Praktikum Wissenschaftliches Rechnen (CFD, Final Project)

Group 9: Breznik, E., Cheng, Z., Ni, W., Schmidbartl, N.

1 The Topic

For the final project, our group chose to extend the Navier Stokes code to be used in 3D for arbitrary geometries, with the ability of handling the Free Surface flow scenarios. We put a lot of attention to designing a code that will work in 3D for as many different scenarios as possible. That is why we did not code a special set of boundaries for just a few more known and frequently simulated situations, but rather made them completely dependent on the input geometry file.

1.1 Running the program

Program is called in the same way as its 2D version;

```
./sim parameter_file
```

with parameter_file denoting the usual (only slightly extended) parameter file, where (under variable name *problemGeometry*) also the geometry file is given - see next section for more information on input.

1.2 Input

The parameter file that was used as input in our previous worksheets remained, with just slight changes due to the additional dimension. Only notable difference is, that we now take as an input also a scalar $velIN$, and a vector $velMW$. First one represent the velocity at the inflow boundaries, and the second one the wall velocity for the moving wall boundaries.

To make our code able to handle truly arbitrary scenarios, we designed so that it allows for any sort of implemented boundary conditions to be employed in any domain cell - so even the obstacles inside the domain can have arbitrary boundaries, as opposed to only allowing that on the domain walls (as in worksheet 3). The standard boundary conditions, that we implemented, are

- no slip,
- free slip,
- inflow,
- outflow,
- moving wall.

To specify, where which of them is applied, we used special numbering of cells when generating our input pgm files, which can be seen from table 1, so our geometries are represented by a grayscale image with 7 levels of brightness. Since we are working in three dimensions, this image consists of

Cell type	**Number code**
water	0
air	1
no-slip	2
free-slip	3
inflow	4
outflow	5
moving wall	6

Table 1: Number representations of different possible cell types.

a sequence of 2D (xy plane) images. The orientation of coordinate system and an example picture for a small lid driven cavity case can be seen on figure 1. One geometry picture set thus needs to contain $(imax + 2) \times (jmax + 2) \times (kmax + 2)$ numbers in $(jmax + 2) \times (kmax + 2)$ rows, and is typically very big.

2 Implementation

2.1 Boundary conditions

As mentioned before, we have implemented 5 different boundary conditions for the 6 boundaries of our 3D domain, as well as for internal boundaries.

2.1.1 No-Slip boundary condition

For no-slip conditions, the fluid vanishes at the boundary. As a consequence, both the velocity components are zero.

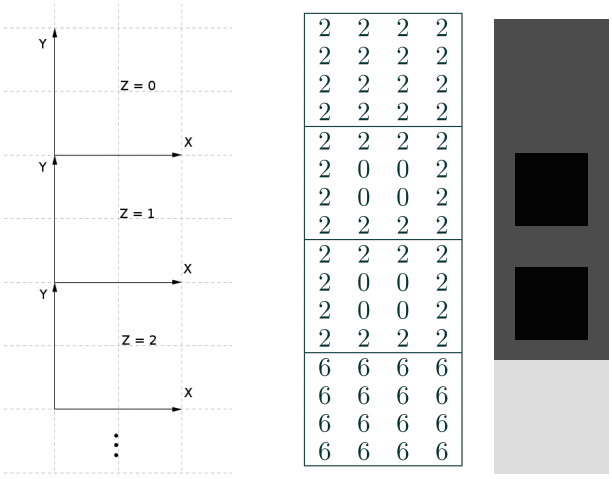


Figure 1: The structure of our input pgm files and a lid driven cavity example case.

For the `B_NOU` case, we set the following boundary conditions:

$$\begin{aligned} u_{i,j,k} &= 0, \quad u_{i-1,j,k} = -\tfrac{1}{2}(u_{i-1,j+1,k} + u_{i-1,j,k+1}), \\ v_{i,j,k} &= 0, \quad v_{i,j-1,k} = -\tfrac{1}{2}(v_{i+1,j-1,k} + v_{i+1,j-1,k+1}), \\ w_{i,j,k} &= 0, \quad w_{i,j,k-1} = -\tfrac{1}{2}(w_{i+1,j,k-1} + w_{i,j+1,k-1}), \end{aligned} \tag{1}$$

and we analogously set the boundary conditions for the following 7 cases: `B_NWU`, `B_NOD`, `B_NWD`, `B_SOU`, `B_SWU`, `B_SOD`, and `B_SWD`.

Boundary conditions of the cells that only have one or two fluid neighbours are analogous to their 2D counterparts, and are omitted here for the sake of brevity; same applies for the discussion in 2.1.2, 2.1.3, and 2.1.6.

2.1.2 Free-Slip boundary condition

For free-slip conditions, the fluid flows freely parallel to the boundary, but does not cross the boundary. As a consequence, the velocity component normal to the boundary is zero, as well as the normal derivative of the velocity component parallel to the wall.

For the `B_NOU` case, we set the following boundary conditions:

$$\begin{aligned} u_{i,j,k} &= 0, \quad u_{i-1,j,k} = \tfrac{1}{2}(u_{i-1,j+1,k} + u_{i-1,j,k+1}), \\ v_{i,j,k} &= 0, \quad v_{i,j-1,k} = \tfrac{1}{2}(v_{i+1,j-1,k} + v_{i+1,j-1,k+1}), \\ w_{i,j,k} &= 0, \quad w_{i,j,k-1} = \tfrac{1}{2}(w_{i+1,j,k-1} + w_{i,j+1,k-1}), \end{aligned} \tag{2}$$

and we analogously set the boundary conditions for the following 7 cases: `B_NWU`, `B_NOD`, `B_NWD`, `B_SOU`, `B_SWU`, `B_SOD`, and `B_SWD`.

2.1.3 Outflow boundary condition

For outflow conditions, the normal derivatives of both the velocity components are zero.

For the `B_NOU` case, we set the following boundary conditions:

$$\begin{array}{ll} u_{i,j,k} = u_{i+1,j,k}, & u_{i-1,j,k} = \frac{1}{2}(u_{i-1,j+1,k} + u_{i-1,j,k+1}), \\ v_{i,j,k} = v_{i,j+1,k}, & v_{i,j-1,k} = \frac{1}{2}(v_{i+1,j-1,k} + v_{i+1,j-1,k+1}), \\ w_{i,j,k} = w_{i,j,k+1}, & w_{i,j,k-1} = \frac{1}{2}(w_{i+1,j,k-1} + w_{i,j+1,k-1}), \end{array} \tag{3}$$

and we analogously set the boundary conditions for the following 7 cases: `B_NWU`, `B_NOD`, `B_NWD`, `B_SOU`, `B_SWU`, `B_SOD`, and `B_SWD`.

2.1.4 Inflow boundary condition

For inflow conditions, we assume the inflow velocity is perpendicular to the inflow boundary. Thus, the input for the velocity is a positive scalar. We implement the following 6 cases, `B_O`, `B_W`, `B_N`, `B_S`, `B_U` and `B_D`, and forbid the other cases (see section 2.2 for forbiden cells). For the `B_O` case, the boundary velocity that we set to the input value is $u_{i,j,k} =$ `velIN`, while other velocities for this cell are set to zero. We analogously set the boundary conditions for the other 5 cases.

2.1.5 Moving wall boundary condition

For moving wall conditions, we need to restrict the moving wall directions. When the boundary cell is like `B_O`, we allow the moving wall direction to be $+y$ or $-y$. Consequently, the boundary velocities are equal to zero, except for the ones along this moving wall direction and parallel to the boundary. The only velocities that are thus set to a non-zero value in this case are:

$$v_{i,j-1,k} = 2 \cdot \texttt{velMW}_y - v_{i+1,j-1,k}, \quad v_{i,j,k} = 2 \cdot \texttt{velMW}_y - v_{i+1,j,k}, \tag{4}$$

where $\texttt{velMW} = [\texttt{velMW}_x, \texttt{velMW}_y, \texttt{velMW}_z]$ is the moving wall velocity vector, and $\texttt{velMW}_x$ is its component along the x axis. We analogously set the boundary conditions for the following 5 cases: .

When the boundary cell is like `B_NO`, we set the directions of moving walls `N` and `O` in a "circular" way. For example, if `N` moves in the $+x/-x$ direction, then `O` moves in the $-y/+y$ direction. Therefore, the following boundary conditions differ from those of the no-slip condition:

$$\begin{array}{ll} u_{i,j,k} = 2 \cdot \texttt{velMW}_x - u_{i,j+1,k}, & u_{i-1,j,k} = 2 \cdot \texttt{velMW}_x - u_{i-1,j+1,k}, \\ v_{i,j,k} = 2 \cdot \texttt{velMW}_y - v_{i+1,j,k}, & v_{i,j-1,k} = 2 \cdot \texttt{velMW}_y - v_{i+1,j-1,k} \end{array} \tag{5}$$

We analogously set the boundary conditions for the following 16 (5 + 11) cases: `B_W`, `B_N`, `B_S`, `B_U`, `B_D`, `B_NW`, `B_NU`, `B_ND`, `B_SO`, `B_SW`, `B_SU`, `B_SD`, `B_OU`, `B_WU`, `B_OD`, `B_WD`, and we forbid the boundary conditions for the following 8 cases: `B_NOU`, `B_NWU`, `B_NOD`, `B_NWD`, `B_SOU`, `B_SWU`, `B_SOD`, `B_SWD`.

2.1.6 Boundary condition for the pressure

On the other hand, for all the 5 aforementioned boundary conditions, the boundary values for the pressure are derived from the discretized momentum equation and result in discrete *Neumann* conditions (the same as in the 2D case).

For the `B_NOU` case, we set the following boundary conditions

$$\begin{aligned} F_{i,j,k} = u_{i,j,k}, \quad G_{i,j,k} = v_{i,j,k}, \quad H_{i,j,k} = w_{i,j,k}, \\ p_{i,j,k} = \tfrac{1}{3}(p_{i+1,j,k} + p_{i,j+1,k} + p_{i,j,k+1}) \end{aligned} \tag{6}$$

and we analogously set the boundary conditions for the following 7 cases: `B_NWU`, `B_NOD`, `B_NWD`, `B_SOU`, `B_SWU`, `B_SOD`, and `B_SWD`.

2.2 Allowed cells and their representation

As in the 2D case, we again forbade geometries with cells, that are of boundary type and border on fluid in two opposite directions. However, we added two more types of forbidden cells, to make implementation easier and results more physical. Firstly, the inflow cells that border on fluid on more than one side are forbidden due to the fact that we're reading the inflow velocity as a scalar. Setting that for a cell with more fluid neighbours would mean having to choose the vector (u, v or w) of the inflow. And secondly, since an obstacle that has a corner and all of its sides moving isn't really physical, we decided not to deal with cells that have three fluid neighbours and moving wall boundary condition.

Cells are in the code represented by flags, which consist of 16 bits, instead of 5 (see following table). That extension was necessary because we planned

center	east	west	north	south	bottom	top
4bits	2bits	2bits	2bits	2bits	2bits	2bits

to add the possibility of free flow scenarios, fo which we needed the additional (apart from water and obstacle) state of cells - air. We also extended representation of center cell by three bits, to be able to store the information on special boundary conditions.

For neighbour cells we used the following 2bit representation:

01	$\cdots$	water
10	$\cdots$	air
11	$\cdots$	obstacle

And for the center cell we used 4bit sequences:

00|01	$\cdots$	water
00|10	$\cdots$	air
00|00	$\cdots$	no slip
00|11	$\cdots$	free slip
01|00	$\cdots$	inflow
01|11	$\cdots$	outlow
10|00	$\cdots$	moving wall

2.3 Output

As previously, results for chosen times are written into vtk files with corresponding names. However, there is an additional output of csv files, for the particle tracking part. Into those, particle position, velocity, and index of particle set are written. The particle lines can be easily visualized from this data sets in paraview as well.

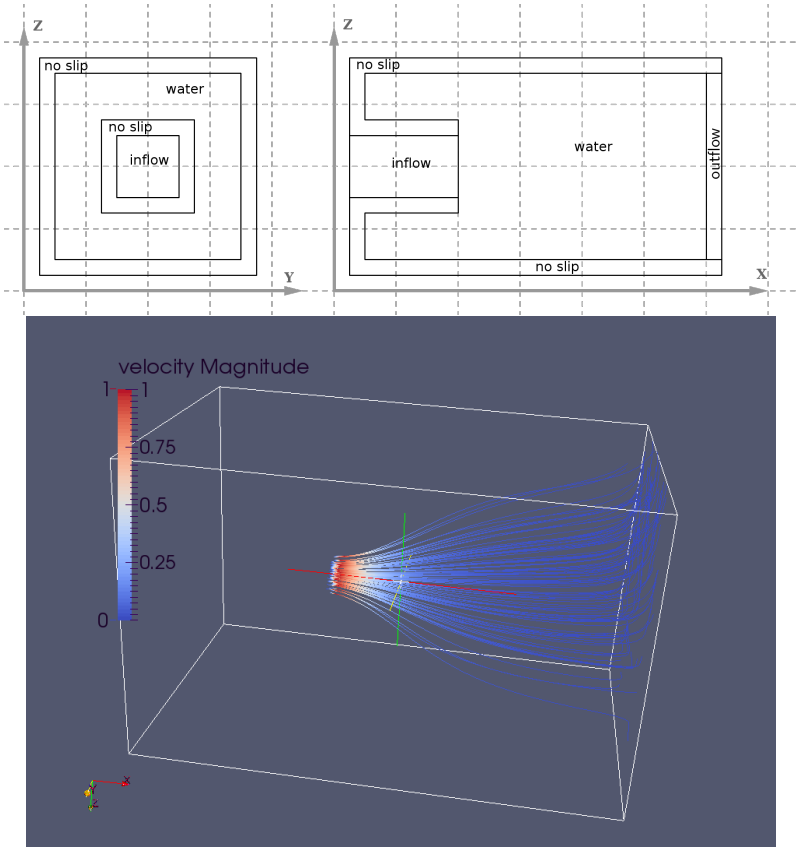
3 Problems, current state and future work

Implementation of truly arbitrary geometries took us much longer than expected; we had problems at deciding which cells to forbid and how to calculate the allowed ones for more complicated boundary conditions. We have therefore spent less time on free surfaces; and have not been able to finish it completely. Currently, tracking of particles is implemented, and is done automatically within simulation (no special setting or call needed). There is a lot of space for improvement of our code, for example finishing the free surface extention, parallelization, and so on.

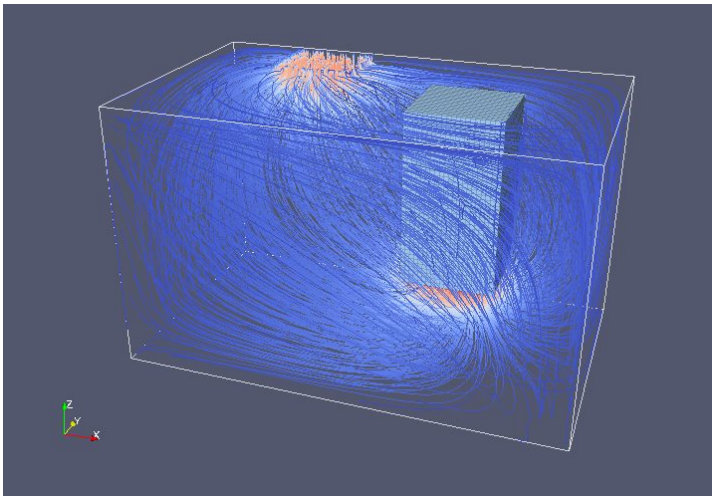
4 Results

Here are a few show cases of what our code can handle: inflow through just part of the outer wall, inflow on the surface of interior obstacle, particle tracking, etc.

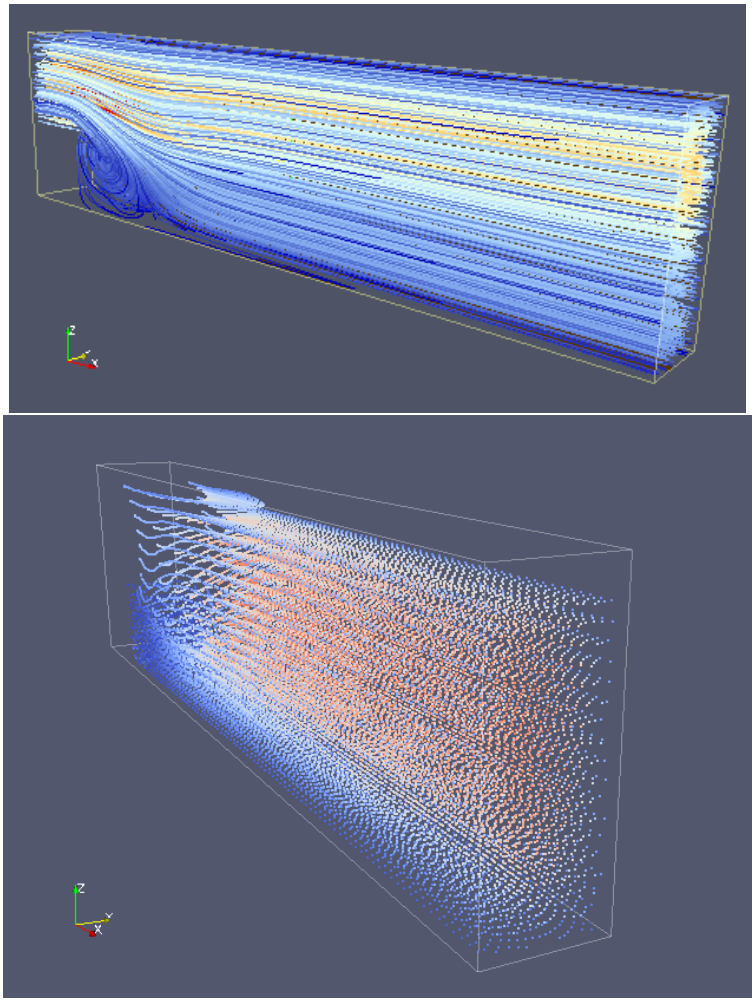
- Inflow through a pipe: geometry and streamlines.



- Inflow through part of wall, outflow through pipe.



- Flow over step: streamlines and particle paths.



References

[1] Griebel, M., Dornsheifer, T., Neunhoeffer, T.: *Numerical Simulation in Fluid Dynamics: A Practical Introduction*. SIAM, **1998**.

[2] Hirt, C. W., Nichols, B. D.: *Volume of Fluid Method for the Dynamics of Free Boundaries*. Journal of Computational Physics **39** (1981).

[3] Hirt, C. W., Nichols, B. D., Hotchkiss, R. S.: *SOLA-VOF: A solution Algorithm for Transient Fluid Flow with Multiple Free Boudaries*. LASL, **1980**.