Tutorial 4: Build your own OpenFOAM® simulation cases

Objective:

Based on the previous tutorials, the objective here is to use existing fluid dynamics solvers in OpenFOAM® for certain problems. You will do hands-on exercise to build simulation cases, i.e., prepare a mesh, setup boundary conditions, specify initial conditions, and run the simulations. You will also do some variations as you may encounter in practice.

The example cases provided in this tutorial are relatively complex. They contain multiple blocks (in comparison with single block in the cavity case shown in class). We will still use the `blockMesh` tool to generate the meshes. However, we will introduce "scripting" to automate, parameterize, and expedite the meshing process. In particular,

- `m4` scripting
- OpenFOAM® inline expression and calculation: *#eval*, *#calc*, and *codeStream*. See Appendix Section A for a brief introduction to these.

The requirement of this exercise is that you are comfortable with Linux command operations and familiar with OpenFOAM® simulation process.

The finished cases are provided as reference. However, you should go through the tutorial by yourself. For this tutorial, two zipped files are provided:

- `flow_over_step.tar.gz` for the flow-over-a-step case. It contains several sub-directories, each of which demonstrates the creation of a mesh with different approaches.
- `meander_channel.tar.gz` for the meander channel case.

1 Exercises:

1.1 Run the provided "Flow over a step" case

You are provided a simulation case as in the lecture. This case is called "flow over a step". It is a 3D case with RANS turbulence model. This is a very simple case. However, it contains almost everything for a typical OpenFOAM® simulation.

Let's first unzip the case and go to the directory (the following is just an example, you can put the case wherever you want):

```
$ run
$ cd tutorial/
$ mkdir CE597
```

```
$ mv path/to/flow_over_step.tgz ./
$ tar xzvf flow_over_step.tgz
$ cd  flow_over_step
$ ls
```

You will see there is a directory named "RAS", corresponding to RANS modeling (don't worry about the turbulence modeling yet; we have one chapter covering this later). Type the following to go to the 3D RANS case,

```
$ cd RAS
$ cd step_3D_m4
$ ls
```

In this case, a `m4` script is used to automatically generate the `blockMeshDict` file. You don't have to use `m4`. However, it will save you a lot of time when you want to change the mesh definition. You can have a look at the `m4` script by the following

```
$ cd system
$ more blockMeshDict.m4
```

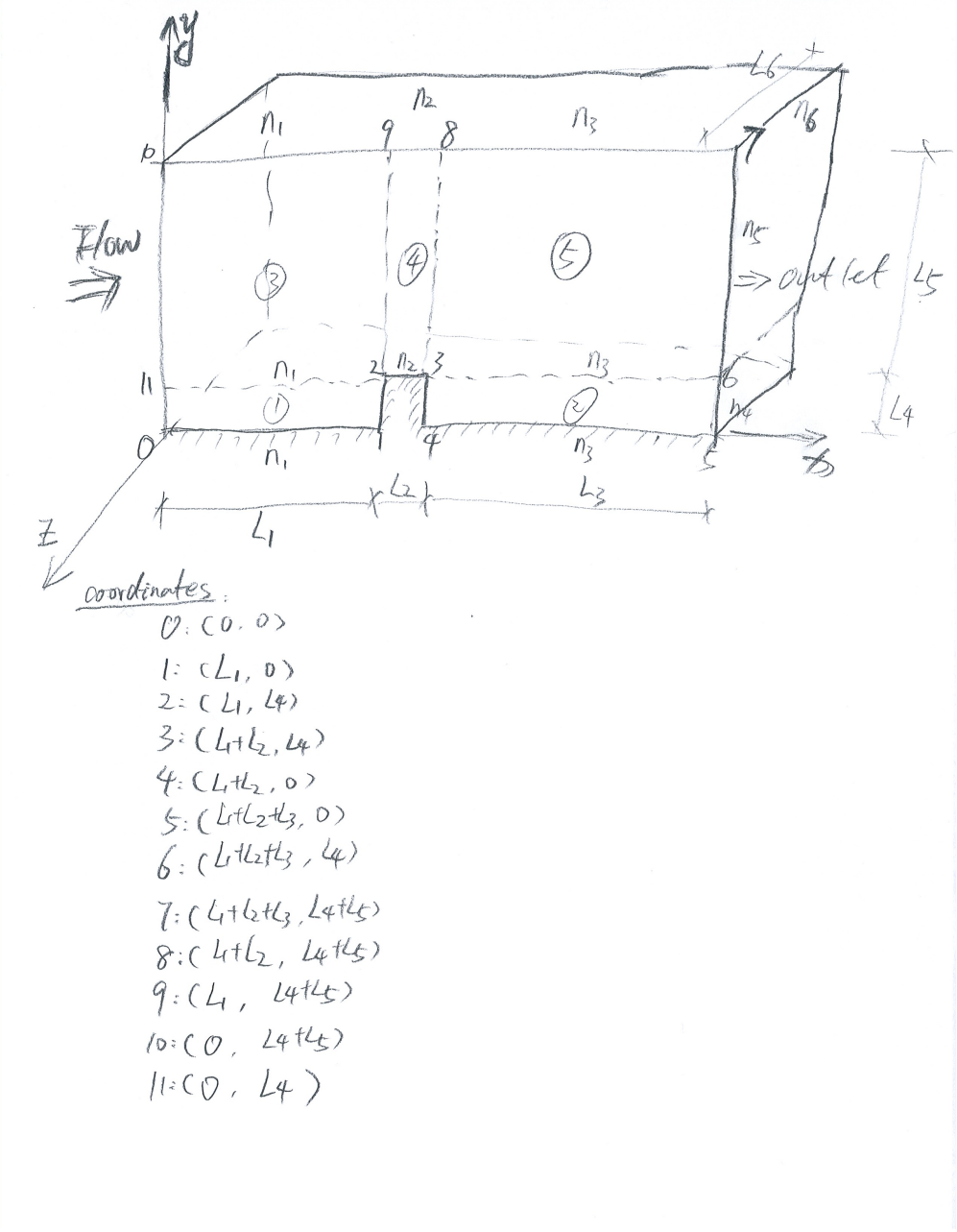


Figure 1: Sketch for the scheme of the flow over step case.

This `m4` definition file corresponds to the hand-drawing sketch (Figure 1). Now you need to use this `m4` file to generate the `blockMeshDict` file:

```
$ m4 blockMeshDict.m4 > blockMeshDict
$ cd ..     (Note: goes back to the case root directory)
```

```
$ blockMesh
```

This will generate the mesh needed. As a good practice, it is better to examine the mesh after you generate it. You can use `ParaView` to do this,

```
$ touch case.foam
$ paraview case.foam&    (Note: the "&" sign will let the shell go back right away;
                          otherwise it will wait until paraView to finish. )
```

In `ParaView`, just show the meshes by choose "Solid Color" and "Wireframe" as in Figure 2. You can also choose to show different regions of the mesh. I usually turn off everything and turn on each of the boundary patches to see whether they are defined correctly.

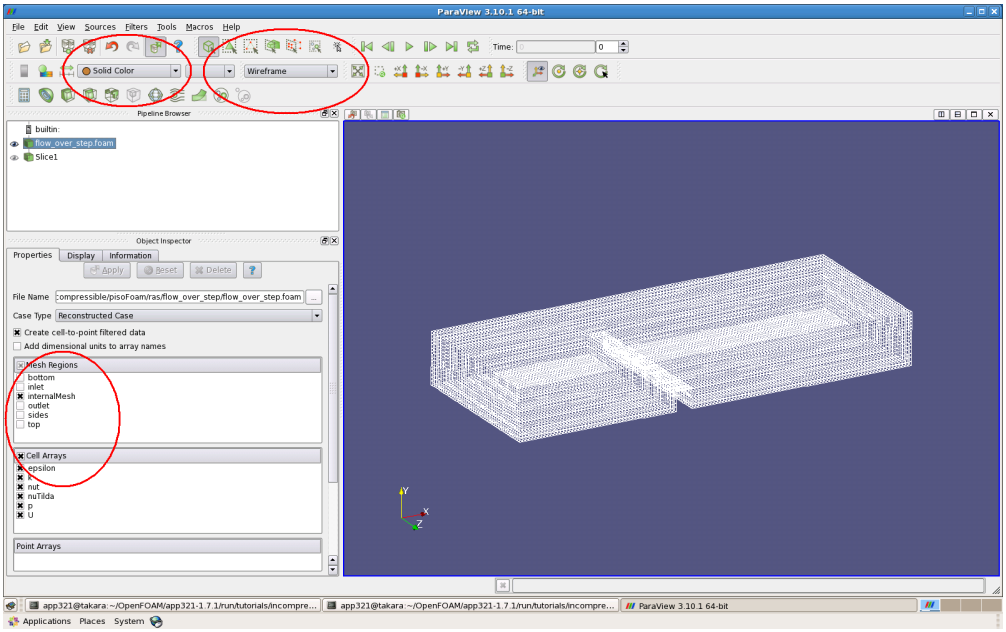


Figure 2: Visualization of mesh for the flow over step case in ParaView

You can also use the `checkMesh` tool to verify everything is ok with the mesh by typing

```
$ checkMesh
```

Now, you can run the simulation since everything else has been set up,

```
$ pisoFoam
```

Depending on the configuration of your computer, it might take a while to finish the run. When it is done, you are ready to visualize the results by using `ParaView`. Even when it is still running, you can examine the intermediate results since we specified output every 10 time steps.

```
$ touch case.foam
$ paraview case.foam&
```

Play around with the `ParaView` interface. The icons and buttons are straightforward. Try to accomplish the following tasks:

- Plot a slice of the result through the center of channel and show the velocity magnitude (as in Figure 3).
- Plot velocity vector on the center slice and adjust your view to be front. Zoom in to see the recirculation zone behind the step (as in Figure 4).
- Generate an animation as you saw in the lecture. You need to go the menu `File->Save Animation`.

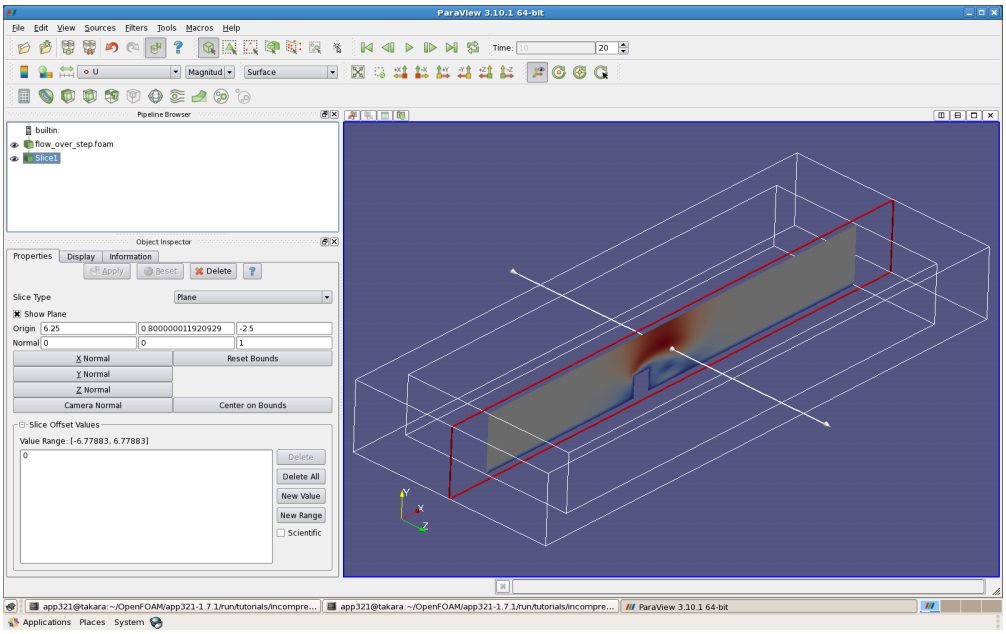


Figure 3: Velocity contour plot on the center slice of the flow over step case in ParaView

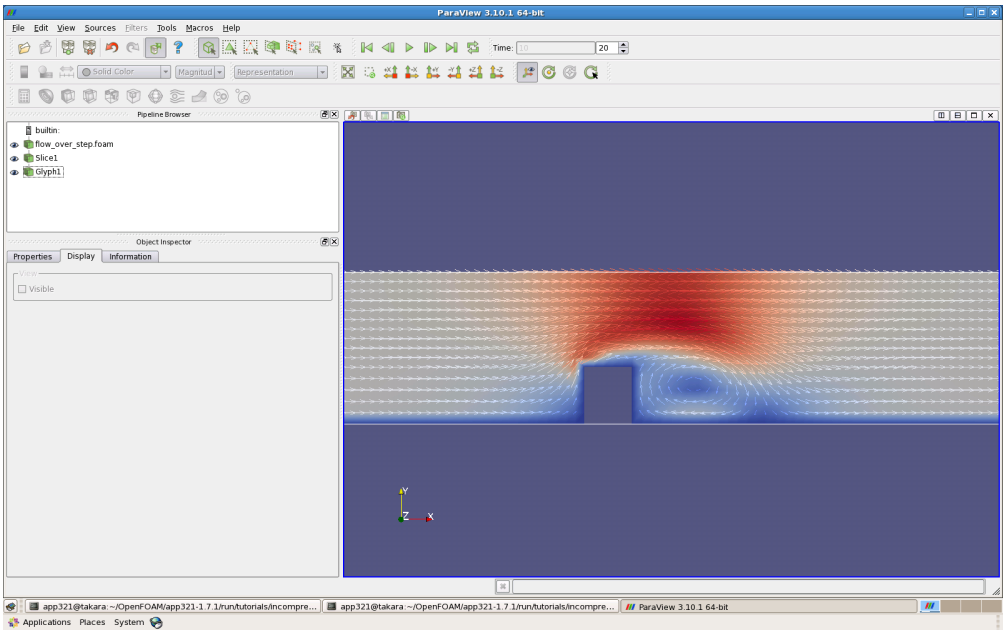


Figure 4: Velocity contour and vector plot on the center slice of the flow over step case in ParaView

There are other things worth exploring, such as generation of streamlines, contour surface (which is useful for example to plot free surface), clip, threshold, and other post-processing filters provided by `ParaView`. For a list of available filters, go to menu `Filters`.

`ParaView` is free and open source. It is the quickest way to see your result from OpenFOAM® . However, I found the visualization effect is less appealing comparing to other professional software. If you prefer, you can export the simulation results to the format of the visualization software of your choice.

1.1.1 Refine mesh for the "Flow over a step" case

Often times, you will run a case on a coarse mesh first to check whether everything is setup correctly. To increase the accuracy of your result, and sometime more importantly to reveal the real physics, you need to refine your mesh. OpenFOAM® is a finite volume method code based on unstructured mesh. In general, the maximum accuracy is second order (proportional to the mesh size). So refining the mesh will reduce the numerical error accordingly.

For this case, we want to reduce the cell size to 1/8 of the previous one. To do that, you can change the `blockMeshDict` directly or modify the `m4` script provided. If you want change the `blockMeshDict`, you need to edit the `block` definition part for each of the block. I will show how to change the `m4` script file. It is very simple. First, clean the simulation by typing

```
$ ./Allclean
```

This will remove all simulation result time directories and the mesh. There is another script called `CleanTimeDir` which will only clean the result time directories but not the mesh.

Now edit the `m4` script file `blockMeshDict.m4` using the text editor of your choice. Find the line saying `define(refine, 1)` and change it to `define(refine,2)`. This will double the cell numbers all three directions. The resulting cell size will be $1/2^3 = 1/8$ of the original one. Don't be too ambitious to refine the mesh as it could get very large quickly.

Follow the steps as you did in the previous exercise to: (a) generate the mesh, (b) run the simulation, and (c) visualize the result. And compare the results with two different resolutions.

There are a few things to note when you refine the mesh:

- The simulation will take longer time.
- You might need to reduce the time step size accordingly to lower the Courant number. A Courant number less than 0.5 is recommended though some of the temporal discretization schemes do not put any restriction on time step size. In addition, larger time step size Δt will increase the temporal error. The time step size can be modified in the `controlDict` file under the `system` directory. You can monitor the Courant number from the output of the simulation.

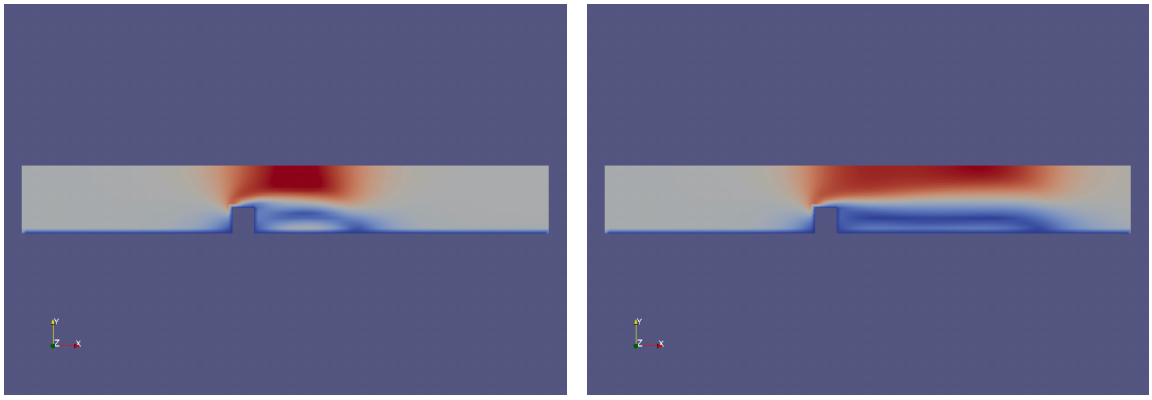
1.1.2 Run the "Flow over a step" case longer

From the animation in the lecture, you may notice that the recirculation zone behind the step is not fully developed. The reason is that we only run the case for 10 seconds. To see a fully developed flow, you need to run longer.

You don't need to run everything from time "0" again. In OpenFOAM® , you can continue the simulation from where you stopped last time (a.k.a., hot start). In fact, you could start from any intermediate time as long as the intermediate results have been written into a time directory.

To extend the run, modify the `controlDict` file under the `system` directory. The entries in this file are self-explaining. Assume you have finished your previous run for 10 seconds. You can instruct OpenFOAM® to continue the run by changing `startTime` to 10. You also need to change the `endTime` to say 40 so we will run additional 30 seconds.

Compare your results at 10 s and 40 s to see whether it is fully developed. Otherwise, you need to continue the simulation.



(a) $t = 10\ s$ (b) $t = 40\ s$

Figure 5: Comparison of velocity magnitude at time 10 s and 40 s for the flow over step case.

1.1.3 Run the "Flow over a step" case as a 2D case

If the channel is straight and very wide, you can simplify it as a 2D case where we simulate the flow in the vertical plane (streamwise and vertical directions). Note this is different from the other sense of "2D" model which uses depth-averaged shallow water equation. In shallow water equation model, the plane is horizontal (streamwise and across the river) and pressure is hydrostatic in the vertical direction.

In OpenFOAM® , at the time of solver development, we do not know or even care whether a simulation case with this solver will be 2D or 3D. You specify the dimensionality through boundary conditions for a case. For a 2D case, you need to use the `empty` boundary condition for the two patches in the front and back. Referring to the hand drawing sketch, the two patches perpendicular to the z axis needs to be specified as `empty`. In addition, there can only have one cell across the z direction. The exact length in z does not matter, however.

Accordingly, we need to do the following modifications based on our 3D case.

- First, you need to change the definition of the mesh. You can change the `blockMeshDict` file directly. Or you can change the `m4` script file. For example, if you change the `m4` script, you need to edit the `blockMeshDict.m4` file and find the line defines `n6` (number of cells along z direction). Change it to `define(n6, 1)`.
- Then, you need to change all `sides` boundary conditions for the variable files in the initial condition directory "0". Change the `sides` boundary condition type to `empty`.

  Sometime it is also beneficial to check the `boundary` file in directory `constant/polyMesh`. You need to change the boundary type of patch `sides` to `empty`. If this file exists before you use `blockMesh` to generate the mesh, OpenFOAM® assumes that it is your intent to keep whatever boundary conditions you have there. So it is your responsibility to

ensure consistency. If not, OpenFOAM® will complain and abort the simulation.

The 2D cases have been provided. If you are still in the 3D case directory, you can go up one level to the "RAS" directory and have a look,

```
$ cd   ..
$ ls
```

You should see the following 2D cases:

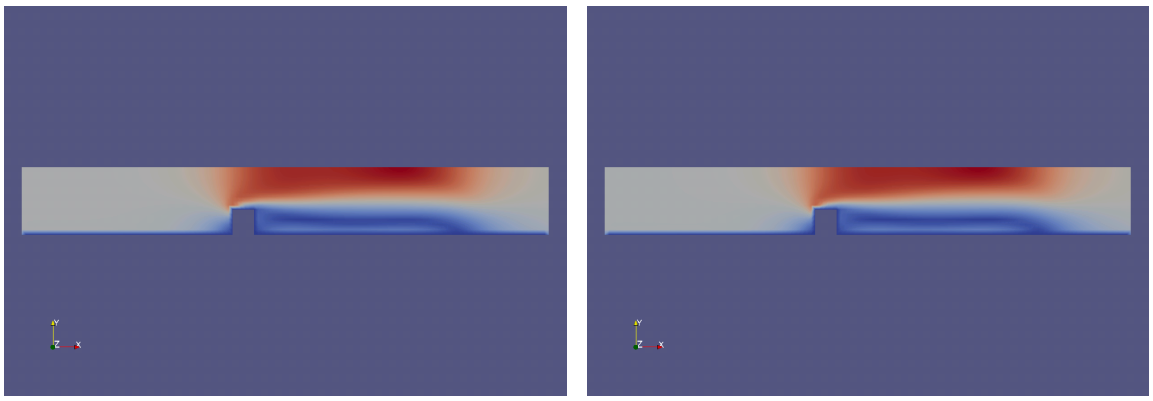
- `step_2D_m4`: 2D case with `m4` script to generate the `blockMeshDict` file. You should run the `m4` script as above to generate the mesh dictionary file first and then run `blockMesh`.
- `step_2D_calc`: 2D case with OpenFOAM's `#calc` dynamic compilation. You can directly run `blockMesh`.
- `step_2D_eval`: 2D case with OpenFOAM's `#eval` expression. You can directly run `blockMesh`.

See Appendix A for more details about `#eval` and `#calc`.

You can run any of the 2D cases by first generating the mesh, and then

```
$ blockMesh
$ pisoFoam
```

You can compare results from 3D and 2D simulations as in the following figure. It seems that the difference is very minor.



(a) 2D result (b) 3D result

Figure 6: Comparison between 2D and 3D results for the flow over step case.

Regarding whether to simulate a case in 2D or 3D, note the following:

- 2D simplification assumes no variation across the channel. In reality, it is usually not the case.
- If you are interested in things like shear stress on the channel side banks, you need to add the third dimension.

If you don't want to learn another script language `m4`, you can achieve the same mesh with the native support of "inline expression and calculation", i.e., `#eval` and `#calc`. However, at least currently, scripting languages like `m4` or Python are more powerful than the OpenFOAM® native support of inline expression and calculation. **Bottom line is that you should know at least one of the approaches demonstrated in this example for meshing**.

1.2 A more complicated case - flow in meander channel

This is a further example, and probably more meaningful one, to show how to use OpenFOAM® to simulate flows in meander (curved) channels. A section of an artificial meander channel with a bend is modeled. You can go to the case directory by

```
$ run
$ cd tutorials/CE597
$ mv path/to/meander_channel.tgz ./
$ tar xzvf meander_channel.tgz
$ cd meander_channel
$ ls
```

The channel geometry is shown in Figure 7 and the sketch for `blockMesh` is in Figure 8.

(a) Plain view (b) 3D view

Figure 7: Plane view and 3D view of the geometry of the meander channel with two bends.

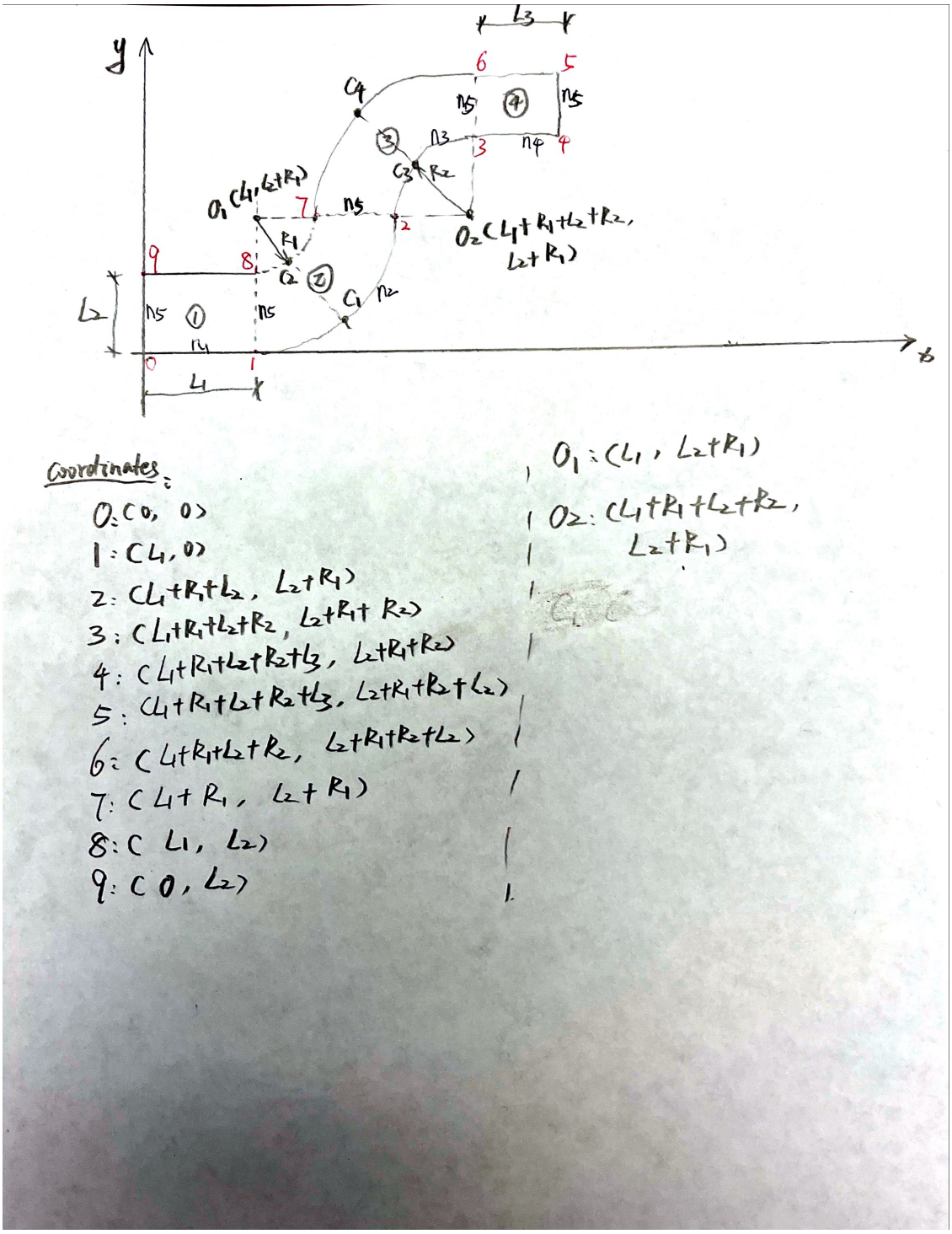


Figure 8: Sketch for the scheme of the flow in meander bends .

The tool `blockMesh` is used to generate the mesh. To automate the meshing process, I used a `m4` script to generate the `blockMeshDict` file. The definitions for the dimensions are plotted in a companion PDF file named `scheme_meander_channel_bends.pdf`. The case provided only generates a moderately sized mesh. In reality, you probably need much higher resolution to reveal important physics, such as secondary flow. In addition, the `blockMesh` tool needs to divide the domain into four blocks, which might be not so convenient or even possible for real rivers. For real rivers with more complicated bathymetry, you probably need a professional meshing software.

To simplify things, we do not consider the free surface. Instead, a shear-free rigid lid is put on the top. At the inlet into the bends, a uniform velocity of 0.2 m/s is specified. At the downstream end (the outlet), a far field is assumed. For the turbulence, the k-ϵ model is used.

You should follow the general steps of performing a simulation as you did in the previous examples.

- First, you need to write the `m4` script according to the provided sketch in the PDF file. Of course, a `m4` script has been provided for your reference. However, it is better for your learning if you do it by yourself.
- Then you need to use the `m4` script to generate the `blockMeshDict` file.
- Then use the `blockMesh` tool to generate the mesh.
- Now you should run the simulation using the `pisoFoam` solver.

I will not show you the exact commands. Note this example is for demonstration purpose only. The mesh is very coarse. It does not necessary fully capture the real flow phenomenon.

When the simulation is finished, you should be able to visualize the results and do some post processing. For example, try to visualize the flow patterns by using streamlines and cross sections as in Figure 9.

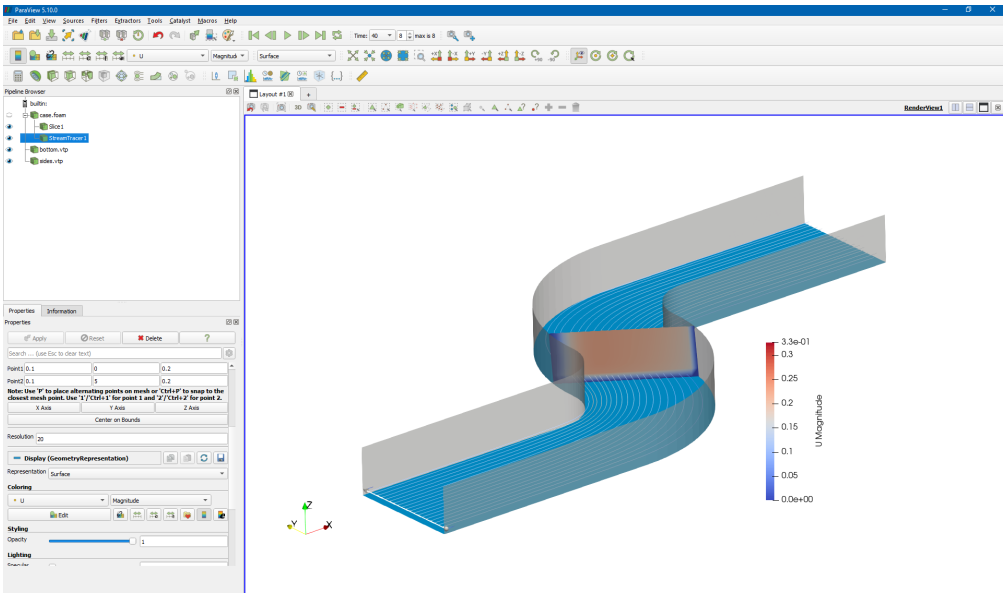


Figure 9: Simulation results showing the streamlines and one cross section in the first bend.

In fluid mechanics, you are probably interested in the shear stresses along the boundaries. In the context of river hydraulics, the wall shear stresses are the driving forces for sediment transport and therefore the stability of the channel. For blood flow within an artery, they are important for the longevity of the blood vessels. You can easily get these results by using the postprocessing tool `wallShearStress`,

```
$ pisoFoam -postProcess -func wallShearStress -time 40
```

We will explain the details of the `postProcess` functionality later. Here, the option `time 40` specifies the particular time that you want to calculate the wall shear stress. If not specified, it will calculate for all time steps. Afterwards, you can examine the content inside directory "40" and there should be a newly created field file named `wallShearStress`. Have a look at the content of this file. You can also plot the color contour of the wall shear stresses. To do that, you need to make sure to load the results for the boundaries. As in Figure 5, you need to specify the `bottom` and `sides` as the `Mesh Regions` to be loaded.

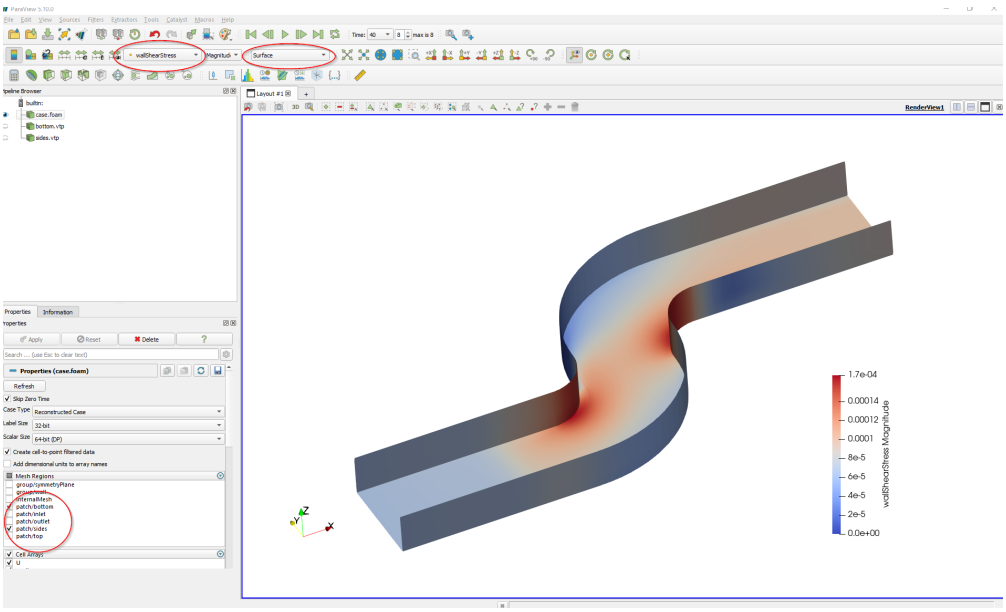


Figure 10: Load and show the wall shear stress on the bottom and the two banks.

The other way to do this is to convert the results into VTK (Visualization Toolkit) format which is widely supported (including `ParaView`). You could use the generic tool called "formToVTK",

```
$ foamToVTK -time 40
```

There are other options such as specify which field variable to be converted. For a complete list of options, you can type `foamToVTK -help`. In `ParaView`, you can directly load the generated VTK files from the menu `File->Open`.

Appendix A *#eval*, *#calc*, and codeStream

A.1 *#eval*

ESI OpenFOAM provides the functionality of *expression* syntax starting from v1912, which enables users to define custom formulas (a.k.a expressions) within a dictionary. *expression* can be used in a variety of places, including input dictionaries, boundary conditions, and utilities. It is noted that at present, *expression* is not available in OpenFOAM Foundation version.

The documentation for *expression* can be found here:

https://www.openfoam.com/documentation/guides/latest/doc/openfoam-guide-expression-syntax.html

An example of using *expression* is as follows:

```
x 2.0;
y #eval "$x / 2";
```

which assigns "x/2" to "y".

In contrast to *#calc* (explained below), *expression* does not need to dynamically compile C++ code, though its syntax is similar to C++. Instead, it uses predefined grammar rules and parsing operations to evaluate the expression. *expression* supports comments, macros, common operators, typical math functions, and many other things. More details on how to use this powerful functionality can be found in its documentation.

A.2 *#calc*

#calc uses dynamic compilation to provide calculating capability for dictionary entries. For example,

```
x 2.0;
y #calc "$x / 2.0";
```

This example achieve the same effect as the one for *#eval* above. However, the difference here is that it needs compilation. Within the OpenFOAM source code, *#calc* is a wrapper around the *codeStream* functionality. Because of the need to do compilation, the run of the simulation will be slightly slow at the beginning. In addition, a compilation environment (e.g., `gcc`) has to exist obviously.

A.3 codeStream

codeStream is an OpenFOAM dictionary that contains OpenFOAM's C++ code which can be compiled to generate dictionary entries. This sounds abstract. But in essence, *codeStream* is a mechanism that you can write a snippet of OpenFOAM code within a dictionary. You can include "code", "codeInclude" (optional), and "codeOptions" (optional). The code snippet will be compiled by "wmake libso". *codeStream* can be used in any place within a dictionary file. For example, it can be used to implement a new boundary condition and the initialization of a field.