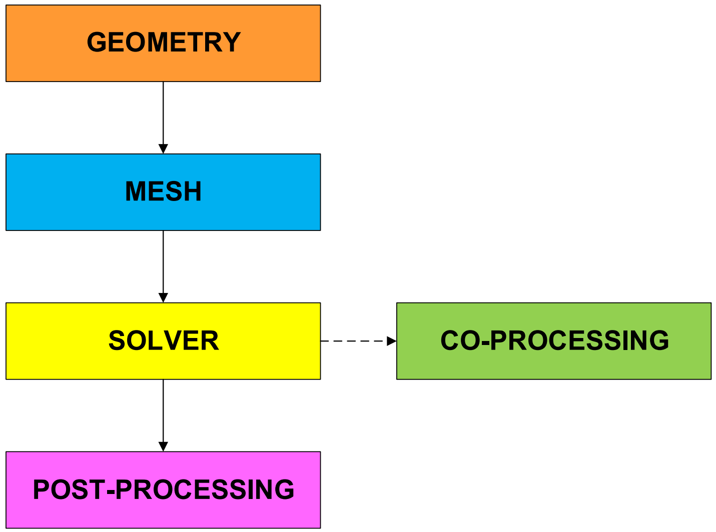
Supplement

Qualitative postprocessing – Coprocessing

Coprocessing

- CFD simulations have the potential to overwhelm any computer with the output obtained from simulations.
- The traditional approach is to run a simulation and save the solution at given time-steps or intervals for post processing at a later time.
- An alternative way to do post processing, is to extract results while the simulation is running (on-the-fly), this is coprocessing.
- For unsteady and big simulations, coprocessing is an alternative if we do not want to overflow the system with tons of data.
- In principle, coprocessing is similar to doing sampling using **functionObjects**, but when we do coprocessing we output pretty pictures (*e.g.*, streamlines, iso-surfaces, cut-planes).
- An added benefit of coprocessing is that results can be immediately reviewed, and problems can be immediately addressed.
- Coprocessing requires that you identify what you want to see before running the simulation. You need to plan everything in advanced.
- In OpenFOAM®, you can output on-the-fly streamlines, cutting planes, iso-surfaces, near surface fields, and forces data bins.



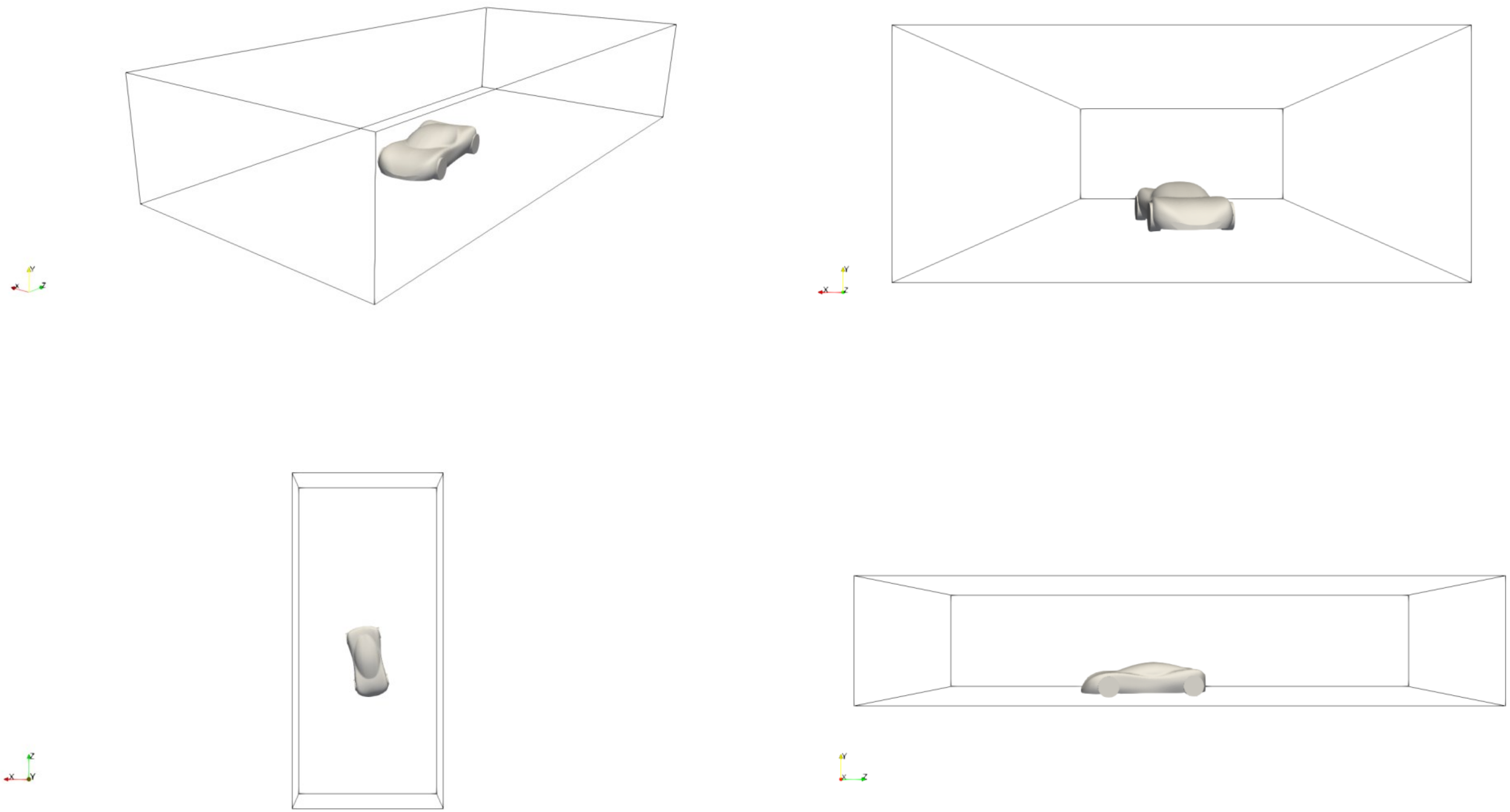
Coprocessing

- Let us do some coprocessing. Go to the directory:

```
$PTOFC/advanced_postprocessing/sport_car/
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_solver`.
- We highly recommend to open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM® commands.
- If you are already comfortable with OpenFOAM®, use the automatic scripts to run the cases.

Coprocessing



Geometry and computational domain

Coprocessing

What are we going to do?

- We will use this case to do coprocessing using **functionObjects**.
- We do not need to run the simulation for a long time, we just need to run a few iterations in order to do coprocessing.
- We will run the simulation for 100 iterations and then we will visualize the solution.
- In this case we will use the solver `potentialFoam` to initialize the solution.
- Then we will use the solver `simpleFoam` with turbulence modeling enabled.
- You can run in serial or parallel.
- To run the case just execute the script `run_solver.sh`
- All the coprocessing **functionObjects** are defined in the dictionary *`controlDict`*.

Coprocessing

The `controlDict` dictionary

```
58    functions
59    {
286       isoSurfaces1

332       isoSurfaces2

379       cuttingPlanes1

444       nearWallField1

471       patch_surface1

504       patch_surface2

537       streamlines1

577       streamlines2

614       wallBoundedStreamLines1

717   };
```

- Let us take a look at the definition of the **functionObjects** in the dictionary `controlDict`.
- In this case, we have defined many **functionObjects**.
- We will only comment on the **functionObjects** related to coprocessing.
- In lines 286 and 332 we defined the **functionObjects** to compute iso-surfaces.
- In line 379 we defined the **functionObjects** to compute cut-planes.
- In line 444 we defined the **functionObjects** to compute near wall fields.
- In lines 471 and 504 we defined the **functionObjects** to compute fields on patches.
- In lines 537, 577, and 614 we defined the **functionObjects** to compute streamlines released from different locations.
- It is important to stress that in coprocessing we are only saving the requested information, we do not save the whole mesh with all fields.

Coprocessing

The *controlDict* dictionary – Iso-surfaces **functionObject**

```
286   isoSurfaces1
287   {
288       type surfaces;
289       functionObjectsLibs ("libsampling.so")
290
291       enabled true;

295       writeControl timestep;
296       writeInterval 10;

298       surfaceFormat vtk;
299       fields ( p U k omega );

301       interpolationScheme cellPoint;

304       surfaces
305       (
306
307           p_constantIso
308           {
309               type isoSurface;
310               isoField p;
311               isoValue 30;
312               Interpolate false;
313           }

              ...
              ...
              ...

323       );

325   }
```

- Let us take a look at the iso-surfaces definition.
- In lines 288-289 we select the library and type of **functionObject**.
- In line 291 we can turn-on and turn-off the **functionObject**. This can be done on-the-fly.
- In lines 295-296 we select the saving frequency. The saving frequency can be different from the saving frequency of the solution.
- In line 298 we select the output format (many formats are available).
- In line 299 we select the fields to save with the iso-surface. No need to mention that the fields must exist.
- In lines 301 we select the interpolation method.
- In lines 304-323 we define the iso-surfaces. You can add as many as you like.
- Remember, to define the iso-surface we need to know the iso value a priori or at least have a rough reference of the value of the iso-surface.

Coprocessing

The *controlDict* dictionary – Iso-surfaces **functionObject**

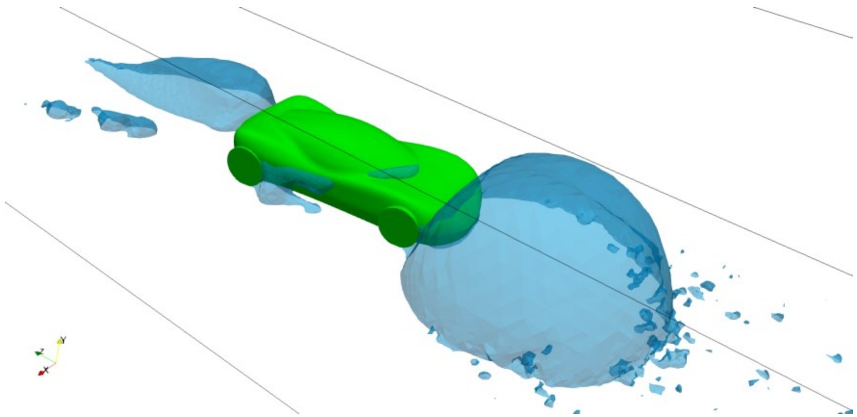
```
286    isoSurfaces1
287    {
288        type surfaces;
289        functionObjectsLibs ("libsampling.so")
290
291        enabled true;

295        writeControl timestep;
296        writeInterval 10;

298        surfaceFormat vtk;
299        Fields ( p U k omega );

301        interpolationScheme cellPoint;

304        surfaces
305        (
306
307            p_constantIso
308            {
309                type isoSurface;
310                isoField p;
311                isoValue 30;
312                Interpolate false;
313            }

               ...
               ...
               ...

323        );

325    }
```

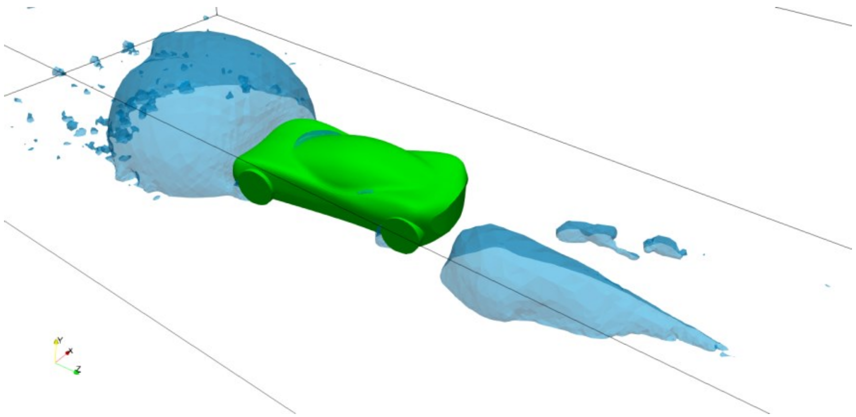
- In lines 307-313 we define the **p_constantIso** object.
  - In line 307 we give a unique name to this object.
  - In line 309 we define the type (iso-surface).
  - In line 310 we select the field to compute the iso-surface.
  - In line 311 we select the iso value.
  - In this case we are saving an iso-surface of the pressure field pressure with a value of 30.
  - The iso-surfaces contain the information of the fields defined in line 299.
- The output of this **functionObject** is saved in the directory **`postProcessing/isoSurface1`**
- The output is saved in this directory because in line 286 we defined a unique name for the **functionObject**.
- In this directory, you will find many time directories with the sampled data.
- Inside each directory you will find a series of files with the VTK extension, you can open these files in paraFoam/paraview.
- The rest of the iso-surfaces **functionObjects** are defined in a similar way.
- As usual, to know all the options available, you can use the banana trick.

Coprocessing

Iso-surfaces of pressure field

- Iso-surfaces sampled using **functionObjects**.
- By using coprocessing, we only saved this specific iso-surface information.
- There is not need to save the whole solution.
- This can significantly reduce the amount of data stored and help us in doing faster post-processing.

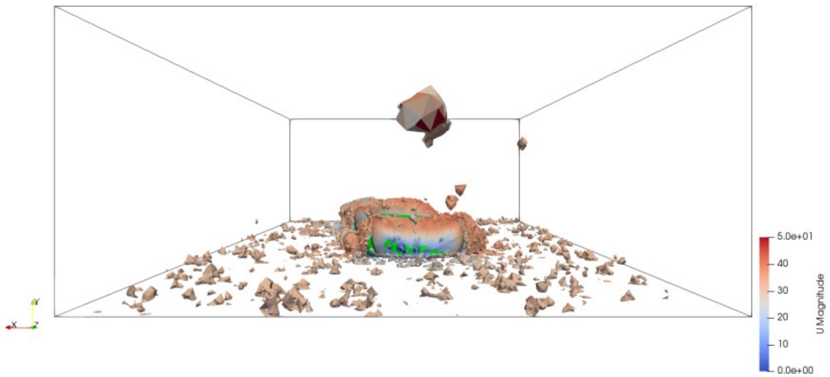


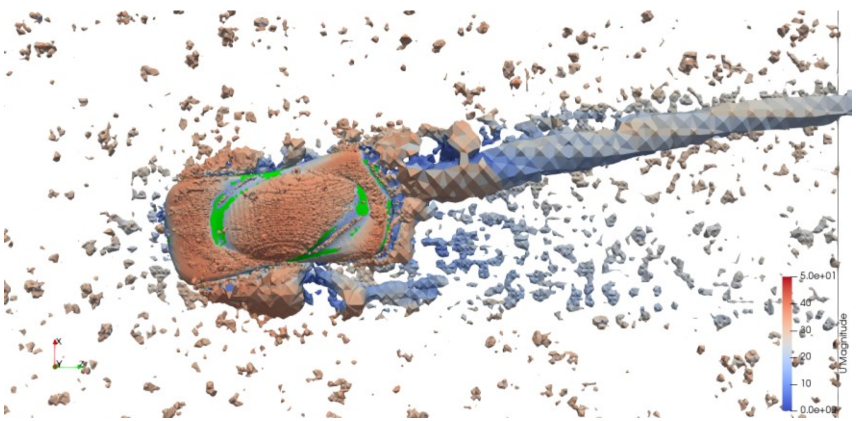


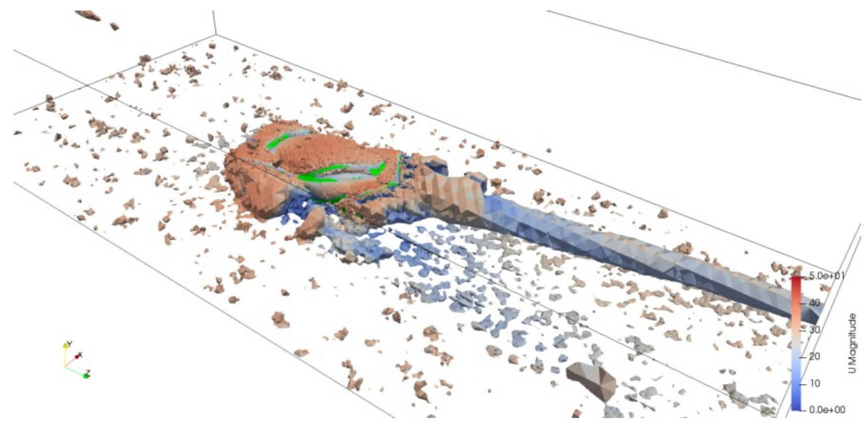
Coprocessing

Iso-surfaces of Q criterion

- Iso-surfaces of Q criterion colored using the velocity field.







Coprocessing

The *controlDict* dictionary – Cut-planes **functionObject**

```
379    cuttingPlanes1
380    {
381        type surfaces;
382        functionObjectsLibs (“libsampling.so”)

384        enabled true;

388        writeControl timestep;
389        writeInterval 10;

392        surfaceFormat vtk;
393        fields ( p U k omega );

395        interpolationScheme cellPoint;

397        surfaces
398        (
399            xNormal
400            {
401                type cuttingPlane;
402                planeType pointAndNormal;
403                pointAndNormalDict
404                {
405                    basePoint (0 0 0);
406                    normalVector (1 0 0);
407                }
408                Interpolate true;
409            }

               ...
               ...
               ...

435        );

437    }
```

- Let us take a look at the cut planes definition.
- The options in lines 381-395 are similar to the iso-surfaces **functionObject**.
- Remember, the saving frequency can be different from the saving frequency of the solution and other **functionObjects**.
- In lines 397-435 we define the cut-planes. You can add as many as you like.
- In lines 399-409 we define the **xNormal** object.
  - In line 399 we give a unique name to this object.
  - In lines 402-408 we define the cut-plane.
- To define cut-planes, there are many options available.
- To know all the options, you can use the banana trick or read the source code.
- Remember, to define the cut-planes we need to know their location a priori or at least have a rough reference of the domain dimensions.

Coprocessing

The `controlDict` dictionary – Cut-planes **functionObject**

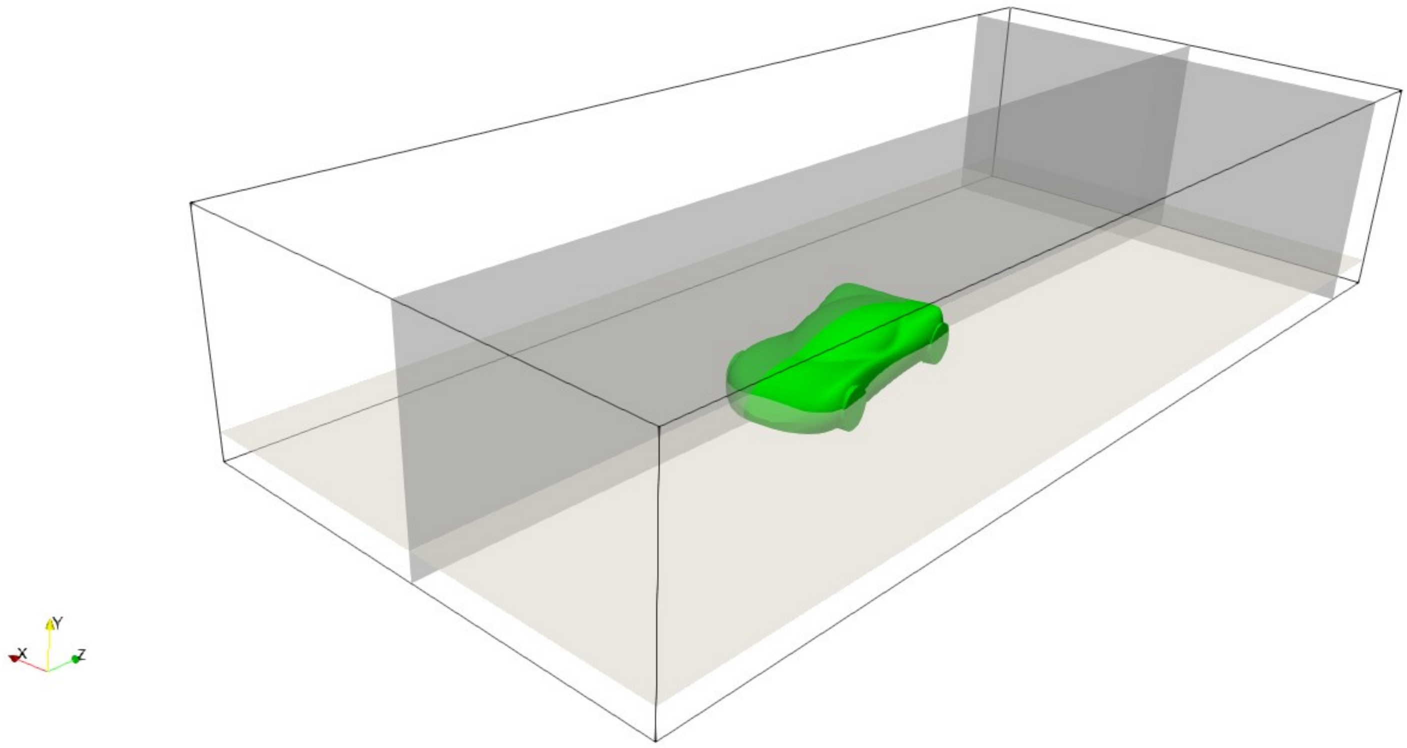
```
379   cuttingPlanes1
380   {
381       type surfaces;
382       functionObjectsLibs ("libsampling.so");

384       enabled true;

388       writeControl timestep;
389       writeInterval 10;

392       surfaceFormat vtk;
393       fields ( p U k omega );

395       interpolationScheme cellPoint;

397       surfaces
398       (
399           xNormal
400           {
401               type cuttingPlane;
402               planeType pointAndNormal;
403               pointAndNormalDict
404               {
405                   basePoint (0 0 0);
406                   normalVector (1 0 0);
407               }
408               Interpolate true;
409           }

              ...
              ...
              ...

435       );

437   }
```

- The output of this **functionObject** is saved in the directory **`postProcessing/cuttingPlanes1`**
- The output is saved in this directory because in line 379 we defined a unique name for the **functionObject**.
- In this directory, you will find many time directories with the sampled data.
- Inside each directory you will find a series of files with the VTK extension, you can open these files in paraFoam/paraview.
- The rest of the cut-planes **functionObjects** are defined in a similar way.
- As usual, to know all the options available, you can use the banana trick.

Coprocessing

Cut-planes location

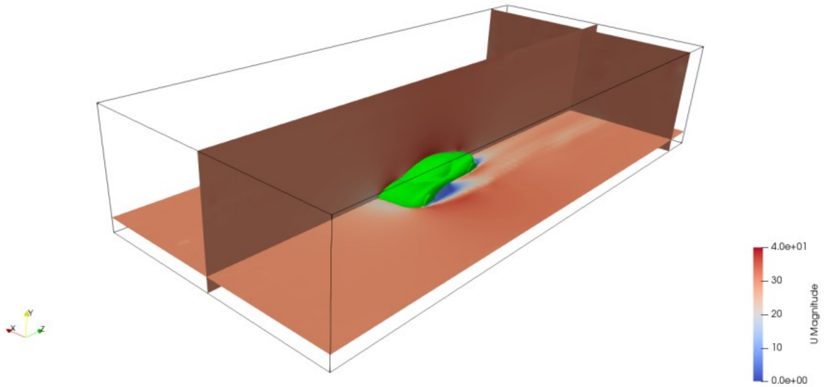
- By using coprocessing, we only saved this specific information.
- There is not need to save the whole solution.
- This can significantly reduce the amount of data stored and help us in doing faster post-processing.

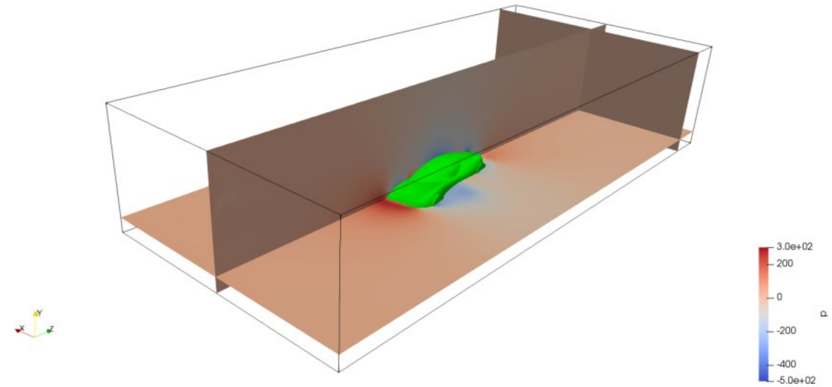


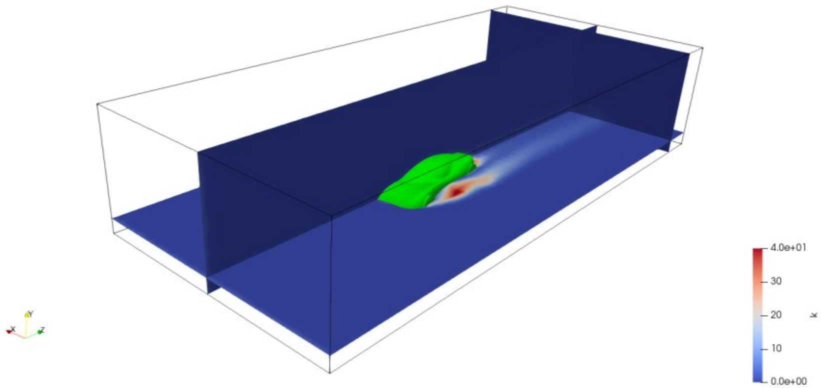
Coprocessing

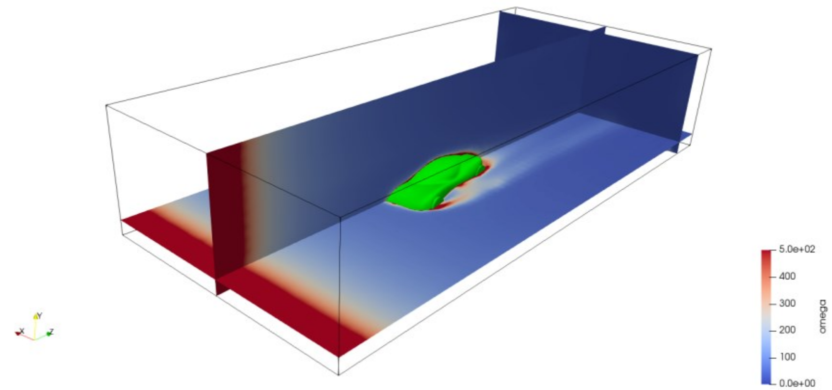
Cut-planes – Field variables contours

- Cut-planes colored using field variables (U, p, k, omega).









Coprocessing

The `controlDict` dictionary – Patch sampling **functionObject**

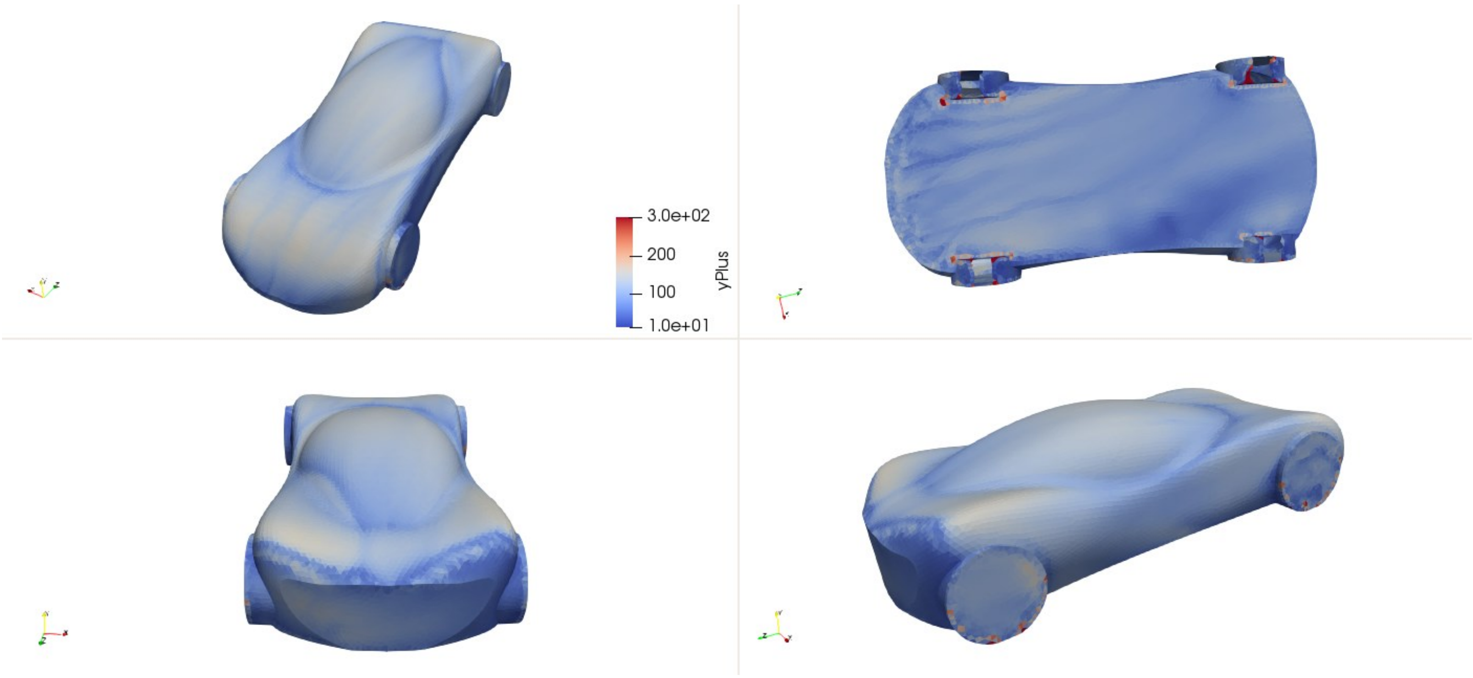
```
471    patch_surface1
472    {
473        type surfaces;
474        functionObjectsLibs ("libsampling.so")
475
476        enabled true;

479        writeControl timestep;
480        writeInterval 10;

482        surfaceFormat vtk;
483        fields ( p U k omega yPlus );

485        interpolationScheme cellPoint;

487        surfaces
488        (
489
490            patch_car
491            {
492                type patch;
493                Patches ("car");
494            }
495        );

497    }
```

- Let us see how to save the information at a given patch.
- The options in lines 473-485 are similar to those of the previous **functionObjects**.
- In lines 487-495 we define the sampling at a given patch.
- In line 493, we select the patch where we want to save the fields information.
- The fields used are defined in line 483.
- The patch (or patches) where you want to sample must exist.
- No need to say that the fields must exist as well.
- The output of this **functionObject** is saved in the directory **`postProcessing/patch_surface1`**
- The output is saved in this directory because in line 471 we defined a unique name for the **functionObject**.
- In this directory, you will find many time directories with the sampled data.
- Inside each directory you will find a series of files with the VTK extension, you can open these files in paraFoam/paraview.
- The rest of the **functionObjects** are defined in a similar way.

Coprocessing

Surface patches – y^+ contours

- Surface patches sampled using **functionObjects**.
- By using coprocessing, we only saved this specific iso-surface information.
- There is not need to save the whole solution.
- This can significantly reduce the amount of data stored and help us in doing faster post-processing.



Coprocessing

The *controlDict* dictionary – Streamlines **functionObject**

```
537  streamlines1
538  {
539      functionObjectsLibs ("libfieldFunctionObjects.so")
540      type streamLine;

542      enabled true;

545      writeControl timestep;
546      writeInterval 20;

548      setFormat vtk;

550      direction forward;

552      U U;

554      fields (U p);

556      lifetime 10000;

560      nSubCycle 5;

562      sedSampleSet
563      {
564          type lineUniform;
565          axis x;
566          start (-2 0.7 4);
567          end   ( 2 0.7 4);
568          nPoints 100;
569      }
570  }
```

- Let us take a look at the streamlines definition.
- In lines 539-540 we select the library and type of **functionObject**.
- In line 542 we can turn-on and turn-off the **functionObject**. This can be done on-the-fly.
- In lines 545-546 we select the saving frequency. The saving frequency can be different from the saving frequency of the solution or other **functionObjects**.
- In line 548 we select the output format (many formats are available).
- In line 550 we select the tracking direction of the streamlines (forward, backward, or both).
- In line 552 we select the velocity field used to compute the streamlines.
    - Most of the times you will use the field **U**, but have in mind that you can use **Umean** (computed using average values **functionObject**), **UNear** (computed using nearWallFields **functionObject**), and so on.
- In line 554 we select the fields to save with the streamlines. No need to mention that the fields must exist.

Coprocessing

The *controlDict* dictionary – Streamlines **functionObject**

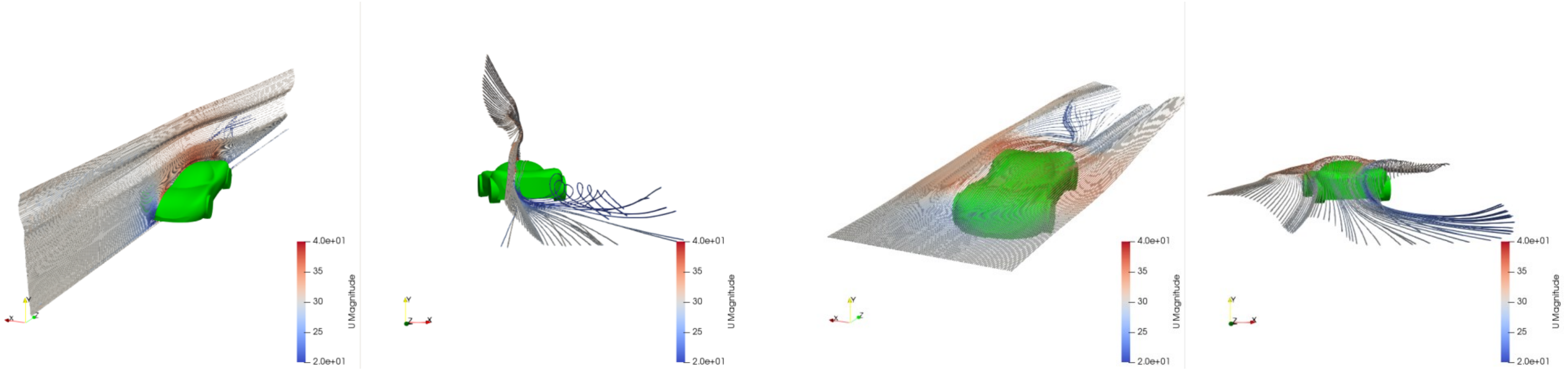
```
537  streamlines1
538  {
539      functionObjectsLibs ("libfieldFunctionObjects.so")
540      type streamLine;

542      enabled true;

545      writeControl timestep;
546      writeInterval 20;

548      setFormat vtk;

550      direction forward;

552      U U;

554      fields (U p);

556      lifetime 10000;

560      nSubCycle 5;

562      sedSampleSet
563      {
564          type lineUniform;
565          axis x;
566          start (-2 0.7 4);
567          end   ( 2 0.7 4);
568          nPoints 100;
569      }
570  }
```

- In lines 554-560 we select the options related to the streamlines tracking.
  - **lifetime** - Steps particles can travel before being removed.
  - **trackLength** - Size of single track segment.
  - **nSubCycle** - Number of steps per cell (estimate). Set to 1 to disable subcycling.
  - **trackLength** and **nSubCyce** are mutually exclusive.
- In lines 562-569 we define the seeding method. The streamlines will be released from this location.
- The output of this **functionObject** is saved in the directory **`postProcessing/sets/streamlines1`**
- The output is saved in this directory because,
  - Seeding method belong to sets.
  - In line 537 we defined a unique name for the **functionObject**,
- In this directory, you will find many time directories with the sampled data.
- Inside each directory you will find a series of files with the VTK extension, you can open these files in paraFoam/paraview.
- As usual, to know all the options available, you can use the banana trick.
- The rest of the **functionObjects** are defined in a similar way.

Coprocessing

Streamlines

- By using coprocessing, we only saved this specific information.
- There is not need to save the whole solution.
- This can significantly reduce the amount of data stored and help us in doing faster post-processing.



Streamlines

- Streamlines can also be released from a surface and constrained to a patch.