Vortexy fluid simulator

Roni Koitermaa *

May 26, 2020

Software documentation

*roninkoi@iki.fi

Contents

1 Introduction

Vortexy is a computational fluid dynamics (CFD) simulation package. It is written in C and uses the finite volume method with the SIMPLE algorithm to calculate flow of incompressible fluids, namely liquids.

The simulator is based on irregular tetrahedral meshes. These meshes can be computed from surfaces using the program Tetgen. The simulator takes a configuration file as input that contains paths to the simulation mesh and boundary conditions in addition to other settings. The state of the system is periodically written to an output file specified in the config. Included is also a renderer that uses OpenGL to visualize results.

2 Background

2.1 Navier-Stokes equations

The Navier-Stokes equations form the basis for all of fluid dynamics. The momentum equation is typically written as [1, p. 59]

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + g, \tag{1}$$

where $\mathbf{u} = (u, v, w)$ is velocity in m s^{-1}, t time in s, ρ density in kg m^{-3}, p pressure in Pa, $\nu = \frac{\mu}{\rho}$ kinematic viscosity in m^2 s^{-1}, g gravity in m s^{-2}.

The continuity equation must be satisfied for incompressible fluids that have no sinks of sources

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

where $\mathbf{u} = (u, v, w)$ is velocity in m s^{-1}.

2.2 Turbulence

A simple way of predicting onset of turbulence is the Reynolds number [1]:

$$\mathrm{Re} = \frac{\mu u L}{\rho} = \frac{uL}{\nu}$$

Turbulence models in simulations include RANS (Reynolds Averaged), LES (Large Eddy) and DNS (Direct).

2.3 Finite volume method

The *finite volume method* (FVM) is based on a simulation mesh with volume elements. This enables evaluation of partial differential equations (PDEs) prevalent in physics. The divergence theorem allows us to convert volume integrals to surface integrals

$$\int_V \nabla \cdot \mathbf{F} \, \mathrm{d}V = \oint_S \mathbf{F} \cdot \mathrm{d}\mathbf{S},$$

so volume terms can be computed from fluxes at element faces.

2.4 SIMPLE algorithm

The SIMPLE algorithm solves the Navier-Stokes equations using an iterative procedure. First the momentum equation is solved, producing a momentum-conserving field u^*. However, this resultant field is not necessarily divergence free, meaning it does not satisfy the continuity equation. The SIMPLE algorithm solves this by calculating a correction u' to the intermediate field by solving the *pressure equation*. The pressure equation is derived from the continuity equation.

The SIMPLE algorithm can be summarized as follows [2, 6]:

1. Set boundary conditions, set u and p
2. Compute gradients ∇u and ∇p
3. Compute mass fluxes j_m, flow rate $\dot{m} = j_m \cdot A$
4. Solve *momentum equation* using velocity guess u^0
5. Solve *pressure correction equation* to get p'
6. Correct pressure $p = p^* + p'$ and velocity $v = v^* + v'$
7. Increment time $t^{n+1} = t^n + \Delta t$
8. Repeat

2.5 Discretization

The momentum equation is written in a form conducive for discretization [2]:

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = -\frac{\nabla p}{\rho} + \frac{\mu}{\rho} \nabla \cdot (\nabla u) + \nabla \cdot (\nabla u)^T + \mathbf{f}_b$$

$$\textbf{transient} + \textbf{convective} = \textbf{diffusive} + \textbf{sources}$$

Continuity equation:

$$\nabla \cdot \mathbf{u} = 0$$

This saddle-point problem can be represented in matrix form as [2, 4]:

$$A\mathbf{u} = \begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{f}_b \\ 0 \end{pmatrix}$$

Discretization for the one-dimensional momentum equation [2]:

$$a_v u_v + \sum_{f \in f_{nb}} a_f u_f = b_v, \tag{3}$$

where a_v refers to a volume element and a_f to a face element coefficient. This can be represented in matrix form as:

$$\begin{pmatrix} a_{00} & a_{01} & 0 & \dots & 0 \\ \vdots & \ddots & & & \vdots \\ a_f & a_f & a_v & a_f & a_f \\ \vdots & & & \ddots & \vdots \\ 0 & \dots & 0 & a_{n(n-1)} & a_{nn} \end{pmatrix} \begin{pmatrix} u_0 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix},$$

where $u_0 \dots u_n$ represents volume element velocities. In the pressure calculation these are referred to as u^*, i.e. momentum conserving. The coefficients a and b are calculated for each volume element and assembled into a matrix. Components X, Y and Z are calculated one after another.

Face fluxes:

$$\phi_f = \max(\dot{m}_f, 0) + \mu \frac{E_f}{d_{vf}} \tag{4}$$

$$\Phi_f = -\max(-\dot{m}_f, 0) - \mu \frac{E_f}{d_{vf}} \tag{5}$$

$$\vec{\psi}_f = -\mu(\nabla \mathbf{u}_f) \cdot \mathbf{T}_f + \dot{m}_f(\mathbf{u}_f^{\mathrm{hr}} - \mathbf{u}_f^{\mathrm{uw}}), \tag{6}$$

where $\mathbf{S}_f = A_f \hat{n}_f = \mathbf{E}_f + \mathbf{T}_f$ and $\dot{m}_f = \rho \mathbf{u}_f \cdot \mathbf{S}_f$. High-resolution model velocity $\mathbf{u}_f^{\mathrm{hr}} = \mathbf{u}_f^{\mathrm{uw}}$.

Volume fluxes:

$$\phi_v = \frac{\rho V}{\Delta t} \tag{7}$$

$$\vec{\psi}_v = \frac{\rho V}{\Delta t} \mathbf{u}_v - \mathbf{f}_b V \tag{8}$$

Coefficients:

$$a_v = \phi_v + \sum_{f \in f_{nb}} \phi_f = \frac{\rho V}{\Delta t} + \sum_{f \in f_{nb}} \left(\max(\dot{m}_f, 0) + \mu \frac{E_f}{d_{vf}} \right) \tag{9}$$

$$a_f = \Phi_f = -\max(-\dot{m}_f, 0) - \mu \frac{E_f}{d_{vf}} \tag{10}$$

$$\mathbf{b}_v = -\vec{\psi}_v - \sum_{f \in f_{nb}} \vec{\psi}_f + \sum_{f \in f_{nb}} \left(\mu (\nabla \mathbf{u}_f)^T \cdot \mathbf{S}_f \right) - V \nabla p_v \tag{11}$$

$$= -\frac{\rho V}{\Delta t} \mathbf{u}_v - \mathbf{f}_b V - \sum_{f \in f_{nb}} \left(-\mu(\nabla \mathbf{u}_f) \cdot \mathbf{T}_f + \dot{m}_f(\mathbf{u}_f - \mathbf{u}_f^{\mathrm{uw}}) \right) + \sum_{f \in f_{nb}} \left(\mu (\nabla \mathbf{u}_f)^T \cdot \mathbf{S}_f \right) - V \nabla p_v$$

$$\mathbf{D}_v = \frac{V}{a_v}$$

Intermediate momentum equation:

$$u_v^* + \sum_{f \in f_{nb}} \frac{a_f}{a_c} \mathbf{u}_f^* = -\frac{V}{a_v} \nabla p_v + \frac{b_v + V \nabla p_v}{a_v} \tag{12}$$

Similarly, the pressure correction equation can be discretized [2]:

$$a_v p'_v + \sum_{f \in f_{nb}} a_f p'_f = b_v \tag{13}$$

With coefficients:

$$a_f = -\rho \mathcal{D}_f \tag{14}$$

$$a_v = \sum_{f \in f_{nb}} \rho \mathcal{D}_f \tag{15}$$

$$b_v = -\sum_{f \in f_{nb}} \dot{m}_f^* = -\sum_{f \in f_{nb}} \rho \mathbf{u}_f^* \cdot \mathbf{S}_f, \tag{16}$$

where $\mathcal{D}_f$ is calculated using the minimum correction approach:

$$\mathcal{D}_f = \frac{d_{vf}^x S_f^x \overline{D_f^x} + d_{vf}^y S_f^y \overline{D_f^y} + d_{vf}^z S_f^z \overline{D_f^z}}{(d_{vf}^x)^2 + (d_{vf}^y)^2 + (d_{vf}^z)^2}.$$

All of these are $n \times n$ diagonally dominant matrices which can be solved using a numerical matrix solver, such as Gauss-Seidel. The off-diagonal coefficients represent neighbouring elements, so they will be mostly zero.

2.6 Gauss-Seidel method

The Gauss-Seidel method is an iterative procedure to solve a linear system of equations. It belongs to a class of successive over-relaxation methods (SOR). The linear system is represented as a diagonally dominant square matrix A and given a vector $\mathbf{b}$ the solution vector $\mathbf{x}$ to the equation

$$A\mathbf{x} = \mathbf{b}$$

is found by an iterative algorithm. GS has better efficiency for large systems than other methods such as matrix inversion.

The iteration for the GS method can be defined as [3, p. 510]:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}}{a_{ii}} \tag{17}$$

In the simulator the GS solver is implemented thusly:

Algorithm 1: Gauss-Seidel method

function GaussSeidel (A, b, m, ε)
Input : $N \times N$ matrix A, N-vector $\mathbf{b}$, maximum iteration count m and convergence criterion ε
Output: Solution N-vector x
for $k \in [0, m[$ **do**
    $\delta = 0$
    for $i \in [0, N[$ **do**
        $s_0 = \frac{1}{A_{ii}}$
        $s_1 = 0$
        for $j \in [0, i-1]$ **do**
            $s_1 \mathrel{+}= A_{ij} x_j$
        end
        $s_2 = 0$
        for $j \in [i+1, N[$ **do**
            $s_2 \mathrel{+}= A_{ij} x_j$
        end
        $s_0 \mathrel{+}= b_i - s_1 - s_2$
        $\delta \mathrel{+}= x_i - s_0$
        $x_i = s_0$
    end
    if $|\delta| < \varepsilon$ **then**
        break
    end
end
return x

2.7 Boundary conditions

Type	Specified quantity
No-slip wall	u
Slip wall	u
Inlet	u or p
Outlet	$\dot{m}$ or p

Table 1: Four main types of boundary conditions

3 Implementation

3.1 Simulation mesh

Triangular/tetrahedral, closed, connected, Delaunay tetrahedralization with spatial Hilbert curve sorting

3.2 Solver

Gauss-Seidel with some sparse matrix optimizations

3.3 Rendering

OpenGL 3.1, GLSL 150

4 Software

4.1 Compilation

The software is compiled using CMake. The `CMakeLists.txt` file is found in the root directory. There are two main ways to configure the simulator: with rendering and without. This is achieved by passing the `-DSIMONLY=ON` flag to CMake. The non-graphical version is linked entirely statically. The graphical version has its `RPATH` set to `./lib/`, so required dynamically linked libraries can be placed there.

The root directory contains shell scripts that can be used for compilation. To build the simulator, renderer and libraries use the script `buildall.sh`. To build the simulator without the renderer use the script `buildsim.h` and set `RENDER_ENABLED` to 0 in `sim.h`.

Dependencies: libc, libm, (OpenGL, GLEW, GLFW)

Building using default options:

```
cmake .
make
```

4.2 Configuration

4.2.1 Simulation config

4.2.2 Fluid config

4.3 Output data

Format:

```
s <sim tick>
o <object id> t <time in s> f <face id> x <face centroid x in m> <y> <z>
v <face velocity x in m/s> <y> <z> p <face pressure in Pa>
e
```

4.4 Examples

4.4.1 Lid-driven cavity

4.4.2 Pipe flow

4.5 Problems

References

[1] D. J. Tritton. (1977). Physical Fluid Dynamics. Springer.

[2] F. Moukalled, L. Mangani & M. Darwish. (2016). The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM and Matlab. Fluid Mechanics and Its Applications Volume 113. Springer International Publishing, Cham. `https://doi.org/10.1007/978-3-319-16874-6`

[3] G. H. Golub & C. F. Van Loan. (1996). Matrix Computations. The John Hopkins University Press.

[4] `https://quickersim.com/tutorial/tutorial-2-numerics-simple-scheme/`

[5] `https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-simple.html`

[6] `https://www.cfd-online.com/Wiki/SIMPLE_algorithm`