Vortexy fluid simulator

Roni Koitermaa *

April 6, 2021

Software documentation

*roninkoi@iki.fi

Contents

1 Introduction

Vortexy is a computational fluid dynamics (CFD) simulation code. 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

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \rightarrow \nabla \cdot \mathbf{u} = 0, \tag{2}$$

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]:

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

Turbulence models in simulations include RANS (Reynolds Averaged) e.g. k-ε, LES (Large Eddy) and DNS (Direct).

3 Methods

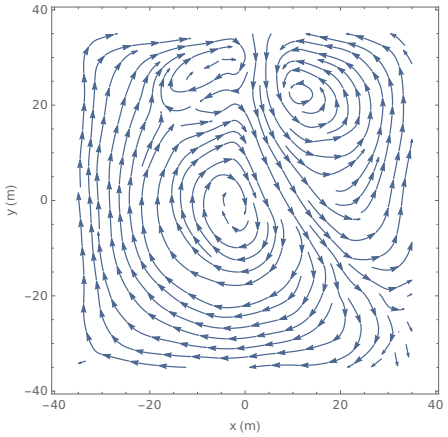

Figure 1: Stream plot example.

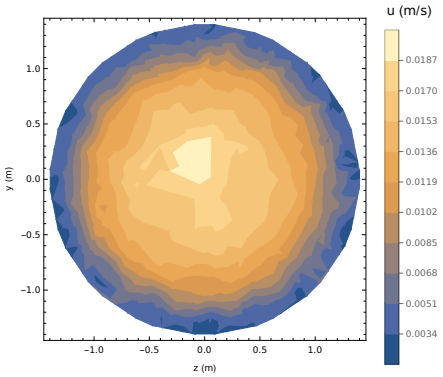

Figure 2: Poiseuille, velocity x-component, cross section.

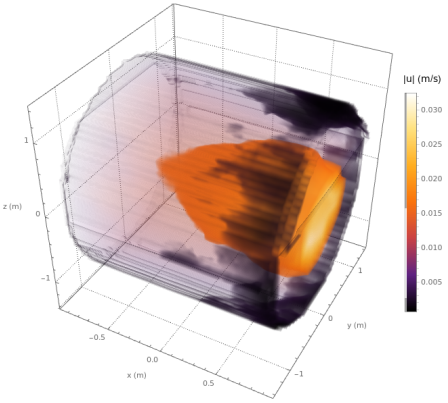

Figure 3: Pipe flow velocity magnitude, 3D view.

3.1 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.

Gradients need to be calculated for both fields. This can be achieved using the Green-Gauss gradient, which can be calculated as

$$\nabla\phi = \frac{1}{V} \int_S \phi \, \mathrm{d}\mathbf{S} = \frac{1}{V} \sum_{f \in F_{\mathrm{nb}}} \phi_f \, \mathbf{S}_f.$$

Face values can be computed from the volume element values by linear interpolation. Second-order gradients are computed by iterative correction.

3.2 SIMPLE algorithm

The *Semi-Implicit Method for Pressure-Linked Equations algorithm* solves the Navier-Stokes equations using an iterative procedure [4]. 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, 4]:

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

Rhie-Chow interpolation is performed as [2]

$$\mathbf{u}_f = \overline{\mathbf{u}_f} - \underbrace{\overline{C_f}\left(\nabla p_f - \overline{\nabla p_f}\right)}_{\text{Pressure correction}} + \underbrace{(1-\alpha_u)\left(\mathbf{u}_f^{(i)} - \overline{\mathbf{u}_f^{(i)}}\right)}_{\text{Relaxation correction}} + \underbrace{\frac{\overline{C_f}}{\overline{C_f^{(t)}}}\left(\mathbf{u}_f^{(t)} - \overline{\mathbf{u}_f^{(t)}}\right)}_{\text{Transient correction}}.$$

3.3 Discretization

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

$$\underbrace{\frac{\partial \mathbf{u}}{\partial t}}_{\text{Transient}} + \underbrace{(\mathbf{u}\cdot\nabla)\mathbf{u}}_{\text{Convection}} = \underbrace{\nu\nabla\cdot(\nabla\mathbf{u})}_{\text{Diffusion}} + \underbrace{\nu\nabla\cdot(\nabla\mathbf{u})^{\mathrm{T}} + \frac{1}{\rho}\nabla p + \mathbf{g}}_{\text{Source}}.$$

The continuity equation for liquids is

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

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

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

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

$$a_V u_V + \sum_{f\in F_{\mathrm{nb}}} a_f u_f = b_V, \tag{3}$$

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

$$\begin{pmatrix} a_{11} & \dots & 0 & \dots & 0 \\ \vdots & \ddots & & & \vdots \\ a_f & a_f & a_V & a_f & a_f \\ \vdots & & & \ddots & \vdots \\ 0 & \dots & 0 & \dots & a_{nn} \end{pmatrix}\begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix},$$

where $u_1 \dots u_n$ 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:

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

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

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

where $\mathbf{S}_f = A_f\hat{n}_f = \mathbf{E}_f + \mathbf{T}_f$ and $\dot{m}_f = \rho\mathbf{u}_f\cdot\mathbf{S}_f$. Volume fluxes:

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

$$\psi_V = \frac{\rho V}{\Delta t}\mathbf{u}_V - \mathbf{f}_b V. \tag{8}$$

Coefficients:

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

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

$$\mathbf{b}_V = -\psi_v - \sum_{f\in F_{\mathrm{nb}}}\psi_f + \sum_{f\in F_{\mathrm{nb}}}\left(\mu(\nabla\mathbf{u}_f)^T\cdot\mathbf{S}_f\right) - V\nabla p_v, \tag{11}$$

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

For interpolation we define:

$$\mathbf{C}_V = \frac{V}{a_V}.$$

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

$$a_V p'_v + \sum_{f\in F_{\mathrm{nb}}} a_f p'_f = b_V \tag{12}$$

with coefficients

$$a_f = -\rho\mathcal{D}_f, \tag{13}$$

$$a_V = \sum_{f\in F_{\mathrm{nb}}}\rho\mathcal{D}_f, \tag{14}$$

$$b_V = -\sum_{f\in F_{\mathrm{nb}}}\dot{m}_f^* = -\sum_{f\in F_{\mathrm{nb}}}\rho\mathbf{u}_f^*\cdot\mathbf{S}_f, \tag{15}$$

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

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

where the overlined values have been interpolated from volume elements to faces.

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.

3.4 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

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

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]

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}}{a_{ii}}. \tag{16}$$

3.5 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

The boundary conditions at each face define the matrix coefficients a_f. For example, a slip wall would have a coefficient $a_f = 0$ at the corresponding column in the matrix A.

3.6 Simulation mesh

The simulation mesh is an irregular, non-orthogonal Delaunay tetrahedralization that is generated from a closed boundary surface. A triangular surface can be used to generate tetrahedra that fill up the bounded volume with constraints such as angle, volume and edge length. Each volume element has four faces and up to four neighbouring volume elements.

4 Software

Vortexy consists of two parts: the simulator and the renderer. The renderer can be used to view output of the simulator in real time and from files. It can also be used to set boundary conditions, but this functionality is limited. The simulator is implemented in the folder `src/phys`. This contains code for finite volume mesh computation (`volume.c`), gradient computation using the Green-Gauss theorem (`gradient.c`) and interpolation (`interpolate.c`). Finally, there is the solver itself that assembles the matrix equations (`solver.c`). The SIMPLE algorithm is implemented in file `sys.c` function `p_sysTick()`. This is the main loop that solves the matrix equations at every time step. The equations are solved iteratively until the desired residual is reached.

4.1 Rendering

OpenGL 3.1, GLSL 150.

4.2 Compilation

The software is compiled using CMake. The `CMakeLists.txt` file is found in the root directory. The simulator can be configured either in graphical or non-graphical mode. Rendering can be disabled by passing the `-DRENDER=OFF` flag to CMake. The non-graphical version is linked entirely statically, so static C development libraries are needed. 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 without the renderer use the script `buildsim.h` and set `RENDER_ENABLED` to 0 in `sim.h`. Building using default options (dependencies: libc, libm, OpenGL, GLEW, GLFW):

```
cmake .
make
```

4.3 Configuration

4.3.1 Simulation config

```
# simulator options
render 0 # use renderer
simulating 1 # start simulation right away
autoquit 1 # quit when not simulating
divhalt 0 # halt simulation upon divergence

fluid data/lid16/lid100.cfg # fluid config file
endt 0.15 # end time / sim duration
maxit 10000 # max solver iterations
epsilon 1.0e-6 # solver target
dtmaxit 15 # max time-step iterations
residual 5.0e0 # solution residual
transient 0 # include Chie-Chow transient term
convsch 0 # convection scheme, 0 = upwind, 1 = second-order upwind
gradit 2 # gradient iterations, if 0 -> first-order
relaxm 1.0 # relax mass flow

# IO options
file data/lid16/out100.dat # input/output file
printitn 1 # print time-step iteration number
outputting 1 # writing to output file?
outputt 0 # outputting time data
outputf 1 # frequency of output (1/cycle)
inputf 20 # frequency of input (1/cycle)
inputram 0 # bytes to load into memory, 0 for direct file access

# rendering options
```

```
# f  v  t  l  w  m
# 64 32 16 8  4  2
rmode 10 # rendering mode (see above)
rz -4.0 # camera z
rs 0.3 # render scale
bgcol 1.0 1.0 1.0 # background color

vs 1.0 # velocity scale (m/s)
ps 100000.0 # pressure scale (Pa)

end # optional
```

4.3.2 Fluid config

```
mu 2.0e-1 # viscosity
rho 1.0e-1 # density

dt 0.001 # time step

vr 0.7 # velocity urf
pr 0.3 # pressure under-relaxation factor

mesh data/lid16/plane_16x16_s2.1.mesh

f 0.0 0.0 0.0

bp 1.0e3 # base pressure, p = bp + ip

ebc 1 # boundary condition for all edges

# bc <f id> <bc> = boundary condition
# 0 = open, 1 = no-slip wall, 2 = slip wall
# 3 = velocity inlet, 4 = pressure inlet
# 5 = mass flow outlet, 6 = pressure outlet
# iv <f id> <v> = initial velocity, ip <f id> <p> = initial pressure
# cv <f id> <v> = constant velocity, cp <f id> <p> = constant pressure

bc 244 3 # velocity inlet for face id 244
cv 244 10.0e2
iv 244 10.0e2

end # optional
```

4.4 Output data

```
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
```

References

[1] D. J. Tritton. (1977). Physical Fluid Dynamics. Oxford University Press.

[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] S. V. Patankar. (1980). Numerical Heat Transfer and Fluid Flow. Hemisphere Publishing Corporation.