PWS/Meesterproef vloeistofdynamica

Wiebe Derksen and Daniël Visser

February 20, 2022

PWS begeleiders meneer Straatman en meneer Matena, expertbegeleider Aron Van Den Bogaard

Contents

1 Introduction

People have been simulating liquids and gasses for decades using computers. We thought this would be an interesting subject to write our PWS about.

2 The Navier-Stokes equations

The Navier Stokes equations are [20]:

$$\begin{aligned}\rho(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z}) \\ = \rho g_x - \frac{\partial P}{\partial x} + \frac{\partial}{\partial x}\left[2\mu\frac{\partial u}{\partial x} + \lambda div\vec{V}\right] \\ + \frac{\partial}{\partial y}\left[\mu(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial z})\right] + \frac{\partial}{\partial z}\left[\mu(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x})\right]\end{aligned} \tag{1}$$

$$\begin{aligned}\rho(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z}) \\ = \rho g_y - \frac{\partial P}{\partial y} + \frac{\partial}{\partial x}\left[\mu(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x})\right] \\ + \frac{\partial}{\partial y}\left[2\mu\frac{\partial v}{\partial y} + \lambda div\vec{V}\right] + \frac{\partial}{\partial z}\left[\mu(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y})\right]\end{aligned} \tag{2}$$

$$\begin{aligned}\rho(\frac{\partial v}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial x}) \\ = \rho g_z - \frac{\partial P}{\partial z} + \frac{\partial}{\partial x}\left[\mu(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x})\right] + \\ \frac{\partial}{\partial y}\left[\mu(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y})\right] + \frac{\partial}{\partial x}2\mu\frac{\partial w}{\partial y} + \lambda div\vec{(}V)\end{aligned} \tag{3}$$

3 What algorithms are feasible to implement?

3.1 Introduction to differential equations

There are many algorithms to execute fluid simulations, each has its own advantages and disadvantages. In this chapter we will discuss these algorithms and their pro's and cons.

A differential equation can be solved either numerically or analytically. An analytical solution is a solution for the entire domain of the function. A numerical solution only gives an approximation for some discrete values. Many differential equations are not analytically solvable, they have to be solved numerically. A numerical solution is often obtained using computers. [2]

3.2 The algorithms

In this section we will discuss some of the most used algorithms for solving the Navier-Stokes Equations, their advantages and disadvantages.

3.2.1 Finite Difference Method(FDM)

The Finite Difference Method(FDM) approximates the derivatives of a function using a truncated Taylor series. For a Taylor series around a point x=a the following formula applies. :[4]:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}}{i!}(x-a)^i$$

A computer cannot keep computing forever. Therefore we have to use a truncated Taylor series. The truncated Taylor series unto i=3 is given by [8]

$$f(x) = f(a) + f'(a)(x-a) + \frac{f``(a)(x-a)^2}{2} + \frac{f^{(3)}(a)(x-a)^3}{6} + O(\Delta x)^4$$

$O(\Delta x)^4$ are the truncated terms.[10]. Using the above formula, we can calculate $f(x+\Delta x)$ by taking a=x[9]

$$f(x+\Delta x) = f(x) + f`(x)\Delta x + \frac{f``(x)(\Delta x)^2}{2} + \frac{f^{(3)}(\Delta x)^3}{6} + O(\Delta x)^4 \tag{4}$$

In the same way we can calculate $f(x-\Delta x)$:

$$f(x-\Delta x) = f(x) - f`(x)\Delta x + \frac{f``(x)(\Delta x)^2}{2} - \frac{f^{(3)}(x) * (\Delta x)^3}{6} + O(\Delta x)^4 \tag{5}$$

Using equation 4 we can approximate f'(x):

$$f(x+\Delta x) - f(x) = f'(x)\Delta x + O(x)^2$$

$$f`(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} + O(\Delta x)$$

$O(x)^2$ changes in $O(x)$ if it is divided by Δx. The equation above is called a first order approximation , because the lowest power in the big O is 1. The lowest power of the big O of a second order approximation is 2. $O(\Delta x)^n$ decreases to zero when Δx becomes smaller, because $O(\Delta x)^n$ consists only of powers of Δx. The higher the lowest power, the fastest $O(\Delta x)^n$ declines and the better the derivative is approximated. We can calculate the second order approximation using equations 4 and 5

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \tag{6}$$

We can also use 4 and 5 to calculate the second approximation of the second derivative.

$$f''(x) = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{(\Delta x)^2} \tag{7}$$

Equation 6 can be used to turn equations 1 , 2 and 3 into a set of equations without deriatives. The values of these equations can be calculated.

3.2.2 Finite Element Method(FEM)

The Finite Element Method(FEM) is a commonly used to solve differential equations. It has a lot of applications, CFD's being only one of them. Unlike the FDM the FEM does not solve the equations for only a few points, but for the entire domain. It does so by approximating the differential equation with linear subfunctions. The linear subfunctions will have unknown coefficients. These coefficients are put into a matrix.

3.2.3 Finite Volume Method(FVM)

The finite volume method calculates the movement of "fluid particles". These particles are discretized points of a certain volume.[14] Therefore the FVM gives the exact expression for the average value of the solution (over the hence described volume). For each volume there is a flow in (Diffusion), a flow out (Convection), a transient term(the disturbance of the volume) and a source term resulting in the formula below.

$$\frac{\partial \rho\phi}{\partial t} + \nabla \cdot (\rho \boldsymbol{v} \phi) = \nabla \cdot (\Gamma^{\phi} \nabla \phi) + Q^{\phi}$$

3.2.4 Spectral Element Method

The spectral element method is a solution to the partial differentiation equation derived from the finite element method.

3.2.5 Lattice Boltzmann Method

The Lattice Boltzmann Method models liquids as if the liquid exists of a n number of fictive particles. The method divides the simulation in two interchanging steps, collision and streaming. The steps change the density $\rho(\vec{x}, t)$ at position $\vec{x}$ at time t. Because the Lattice Boltzmann method works within a lattice (a grid with a n amount of points in a m number of dimensions), you can draw up a formula $f_i(\vec{x}, t)$. In this formula f_i is the the influence of a neighbouring point i on $\vec{x}$ (a point on the lattice). The total movement vector in a point then becomes:

$$P_i(\vec{x}, t) = \sum_{i=0}^{a} f_i(\vec{x}, t)$$

Where a is the number of adjacent points on the lattice and P the total movement of the particle.

Collision For simulating the collision of particles within the lattice, the formula is [7]:

$$f_i(\vec{x}, t + \delta_t) = f_i(\vec{x}, t) + \frac{f_i^{eq}(\vec{x}, t) - f_i(\vec{x}, t)}{\tau_f}$$

$f_i(\vec{x}, t + \delta_t)$ is the influence at time $t + \delta_t$ with δ_t the size of the time step. A smaller $\delta_t$leads to a more accurate simulation but also lead to more computing time required to simulate a certain situation.
τ_f is a constant that depends on the viscosity of the simulated liquid. $f_i^{eq}(\vec{x}, t)$ is the equilibrium density.

Streaming For simulating the actual movement of the particles the formula is:

$$f_i(\vec{x} + \vec{e_i}, t + \delta_t) = f_i(\vec{x}, t)$$

Here $f_i(\vec{x} + \vec{e_i}, t + \delta_t)$ is the fluid density at point $\vec{x}$ and $\vec{e}$ is the movement vector(the change in perceived density) of the density.

3.2.6 Boundary Element Method

The boundary element method is a method of simulating fluid dynamics with the presence of a influence of the boundary of the simulation, on the simulation. For this reason the boundary element method is quite complex and not really in the scope of this project. [12]

3.2.7 Turbulence methods

Apart from the above mentioned methods, there are also turbulence methods. These methods are able to simulate turbulent flows of liquids. These methods use a huge amount of computing power and thus are not really feasible for this paper.

3.3 Some Choices

Due to the finite difference method being relatively easy to implement, we have decided to start our simulation by implementing that method. For a programming language we have chosen (provisionally) for Rust as it is fast and reasonably easy.

3.4 Neglections in the Navier Stokes Equations

3.4.1 The dimension

It is easier for computers to solve a 2D problem than a 3D problem, because for a 2D problem the number of grid points is smaller then for a 3D problem. For example, a 10 by 10 grid contains 100 points, while a 10 by 10 by 10 grid contains 1000 points. Furthermore, 2D rendering is easier then 3D rendering. However, nowadays computers are very fast. Therefore, our computers will be able to solve 3D problems. Since we intend to simulate a container with water, we will do a 3D simulation.

3.4.2 Compressible and incompressible Navier Stokes Equations

All fluids can be compressed. Their volume gets smaller when the pressure on the gas or fluid increases. However, in many cases the change in volume is not significant, then the flows can be approximated as incompressible flows. The Navier Stokes Equations can be simplified to the incompressible Navier Stokes Equations when a fluid is not compressible. Water is not very compressible[21], so we will use the incompressible Navier Stokes Equations. The incompressible Navier Stokes equations for a Newtonian fluid, such as water, are:[18] [26]

$$\begin{aligned}
&\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}+\frac{\partial v_z}{\partial z}=0\\
&\left(\frac{\partial v_\mathrm{x}}{\partial t}+\frac{\partial v_\mathrm{x}}{\partial x}v_\mathrm{x}+\frac{\partial v_\mathrm{x}}{\partial y}v_\mathrm{y}+\frac{\partial v_\mathrm{x}}{\partial z}v_\mathrm{z}\right)\rho=-\frac{\partial p}{\partial x}+\eta\left(\frac{\partial^2 v_\mathrm{x}}{\partial x^2}+\frac{\partial^2 v_\mathrm{x}}{\partial y^2}+\frac{\partial^2 v_\mathrm{x}}{\partial z^2}\right)+\rho g_\mathrm{x}\\
&\left(\frac{\partial v_\mathrm{y}}{\partial t}+\frac{\partial v_\mathrm{y}}{\partial x}v_\mathrm{x}+\frac{\partial v_\mathrm{y}}{\partial y}v_\mathrm{y}+\frac{\partial v_\mathrm{y}}{\partial z}v_\mathrm{z}\right)\rho=-\frac{\partial p}{\partial y}+\eta\left(\frac{\partial^2 v_\mathrm{y}}{\partial x^2}+\frac{\partial^2 v_\mathrm{y}}{\partial y^2}+\frac{\partial^2 v_\mathrm{y}}{\partial z^2}\right)+\rho g_\mathrm{y}\\
&\left(\frac{\partial v_\mathrm{z}}{\partial t}+\frac{\partial v_\mathrm{z}}{\partial x}v_\mathrm{x}+\frac{\partial v_\mathrm{z}}{\partial y}v_\mathrm{y}+\frac{\partial v_\mathrm{z}}{\partial z}v_\mathrm{z}\right)\rho=-\frac{\partial p}{\partial z}+\eta\left(\frac{\partial^2 v_\mathrm{z}}{\partial x^2}+\frac{\partial^2 v_\mathrm{z}}{\partial y^2}+\frac{\partial^2 v_\mathrm{z}}{\partial z^2}\right)+\rho g_\mathrm{z}
\end{aligned} \tag{8}$$

The equation at the top is called the continuity equation. The others are the momentum equations. The spatial derivatives of the left hand sides are called the convection terms. The parts on the right hand sides between brackets are called the diffusion terms. The parts involving pressure are the pressure terms and ρg is the gravitational force.

3.4.3 Viscosity

All fluids have viscosity. But, like compressibility, it can be neglected sometimes. Viscosity is: $\eta = \frac{F/A}{dv_x/dz}$. Here F is the shear force, that is the force parallel to a surface. A is the surface and dv_x/dz is the change in velocity.[22]. We will not neglect viscosity, because we do intend to simulate a water container with boundaries and viscosity is important near the boundaries, because there is dissipation over there.[21] Dissipation is the conversion of some for of energy to heat energy[23]. Without viscosity the Navier Stokes Equations reduce to the Euler Equations[21].

3.4.4 Laminar and Turbulent flows

Flows can be either laminar or turbulent[21]. In laminar flows there is no disruption between adjacent layers, but in turbulent flows there is. Turbulent flows have chaotic paths and they swirl. The Navier Stokes Equations for laminar and turbulent flows are the same. But there are very small changes in pressure, velocity and temperature in turbulent flows, which should be dealt with. Although a turbulent flow can be modeled using the Navier Stokes Equations, this costs a lot of computation power due to the small changes.[24] Therefore, they are often modeled in another way. The simulation of turbulent flows is beyond the scope of this project.

3.4.5 Steady and unsteady

A flow is either steady or unsteady[21]. In a steady flow parameters, such as velocity, do not change over time. In a unsteady flow parameters can change over time.[25] We will simulate a unsteady flow, because we want that the water in the container is able to move.

3.5 Boundary conditions

To solve a partial differential equation it is necessary to enforce boundary conditions. A container with water has two types of boundaries. One type are the walls and the floor. The other is the free surface of the water. The free surface is the surface where the water and air touch each other. The boundary condition at the free surface is trickier than that at the walls and floor, since the free surface can move, while the walls and floors can not.

3.5.1 No-slip boundary condition

The no-slip boundary condition supposes the velocity of a fluid that touches a solid is equal to the velocity of the solid [16].

3.5.2 Free slip boundary condition

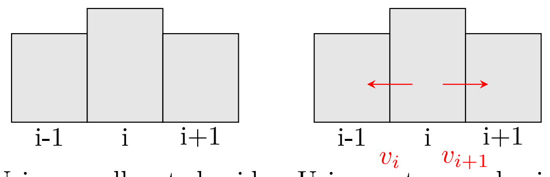
The free-slip boundary condition presumes the velocity of the fluid perpendicular to the wall is zero, so the fluid can not move through the wall. But, it does not restrict the velocity parallel to the wall.[17]

3.5.3 Dynamic boundary condition

The dynamic boundary condition assumes the pressure on the free surface must be constant.

4 The implementation of the finite difference method

4.1 Odd-even decoupling



Using a collocated grid   Using a staggered grid

Figure 1: Collocated and staggered grids

The Finite Difference Method calculates the pressure and velocity at discrete grid points. We can choose ourselves which variables will be calculated at which points. It would be the most easy to store all variables at every point. Such a grid is called a collocated grid[27]. However, as figure 1 shows, there will be no fluid flow between cell i-1 and i and cell i and i+1 when the pressure in cells i-1 and i+1 is equal in such a grid. This phenomenon is called odd-even decoupling and happens because the derivative in cell i will be zero[27]. It is called so because information from odd and even cells are not combined properly[27]. To solve that problem we will use a staggered grid, such as in the right side of figure 1. A staggered grid consists of cells. The pressure is stored in the cell center and the velocities in the cell edges that are perpendicular to them[27]. As one can see, the derivatives will not be zero anymore, since the flow between two cells will not depend on the pressure in other cells anymore.

4.2 Boundary conditions and walls

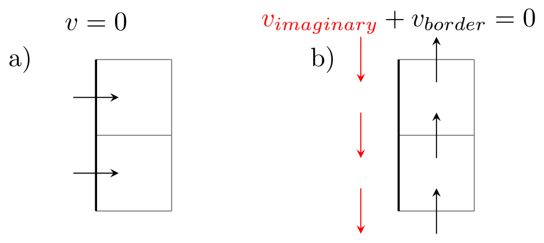


Figure 2: Wall boundary conditions

The water container is surrounded by walls on all sides. We will use a no-slip boundary conditions at the walls, as discussed in paragraph 3.5.1[28]. This means the velocities of the water at the wall will be equal to the velocity of the wall. The wall velocity is always zero, so the velocity of the water is zero at points on the wall. Therefore, all velocity components should be zero for wall points[28]. Unfortunately, as can be seen in figure 2.a only the velocity perpendicular to the wall has points that are at the wall. However, by using an imaginary velocity outside the box we can obtain the velocity at the wall using the average[28]:

$$v_{wall} = \frac{v_{imaginary} + v_{border}}{2}$$

Here v is parallel to the wall(so not necessarily in the y-direction), v_{wall} is the velocity at a wall point, v_{border} is the velocity that is closest to the certain wall point and $v_{imaginary}$ is the imaginary velocity. We can modify the formula to obtain $v_{imaginary}$ quite easily:

$$\begin{aligned}
&v_{wall} = 0 \\
&=> \frac{v_{imaginary} + v_{border}}{2} = 0 \\
&=> v_{imaginary} + v_{border} = 0 \\
&=> v_{imaginary} = -v_{border}
\end{aligned}$$

Because of the imaginary velocity grid points, we will have to make the grid larger in some directions.

In short, we have the following boundary conditions:
1) The velocities on the edge of the box that are perpendicular to the box are zero.
2) The imaginary velocities are equal to the inverse of the velocities opposite of the wall.

4.3 The sizes of the grids

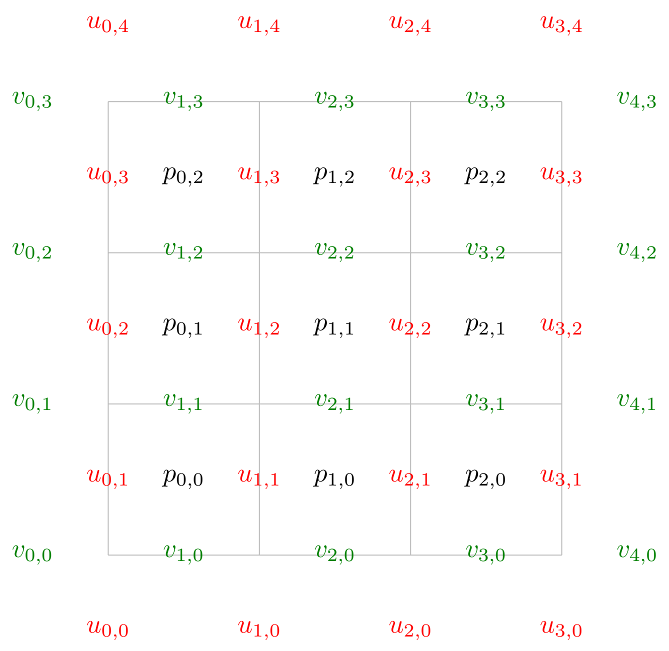

$u_{0,4}$ $u_{1,4}$ $u_{2,4}$ $u_{3,4}$
$v_{0,3}$ $v_{1,3}$ $v_{2,3}$ $v_{3,3}$ $v_{4,3}$
$u_{0,3}$ $p_{0,2}$ $u_{1,3}$ $p_{1,2}$ $u_{2,3}$ $p_{2,2}$ $u_{3,3}$
$v_{0,2}$ $v_{1,2}$ $v_{2,2}$ $v_{3,2}$ $v_{4,2}$
$u_{0,2}$ $p_{0,1}$ $u_{1,2}$ $p_{1,1}$ $u_{2,2}$ $p_{2,1}$ $u_{3,2}$
$v_{0,1}$ $v_{1,1}$ $v_{2,1}$ $v_{3,1}$ $v_{4,1}$
$u_{0,1}$ $p_{0,0}$ $u_{1,1}$ $p_{1,0}$ $u_{2,1}$ $p_{2,0}$ $u_{3,1}$
$v_{0,0}$ $v_{1,0}$ $v_{2,0}$ $v_{3,0}$ $v_{4,0}$
$u_{0,0}$ $u_{1,0}$ $u_{2,0}$ $u_{3,0}$

Figure 3: A 3*3 pressure grid

If we were using a collocated grid we would store our data in an array with the desired dimensions. However, we are using a staggered grid. This complicates the grid sizes a lot. We will have to use three seperate grids for velocity with different sizes and another grid for the pressure. Let u, v and w denote the velocities in respectively the x, y and z direction. The size of the pressure grid are m*n*o. In the directions that are orthogonal to the velocities of a velocity grid the size of that grid will be the size of the pressure grid plus one. We have illustrated this in figure 3. The size of the grid in the dimension that is parallel to the velocities that the grid contains is the pressure grid size plus two. The sizes in those directions are that size because of the imaginary velocities we need to store.

4.4 Discretising the convection term and implementing the grid

We will express the convection terms of equation 8 as ∂_c[28].

$$\begin{aligned}
\partial_c u &= \rho(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z}) \\
\partial_c v &= \rho(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z}) \\
\partial_c w &= \rho(u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z})
\end{aligned} \tag{9}$$

Computers can not exactly calculate the derivatives, so we will have to approximate them to solve equation 8. We will be using Forward Time Central Space(FTCS) for approximating the derivatives[28]. This means we will use first order forward derivatives for time and second order central derivatives for space. We can calculate the convection terms $u\frac{\partial u}{\partial x}$, $v\frac{\partial v}{\partial y}$ and $w\frac{\partial w}{\partial x}$ using equation 6. In the equations below, one might notice that we have used Δx everywhere and have not used Δy or Δz. We have done this because we set $\Delta x = \Delta y = \Delta z$.

$$\begin{aligned}
u\frac{\partial u}{\partial x} =& u_{i,j,k}\frac{u_{i+1,j.k} - u_{i-1,j,k}}{2\Delta x} \\
v\frac{\partial v}{\partial y} =& v_{i,j,k}\frac{v_{i,j+1.k} - v_{i,j-1,k}}{2\Delta x} \\
w\frac{\partial w}{\partial z} =& w_{i,j,k}\frac{w_{i,j.k+1} - w_{i,j,k-1}}{2\Delta x}
\end{aligned}$$

However, calculating the other convection terms is more complex, because we have to estimate a velocity at point (i,j,k) of a grid which is not defined at that grid point. We can however estimate those velocities by taking the average of some nearby velocities, as in figure 4[28].

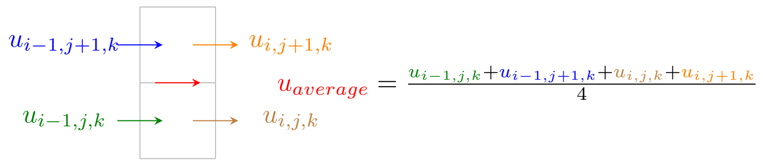

$u_{i-1,j+1,k}$
$u_{i,j+1,k}$
$u_{average} = \frac{u_{i-1,j,k}+u_{i-1,j+1,k}+u_{i,j,k}+u_{i,j+1,k}}{4}$
$u_{i-1,j,k}$
$u_{i,j,k}$

Figure 4: average of velocity

$$v\frac{\partial u}{\partial y} = \frac{v_{i,j-1,k} + v_{i+1,j-1,k} + v_{i,j,k} + v_{i+1,j,k}}{4}\frac{u_{i,j+1,k} - u_{i,j-1,k}}{2\Delta x}$$
$$w\frac{\partial u}{\partial z} = \frac{w_{i,j,k-1} + w_{i+1,j,k-1} + w_{i,j,k} + w_{i+1,j,k}}{4}\frac{u_{i,j,k+1} - u_{i,j,k-1}}{2\Delta x}$$
$$u\frac{\partial v}{\partial x} = \frac{u_{i-1,j,k} + u_{i-1,j+1,k} + u_{i,j,k} + u_{i,j+1,k}}{4}\frac{v_{i+1,j,k} - v_{i-1,j,k}}{2\Delta x}$$
$$w\frac{\partial v}{\partial z} = \frac{w_{i,j,k-1} + w_{i,j+1,k-1} + w_{i,j,k} + w_{i,j+1,k}}{4}\frac{v_{i,j,k+1} - v_{i,j,k-1}}{2\Delta x}$$
$$u\frac{\partial w}{\partial x} = \frac{u_{i-1,j,k} + u_{i-1,j,k+1} + u_{i,j,k} + u_{i,j,k+1}}{4}\frac{w_{i+1,j,k} - w_{i-1,j,k}}{2\Delta x}$$
$$v\frac{\partial w}{\partial y} = \frac{v_{i,j-1,k} + v_{i,j-1,k+1} + v_{i,j,k} + v_{i,j,k+1}}{4}\frac{w_{i,j+1,k} - w_{i,j-1,k}}{2\Delta x}$$

We do not want to use these long equations in the code everywhere, since that would be a lot of work and unclear. Therefore, we will create a function that will calculate the partial derivative and a function that will calcute the avergage velocity at at point. To do this, we first need to create a struct that stores a velocity grid and the dimension of that grid.

```rust
pub struct VelocityGrid{
    grid: Vec<Vec<Vec<f32>>>,
    dimension: usize,
}
```

As one can see, the grid is stored in a vector(Vec means vector). In Rust, a vector is an array that can have different sizes. Furthermore, the dimension of the array is stored as a usize, which basically is an integer with a maximum length that corresponds to the maximum length of an array. The values 0,1 and 2 for dimension denote the x-, y-, and z-direction respectively. To prevent the use of a lot of if-statement, we create a function that can convert the dimension number to an array for which all elements, except the element corresponding to the dimension number are zero.

```rust
//Gives you the unit vector of the dimension with the given
    number.
//x - 0, y - 1, z - 2
fn get_dimension(dimension_number:usize)->[usize; 3]{
    let mut dim = [0,0,0];
    dim[dimension_number]=1;
    return dim;
}
```

Next, we define the sizes of the grid, the velocity grid sizes will depend on the pressure grid sizes, so we will define the pressure grid sizes:

```rust
//Grid size(e.g. number of elements in each dimension)
const PRESSUREGRIDSIZE: [usize; 3] = [10,10,10];//x,y,z
```

Now we can create a function for the second order spatial derivative.

```
fn second_order_spatial_derivative(f:&VelocityGrid, x: usize, y:usize, z:usize, dimension_number:usize) -> f32{
    let dim= get_dimension(dimension_number);
    return (f.grid[x+dim[0]][y+dim[1]][z+dim[2]] - f.grid[x-dim[0]][y-dim[1]][z-dim[2]])/(2.0*GRIDELEMENTSCALE);
}
```

Besides that, we can now also calculate the average velocity, as in figure 4.

```
//This function will retrieve the velocity of an orthogonal grid a grid point of another grid.
fn get_velocity_from_orthogonal_grid(orthogonal_grid: &VelocityGrid, x:usize, y:usize, z:usize, other_grid_dimension:usize) -> f32{
    let dim_to=get_dimension(other_grid_dimension);
    let dim_from=get_dimension(orthogonal_grid.dimension);
    return 0.25*(orthogonal_grid.grid[x-dim_from[0]][y-dim_from[1]][z-dim_from[2]]//Left down
        +orthogonal_grid.grid[x-dim_from[0]+dim_to[0]][y-dim_from[1]+dim_to[1]][z-dim_from[2]+dim_to[2]]//left up
        +orthogonal_grid.grid[x][y][z]//right down
        +orthogonal_grid.grid[x+dim_to[0]][y+dim_to[1]][z+dim_to[2]]);//right up
}
```

Now we use the above two functions to write a function that calculates the convections terms $\partial_c u$, $\partial_c v$ and $\partial_c w$ from equation 9. We will write one function that is able to calculate all these terms:

```
fn convection_term(velocity_field_last_time_step: &VelocityGrid,orthogonal_velocity_field_a: &VelocityGrid, orthogonal_velocity_field_b: &VelocityGrid, x: usize, y: usize, z:usize ) -> f32{// calculate the convection term
     return DENSITY*(velocity_field_last_time_step.grid[x][y][z]*second_order_spatial_derivative(&velocity_field_last_time_step, x, y, z, velocity_field_last_time_step.dimension)
                +get_velocity_from_orthogonal_grid(&orthogonal_velocity_field_a, x, y, z, velocity_field_last_time_step.dimension)*second_order_spatial_derivative(velocity_field_last_time_step, x, y, z, orthogonal_velocity_field_a.dimension)
                +get_velocity_from_orthogonal_grid(&orthogonal_velocity_field_b, x, y, z, velocity_field_last_time_step.dimension)*
```

```
    second_order_spatial_derivative(
    velocity_field_last_time_step, x, y, z,
    orthogonal_velocity_field_b.dimension));
}
```

4.5 Discretising the diffusion term

Like the convection term, the diffusion term contains no temporal terms. Of course it varies, since it terms do vary over time, but we can calculate it using information from only one timestep. We will abreviate the diffusion term of u as $\partial_d u$ and the diffusion terms of v and w in a similar way[28]. The diffusion terms of equation 8 are[28]:

$$\begin{aligned}\partial_d u =& \eta(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2})\\ \partial_d v =& \eta(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2})\\ \partial_d w =& \eta(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2})\end{aligned} \tag{10}$$

We can calculate it using equation 7:

```rust
fn second_order_second_spatial_derivative(f: &VelocityGrid, x:
    usize, y:usize, z:usize, dimension_number:usize) -> f32{
    let dim = get_dimension(dimension_number);
    return(f.grid[x+dim[0]][y+dim[1]][z+dim[2]]-2.0*f.grid[x][
    y][z]+f.grid[x-dim[0]][y-dim[1]][z-dim[2]])/(
    GRIDELEMENTSCALE*GRIDELEMENTSCALE);
}
```

The sum of all three second derivatives is called the laplacian. [28]

```rust
//Laplacian velocity grid
fn laplacian(f: &VelocityGrid, x:usize, y:usize, z:usize)->f32
    {
    return second_order_second_spatial_derivative(f, x, y, z,
    0)+second_order_second_spatial_derivative(f, x, y, z, 1)+
    second_order_second_spatial_derivative(f, x, y, z, 2);
}
```

So we are now able to calculate the diffusion term using:

```rust
let diffusion=VISCOSITY*(laplacian(velocity_field, x, y, z));
```

We will not put this into a function already, because we do not know all the other terms yet. We will however add the constant VISCOSITY:

```rust
const VISCOSITY: f32 = 0.001;//Viscosity in Pa*s.
```

4.6 Discretising the pressure

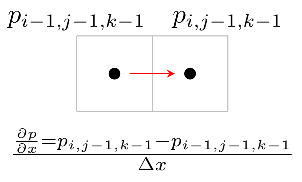

$p_{i-1,j-1,k-1}$
$p_{i,j-1,k-1}$
$\frac{\partial p}{\partial x} = \frac{p_{i,j-1,k-1} - p_{i-1,j-1,k-1}}{\Delta x}$

Figure 5: derivative of pressure

As in figure 5, we can calculate the derivatives $\frac{\partial p}{\partial x}$, $\frac{\partial p}{\partial y}$ and $\frac{\partial p}{\partial z}$

$$\begin{aligned} \frac{\partial p}{\partial x} &= \frac{p_{i,j-1,k-1} - p_{i-1,j-1,k-1}}{\Delta x} \\ \frac{\partial p}{\partial y} &= \frac{p_{i-1,j,k-1} - p_{i-1,j-1,k-1}}{\Delta x} \\ \frac{\partial p}{\partial z} &= \frac{p_{i-1,j-1,k} - p_{i-1,j-1,k-1}}{\Delta x} \end{aligned} \tag{11}$$

4.7 Implicit and explicit methods

Discretising the time can be done by using either an implicit or an explicit methods.[29] In explicit methods, all terms except for one belong to the current timestep and the remanining term that belongs to the next timestep can be calculated easily. [29]In implicit methods, more variables belong to the next timestep. At first, one variable is calculated in the same way as in the explicit methods and the values of the other variables are not changed. [29] Thereafter, the variables are corrected iteratively so that the solution will converge. [29] In the case of the incompressible Navier-Stokes equation, this means the continuity equation will converge to zero. Consequently, in explicit methods the continuity equation will always converge to zero. However, in explicit methods it is just assumed that the continuity equation holds. For the continuity equation to hold the timestep size should be very low in explicit methods. [29] The timestep size can be higher in implicit methods, so less timesteps are needed. [29] However, explicit methods do not need multiple iterations per timestep. Therefore, they compute a timestep faster. [29]We will use an implicit method because we intend to be able to do long simulations and they are better for larger time intervals.[29]

4.8 Discretising the time

Using the formulas for $\partial_d u$ and $\partial_d c$ we can write equation 8 as[28]:

$$\begin{aligned} \rho\frac{\partial u}{\partial t} + \partial_c u &= -\frac{\partial p}{\partial x} + \partial_d u \\ \rho\frac{\partial v}{\partial t} + \partial_c v &= -\frac{\partial p}{\partial y} + \partial_d v \\ \rho\frac{\partial w}{\partial t} + \partial_c w &= -\frac{\partial p}{\partial z} + \partial_d w \end{aligned} \tag{12}$$

We use this notation because it is a lot shorter and, as discussed previously, we will use forward time for the derivatives. Let a^n denote a in the current timestep and a^{n+1} a in the next timestep, where a is a variable. As discussed in chapter 4.7, we will use an implicit method. We will rewrite the Navier Stokes Equations with the spatial pressure derivative of the next timestep. This will allow us to enforce convergence. [28]

$$\begin{aligned} \rho\frac{u^{n+1} - u^n}{\Delta t} + \partial_c u &= -\frac{\partial p^{n+1}}{\partial x} + \partial_d u \\ \rho\frac{v^{n+1} - v^n}{\Delta t} + \partial_c v &= -\frac{\partial p^{n+1}}{\partial y} + \partial_d v \\ \rho\frac{w^{n+1} - w^n}{\Delta t} + \partial_c w &= -\frac{\partial p^{n+1}}{\partial z} + \partial_d w \end{aligned} \tag{13}$$

Unfortunately, we now have two variables for each formula: $\frac{u^{n+1}-u^n}{\Delta t}$ and $\frac{\partial p^{n+1}}{\partial x}$, for x for instance. Therefore, we can not just solve this formula. We can however set the pressure term to the current pressure term and correct the error that is created in that way later. We will denote the velocities obtained in that way $\tilde{u}$, $\tilde{v}$ and $\tilde{w}$[28].

$$\begin{aligned} \rho\frac{\tilde{u} - u^n}{\Delta t} + \partial_c u &= -\frac{\partial p^n}{\partial x} + \partial_d u \\ \rho\frac{\tilde{v} - v^n}{\Delta t} + \partial_c v &= -\frac{\partial p^n}{\partial y} + \partial_d v \\ \rho\frac{\tilde{w} - w^n}{\Delta t} + \partial_c w &= -\frac{\partial p^n}{\partial z} + \partial_d w \end{aligned} \tag{14}$$

These equations contain only one unknown variable(per equation). We can rewrite them to calculate that variable.

$$\begin{aligned} \tilde{u} &= u^n + \frac{\Delta t}{\rho}(-\frac{\partial p^n}{\partial x} - \partial_c u + \partial_d u) \\ \tilde{v} &= v^n + \frac{\Delta t}{\rho}(-\frac{\partial p^n}{\partial y} - \partial_c v + \partial_d v) \\ \tilde{w} &= w^n + \frac{\Delta t}{\rho}(-\frac{\partial p^n}{\partial z} - \partial_c w + \partial_d w) \end{aligned} \tag{15}$$

Now we substract equation 13 from 14.

$$\begin{aligned}
&\rho\frac{\tilde{u}-u^{n+1}}{\Delta t} \\
&= -\frac{p^n_{i,j-1,k-1} - p^{n+1}_{i-1,j-1,k-1} - (p^{n+1}_{i,j-1,k-1} - p^n_{i-1,j-1,k-1})}{\Delta x} \\
&= -\frac{(p^{n+1}_{i-1,j-1,k-1} - p^n_{i-1,j-1,k-1}) - (p^{n+1}_{i,j-1,k-1} - p^n_{i,j-1,k-1})}{\Delta x} \\
&= -\frac{p'_{i-1,j-1,k-1} - p'_{i,j-1,k-1}}{\Delta x} \\
&\rho\frac{\tilde{v}-v^{n+1}}{\Delta t} \\
&= -\frac{p^n_{i-1,j,k-1} - p^{n+1}_{i-1,j-1,k-1} - (p^{n+1}_{i-1,j,k-1} - p^n_{i-1,j-1,k-1})}{\Delta x} \\
&= -\frac{(p^{n+1}_{i-1,j-1,k-1} - p^n_{i-1,j-1,k-1}) - (p^{n+1}_{i-1,j,k-1} - p^n_{i-1,j,k-1})}{\Delta x} \\
&= -\frac{p'_{i-1,j-1,k-1} - p'_{i-1,j,k-1}}{\Delta x} \\
&\rho\frac{\tilde{w}-w^{n+1}}{\Delta t} \\
&= -\frac{p^n_{i-1,j-1,k} - p^{n+1}_{i-1,j-1,k-1} - (p^{n+1}_{i-1,j-1,k} - p^n_{i-1,j-1,k-1})}{\Delta x} \\
&= -\frac{(p^{n+1}_{i-1,j-1,k-1} - p^n_{i-1,j-1,k-1}) - (p^{n+1}_{i-1,j-1,k} - p^n_{i-1,j-1,k})}{\Delta x} \\
&= -\frac{p'_{i-1,j-1,k-1} - p'_{i-1,j-1,k}}{\Delta x}
\end{aligned} \tag{16}$$

Where $p'_{i,j-1,k-1} = p^{n+1}_{i,j-1,k-1} - p^n_{i,j-1,k-1}$ and $p'_{i-1,j-1,k-1} = p^{n+1}_{i-1,j-1,k-1} - p^n_{i-1,j-1,k-1}$. The same is valid for the other directions. The terms denoted by p' are the pressure correction terms. We can calculate the velocity at the next timestep if we know the pressure correction:

$$\begin{aligned}
u^{n+1} &= \tilde{u} - \frac{\Delta t}{\rho\Delta x}(p'_{i,j-1,k-1} - p'_{i-1,j-1,k-1}) \\
v^{n+1} &= \tilde{v} - \frac{\Delta t}{\rho\Delta x}(p'_{i-1,j,k-1} - p'_{i-1,j-1,k-1}) \\
w^{n+1} &= \tilde{w} - \frac{\Delta t}{\rho\Delta x}(p'_{i-1,j-1,k} - p'_{i-1,j-1,k-1})
\end{aligned} \tag{17}$$

However, we still need to know the pressure to solve this equation. The velocity and pressure should satisfy the continuity equation(the upper equation in 8). We will use it to obtain the pressure.[28]

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0$$

We can discretise this equation for the timestep n+1 to obtain a formula for which the sole unknowns are the pressures. The pressure term has the coordinates (i,j,k)

$$\frac{u^{n+1}_{i+1,j+1,k+1} - u^{n+1}_{i,j+1,k+1}}{\Delta x} + \frac{v^{n+1}_{i+1,j+1,k+1} - v^{n+1}_{i+1,j,k+1}}{\Delta x} + \frac{w^{n+1}_{i+1,j+1,k+1} - w^{n+1}_{i+1,j+1,k}}{\Delta x} = 0$$

We can multiply by Δx on both sides.

$$u^{n+1}_{i+1,j+1,k+1} - u^{n+1}_{i,j+1,k+1} + v^{n+1}_{i+1,j+1,k+1} - v^{n+1}_{i+1,j,k+1} + w^{n+1}_{i+1,j+1,k+1} - w^{n+1}_{i+1,j+1,k} = 0$$

Next, we fill in the velocities using equation 17.

$$\begin{aligned}
&\tilde{u}_{i+1,j+1,k+1} - \frac{\Delta t}{\rho \Delta x}(p'_{i+1,j,k} - p'_{i,j,k}) - \left(\tilde{u}_{i,j+1,k+1} - \frac{\Delta t}{\rho \Delta x}(p'_{i,j,k} - p'_{i-1,j,k})\right)\\
&+\tilde{v}_{i+1,j+1,k+1} - \frac{\Delta t}{\rho \Delta x}(p'_{i,j+1,k} - p'_{i,j,k}) - \left(\tilde{v}_{i+1,j,k+1} - \frac{\Delta t}{\rho \Delta x}(p'_{i,j,k} - p'_{i,j-1,k})\right)\\
&+\tilde{w}_{i+1,j+1,k+1} - \frac{\Delta t}{\rho \Delta x}(p'_{i,j,k+1} - p'_{i,j,k}) - \left(\tilde{w}_{i+1,j+1,k} - \frac{\Delta t}{\rho \Delta x}(p'_{i,j,k} - p'_{i,j,k-1})\right)\\
&= 0
\end{aligned}$$

We move all the pressure terms to the right hand side of the equation and keep the velocity terms at the left hand side.

$$\begin{aligned}
&\tilde{u}_{i+1,j+1,k+1} - \tilde{u}_{i,j+1,k+1} + \tilde{v}_{i+1,j+1,k+1} - \tilde{v}_{i+1,j,k+1} + \tilde{w}_{i+1,j+1,k+1} - \tilde{w}_{i+1,j+1,k}\\
&\quad = \frac{\Delta t}{\rho \Delta x}(p'_{i+1,j,k} - p'_{i,j,k} - (p'_{i,j,k} - p'_{i-1,j,k})\\
&\quad + p'_{i,j+1,k} - p'_{i,j,k} - (p'_{i,j,k} - p'_{i,j-1,k})\\
&\quad + p'_{i,j,k+1} - p'_{i,j,k} - (p'_{i,j,k} - p'_{i,j,k-1}))\\
&=> \tilde{u}_{i+1,j+1,k+1} - \tilde{u}_{i,j+1,k+1} + \tilde{v}_{i+1,j+1,k+1} - \tilde{v}_{i+1,j,k+1} + \tilde{w}_{i+1,j+1,k+1} - \tilde{w}_{i+1,j+1,k}\\
&\quad = \frac{\Delta t}{\rho \Delta x}(p'_{i+1,j,k} + p'_{i-1,j,k}\\
&\quad + p'_{i,j+1,k} + p'_{i,j-1,k}\\
&\quad + p'_{i,j,k+1} + p'_{i,j,k-1} - 6p'_{i,j,k})
\end{aligned}$$

We assume that the pressure corrections for p other than (i,j,k) is zero. This will allow us to calculate the pressure correction in that point.[28]

$$\begin{aligned}
\tilde{u}_{i+1,j+1,k+1} - \tilde{u}_{i,j+1,k+1} + \tilde{v}_{i+1,j+1,k+1} - \tilde{v}_{i+1,j,k+1} \\
+ \tilde{w}_{i+1,j+1,k+1} - \tilde{w}_{i+1,j+1,k} = -6\frac{\Delta t}{\rho \Delta x} p'_{i,j,k} \\
=> p'_{i,j,k} = -\frac{\rho \Delta x}{6 \Delta t}(\tilde{u}_{i+1,j+1,k+1} \\
- \tilde{u}_{i,j+1,k+1} + \tilde{v}_{i+1,j+1,k+1} - \tilde{v}_{i+1,j,k+1} \\
+ \tilde{w}_{i+1,j+1,k+1} - \tilde{w}_{i+1,j+1,k})
\end{aligned}$$

The above equation usually reaches convergence too slow, it should therefore be multiplied by a constant, ω_0.

$$\begin{aligned}
p'_{i,j,k} = -\omega_0 \frac{\rho \Delta x}{6 \Delta t}(\tilde{u}_{i+1,j+1,k+1} \\
- \tilde{u}_{i,j+1,k+1} + \tilde{v}_{i+1,j+1,k+1} - \tilde{v}_{i+1,j,k+1} \\
+ \tilde{w}_{i+1,j+1,k+1} - \tilde{w}_{i+1,j+1,k})
\end{aligned}$$

Even though we multiply by ω_0, convergence is often not yet reached after one pressure correction. We therefore need to apply the pressure correction formula until convergence is reached.[28].

Like we did with formulas in other chapters, we will convert the formulas from this chapter into code. We will not rewrite the formulas every time we use them, so we will create functions for them. We start with formula 15:

```rust
fn predict_velocity(provisonal_velocity_field: &mut VelocityGrid, velocity_field_last_time_step: &VelocityGrid, orthogonal_velocity_field_a: &VelocityGrid, orthogonal_velocity_field_b: &VelocityGrid, pressure_grid: [[[f32; PRESSUREGRIDSIZE[2]]; PRESSUREGRIDSIZE[1]]; PRESSUREGRIDSIZE[0]]){
    let dim=get_dimension(provisonal_velocity_field.dimension);
    for x in 1..(PRESSUREGRIDSIZE[0]-dim[0]+1) {
        for y in 1..(PRESSUREGRIDSIZE[1]-dim[1]+1) {
            for z in 1..(PRESSUREGRIDSIZE[2]-dim[2]+1) {
                //convection term
                let convection=DENSITY*(velocity_field_last_time_step.grid[x][y][z]*second_order_spatial_derivative(&velocity_field_last_time_step, x, y, z, velocity_field_last_time_step.dimension)
                +get_velocity_from_orthogonal_grid(&orthogonal_velocity_field_a, x, y, z, velocity_field_last_time_step.dimension)*
    second_order_second_spatial_derivative(
    velocity_field_last_time_step, x, y, z,
    orthogonal_velocity_field_a.dimension)
                +get_velocity_from_orthogonal_grid(&
    orthogonal_velocity_field_b, x, y, z,
    velocity_field_last_time_step.dimension)*
    second_order_second_spatial_derivative(
    velocity_field_last_time_step, x, y, z,
    orthogonal_velocity_field_b.dimension));
                //Diffusion term
                let diffusion=VISCOSITY*(laplacian(
    velocity_field_last_time_step, x, y, z));
                //And finally, the provisional velocity
                provisonal_velocity_field.grid[x][y][z]=
    velocity_field_last_time_step.grid[x][y][z]+TIMESTEPSIZE/
    DENSITY*(-convection+
    first_order_central_spatial_pressure_derivative(
    pressure_grid, x-1, y-1, z-1, velocity_field_last_time_step
    .dimension)+diffusion+EXTERNALFORCE[
    velocity_field_last_time_step.dimension]);
            }
        }
    }

}
```

The function above will create a provisional velocity grid from the last timestep grid. The convection terms is calculated as in section

5 Visualisation

5.1 Drawing

The way we decided to show the movement of the fluids is by drawing arrows in a space of three dimensions. We do this by first loading in the arrow from an OBJ file. and OBJ file is the simplest way of storing three dimensional objects. We define the center of the arrow as (0,0,0). This makes rotation and translation easier. We also make sure that by default the arrow position is up, giving us the vector $\vec{v} = (0, 0, 1)$. We then create multiple instances of that arrow which we multiply with a quaternion that is the cross product of the desired rotation and the neutral rotation and we set the scalar equal to the distance between the endpoints of the vectors with the origin as a starting point. finally we transform the arrow by a vector $\vec{x} = (x, y, z)$, giving us the following transformation: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix}$. Due to the z-component in compute graphics being defined as the distance from the screen and the height as the y-component, we also need to switch the z and y-components. $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ x & y & z & 1 \end{bmatrix}$ The finally multiplication matrix then becomes: $\begin{bmatrix} d & 0 & y & -x \\ 0 & d & x & y \\ -y & -x & d & 0 \\ xd - zy + x & yd - zx - y & 2xy + zd & -x^2 + y^2 + d \end{bmatrix}$ where $d = ||\vec{x} - \vec{v}||$.[32]

5.2 Data capturing

In order to actually show the data, we need to draw the arrows by defining the vertices that make up the arrows. We then multiply it with the transformation matrix from the previous chapter and finally with a standard camera matrix:

6 Conclusion

References

[1] https://resources.pcb.cadence.com/blog/2020-cfd-simulation-types-discretization-approximation-and-algorithms

[2] https://www.manchestercfd.co.uk/post/what-is-discretization

[3] https://www.britannica.com/science/difference-equation

[4] https://courses.engr.illinois.edu/cs357/fa2019/assets/lectures/Lecture8-Sept19.pdf

[5] https://web.iit.edu/sites/web/files/departments/academic-affairs/academic-resource-center/pdfs/Navier_Stokes.pdf

[6] https://www.dive-solutions.de/articles/cfd-methods

[7] https://arxiv.org/pdf/cs/0501021.pdf

[8] https://brilliant.org/wiki/taylor-series-approximation/

[9] https://www.quantstart.com/articles/Derivative-Approximation-via-Finite-Difference-Metho

[10] https://math.stackexchange.com/questions/501735/why-do-we-use-big-oh-in-taylor-series

[11] https://maxwell.ict.griffith.edu.au/jl/Chapter5.pdf

[12] https://pure.tue.nl/ws/files/3372975/696955.pdf

[13] https://www.comsol.com/multiphysics/finite-element-method

[14] https://link.springer.com/chapter/10.1007%2F978-3-319-16874-6_19

[15] http://www.fem.unicamp.br/~phoenics/SITE_PHOENICS/Apostilas/CFD-1_U%20Michigan_Hong/Lecture13.pdf

[16] https://www.quora.com/What-is-the-physics-behind-no-slip-condition-in-fluid-mechanics

[17] https://physics.stackexchange.com/questions/383096/understanding-free-slip-boundary-condition

[18] https://www.tec-science.com/mechanics/gases-and-liquids/derivation-of-the-navier-stokes-equations/

[19] https://www.paramvisions.com/2021/04/what-is-normal-stress-shear-stress.html

[20] http://ingforum.haninge.kth.se/armin/fluid/exer/deriv_navier_stokes.pdf

[21] https://projects.iq.harvard.edu/files/ac274_2015/files/lecture2_3.pdf

[22] https://physics.info/viscosity/

[23] https://www.tec-science.com/thermodynamics/thermodynamic-processes-in-closed-systems/what-is-meant-by-dissipation-of-energy/

[24] https://www.simscale.com/blog/2017/12/turbulence-cfd-analysis/

[25] http://www.nzdl.org/cgi-bin/library?e=d-00000-00---off-0hdl--00-0----0-10-0---0---0direct-10---4-------0-1l--11-en-50---20-
cl=CL1.11&d=HASH011f05bf8734d88d1a080257.12.1&gt=1

[26] https://www.grc.nasa.gov/WWW/k-12/airplane/nseqs.html

[27] https://www.youtube.com/watch?v=Kf_RHzaqFBc

[28] https://www.youtube.com/watch?v=0qtvRjuTihY

[29] https://enterfea.com/implicit-vs-explicit/

[30] https://vulkan-tutorial.com/

[31] http://thevisualroom.com/marker_and_cell_method.html

[32] https://stackoverflow.com/questions/1171849/finding-quaternion-representing-the-rotation-from-one-vector-to-another