++ -----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version: 2.2.1 |
5  |  \ \  /  A n d      | Web: www.OpenFOAM.org |
6  |  \ \  /  M a n i p u l a t i o n |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        fvSchemes;
14 }
15 // *****
16
17 ddtSchemes
18 {
19     default      steadyState;
20 }
21
22 gradSchemes
23 {
24     default      Gauss linear;
25     grad(U)      cellLimited Gauss linear 1;
26 }
27
28 divSchemes
29 {
30     default      none;
31     div(phi,U)   bounded Gauss linearUpwindV grad(U);
32     div(phi,k)   bounded Gauss upwind;
33     div(phi,omega) bounded Gauss upwind;
34     div((nuEff*dev(T(grad(U)))) Gauss linear;
35 }

```



```

36
37 laplacianSchemes
38 {
39     default      Gauss linear corrected;
40 }
41
42 interpolationSchemes
43 {
44     default      linear;
45 }
46
47 snGradSchemes
48 {
49     default      corrected;
50 }
51
52 fluxRequired
53 {
54     default      no;
55     p;
56 }
57
58 // *****

1  /*-----* C++ *-----*\
2  |=====|
3  |  \ \  /  F ield      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O peration  | Version:  2.2.1                      |
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        fvSolution;
14 }
15 // *****

16
17 solvers
18 {
19     p
20     {
21         solver      GAMG;
22         tolerance    1e-7;
23         relTol       0.01;
24         smoother     GaussSeidel;
25         nPreSweeps    0;
26         nPostSweeps   2;
27         cacheAgglomeration on;
28         agglomerator  faceAreaPair;
29         nCellsInCoarsestLevel 10;
30         mergeLevels   1;
31     }
32

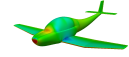
```



```

33     U
34     {
35         solver            smoothSolver;
36         smoother          GaussSeidel;
37         tolerance         1e-8;
38         relTol             0.1;
39         nSweeps            1;
40     }
41
42     k
43     {
44         solver            smoothSolver;
45         smoother          GaussSeidel;
46         tolerance         1e-8;
47         relTol             0.1;
48         nSweeps            1;
49     }
50
51     omega
52     {
53         solver            smoothSolver;
54         smoother          GaussSeidel;
55         tolerance         1e-8;
56         relTol             0.1;
57         nSweeps            1;
58     }
59 }
60
61 SIMPLE
62 {
63     nCorrectors            1;
64     nNonOrthogonalCorrectors 2;
65     pRefCell               0;
66     pRefValue              0;
67 }
68
69 potentialFlow
70 {
71     nNonOrthogonalCorrectors 10;
72 }
73
74 relaxationFactors
75 {
76     fields
77     {
78         p            0.3;
79     }
80     equations
81     {
82         U            0.5;
83         k            0.5;
84         omega        0.5;
85     }
86 }
87
88 cache
89 {

```



```

90     grad(U);
91 }
92
93 // *****

```

### 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

```

1  /*-----* C++ *-----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version:  2.2.1                    |
5  |  \ \  /  A n d              | Web:      www.OpenFOAM.org         |
6  |  \ \  /  M a n i p u l a t i o n |
7  \*-----*/
8
9  // Make sure all fields for functionObjects are loaded. Prevents any
10 // problems running with execFlowFunctionObjects.
11 readFields
12 {
13     // Where to load it from (if not already in solver)
14     functionObjectLibs ("libfieldFunctionObjects.so");
15
16     type                readFields;
17     fields              (p U k);
18 }
19
20
21 // *****

```

#### forceCoeffs

```

1  /*-----* C++ *-----*/
2  |=====|
3  |  \ \  /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O p e r a t i o n | Version:  2.2.1                    |
5  |  \ \  /  A n d              | Web:      www.OpenFOAM.org         |
6  |  \ \  /  M a n i p u l a t i o n |
7  \*-----*/
8
9  forces
10 {
11     type                forces;
12     functionObjectLibs  ("libforces.so"); //Lib to load

```

```

13         outputControl      timeStep;
14         outputInterval     1;
15         patches             (aircraftPatch); //Patch name over forces will be
            calculated
16         pName               p;
17         UName               U;
18         rhoName             rhoInf; //Reference density
19         log                 true;
20         rhoInf              1.225; //Air density
21         CofR                (0 0 0); //Origin for moment calculations
22     }
23
24     forceCoeffs
25     {
26     type forceCoeffs;
27     functionObjectLibs ("libforces.so");
28     patches (aircraftPatch);
29     // pName p;
30     // UName U;
31     rhoName rhoInf;
32     rhoInf 1.225;
33     CofR (0 0 0);
34     liftDir (0 1 0);
35     dragDir (-1 0 0);
36     pitchAxis (0 0 1);
37     magUInf 45; // Free stream velocity
38     lRef 0.01276; // Mean Chord
39     Aref 0.001218; // Ref. Area
40     outputControl timeStep;
41     outputInterval 1;
42     }
43
44     // *****

```

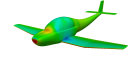
## cuttingPlane

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 practical and rapid view of the  $p$  and  $\mathbf{U}$  fields.

```

1  /*-----* C++ *-----*/
2  |=====|
3  |  \ \  /  F ield      | OpenFOAM: The Open Source CFD Toolbox |
4  |  \ \  /  O peration  | Version: 2.2.1 |
5  |  \ \  /  A nd        | Web: www.OpenFOAM.org |
6  |  \ \ /  M anipulation | |
7  /*-----*/
8
9  cuttingPlane
10 {
11     type          surfaces;
12     functionObjectLibs ("libsampling.so");
13     outputControl  outputTime;
14

```

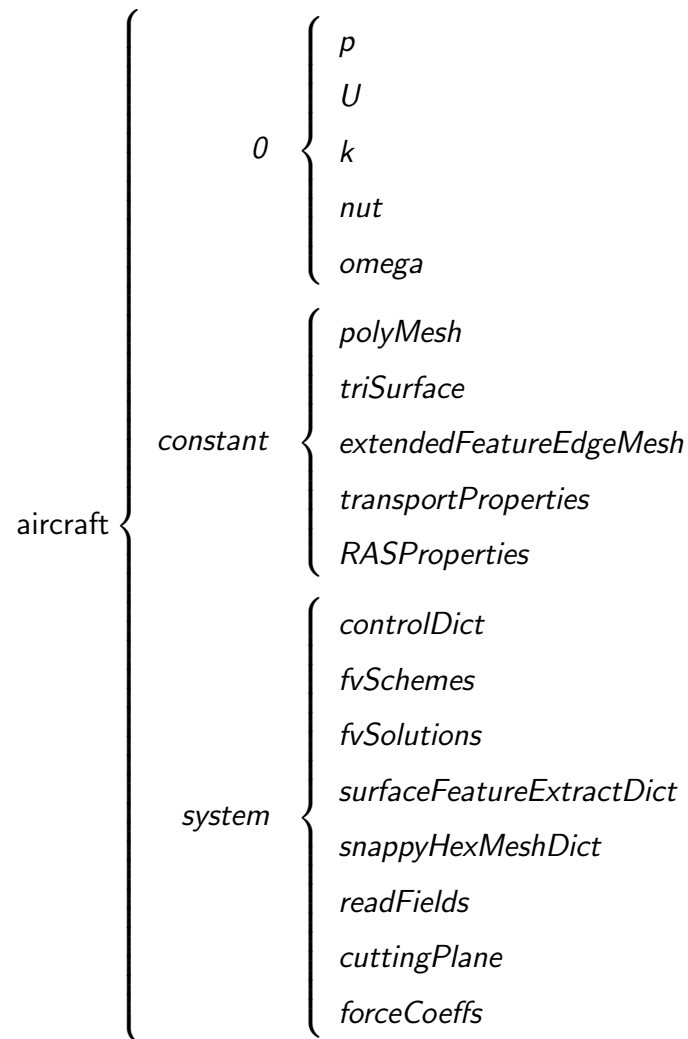



---

```

15     surfaceFormat    vtk;
16     fields           ( p U );
17
18     interpolationScheme cellPoint;
19
20     surfaces
21     (
22         yNormal
23         {
24             type          cuttingPlane;
25             planeType      pointAndNormal;
26             pointAndNormalDict
27             {
28                 basePoint    (0 0 0);
29                 normalVector  (0 1 0);
30             }
31             interpolate      true;
32         }
33     );
34 }
35
36 // ***** //
```

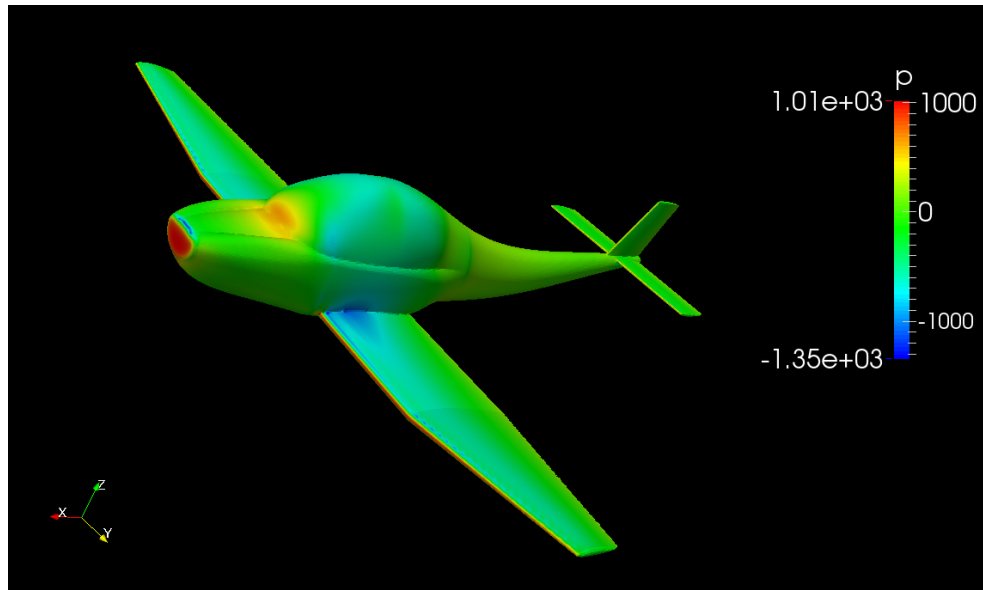
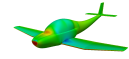
At the end of the pre-processing, the structure of directories, subdirectories and files within `aircraft` should be as follows:



## 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:

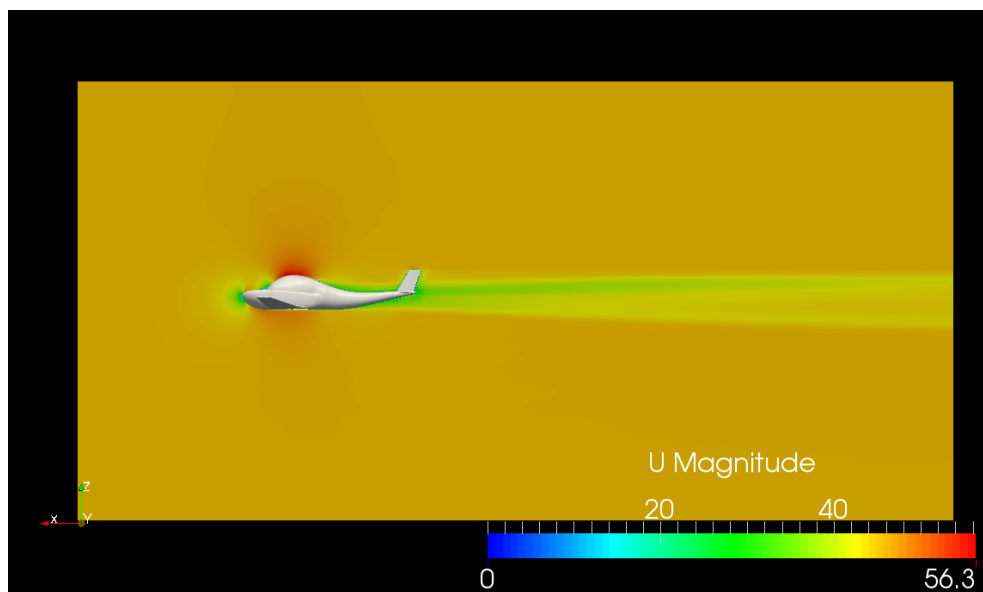


Figure 6.11: Velocity field in the domain of the aircraft case (m/s)

The velocity field around the aircraft:

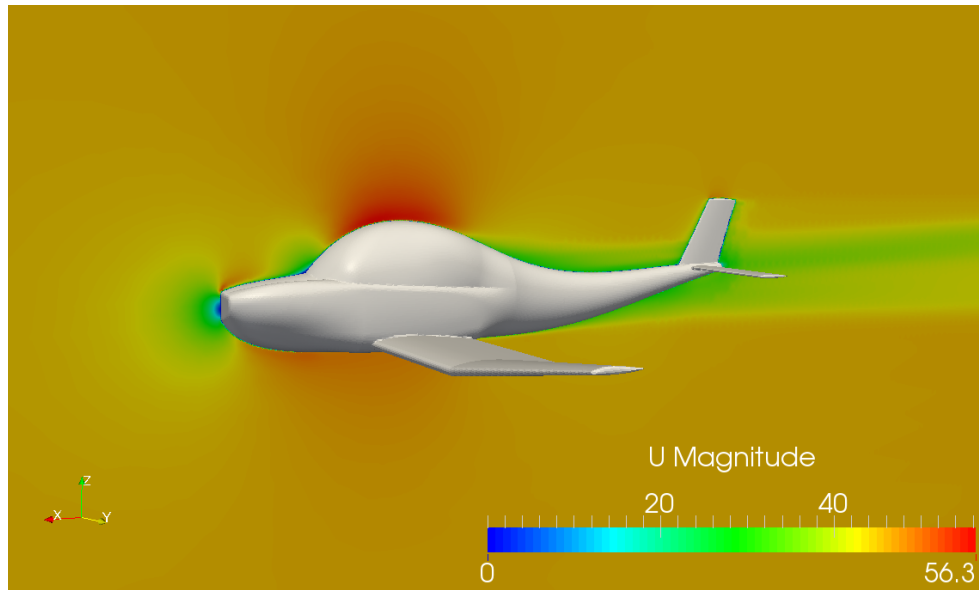


Figure 6.12: Velocity field around the aircraft (m/s)

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.