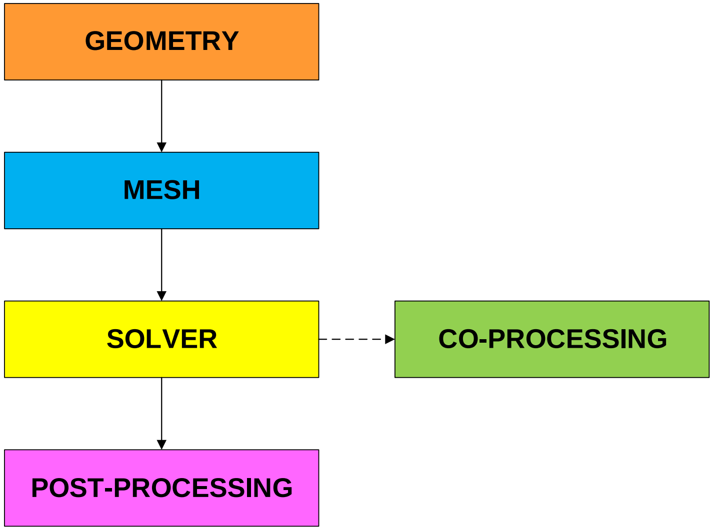
Supplement 4

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.



GEOMETRY

MESH

SOLVER

CO-PROCESSING

POST-PROCESSING

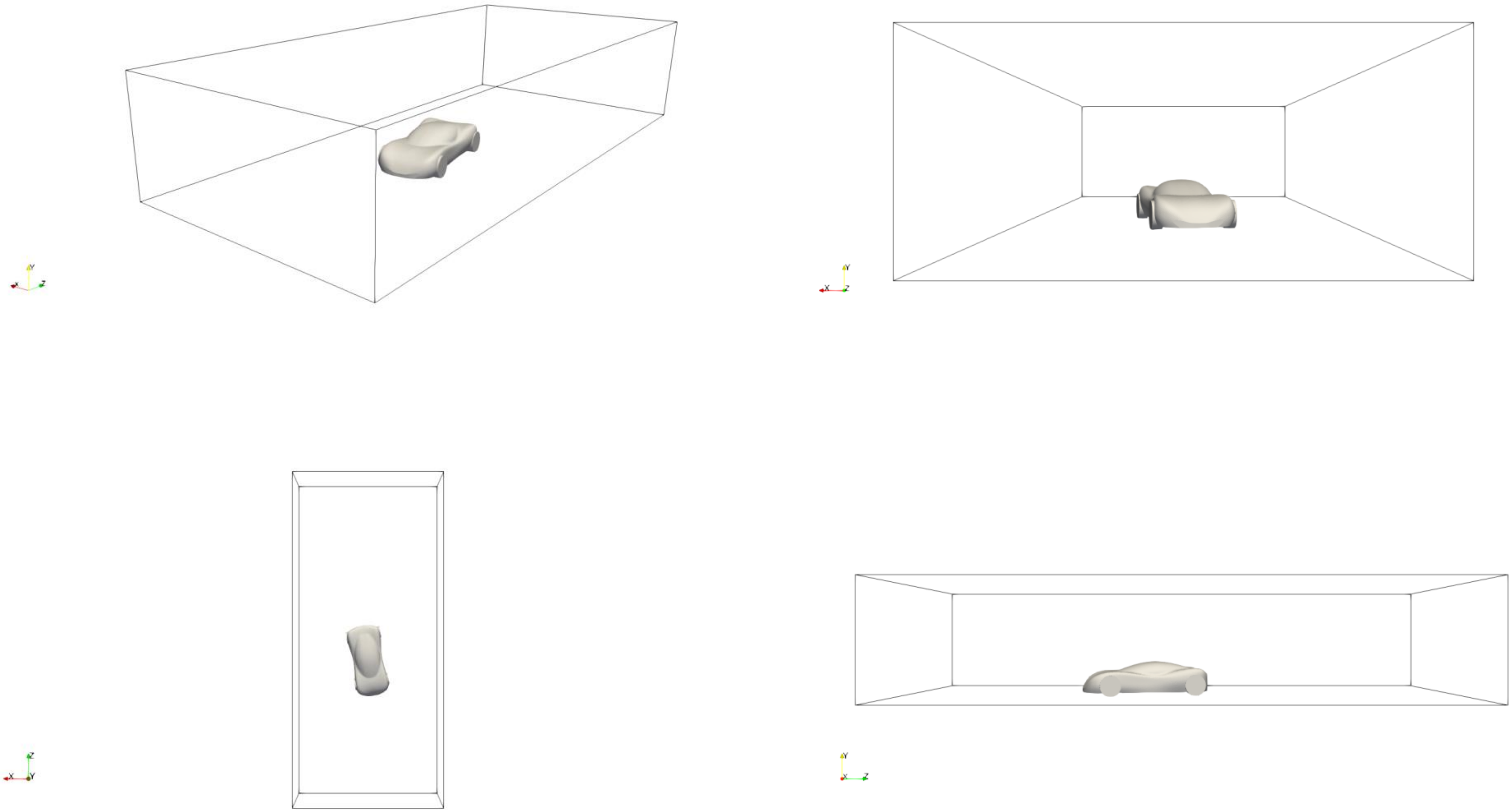
Coprocessing

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

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

- In the case directory, you will find a few scripts with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on.
- These scripts can be used to run the case automatically by typing in the terminal, for example,
  - `$> sh run_solver`
- These scripts are human-readable, and we highly recommend you open them, get familiar with the steps, 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, run the cases automatically using these scripts.
- In the case directory, you will also find the `README.FIRST` file. In this file, you will find some additional comments.

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

```
180    functions
181    {
358        isoSurfaces1


403        isoSurfaces2


450        cuttingPlanes1


521        nearWallField1


549        patch_surface1


584        patch_surface2


618        streamlines1


659        streamlines2


697        wallBoundedStreamLines1


941   };
```

- 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 358 and 403 we defined the **functionObjects** to compute iso-surfaces.
- In line 450 we defined the **functionObjects** to compute cut-planes.
- In line 521 we defined the **functionObjects** to compute near wall fields.
- In lines 549 and 584 we defined the **functionObjects** to compute fields on patches.
- In lines 618, 659, and 697 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**

```
358    isoSurfaces1
359    {
360        type surfaces;
361        functionObjectsLibs (“libsampling.so”)

363        enabled true;

368        writeControl timestep;
369        writeInterval 10;

371        surfaceFormat vtk;
372        fields ( p U k omega );

374        interpolationScheme cellPoint;

376        surfaces
377        (

379            p_constantIso
380            {
381                type isoSurface;
382                isoField p;
383                isoValue 30;
384                Interpolate false;
385            }

               ...
               ...
               ...

395        );

397    }
```

- Let us take a look at the iso-surfaces definition.
- In lines 360-361 we select the library and type of **functionObject**.
- In line 363 we can turn-on and turn-off the **functionObject**. This can be done on-the-fly.
- In lines 368-369 we select the saving frequency. The saving frequency can be different from the saving frequency of the solution.
- In line 371 we select the output format (many formats are available).
- In line 372 we select the fields to save with the iso-surface. No need to mention that the fields must exist.
- In lines 374 we select the interpolation method.
- In lines 376-395 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**

```
358    isoSurfaces1
359    {
360        type surfaces;
361        functionObjectsLibs (“libsampling.so”)

363        enabled true;

368        writeControl timestep;
369        writeInterval 10;

371        surfaceFormat vtk;
372        fields ( p U k omega );

374        interpolationScheme cellPoint;

376        surfaces
377        (

379            p_constantIso
380            {
381                type isoSurface;
382                isoField p;
383                isoValue 30;
384                Interpolate false;
385            }

               ...
               ...
               ...

395        );

397    }
```

- In lines 379-385 we define the **p_constantIso** object.
    - In line 379 we give a unique name to this object.
    - In line 381 we define the type (iso-surface).
    - In line 382 we select the field to compute the iso-surface.
    - In line 383 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 372.
- 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

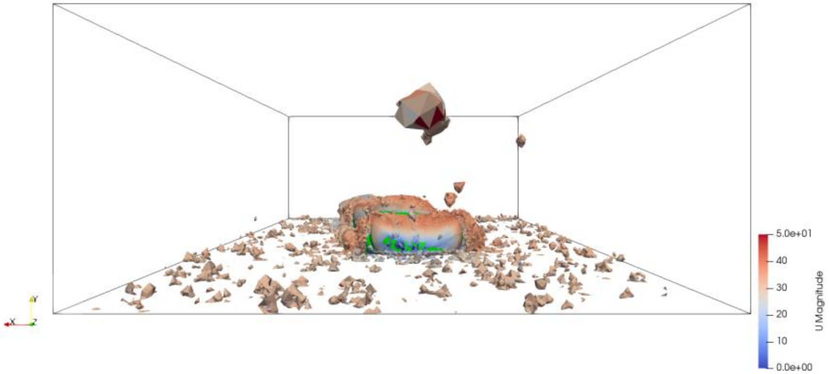
Iso-surfaces of pressure field

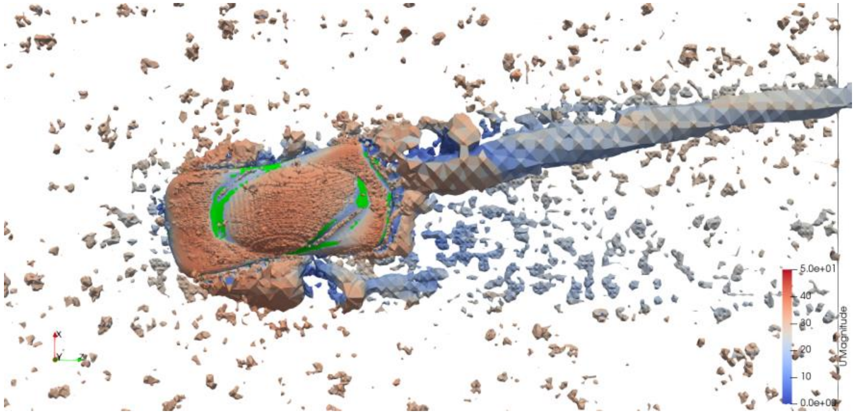
- Iso-surfaces sampled using **functionObjects**.
- By using coprocessing, we only saved this specific iso-surface information.
- There is not need to save the whole solution.
- This can significantly reduce the amount of data stored and help us in doing faster post-processing.

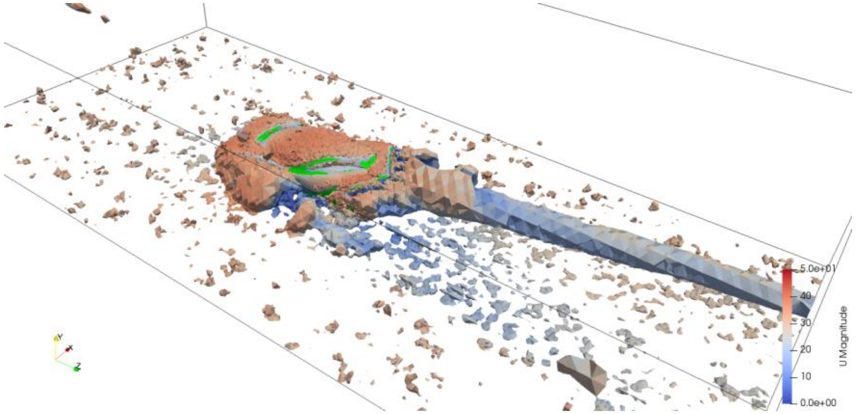
Coprocessing

Iso-surfaces of Q criterion

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







Coprocessing

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

```
450    cuttingPlanes1
451    {
452        type surfaces;
453        functionObjectsLibs (“libsampling.so”)

455        enabled true;

460        writeControl timestep;
461        writeInterval 10;

463        surfaceFormat vtk;
464        fields ( p U k omega );

466        interpolationScheme cellPoint;

468        surfaces
469        (
470            xNormal
471            {
472                type cuttingPlane;
473                planeType pointAndNormal;
474                pointAndNormalDict
475                {
476                    basePoint (0 0 0);
477                    normalVector (1 0 0);
478                }
479                Interpolate true;
480            }

               ...
               ...
               ...

506        );

508    }
```

- Let us take a look at the cut planes definition.
- The options in lines 452-466 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 466-506 we define the cut-planes. You can add as many as you like.
- In lines 470-480 we define the **xNormal** object.
    - In line 470 we give a unique name to this object.
    - In lines 471-480 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**

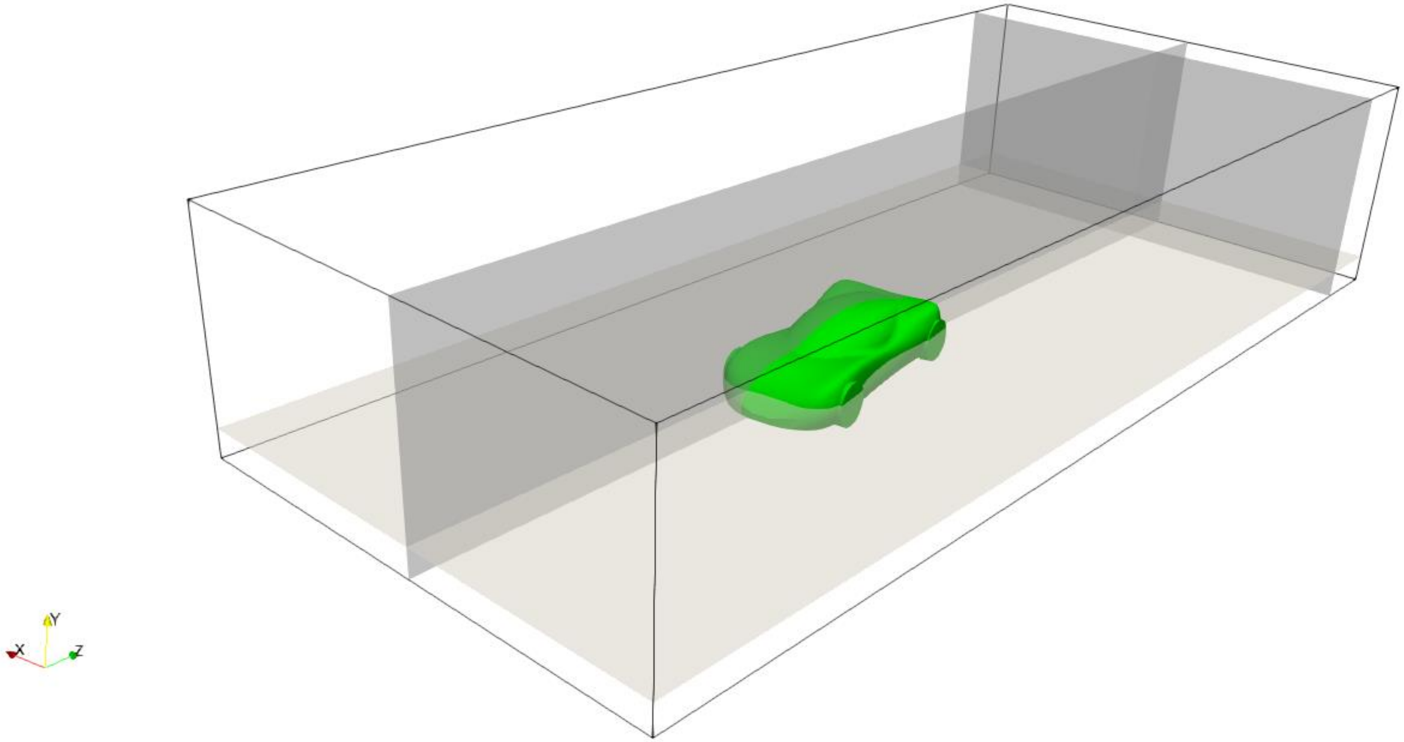
```
450   cuttingPlanes1
451   {
452       type surfaces;
453       functionObjectsLibs (“libsampling.so”)

455       enabled true;

460       writeControl timestep;
461       writeInterval 10;

463       surfaceFormat vtk;
464       fields ( p U k omega );

466       interpolationScheme cellPoint;

468       surfaces
469       (
470           xNormal
471           {
472               type cuttingPlane;
473               planeType pointAndNormal;
474               pointAndNormalDict
475               {
476                   basePoint (0 0 0);
477                   normalVector (1 0 0);
478               }
479               Interpolate true;
480           }

              ...
              ...
              ...

506       );

508   }
```

- The output of this **functionObject** is saved in the directory **`postProcessing/cuttingPlanes1`**
- The output is saved in this directory because in line 450 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

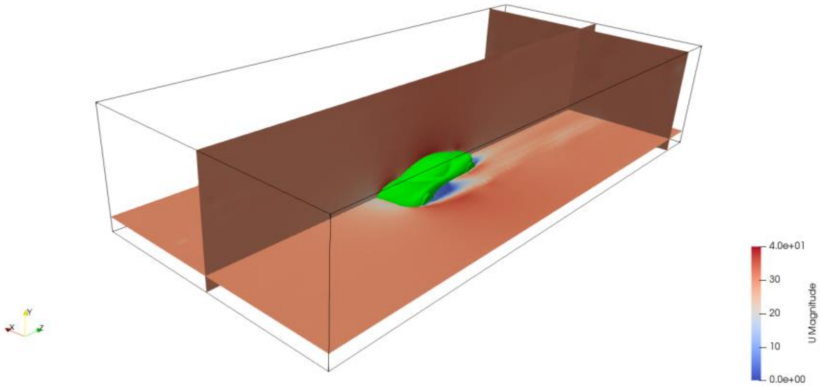
- By using coprocessing, we only saved this specific information.
- There is not need to save the whole solution.
- This can significantly reduce the amount of data stored and help us in doing faster post-processing.

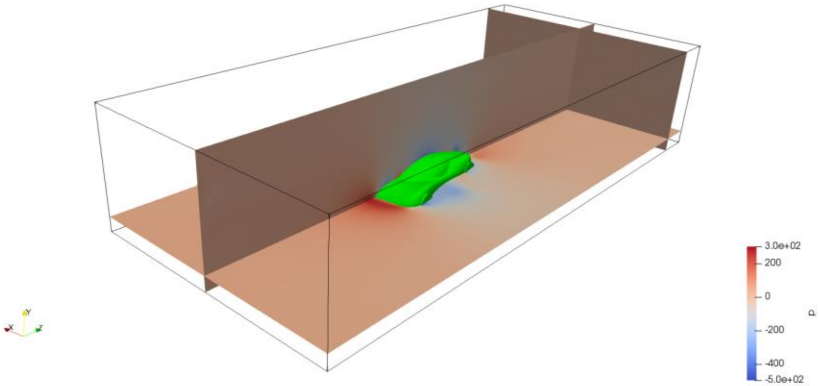


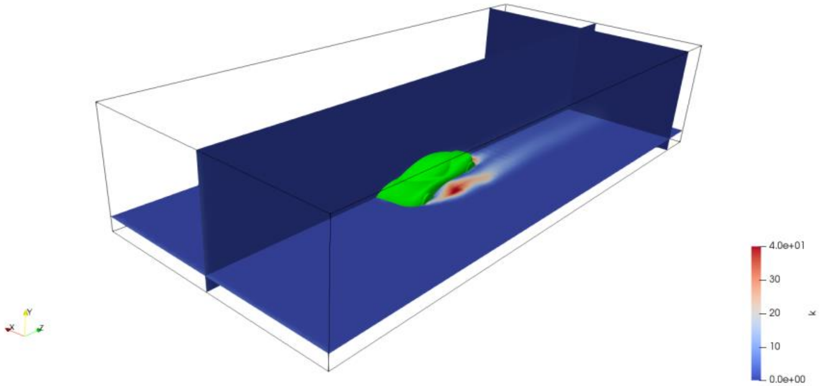
Coprocessing

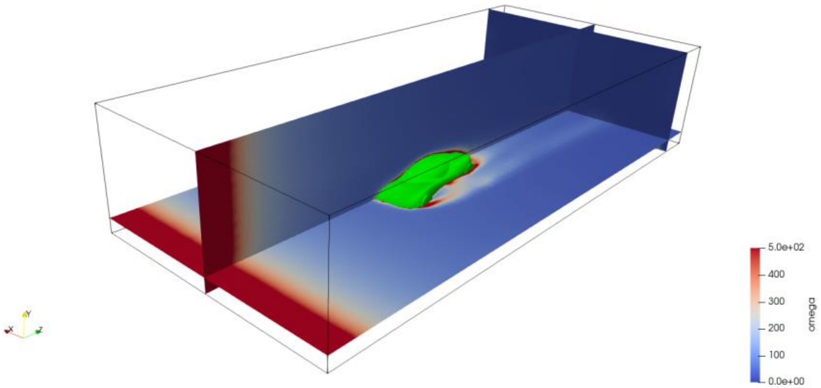
Cut-planes – Field variables contours

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









Coprocessing

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

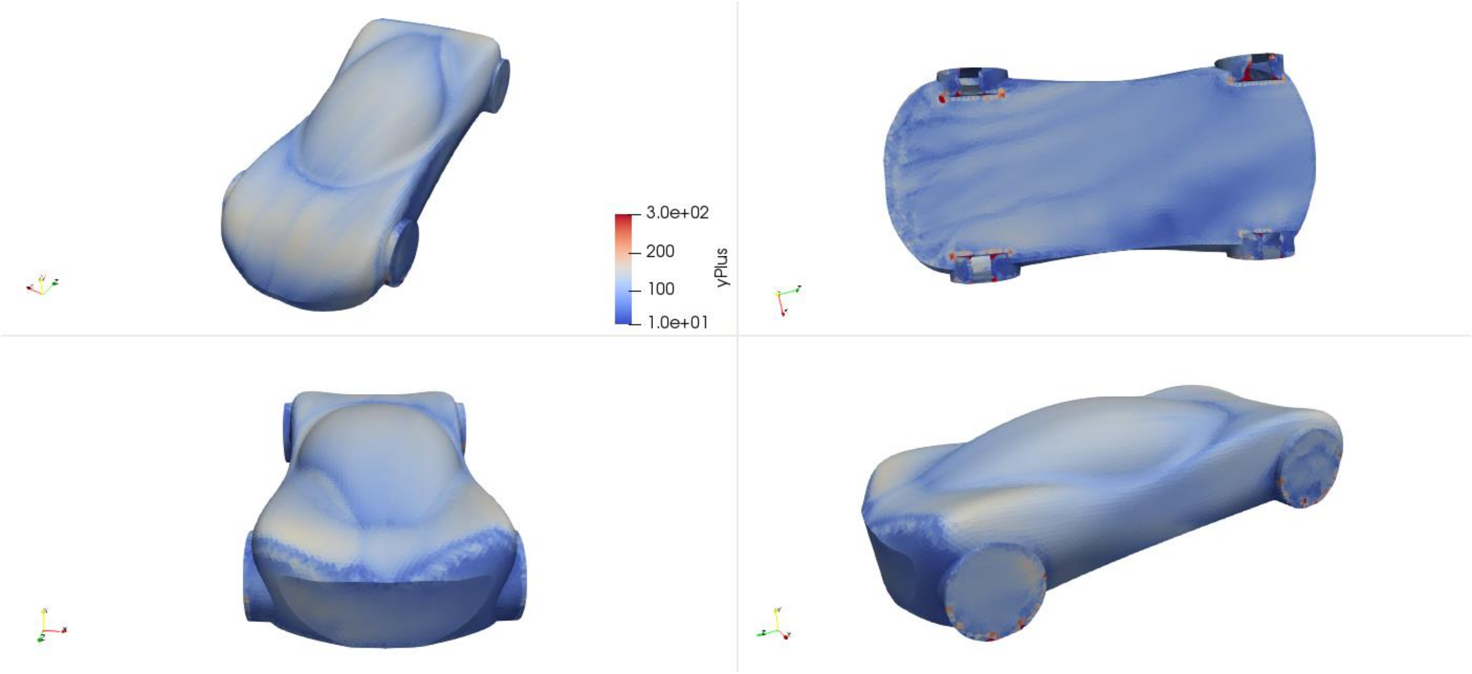
```
549    patch_surface1
550    {
551        type surfaces;
552        functionObjectsLibs (“libsampling.so”)

554        enabled true;

559        writeControl timestep;
560        writeInterval 10;

562        surfaceFormat vtk;
563        fields ( p U k omega yPlus );

566        interpolationScheme cellPoint;

568        surfaces
569        (

571            patch_car
572            {
573                type patch;
574                Patches (“car”);
575            }
576        );

578    }
```

- Let us see how to save the information at a given patch.
- The options in lines 551-566 are similar to those of the previous **functionObjects**.
- In lines 568-576 we define the sampling at a given patch.
- In line 574, we select the patch where we want to save the fields information.
- The fields used are defined in line 563.
- 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 549 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**

```
618    streamlines1
619    {
620        functionObjectsLibs (“libfieldFunctionObjects.so”)
621        type streamLine;

623        enabled true;

628        writeControl timestep;
629        writeInterval 20;

631        setFormat vtk;

633        direction forward;

635        U U;

637        fields (U p);

639        lifetime 10000;

643        nSubCycle 5;

645        sedSampleSet
646        {
647            type lineUniform;
648            axis x;
649            start (-2 0.7 4);
650            end   ( 2 0.7 4);
651            nPoints 100;
652        }
653    }
```

- Let us take a look at the streamlines definition.
- In lines 620-621 we select the library and type of **functionObject**.
- In line 623 we can turn-on and turn-off the **functionObject**. This can be done on-the-fly.
- In lines 628-629 we select the saving frequency. The saving frequency can be different from the saving frequency of the solution or other **functionObjects**.
- In line 631 we select the output format (many formats are available).
- In line 633 we select the tracking direction of the streamlines (forward, backward, or both).
- In line 635 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 637 we select the fields to save with the streamlines. No need to mention that the fields must exist.

Coprocessing

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

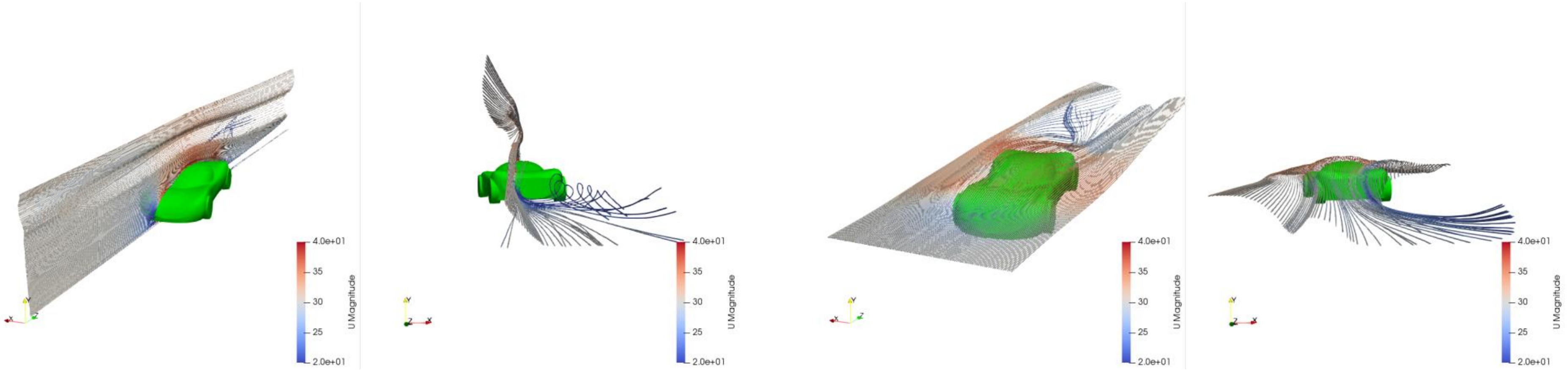
```
618    streamlines1
619    {
620        functionObjectsLibs (“libfieldFunctionObjects.so”)
621        type streamLine;

623        enabled true;

628        writeControl timestep;
629        writeInterval 20;

631        setFormat vtk;

633        direction forward;

635        U U;

637        fields (U p);

639        lifetime 10000;

643        nSubCycle 5;

645        sedSampleSet
646        {
647            type lineUniform;
648            axis x;
649            start (-2 0.7 4);
650            end   ( 2 0.7 4);
651            nPoints 100;
652        }
653    }
```

- In lines 639 and 643 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 647-651 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 618 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.



Coprocessing

Streamlines

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