

Exercise 5: steady state 2D incompressible turbulent flow

Group 12: Giulia Fazzino, Magnus Moldekleiv

October 5, 2026

1 Introduction

The primary objective of this exercise is to calculate and validate the aerodynamic drag of a 2D vehicle profile. For bluff bodies, aerodynamic resistance is heavily dominated by pressure drag due to boundary layer separation and wake formation. In some bodies and Re combinations, a vortex street often forms, preventing steady-state convergence. As this was the case here, we used an unsteady solver (pimpleFoam), and the drag force was calculated both locally on the vehicle surface as a period average, and verified globally using control-volume analysis for momentum balance at the last timestep.

2 Results

The C_d for a 2D donald duck riding in his 313 while hovering 1m above the road, was investigated. The direct surface integration, performed internally by OpenFOAM computing both pressure and viscous forces on the vehicle's boundary, resulted in a total drag coefficient of $C_d \approx 2.46$ (2.1 for SA model, see discussion). Furthermore, a Control Volume analysis was carried out, in order to verify the results obtained. This method resulted in a value of $C_d \approx 2.25$. This falls within the range of temporal variations of C_d , and is thus a reasonable validation. These results align perfectly with what was expected, even though they do not match the typical drag coefficient expected for a car of $C_d \approx 0.25$. That is because the vehicle profile acts as bluff body, causing boundary layer separation. The simulation is also two-dimensional, causing the flow to lack of the 3D effect.

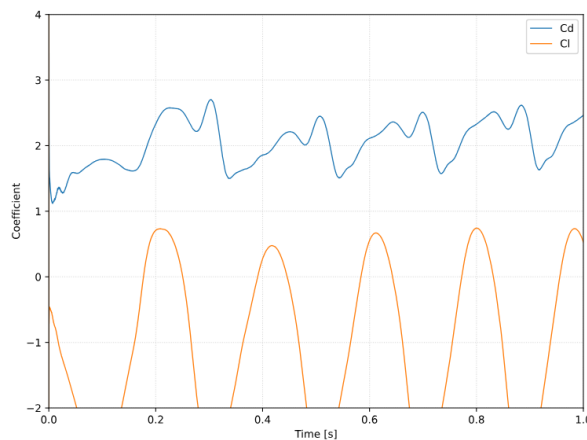


Figure 1: C_d and C_l as calculated by openFOAM for different timesteps

3 Validation

3.1 Simulating known shapes

As there is no experimental data (that we know of) available for a 2D Donald Duck in his 313 car hovering above the road, we decided to simulate two known shapes to get a baseline for whether our simulation captures the real world. The shapes we compared against was a half cylinder and a 45 degree rotated square. These benchmark tests served to validate the chosen mesh resolution, boundary conditions, the `pimpleFoam` solver settings, and the accuracy of both the surface integration and the control volume extraction routines against experimental data. Achieving consistent drag coefficients for these reference shapes provided the necessary confidence to apply the same methodological approach to the more complex, custom geometry. Using the mean value theorem for integrals with trapezoidal quadrature (account for differing timestep lengths), we calculated the average drag coefficients for all shapes. This was done for the time interval $[0.5-1]$, as this was when the drag coefficient seemed to settle into a periodic motion.

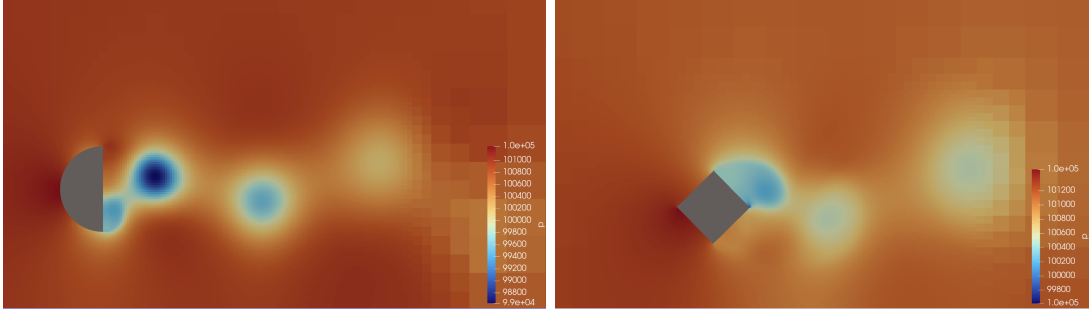


Figure 2: The baseline shapes used for the validation simulations

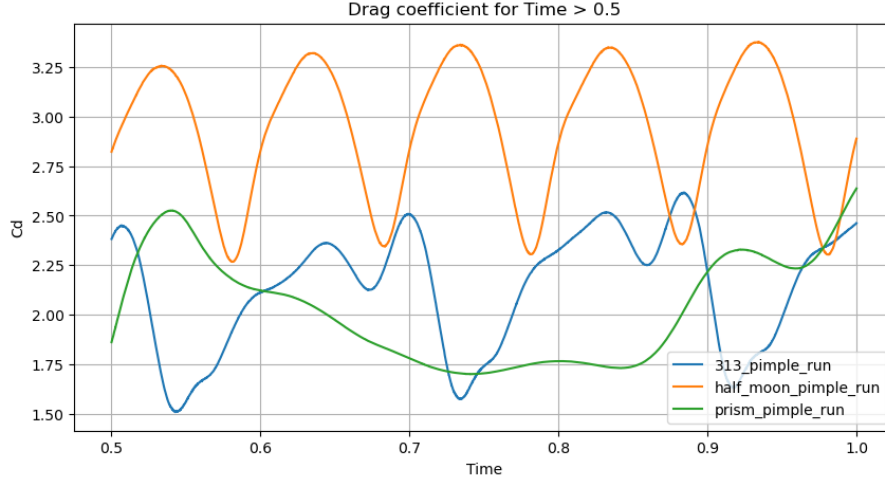


Figure 3: Drag coefficients for the different shapes after approximate steady state reached

run	average_cd	Validation target
313_pimple_run	2.131732	N/A
half_moon_pimple_run	2.926085	1.2
prism_pimple_run	2.031149	1.55

Table 1: Caption

Note that that the cycle mean between for 313 varies by about 5.27% between cycles, thus we cannot truly say that we have reached steady state, and these values should be considered fairly imprecise.

4 Verification

4.1 Simulation setup

The simulation was based on the motorcycle case file, further modified to fit our problem, BCs, yPlus values ect. AI assistance was then used to develop an Allrun file that would copy this base case, and insert each of our test geometries to run cases with identical settings. Assistance was also used to write the code that

Parameter	Value
U_∞	22.222222... m/s
ν	$1.5 \cdot 10^{-5} \frac{m^2}{s}$
ρ	1,0 kg/m ³

Table 2: Physical properties

4.1.1 Boundary conditions

The boundary conditions were applied to simulate a virtual wind tunnel environment. A uniform velocity profile was prescribed at the domain inlet, while zeroGradient condition kinematic pressure was maintained at every boundary. To prevent artificial boundary layer growth along the tunnel walls, slip conditions were applied to the bottom patch for the velocity. A standard no-slip condition was enforced on the vehicle's solid surface. Given the two-dimensional nature of the simulation, the lateral planes were designated as empty. The complete setup for the velocity (U) and kinematic pressure (p) fields is summarized in Table 3.

Table 3: Summary of boundary conditions for the velocity and pressure fields.

Boundary Patch	Velocity (U)	Kinematic Pressure (p)
inlet	fixedValue ($U_x = 22.22$ m/s)	zeroGradient
outlet	zeroGradient	zeroGradient
lowerWall	slip	zeroGradient
Vehicle Surface	noSlip	zeroGradient
frontAndBack	empty	empty

4.1.2 Meshing

For our background mesh, we used a box dimensioned based on car lengths, so we had 5 car-lengths upstream of origo, 11 downstream of origo, and 6 over origo. One car length was 1.09585 giving a total box size of 17.53×6.58 m. Around the geometrical shapes, we had a refinement box placed one car length in front of DD, three behind him and t1 above him. This we generated using snappyHexMesh with three surface layers for the bodies, and otherwise the standard settings from the motorcycle tutorial case.

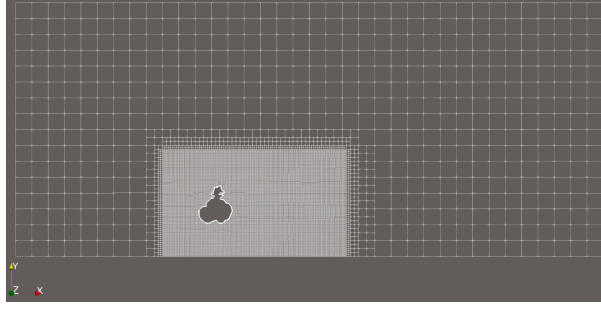


Figure 4: Detail of the computational mesh around the 2D car profile



Figure 5: Highlight of the computational mesh around the 2D car profile

4.1.3 yPlus values

The snappyHexMesh was adjusted to give yPlus values suitable for using wall functions (and scalable wall functions) for the kOmegaSST model, calculated for the patches on the stl figure. Hitting all yPlus values within the range (30-300) for wall functions proved difficult, as it naturally becomes lower for stagnation points and higher for extreme points, so we settled for trying to hit the average quite close too the relevant range (we also expect these values to fluctuate from timestep to timestep - below are the values for the final timestep). By the colouring of the figure below, we see that the spikes are limited, and as such it appears most values are in the vicinity of the average. We will also note that k-omega SST supports scalable wall functions, and as such values down to 1 (below 30) are less of an issue.

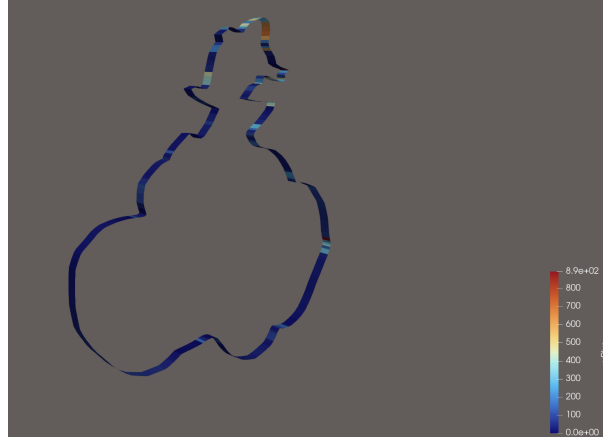


Figure 6: Detail of the yPlus value around the 2D profile

Shape	yPlus min	yPlus max	yPlus avg
313 DD	1.245	892.682	95.7469
half cylinder	4.147	491.139	49.463
45° square	1.894	1333.26	46.915

Table 4: yPlus values

4.1.4 Simulation control

Parameter	value
Max Courant number	0.5
Duration	1 s
Max timestep	0.005s*
number of outer correctors per timestep	2
number of correctors	2
Relaxation factors	U (0.9), k (0.7), omega (0.7)
p	GAMG: tol 1e-7, relTol 0.01
U, k, omega	smoothSolver: tol 1e-8, relTol 0.1

Table 5: Key simulation parameters

* Note: a quick check verified that Courant number was binding, not max dt.

4.1.5 Turbulence modelling

A kOmegaSST model was used for turbulence modeling. This model was selected due to its exceptional accuracy in predicting flow detachment and behavior under severe adverse pressure gradients. The $k - \omega$ SST model effectively combines the robustness of the $k - \omega$ formulation near the solid walls—allowing for a precise resolution of the boundary layer without overly restrictive y^+ constraints—with the free-stream independence of the $k - \epsilon$ model in the outer regions of the domain.

4.2 Parameter study

Using codex (AI assistance tool from OpenAI), a python script was developed that ran the simulation for 313 while changing one parameter at a time. Since this was using pimpleFoam, this took about 4,5 h on my PC. In the interest of time, and keeping consistency with the

validation cases, changing the mesh refinement box size was not investigated - this is a plausible future avenue for investigating which factors might impact C_d , especially since we got the largest changes from changing the total domain size. For the numerical sensitivity study, we took the periodic average of C_d for two periods, similar to the original one (as the results changed so little that we were able to find the same pattern to average over)

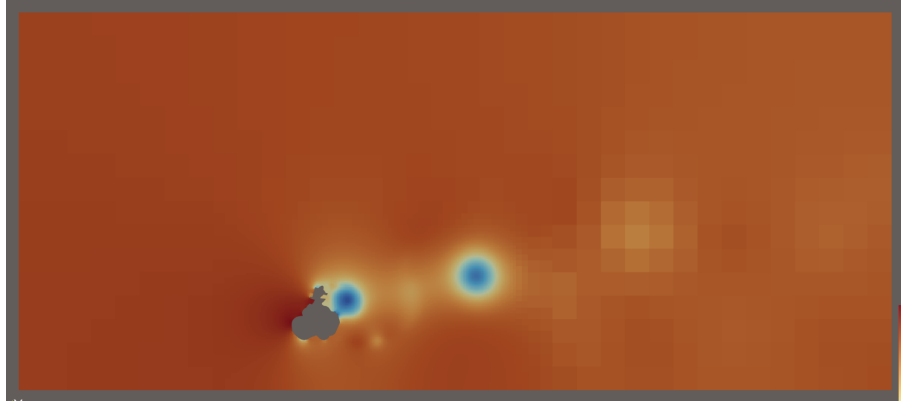


Figure 7: Pressure field for final timestep of 313 simulation - note von karman vorticies forming behind it preventing convergence of a steady state solver

Parameter	Old value	New lower	Relative change ΔC_d (%)	New higher	Relative change ΔC_d (%)
Linear tolerance multiplier	1	0.1	0.001 %	10	0.001 %
Maximum Courant number	0.5	0.25	-0.601 %	1.0	0.072 %
Inlet turbulence intensity	5%	1%	0.149 %	10%	-0.011 %
PIMPLE outer correctors	2	—	—	4	-0.057 %

Table 6: Changing numerical parameters

Parameter	Old value	New lower value	ΔC_d lower (%)	New higher value	ΔC_d higher (%)
Inlet distance (L)	5	4	Mesh quality issues	6	-2.147
Outlet coordinate (L)	11	10	-2.844	12	-8.474
Roof coordinate (L)	6	5	-7.392	7	-9.213

Table 7: Changing domain area

From this, our results seem fairly insensitive to changes in the numerical parameters, while most sensitive to changes in domain size. We do find it a bit suspicious that C_d decreases for changes in both directions, as this would imply that both changes could bring us closer to the validation values (assuming that our C_d is too high given our validation values are too high). We will primarily look closer at the results from changing the roof height, as this gave the largest differences, and as the symmetry-plane BC could create interactions between the roof and the rest of the domain. Below we compare the graphs of how the force coefficients develop over time (small differences in how the plots were generated).

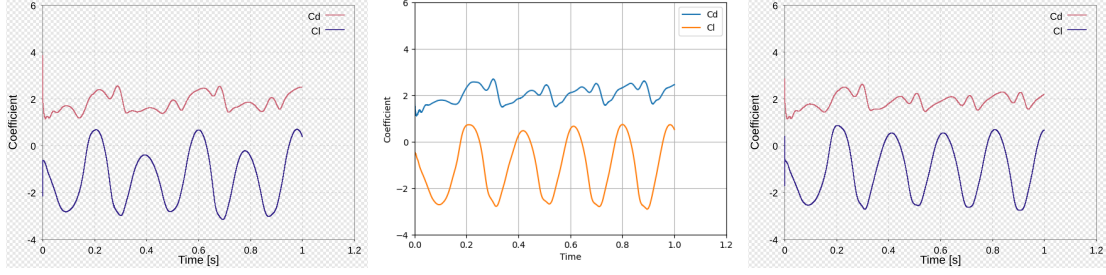


Figure 8: CD development for 313 for changing roof height. From left to right -1L, original, +1L

From these plots, we see that how the periodicity develops appears quite influenced by the boundaries. From our fixed averaging window, a large part of the difference in Cd might be explained from differences in how transients develop affecting which values are included in our average. From visual inspection, the difference in the Cd pattern appear small for increasing the roof height, compared to decreasing it. With the differences in cycle patterns, and inter cycle differences, still seen for these timesteps, we do not think it we have enough data to properly conclude whether changing the domain parameters would affect our study.

However, if we assume that a bigger domain will give a better result, due to less interference from inlets, enough space for wake to develop and smaller interactions with the roof ect, we can try running one more simulation - this time for the validation prism as we know what the right answer should be for this one - where we increase inlet length, roof height and outlet length, as well as refinement box size. This gave a Cd average of 2.4, compared to 2.03 for the original one, a relative change of 18 % in the wrong direction. We suspect this to be caused by our fixed avegaging window, rather than actual large changes in the drag coefficient.

4.3 CV Analysis

A transverse line probe was placed at the outlet plane using ParaView to extract the horizontal velocity (u_x) and kinematic pressure (p) profiles. The 1 and 2 equations were used to get the results (see Appendix 7.3).

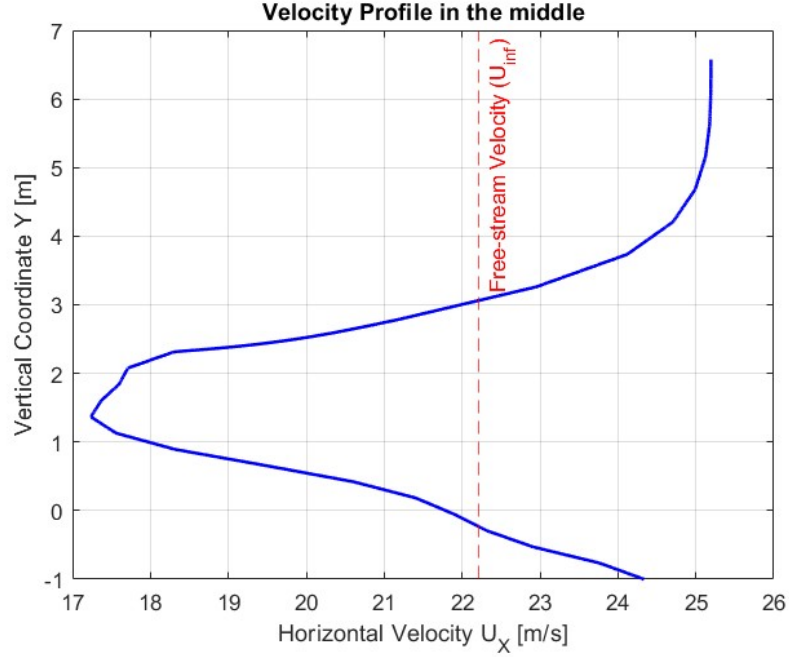


Figure 9: Velocity profile for the simulation

--- ADVANCED CONTROL VOLUME ANALYSIS ---

Reference Free-stream Pressure (p_{inf}): 101640.0000 m^2/s^2

Momentum Drag Contribution: -57.5302 N/m

Outlet Drag Contribution: 541.0523 N/m

Calculated Total Drag (D) : 23.9433 N

Final Drag Coefficient (C_d): 2.2127

It should be noted that the transverse line probe used for the data extraction was not exactly positioned at the numerical outlet of the domain. Sampling the velocity and pressure fields directly at the domain boundary can occasionally introduce non-physical artifacts. To ensure the physical fidelity of the momentum integral and to avoid boundary-induced disturbances, the control volume evaluation plane was placed at an intermediate downstream location ($x = 5.5$). This position is sufficiently far from the bluff body to capture the fully developed wake, yet safely upstream of the domain's exit plane. This analytical result matches the value calculated by openFoam.

5 Discussion

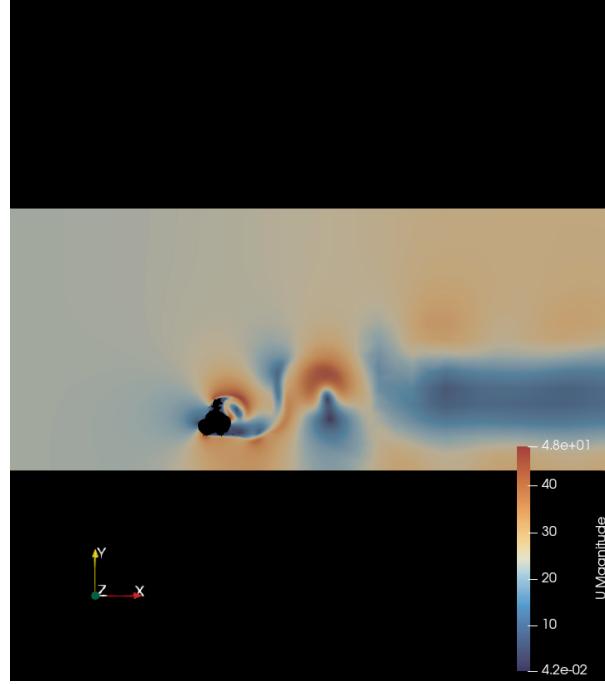


Figure 10: Velocity field for final timestep of 313 simulation - note von Karman vortices forming behind it preventing convergence of a steady state solver

The transient simulation captured the fundamental fluid dynamic behavior typical of a bluff body. Analysis of the kinematic pressure field reveals a distinct stagnation point at the front of the vehicle, generating a strong high-pressure zone. Downstream, boundary layer separation occurs, leading to the formation of a pronounced von Kármán vortex street in the wake. The alternating vortex cores are characterized by significant low-pressure regions (suction zones), visually confirming that the total aerodynamic resistance is heavily pressure-dominated.

5.1 Possibility of Floor effects - investigating a free-stream setup

5.1.1 k-omega SST for prism in free-stream

In our efforts to make sure that we keep everything similar between runs, we forgot to account for the fact that the floor might be impacting the drag of our free shapes. To investigate this, we made one run keeping all else standard (hexMesh ect) (but max Courant of 1 as this was seen to not affect the results), where we removed the bottom floor, replaced this with a symmetry plane, and moved this as far away as the top plane. As seen in fig. 11, this has a large effect on the development of the transients, and while they seem lower, we also from the Cd plot cannot conclude that the transients have fully developed. A better choice for this investigation might have been the half-cylinder, as there the transients seemed to settle down faster.

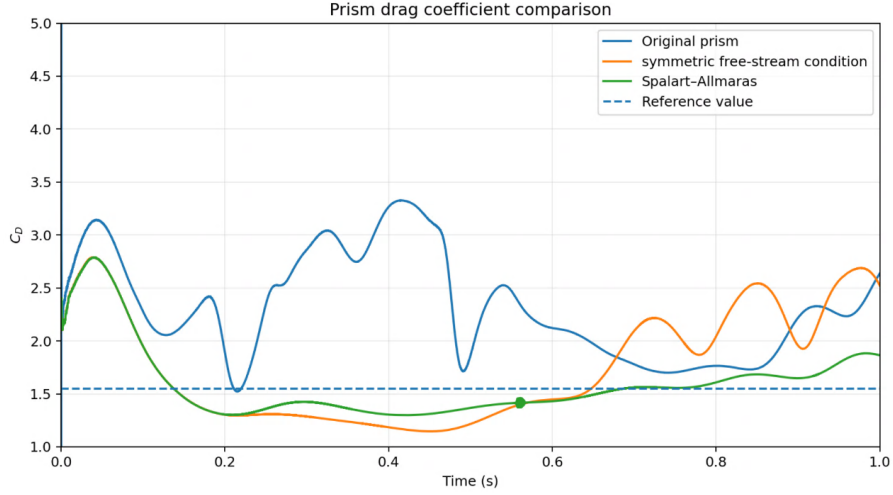


Figure 11: Free stream compatison for prism: Spalart-Allmaras, k-omega SST

Using the free-stream setup, we also perform one run using the Spalart-Allmaras turbulence model, which did a better job of reproducing out reference C_d over the timescales we simulated. On the other hand, we are now testing turbulence models for external aerodynamics (i.e both) on free-stream conditions, when we in our real case have apply them to a case with substantial channel flow (between figure and floor). This raises the question of how valid this validation actually is. In any case, we quickly also ran our 313 setup using SA, which gave a C_d cycle average of 2.1 (and also less intense vortexes, similar as what we saw for the prism fig. 12).

Our key takeaway is that a a better/faster approach might have been to first test most parameters with a steady solver, accepting that we do not properly converge, then choosing a *reasonable setup* which give the closest results. And only then check all validation cases with an unsteady solver

6 Conclusions

While we were able to evaluate the drag of a 2D cartoon car and gotten out a number, our main takeaways are on how to do CFD analysis properly. First is the importance of doing a parameter study *early*, for the *validation cases*, on as cheap test cases as one can get away with. As we have shown, parameters such as turbulence model, or domain, can have a large effect on the results. Second, while these validation cases should be as close as possible to the original tests, but this is often far from trivial. Third, including the pressure deficit and accounting for the blockage effect when applying the Control Volume method in confined, near-wake computational domains, can make a large difference for C.V analysis.

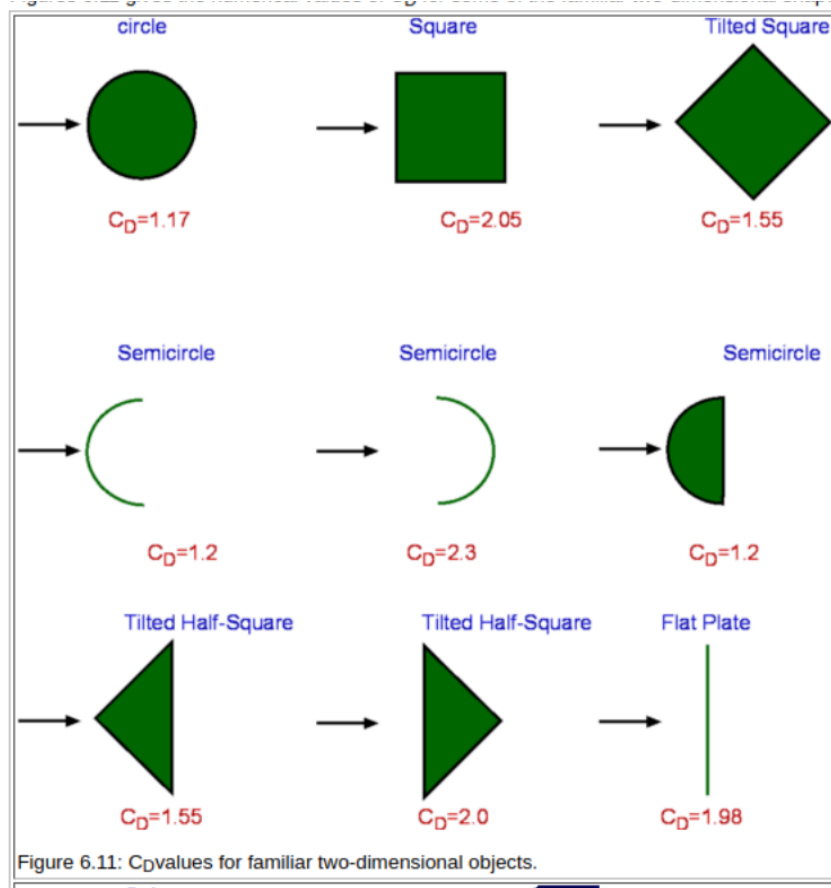
7 Appendix

7.1 Source-code

<https://github.com/M4gnusMJ/313-2D-CFD>

7.2 Reference drag coefficients

https://www-mdp.eng.cam.ac.uk/web/library/enginfo/aerothermal_dvd_only/aero/fprops/introvisc/node11.html



7.3 Supplementary formulations

The total drag per unit depth (D) was evaluated using the momentum conservation equation:

$$D = -\rho U_{\text{inf}}^2 A + \iint_{\text{Outlet}} \rho u^2 dA \quad (1)$$

where ρ is the fluid density and U_{in} is the free-stream velocity.

The drag coefficient was evaluated using the following equation:

$$C_d = \frac{2F_d}{\rho U_{\text{inf}}^2 A} \quad (2)$$

7.4 MATLAB script

```

1
2 % =====
3 % Control Volume Analysis for Drag Coefficient (Cd) Calculation
4 % INCLUDES DYNAMIC PRESSURE DRAG CORRECTION FOR NEAR-WAKE MEASUREMENTS
5 % =====
6
7 clear; clc; close all;
8
9 % 1. DEFINE SIMULATION PARAMETERS
10 % -----
11 U_inf = 22.22;          % Free-stream velocity [m/s]
12 rho = 1;               % Air density (kinematic pressure in OpenFOAM) [kg/m^3]
13 z_thickness = 0.04;    % Extrusion thickness in the Z direction [m]
14 H_ref = 1.09585;       % Reference height of the "Donald Duck" model [m]
15
16 A_ref = H_ref * z_thickness;
17
18 % 2. LOAD DATA FROM PARAVIEW CSV
19 % -----
20 filename = 'data.csv'; % Ensure you export 'p' and 'U_X'
21
22 data = readtable(filename, 'PreserveVariableNames', true);
23
24 % Try to extract Y coordinates, U_X velocity, and Pressure (p)
25 try
26 y = data('Points:1');
27 u = data('U:0');
28 p = data('p'); % This is the kinematic pressure column
29 catch
30 warning('Check your CSV column headers. Update the names in the script if they
31         differ. ');
32 y = data.Points_1;
33 u = data.U_0;
34 p = data.p;
35 end
36
37 % 4. DYNAMICALLY DEFINE FREE-STREAM PRESSURE (p_inf)
38 % -----
39 p_inf = p(1);
40
41 % 5. COMPUTE THE INTEGRALS (Trapezoidal Rule)
42 % -----
43 % A. Momentum Deficit Term
44 momentum_integrand = rho .* u .* (U_inf - u);
45 drag_momentum = trapz(y, momentum_integrand);
46
47 % B. Pressure Deficit Term (Pressure Drag)
48 drag_pressure = rho.*U_inf^2.*H_ref;
49
50 % Total Drag per unit depth is the sum of both terms
51 drag_per_unit_depth = -drag_momentum + drag_pressure;
52
53 % 6. CALCULATE TOTAL DRAG AND DRAG COEFFICIENT
54 % -----
55 D_total = drag_per_unit_depth * z_thickness;
56 Cd_cv = D_total / (0.5 * rho * U_inf^2 * A_ref);
57
58 % 7. DISPLAY RESULTS
59 % -----
60 fprintf('\n--- ADVANCED CONTROL VOLUME ANALYSIS ---\n');
61 fprintf('Reference Free-stream Pressure (p_inf): %.4f m^2/s^2\n', p_inf);

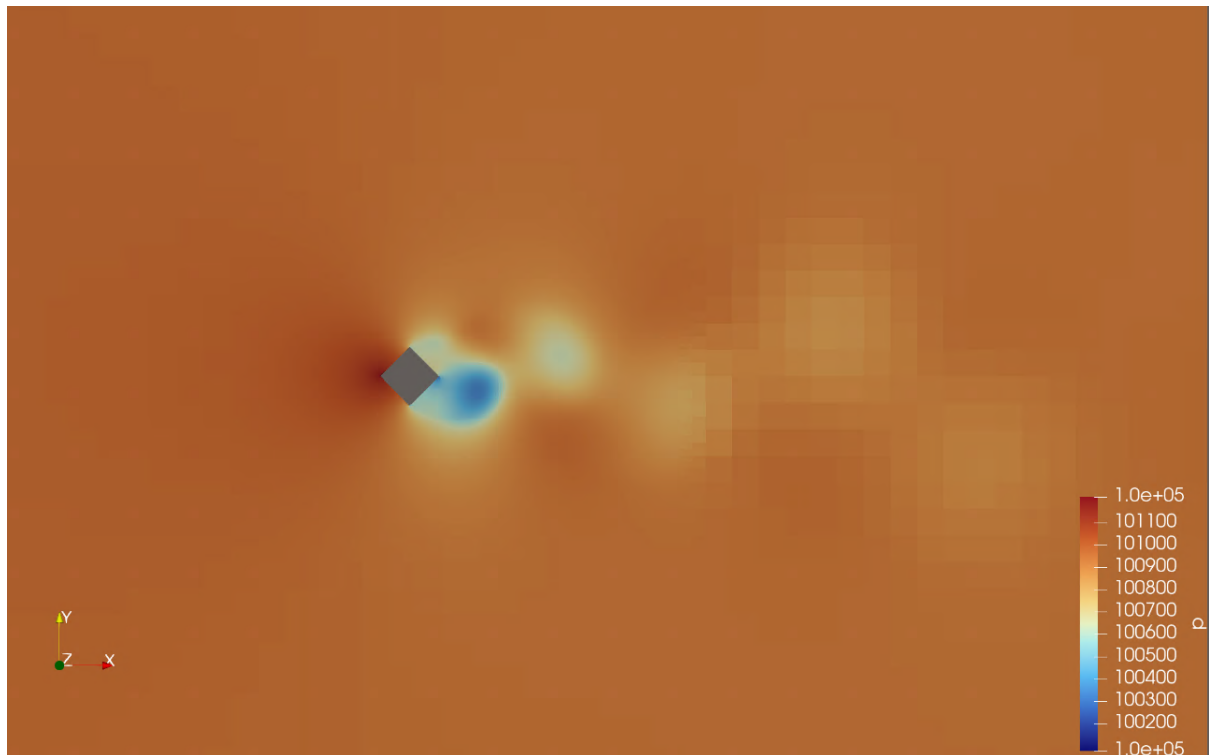
```

```

61 fprintf('Momentum Drag Contribution: %.4f N/m\n', drag_momentum);
62 fprintf('Outlet Drag Contribution: %.4f N/m\n', drag_pressure);
63 fprintf('-----\n');
64 fprintf('Calculated Total Drag (D) : %.4f N\n', D_total);
65 fprintf('Final Drag Coefficient (Cd): %.4f\n\n', Cd_cv);
66
67 %Plot the velocity profile to visually check the wake
68 figure;
69 plot(u, y, 'b-', 'LineWidth', 1.5);
70 hold on;
71 xline(U_inf, 'r--', 'Free-stream Velocity (U_{inf})');
72 xlabel('Horizontal Velocity U_X [m/s]');
73 ylabel('Vertical Coordinate Y [m]');
74 title('Velocity Profile in the middle');
75 grid on;

```

7.5 Final pressure field Spalart-Allmaras freestream run for prism



One might think that it could be possible to use Spalart-Allmaras turbulence model with simple foam instead of our current transient treatment as a reasonable approximation, as the transients seemed less strong for this model. When tested, this failed to converge just the same.

7.6 SA 313 run final timestep image

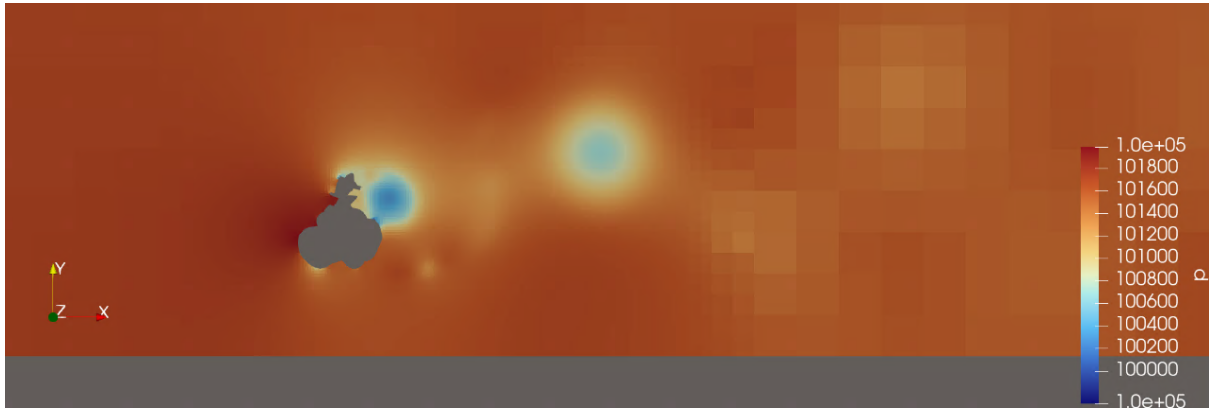


Figure 12: SA 313 run final timestep pressure field

7.7 OpenFoam files

7.7.1 BlockMeshDict

```
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: v2412 /
5  / \ \ / A n d / Website: www.openfoam.com /
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 scale      1;          //All stl's are given in meters
18
19 upstream_dist      #calc "5*1.09585";          // Car lengths (1.09585
    is height & width of 313)
20 downstream_dist    #calc "(10+1)*1.09585";      // +1 to account for
    length of car itself from origin
21 roof_height        #calc "(5+1)*1.09585";      // Roof height above car
22
23 vertices            // Background mesh
24 (
25     (-$upstream_dist      -1          0)        // -5 is 5 car-lengths (
    approx) upstream
26     ($downstream_dist      -1          0)        //
27     ($downstream_dist      $roof_height  0)
28     (-$upstream_dist      $roof_height  0)
29     (-$upstream_dist      -1          0.04)      // Cuts through all bodies
    on both sides
30     ($downstream_dist      -1          0.04)
31     ($downstream_dist      $roof_height  0.04)
32     (-$upstream_dist      $roof_height  0.04)
33 );
34
35 // As per the guide, want an aspect ratio of approximately 1
36 // We therefore parametrise this as another parameter we might want to change
37 // Max cell size
38 max_h      0.5;
39 Nx          #calc "static_cast<int>(std::ceil(($upstream_dist+$downstream_dist)/
    $max_h))";
40 Ny          #calc "static_cast<int>(std::ceil(($roof_height+$upstream_dist)/$max_h))";
41
42 // We have not cared about aspect ratio on z direction
43
44 blocks
45 (
46     hex (0 1 2 3 4 5 6 7) ($Nx $Ny 1) simpleGrading (1 1 1)
47 );
48
49 edges
50 (
51 );
52
53 boundary
54 (
```

```

55     front
56     {
57         type patch;      // Z = 0; empty after extrusion
58         faces
59         (
60             (0 3 2 1)
61         );
62     }
63     back
64     {
65         type patch;      // Z = 0.04; empty after extrusion
66         faces
67         (
68             (4 5 6 7)
69         );
70     }
71     inlet
72     {
73         type patch;
74         faces
75         (
76             (0 4 7 3)
77         );
78     }
79     outlet
80     {
81         type patch;
82         faces
83         (
84             (2 6 5 1)
85         );
86     }
87     lowerWall
88     {
89         type patch;
90         faces
91         (
92             (0 1 5 4)
93         );
94     }
95     upperWall
96     {
97         type symmetryPlane;
98         faces
99         (
100             (2 3 7 6)
101         );
102     }
103 );
104
105
106 // *****

```

7.7.2 SnappyHexMeshDict

```

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: v2412 /
5  / \ \ / A n d / Website: www.openfoam.com /
6  / \ \ / M a n i p u l a t i o n / /
7  /*-----*- C++ -*-----*/
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            true;
20 addLayers       true;
21
22
23 // Geometry. Definition of all surfaces. All surfaces are of class
24 // searchableSurface.
25 // Surfaces are used
26 // - to specify refinement for any mesh cell intersecting it
27 // - to specify refinement for any mesh cell inside/outside/near
28 // - to 'snap' the mesh boundary to the surface
29
30 // Bounding boxes
31 L      1.09585;
32 box_top    #calc "2*$L";
33 box_back   #calc "(1+3)*$L"; // i.e resolving three car lengths of wake
34
35
36 geometry
37 {
38     body.stl
39     {
40         type triSurfaceMesh;
41         name body;
42     }
43
44     refinementBox
45     {
46         type box;
47         min  (-$L      -$L      0.0);
48         max  ($box_back $box_top 0.04);
49     }
50 }
51
52
53 // Settings for the castellatedMesh generation.
54 castellatedMeshControls
55 {
56
57     // Refinement parameters
58     // ~~~~~
59
60     // If local number of cells is >= maxLocalCells on any processor
61     // switches from from refinement followed by balancing

```

```

62 // (current method) to (weighted) balancing before refinement.
63 maxLocalCells 100000;
64
65 // Overall cell limit (approximately). Refinement will stop immediately
66 // upon reaching this number so a refinement level might not complete.
67 // Note that this is the number of cells before removing the part which
68 // is not 'visible' from the keepPoint. The final number of cells might
69 // actually be a lot less.
70 maxGlobalCells 2000000;
71
72 // The surface refinement loop might spend lots of iterations refining just
73 // a
74 // few cells. This setting will cause refinement to stop if <=
75 // minimumRefine
76 // are selected for refinement. Note: it will at least do one iteration
77 // (unless the number of cells to refine is 0)
78 minRefinementCells 10;
79
80 // Allow a certain level of imbalance during refining
81 // (since balancing is quite expensive)
82 // Expressed as fraction of perfect balance (= overall number of cells /
83 // nProcs). 0=balance always.
84 maxLoadUnbalance 0.10;
85
86 // Number of buffer layers between different levels.
87 // 1 means normal 2:1 refinement restriction, larger means slower
88 // refinement.
89 nCellsBetweenLevels 3;
90
91
92 // Explicit feature edge refinement
93 // ~~~~~
94
95 // Specifies a level for any cell intersected by its edges.
96 // This is a featureEdgeMesh, read from constant/triSurface for now.
97 features
98 (
99     {
100         file "body.eMesh";
101         level 6;
102     }
103 );
104
105
106 // Surface based refinement
107 // ~~~~~
108
109 // Specifies two levels for every surface. The first is the minimum level,
110 // every cell intersecting a surface gets refined up to the minimum level.
111 // The second level is the maximum level. Cells that 'see' multiple
112 // intersections where the intersections make an
113 // angle > resolveFeatureAngle get refined up to the maximum level.
114
115
116 refinementSurfaces
117 {
118     body
119     {
120         // Surface-wise min and max refinement level
121         level (5 6);
122     }

```

```

123         // Optional specification of patch type (default is wall). No
124         // constraint types (cyclic, symmetry) etc. are allowed.
125         patchInfo
126         {
127             type wall;
128             inGroups (motorBikeGroup);
129         }
130     }
131 }
132
133 // Resolve sharp angles
134 resolveFeatureAngle 30;
135
136
137 // Region-wise refinement
138 // ~~~~~
139
140 // Specifies refinement level for cells in relation to a surface. One of
141 // three modes
142 // - distance. 'levels' specifies per distance to the surface the
143 //   wanted refinement level. The distances need to be specified in
144 //   descending order.
145 // - inside. 'levels' is only one entry and only the level is used. All
146 //   cells inside the surface get refined up to the level. The surface
147 //   needs to be closed for this to be possible.
148 // - outside. Same but cells outside.
149
150 refinementRegions
151 {
152     refinementBox
153     {
154         mode inside;
155         levels ((1E15 4));
156     }
157 }
158
159
160 // Mesh selection
161 // ~~~~~
162
163 // After refinement patches get added for all refinementSurfaces and
164 // all cells intersecting the surfaces get put into these patches. The
165 // section reachable from the locationInMesh is kept.
166 // NOTE: This point should never be on a face, always inside a cell, even
167 // after refinement.
168 locationInMesh (3.0001 3.0001 0.02);
169
170
171 // Whether any faceZones (as specified in the refinementSurfaces)
172 // are only on the boundary of corresponding cellZones or also allow
173 // free-standing zone faces. Not used if there are no faceZones.
174 allowFreeStandingZoneFaces true;
175 }
176
177
178
179 // Settings for the snapping.
180 snapControls
181 {
182     //- Number of patch smoothing iterations before finding correspondence
183     // to surface
184     nSmoothPatch 3;
185

```

```

186 // - Relative distance for points to be attracted by surface feature point
187 // or edge. True distance is this factor times local
188 // maximum edge length.
189 tolerance 2.0;
190
191 // - Number of mesh displacement relaxation iterations.
192 nSolveIter 30;
193
194 // - Maximum number of snapping relaxation iterations. Should stop
195 // before upon reaching a correct mesh.
196 nRelaxIter 5;
197
198 // Feature snapping
199
200 // - Number of feature edge snapping iterations.
201 // Leave out altogether to disable.
202 nFeatureSnapIter 10;
203
204 // - Detect (geometric only) features by sampling the surface
205 // (default=false).
206 implicitFeatureSnap false;
207
208 // - Use castellatedMeshControls::features (default = true)
209 explicitFeatureSnap true;
210
211 // - Detect points on multiple surfaces (only for explicitFeatureSnap)
212 multiRegionFeatureSnap false;
213 }
214
215
216
217 // Settings for the layer addition.
218 addLayersControls
219 {
220 // Are the thickness parameters below relative to the undistorted
221 // size of the refined cell outside layer (true) or absolute sizes (false).
222 relativeSizes true;
223
224 // Per final patch (so not geometry!) the layer information
225 layers
226 {
227     "body.*"
228     {
229         nSurfaceLayers 3;
230     }
231 }
232
233 // Expansion factor for layer mesh
234 expansionRatio 1.0;
235
236 // Wanted thickness of final added cell layer. If multiple layers
237 // is the thickness of the layer furthest away from the wall.
238 // Relative to undistorted size of cell outside layer.
239 // See relativeSizes parameter.
240 finalLayerThickness 0.9;
241
242 // Minimum thickness of cell layer. If for any reason layer
243 // cannot be above minThickness do not add layer.
244 // Relative to undistorted size of cell outside layer.
245 minThickness 0.1;
246
247 // If points get not extruded do nGrow layers of connected faces that are
248 // also not grown. This helps convergence of the layer addition process

```

```

249 // close to features.
250 // Note: changed(corrected) w.r.t 1.7.x! (didn't do anything in 1.7.x)
251 nGrow 0;
252
253 // Advanced settings
254
255 // When not to extrude surface. 0 is flat surface, 90 is when two faces
256 // are perpendicular
257 featureAngle 60;
258
259 // At non-patched sides allow mesh to slip if extrusion direction makes
260 // angle larger than slipFeatureAngle.
261 slipFeatureAngle 30;
262
263 // Maximum number of snapping relaxation iterations. Should stop
264 // before upon reaching a correct mesh.
265 nRelaxIter 3;
266
267 // Number of smoothing iterations of surface normals
268 nSmoothSurfaceNormals 1;
269
270 // Number of smoothing iterations of interior mesh movement direction
271 nSmoothNormals 3;
272
273 // Smooth layer thickness over surface patches
274 nSmoothThickness 10;
275
276 // Stop layer growth on highly warped cells
277 maxFaceThicknessRatio 0.5;
278
279 // Reduce layer growth where ratio thickness to medial
280 // distance is large
281 maxThicknessToMedialRatio 0.3;
282
283 // Angle used to pick up medial axis points
284 // Note: changed(corrected) w.r.t 1.7.x! 90 degrees corresponds to 130
285 // in 1.7.x.
286 minMedialAxisAngle 90;
287
288
289 // Create buffer region for new layer terminations
290 nBufferCellsNoExtrude 0;
291
292
293 // Overall max number of layer addition iterations. The mesher will exit
294 // if it reaches this number of iterations; possibly with an illegal
295 // mesh.
296 nLayerIter 50;
297 }
298
299
300
301 // Generic mesh quality settings. At any undoable phase these determine
302 // where to undo.
303 meshQualityControls
304 {
305     #include "meshQualityDict"
306
307
308     // Advanced
309
310     //- Number of error distribution iterations
311     nSmoothScale 4;

```

```

312     //- Amount to scale back displacement at error points
313     errorReduction 0.75;
314 }
315
316
317 // Advanced
318
319 // Write flags
320 writeFlags
321 (
322     scalarLevels
323     layerSets
324     layerFields      // write volScalarField for layer coverage
325 );
326
327
328 // Merge tolerance. Is fraction of overall bounding box of initial mesh.
329 // Note: the write tolerance needs to be higher than this.
330 mergeTolerance 1e-6;
331
332
333 // ***** //

```

7.7.3 ControlDict

```

1  /*-----*- C++ -*-----*/
2  / ===== /
3  / \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ / Operation / Version: v2412 /
5  / \ / And / Website: www.openfoam.com /
6  / \ / Manipulation / /
7  /*-----*- C++ -*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class          dictionary;
13     object          controlDict;
14 }
15 // * * * * *
16
17 //- To be more comparable to v2406 bug-fix branch disable the
18 //- near-wall distance fix (see v2412 release notes about 'Wall distance')
19 //OptimisationSwitches
20 //{
21 //    useCombinedWallPatch    0;
22 //}
23
24 application      simpleFoam;
25
26 startFrom         startTime;
27
28 startTime         0;
29
30 stopAt            endTime;
31
32 endTime           500;
33
34 deltaT            1;
35
36 writeControl       timeStep;
37
38 writeInterval      100;
39
40 purgeWrite         0;
41
42 writeFormat        binary;
43
44 writePrecision     6;
45
46 writeCompression  off;
47
48 timeFormat         general;
49
50 timePrecision      6;
51
52 runTimeModifiable true;
53
54 functions
55 {
56     #include "forceCoeffs"
57
58     #include "solverInfo"
59     #include "graphFunctionObject"
60     #include "foamReport"
61 }

```

```
62 |  
63 |  
64 | // ***** //  
   |
```

7.7.4 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: v2412 /
5  / \ \ / A n d / Website: www.openfoam.com /
6  / \ \ / M a n i p u l a t i o n / /
7  /*-----*- C++ -*-----*/
8
9  forceCoeffs1
10 {
11     type                forceCoeffs;
12
13     libs                (forces);
14
15     writeControl         timeStep;
16     timeInterval        1;
17
18     log                 yes;
19
20     patches              (motorBikeGroup);
21     rho                 rhoInf; // Indicates incompressible
22     rhoInf               1; // Required when rho = rhoInf
23     liftDir              (0 1 0);
24     dragDir              (1 0 0);
25     CofR                 (0.72 0 0); // Axle midpoint on ground
26     pitchAxis            (0 1 0);
27     magUInf              22.2222222222;
28     lRef                 1.42; // Wheelbase length
29     Aref                 #calc "0.04*1.09585"; // The same for all shapes,
        projected area
30 }
31
32 /*
33 binField1
34 {
35     type                binField;
36     libs                (fieldFunctionObjects);
37     binModel              singleDirectionUniformBin;
38     fields                (forceCoeff);
39     patches              (motorBikeGroup);
40     decomposePatchValues true;
41     CofR                  ${../forceCoeffs1/CofR};
42
43     binData
44     {
45         nBin              20; // output data into 20 bins
46         direction         (1 0 0); // bin direction
47         cumulative         yes;
48     }
49     writeControl          timeStep;
50 }
51 */
52
53
54 // *****

```

7.7.5 fvSchemes

```

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: v2412 /
5  / \ \ / A n d / Website: www.openfoam.com /
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
32     div(phi,U)      bounded Gauss linearUpwindV grad(U);
33
34     turbulence       bounded Gauss upwind;
35     div(phi,k)       $turbulence;
36     div(phi,omega)   $turbulence;
37
38     div((nuEff*dev2(T(grad(U)))) Gauss linear;
39 }
40
41 laplacianSchemes
42 {
43     default        Gauss linear corrected;
44 }
45
46 interpolationSchemes
47 {
48     default        linear;
49 }
50
51 snGradSchemes
52 {
53     default        corrected;
54 }
55
56 wallDist
57 {
58     method          meshWave;
59 }
60
61

```


7.7.6 fvSolutions

```

1  /*-----*- C++ -*-----*/
2  / ===== /
3  / \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ / Operation / Version: v2412 /
5  / \ / And / Website: www.openfoam.com /
6  / \ / Manipulation / /
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         smoother     GaussSeidel;
23         tolerance    1e-7;
24         relTol       0.01;
25     }
26
27     Phi
28     {
29         $p;
30     }
31
32     U
33     {
34         solver      smoothSolver;
35         smoother     GaussSeidel;
36         tolerance    1e-8;
37         relTol       0.1;
38         nSweeps      1;
39     }
40
41     k
42     {
43         solver      smoothSolver;
44         smoother     GaussSeidel;
45         tolerance    1e-8;
46         relTol       0.1;
47         nSweeps      1;
48     }
49
50     omega
51     {
52         solver      smoothSolver;
53         smoother     GaussSeidel;
54         tolerance    1e-8;
55         relTol       0.1;
56         nSweeps      1;
57     }
58 }
59
60 SIMPLE
61 {

```

```

62     nNonOrthogonalCorrectors 2;
63     consistent yes;
64
65     pRefCell      0;           //Bottom left
66     pRefValue     101e3;       // air density approx 1
67 }
68
69 potentialFlow
70 {
71     nNonOrthogonalCorrectors 10;
72
73     PhiRefCell      0;
74     PhiRefValue     0; //Velocity potential to solve the euler eqs for initial
        starting field for velocity
75 }
76
77 relaxationFactors
78 {
79     equations
80     {
81         U           0.9;
82         k           0.7;
83         omega       0.7;
84     }
85 }
86
87 cache
88 {
89     grad(U);
90 }
91
92
93 // *****

```

7.7.7 Verification control script

```
1  #!/usr/bin/env python3
2  """Prepare and run the 313 PIMPLE sensitivity cases."""
3
4  import argparse
5  import json
6  import math
7  import re
8  import shlex
9  import shutil
10 import subprocess
11 import sys
12 from datetime import datetime, timezone
13 from pathlib import Path
14
15
16 ROOT = Path(__file__).resolve().parents[1]
17 SOURCE = ROOT / "saved_runs/313_pimple_run"
18 OUTPUT = ROOT / "runs/313_study"
19 OPENFOAM_BASHRC = Path("/usr/lib/openfoam/openfoam2412/etc/bashrc")
20 LENGTH = 1.09585
21 END_TIME = 1.0
22 RANKS = 6
23 WARM_START = 0.3
24 WARM_DELTA_T = 6.79727e-05
25
26
27 def now():
28     return datetime.now(timezone.utc).isoformat(timespec="seconds")
29
30
31 def cases():
32     result = [{"name": "maxCo_0p25", "kind": "maxCo", "value": 0.25,
33               "restart_time": WARM_START}]
34     result.append({"name": "maxCo_1", "kind": "maxCo", "value": 1.0,
35                  "restart_time": WARM_START})
36     for factor in (0.1, 10.0):
37         result.append({"name": f"tolerance_x{factor:g}", "kind": "tolerance", "
38                       factor": factor,
39                       "restart_time": WARM_START})
40     for intensity in (0.01, 0.10):
41         result.append({
42             "name": f"turbulenceIntensity_{intensity*100:g}pct",
43             "kind": "turbulence_intensity",
44             "intensity": intensity,
45             "viscosity_ratio": 10,
46             "initial_time": 0,
47         })
48     result.append({"name": "outerCorrectors_4", "kind": "PIMPLE", "value": 4,
49                  "restart_time": WARM_START})
50     for parameter, baseline in (("upstream_dist", 5), ("downstream_dist", 11),
51                                ("roof_height", 6)):
52         for direction, offset in (("minus", -1), ("plus", 1)):
53             result.append({
54                 "name": f"{parameter}_{direction}_1L",
55                 "kind": "domain",
56                 "parameter": parameter,
57                 "value": (baseline + offset) * LENGTH,
58                 "baseline_value": baseline * LENGTH,
59             })
60     return result
```

```

60
61 def load_manifest(path):
62     if path.exists():
63         return json.loads(path.read_text())
64     return {
65         "source": str(SOURCE),
66         "baseline_reference": str(SOURCE),
67         "baseline_settings": {"maxCo": 0.5, "endTime": END_TIME},
68         "tolerance_definition": (
69             "Multiply only tolerance entries for p, U, k, and omega in
70             "system/fvSolution.steady; relTol remains unchanged."
71         ),
72         "length_m": LENGTH,
73         "mpi_ranks": RANKS,
74         "warm_start": {"time_s": WARM_START, "deltaT_s": WARM_DELTA_T},
75         "cases": {},
76     }
77
78
79 def save_manifest(path, manifest):
80     path.write_text(json.dumps(manifest, indent=2) + "\n")
81
82
83 def foam(case_dir, *args):
84     command = "source" + shlex.quote(str(OPENFOAM_BASHRC)) + "&&" + shlex.
85         join(args)
86     return subprocess.run(["bash", "-c", command], cwd=case_dir,
87         stdout=subprocess.DEVNULL, stderr=subprocess.PIPE,
88         text=True, check=False)
89
90 def copy_source(case_dir, mesh_case, warm_start=True):
91     shutil.copytree(SOURCE / "system", case_dir / "system")
92     shutil.copytree(SOURCE / "0.orig", case_dir / "0.orig")
93     shutil.copy2(SOURCE / "Allrun.pimple", case_dir / "Allrun.pimple")
94     if mesh_case:
95         (case_dir / "constant").mkdir()
96         for name in ("transportProperties", "turbulenceProperties"):
97             shutil.copy2(SOURCE / "constant" / name, case_dir / "constant" /
98                 name)
99         shutil.copytree(SOURCE / "constant/triSurface", case_dir / "constant/
100             triSurface")
101     else:
102         shutil.copytree(SOURCE / "constant", case_dir / "constant")
103         for rank in range(RANKS):
104             proc_name = f"processor{rank}"
105             proc_dir = case_dir / proc_name
106             proc_dir.mkdir()
107             shutil.copytree(SOURCE / proc_name / "constant", proc_dir / "
108                 constant")
109             shutil.copytree(SOURCE / proc_name / "0", proc_dir / "0")
110             if warm_start:
111                 shutil.copytree(SOURCE / proc_name / "0.3", proc_dir / "0.3")
112
113 def set_entry(case_dir, dictionary, entry, value):
114     result = foam(case_dir, "foamDictionary", "-entry", entry, "-set", str(
115         value), dictionary)
116     if result.returncode:
117         detail = result.stderr.strip() if result.stderr else ""
118         raise RuntimeError(f"foamDictionary failed for {dictionary}:{entry}: {
119             detail}")

```

```

117
118 def set_block_mesh_dimension(case_dir, entry, value):
119     path = case_dir / "system/blockMeshDict"
120     text = path.read_text()
121     pattern = rf"(?m)^(?!\s)*{re.escape(entry)}[\s\t+][^;\r\n]*(;[^\r\n]*)$"
122     updated, count = re.subn(
123         pattern,
124         lambda match: match.group(1) + format(value, ".15g") + match.group(2),
125         text,
126     )
127     if count != 1:
128         raise RuntimeError(f"expected one {entry} definition in {path}, found {count}")
129     path.write_text(updated)
130
131
132 def configure_warm_start(case_dir):
133     for rank in range(RANKS):
134         proc_dir = case_dir / f"processor{rank}"
135         checkpoint = proc_dir / "0.3"
136         if not checkpoint.exists():
137             shutil.copytree(SOURCE / f"processor{rank}/0.3", checkpoint)
138         set_entry(case_dir, "system/controlDict", "startFrom", "startTime")
139         set_entry(case_dir, "system/controlDict", "startTime", WARM_START)
140         set_entry(case_dir, "system/controlDict", "deltaT", WARM_DELTA_T)
141
142
143 def set_initial_intensity(case_dir, intensity):
144     paths = [case_dir / "0.orig/include/initialConditions"]
145     paths.extend(case_dir / f"processor{rank}/0/include/initialConditions" for
146                  rank in range(RANKS))
147     for path in paths:
148         text = path.read_text()
149         updated, count = re.subn(
150             r"(?m)^(?!\s)*I[\s\t+][^;\r\n]+(;[^\r\n]*)$",
151             lambda match: match.group(1) + f"{intensity:g}" + match.group(2),
152             text,
153         )
154         if count != 1:
155             raise RuntimeError(f"expected one inlet intensity entry in {path}, found {count}")
156         path.write_text(updated)
157
158
159 def prepare_case(spec, manifest):
160     case_dir = OUTPUT / spec["name"]
161     state = manifest["cases"].get(spec["name"])
162     if case_dir.exists():
163         if state is None:
164             manifest["cases"][spec["name"]] = {
165                 **spec, "status": "existing_unmanaged", "updated": now(),
166                 "message": "Existing directory preserved; runner will not modify it.",
167             }
168         elif spec["kind"] in ("maxCo", "tolerance", "PIMPLE") and state.get("status") == "prepared" and "restart_time" not in state:
169             if (case_dir / "log.pimpleFoam").exists():
170                 raise RuntimeError(f"{spec['name']}: solver log exists; warm-start migration left untouched")
171             configure_warm_start(case_dir)
172             state.update(restart_time=WARM_START, restart_deltaT=WARM_DELTA_T, updated=now())
173     return case_dir

```

```

173
174 mesh_case = spec["kind"] == "domain"
175 case_dir.mkdir(parents=True)
176 warm_start = spec["kind"] != "turbulence_intensity"
177 copy_source(case_dir, mesh_case, warm_start=warm_start)
178 if spec["kind"] == "maxCo":
179     set_entry(case_dir, "system/controlDict", "maxCo", spec["value"])
180 elif spec["kind"] == "tolerance":
181     for field, baseline in (("p", 1e-7), ("U", 1e-8), ("k", 1e-8), ("omega"
182         , 1e-8)):
183         set_entry(case_dir, "system/fvSolution.steady", f"solvers.{field}.
184             tolerance",
185                 baseline * spec["factor"])
186 elif spec["kind"] == "PIMPLE":
187     set_entry(case_dir, "system/fvSolution", "PIMPLE.nOuterCorrectors",
188         spec["value"])
189 elif spec["kind"] == "turbulence_intensity":
190     set_initial_intensity(case_dir, spec["intensity"])
191 else:
192     set_block_mesh_dimension(case_dir, spec["parameter"], spec["value"])
193 if not mesh_case and warm_start:
194     configure_warm_start(case_dir)
195 manifest["cases"][spec["name"]] = {**spec, "status": "prepared", "updated":
196     now()}
197 return case_dir
198
199
200 def solver_complete(case_dir):
201     log = case_dir / "log.pimpleFoam"
202     if not log.is_file():
203         return False, "missing_log.pimpleFoam"
204     contents = log.read_text(errors="replace")
205     if not re.search(r"^End\s*$", contents, re.MULTILINE):
206         return False, "log.pimpleFoam_has_no_End_marker"
207     times = [float(value) for value in re.findall(r"^Time\s*=\s*([0-9.eE+-]+)",
208         contents, re.MULTILINE)]
209     if not times or not math.isclose(max(times), END_TIME, rel_tol=0, abs_tol=1
210         e-8):
211         return False, f"log.pimpleFoam_did_not_reach_{END_TIME:g}s"
212     return True, "solver_log_ended_at_1s"
213
214
215 def domain_bounds_match(spec, check_mesh_log):
216     bounding_box = re.findall(
217         r"Overall_domain_bounding_box\s*\(\s*([+-0-9.eE]+)\s+([+-0-9.eE]+)\s
218         +([+-0-9.eE]+)\s*\)"
219         r"\s*\(\s*([+-0-9.eE]+)\s+([+-0-9.eE]+)\s+([+-0-9.eE]+)\s*\)",
220         check_mesh_log,
221     )
222     if not bounding_box:
223         return False, "domain_checkMesh_has_no_overall_bounding_box"
224     values = tuple(float(value) for value in bounding_box[-1])
225     dimensions = {"upstream_dist": 5, "downstream_dist": 11, "roof_height": 6}
226     expected = {
227         "upstream_dist": dimensions["upstream_dist"] * LENGTH,
228         "downstream_dist": dimensions["downstream_dist"] * LENGTH,
229         "roof_height": dimensions["roof_height"] * LENGTH,
230     }
231     expected[spec["parameter"]] = spec["value"]
232     actual = {"upstream_dist": -values[0], "downstream_dist": values[3], "
233         roof_height": values[4]}
234     for parameter, value in expected.items():
235         if not math.isclose(actual[parameter], value, rel_tol=0, abs_tol=1e-4):

```

```

228         return False, (
229             f"domain_checkMesh_{parameter}_extent_{actual[parameter]:g}_m_"
230             f"does_not_match_requested_{value:g}_m"
231         )
232     return True, "domain_checkMesh_bounds_match_requested_extents"
233
234
235 def case_complete(spec, case_dir):
236     complete, message = solver_complete(case_dir)
237     if not complete:
238         return complete, message
239     if spec["kind"] == "domain":
240         log = case_dir / "log.checkMesh"
241         if not log.is_file():
242             return False, "missing_domain_log.checkMesh"
243         contents = log.read_text(errors="replace")
244         if "Mesh_OK." not in contents:
245             return False, "domain_checkMesh_did_not_report_Mesh_OK."
246         complete, message = domain_bounds_match(spec, contents)
247         if not complete:
248             return complete, message
249     return True, message
250
251
252 def run_case(spec, case_dir):
253     mesh_case = spec["kind"] == "domain"
254     if mesh_case:
255         log = case_dir / "execution.log"
256         with log.open("w") as output:
257             command = "source_" + shlex.quote(str(OPENFOAM_BASHRC)) + "_&&_bash"
258             result = subprocess.run(["bash", "-c", command], cwd=case_dir,
259                                     stdout=output, stderr=subprocess.STDOUT,
260                                     check=False)
261             if result.returncode:
262                 return False, f"Allrun.pimple_exited_{result.returncode}"
263             for path in sorted(case_dir.glob("log.*")):
264                 if path.is_file() and not re.search(r"End\s*$", path.read_text(
265                     errors="replace"), re.MULTILINE):
266                     return False, f"{path.name}_has_no_End_marker"
267     else:
268         log = case_dir / "log.pimpleFoam"
269         if log.exists():
270             previous = case_dir / f"log.pimpleFoam.previous-{datetime.now().
271                                     strftime('%Y%m%d%H%M%S')}"
272             shutil.copy2(log, previous)
273         command = (
274             "source_" + shlex.quote(str(OPENFOAM_BASHRC)) +
275             f"_&&_mpirun_-np_{RANKS}_pimpleFoam_-parallel"
276         )
277         with log.open("w") as output:
278             result = subprocess.run(["bash", "-c", command], cwd=case_dir,
279                                     stdout=output, stderr=subprocess.STDOUT,
280                                     check=False)
281             if result.returncode:
282                 return False, f"pimpleFoam_exited_{result.returncode}"
283     return case_complete(spec, case_dir)
284
285
286 def update_report(manifest):
287     completed = [name for name, state in manifest["cases"].items() if state.get
288                   ("status") == "complete"]
289     baseline_coeff = SOURCE / "postProcessing/forceCoeffs1/0/coefficient.dat"

```

```

285     case_paths = [str(baseline_coeff)] + [str(OUTPUT / name) for name in sorted
286         (completed)]
287     script = str(ROOT / "study/analyze.py")
288     fixed = subprocess.run([sys.executable, script, *case_paths,
289         "--csv", str(OUTPUT / "summary_fixed.csv"),
290         "--markdown", str(OUTPUT / "summary_fixed.md")],
291         cwd=ROOT, capture_output=True, text=True, check=
292             False)
293     manifest["fixed_window_analysis_status"] = "complete" if fixed.returncode
294         == 0 else "failed"
295     manifest["fixed_window_analysis_error"] = fixed.stderr.strip() if fixed.
296         returncode else ""
297
298     cycle_cases = []
299     missing_cycles = []
300     for name, path in zip(["saved_baseline"] + sorted(completed), case_paths):
301         cycle = subprocess.run([sys.executable, script, path, "--cycles", "2"],
302             cwd=ROOT, capture_output=True, text=True, check=
303                 False)
304         if cycle.returncode == 0:
305             cycle_cases.append(path)
306             if name != "saved_baseline":
307                 manifest["cases"][name].update(cycle_analysis_status="complete"
308                     , cycle_analysis_error="")
309         else:
310             reason = cycle.stderr.strip()
311             if name != "saved_baseline":
312                 manifest["cases"][name].update(cycle_analysis_status="
313                     incomplete", cycle_analysis_error=reason)
314             missing_cycles.append((name, reason))
315
316     cycle_result = (subprocess.run([sys.executable, script, *cycle_cases, "--
317         cycles", "2",
318         "--csv", str(OUTPUT / "summary.csv"),
319         "--markdown", str(OUTPUT / "summary.md")],
320         cwd=ROOT, capture_output=True, text=True,
321         check=False)
322         if cycle_cases else None)
323     if cycle_result is not None and cycle_result.returncode == 0:
324         with (OUTPUT / "summary.md").open("a") as markdown:
325             if missing_cycles:
326                 markdown.write("\nCycle average omitted for cases without two
327                 complete cycles after 0.5s:\n\n")
328                 markdown.writelines(f"- {name}: {reason}\n" for name, reason in
329                     missing_cycles)
330     manifest["cycle_analysis_status"] = (
331         "complete" if cycle_result is not None and cycle_result.returncode == 0
332         else "failed"
333     )
334     manifest["cycle_analysis_error"] = (
335         cycle_result.stderr.strip() if cycle_result is not None and
336         cycle_result.returncode else ""
337     )
338
339 def main():
340     parser = argparse.ArgumentParser(description=__doc__)
341     parser.add_argument("--prepare-only", action="store_true", help="prepare
342         cases without running simulations")
343     parser.add_argument("--cases", nargs="+", help="run only these case names")
344     parser.add_argument("--manifest", type=Path, default=OUTPUT / "manifest.
345         json",

```

```

332         help="manifest_for_this_batch_(default:_runs/313_study/
333             manifest.json)")
334     args = parser.parse_args()
335     specs = cases()
336     by_name = {spec["name"]: spec for spec in specs}
337     if args.cases:
338         unknown = sorted(set(args.cases) - set(by_name))
339         if unknown:
340             parser.error("unknown_case_name(s):_ " + ",_".join(unknown))
341             specs = [by_name[name] for name in args.cases]
342
343     if not SOURCE.is_dir():
344         parser.error(f"savé_baseline_not_found:_{SOURCE}")
345     if not OPENFOAM_BASHRC.is_file():
346         parser.error(f"OpenFOAM_setup_script_not_found:_{OPENFOAM_BASHRC}")
347     OUTPUT.mkdir(parents=True, exist_ok=True)
348     manifest_path = args.manifest.resolve()
349     manifest = load_manifest(manifest_path)
350     manifest.setdefault("warm_start", {"time_s": WARM_START, "deltaT_s":
351         WARM_DELTA_T})
352
353     for spec in specs:
354         try:
355             case_dir = prepare_case(spec, manifest)
356         except (OSError, RuntimeError) as error:
357             manifest["cases"][spec["name"]] = {**spec, "status": "
358                 prepare_failed",
359                 "updated": now(), "message":
360                     str(error)}
361             save_manifest(manifest_path, manifest)
362             print(f"{spec['name']}:_preparation_failed:_{error}", file=sys.
363                 stderr)
364             continue
365             state = manifest["cases"][spec["name"]]
366             if state.get("status") == "existing_unmanaged":
367                 print(f"{spec['name']}:_existing_directory_left_untouched")
368                 continue
369             if args.prepare_only:
370                 if state.get("status") not in ("failed", "prepare_failed", "
371                     complete"):
372                     state.update(status="prepared", updated=now())
373                     print(f"{spec['name']}:_{state['status']}_preparation_only")
374             elif state.get("status") == "complete":
375                 complete, message = case_complete(spec, case_dir)
376                 if complete:
377                     print(f"{spec['name']}:_already_complete,_skipped")
378                     continue
379                     state.update(status="failed", updated=now(), message=message)
380             else:
381                 state.update(status="running", updated=now())
382                 save_manifest(manifest_path, manifest)
383                 print(f"{spec['name']}:_running", flush=True)
384                 try:
385                     ok, message = run_case(spec, case_dir)
386                 except OSError as error:
387                     ok, message = False, str(error)
388                 state.update(status="complete" if ok else "failed", updated=now(),
389                     message=message)
390                 print(f"{spec['name']}:_{state['status']}_({message})")
391                 if ok:
392                     update_report(manifest)
393             save_manifest(manifest_path, manifest)

```

```

388     if args.prepare_only:
389         update_report(manifest)
390         save_manifest(manifest_path, manifest)
391     return 0 if all(manifest["cases"].get(spec["name"], {}).get("status") in ("
392         prepared", "complete", "existing_unmanaged")
393         for spec in specs) else 1
394
395 if __name__ == "__main__":
396     sys.exit(main())

```

Note: With further subscripts