INSTITUTE OF THERMOMECHANICS
CZECH ACADEMY OF SCIENCES

NARLabs
National Applied Research Laboratories
National Center for
High-performance Computing

Institute of Thermomechanics, Czech Academy of Sciences

Department of Waves in Solids

Air flow and spreading of pollution in an inhabited area

REPORT FROM THE STUDENT INTERNSHIP

AUTHOR	**Ing. Tomáš Hlavatý**
PROGRAMME	**NARLabs student internship programme**
YEAR	**2023**

Acknowledgements

I would like to honestly thank the NARLabs for allowing, and the supervisors for the guidence thoughout the whole internship.

Contents

1 Introduction

The present work arose as the part of the authors student internship in Taiwan National Applied Research Laboratories (NARLabs). The report was written in National Center for High-performance Computing (NCHC).

The main contribution of the present work lies in the development of the OpenFOAM computational framework compatible with the NCHC computational servers. As the tutorial case of the interest, the spreading of a pollution in an inhabited area has been chosen.

The work is structured as follows, (i) mathematical model is described together with the used assumptions (Section 2), (ii) computational framework (environment) is introduced with the usage guideline (Section 3), and (iii) the results are briefly discussed (Section 4).

2 Mathematical model

In the present work, the air-flow in the inhabited area is investigated together with the transport of the pollution emerging on the main road. The chosen simulation approach is a segregated one: (i) a steady state velocity field is pre-calculated using Navier-Stokes equations, and (ii) it is used in scalar transport equation to simulate advancement of the pollution.

The present section describing the used mathematical model is structured as follows: (i) a model geometry and a computational mesh generation are presented, (ii) used boundary conditions are summarized and (iii) the governing equations are briefly outlined.

2.1 Model geometry generation and boundary conditions

As stated, the air flow and the spread of the pollution in the inhabited area is of an interest. A studied part of the city is $L_\mathrm{l} = 500\,\mathrm{m}$ long, $L_\mathrm{w} = 500\,\mathrm{m}$ wide, and the highest building is $L_\mathrm{h} = 105\,\mathrm{m}$ tall. The available *stl* file together with its dimensions and the orientation in the chosen Cartesian coordinate system is depicted in Figure 1.

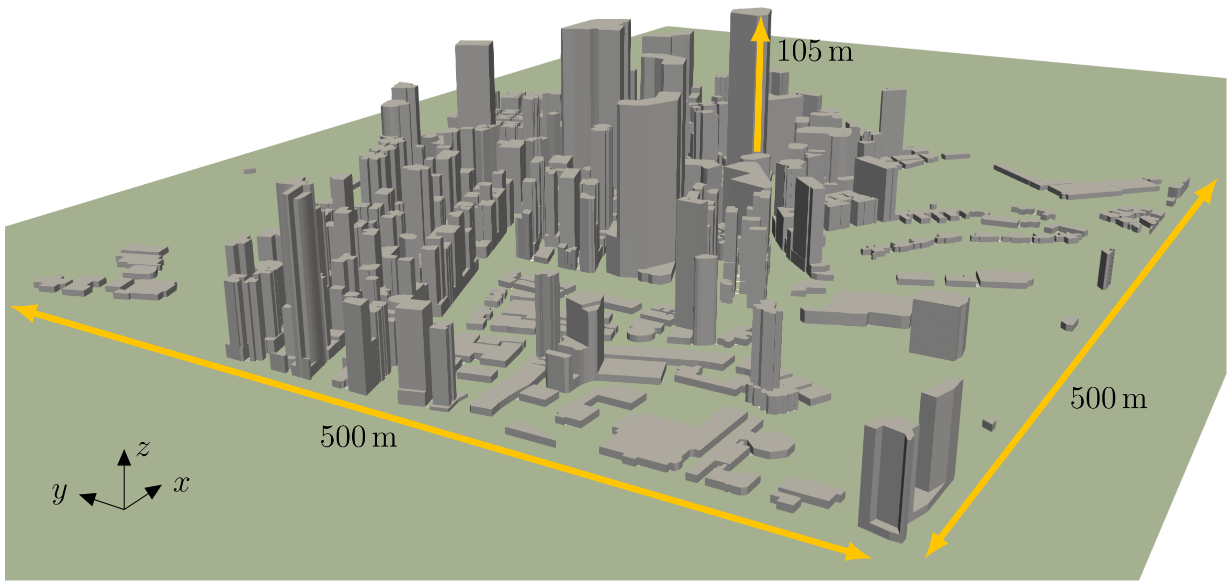
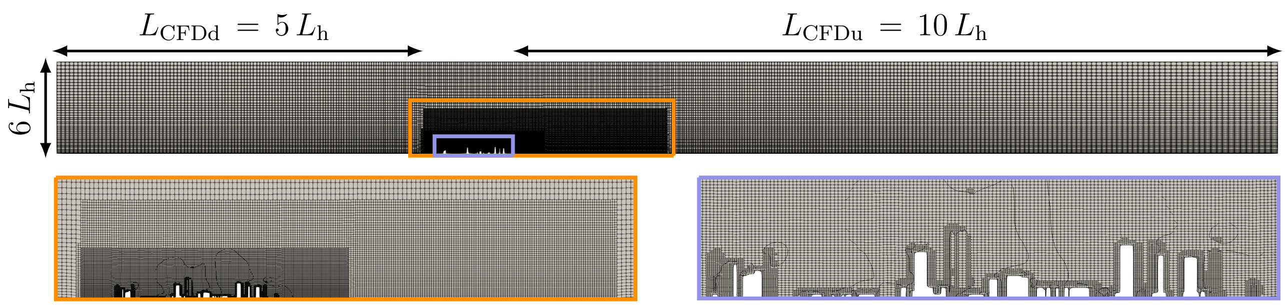


Figure 1: Studied city geometry (used *stl* file) with its dimensions and chosen Cartesian coordinate system.

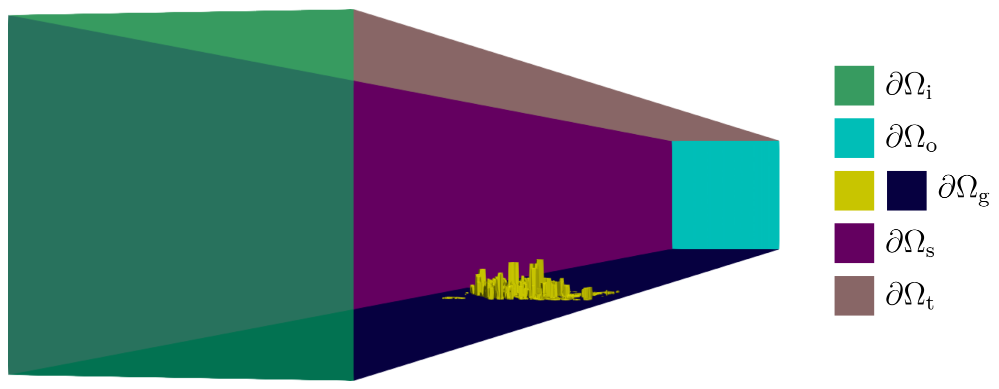
Computational mesh Computational mesh was prepared using the OpenFOAM `blockMesh` and `snappyHexMesh` utilities. The computational domain dimensions were determined based on the review by Pantusheva et al. [1]. In particular, we introduce both upstream, and downstream buffers in front of, and behind the city, which are in order $L_\mathrm{CFDu} = 5\,L_\mathrm{h}$, and $L_\mathrm{CFDd} = 10\,L_\mathrm{h}$ long. Furthermore, the computational domain is $L_\mathrm{CFDh} = 6\,L_\mathrm{h}$ high. The y-normal slices cut in the middle of the used computational mesh are shown in Figure 2. Note the grading of the computational mesh in the z-direction, and its refinement in the vicinity of the city.



$L_{\mathrm{CFDd}} = 5\,L_{\mathrm{h}}$
$L_{\mathrm{CFDu}} = 10\,L_{\mathrm{h}}$
$6\,L_{\mathrm{h}}$

Figure 2: y-normal slices through the half of the computational mesh.

Boundary conditions The model behavior in the computational domain (Ω) is described by partial differential equations outlined in the following subsection 2.2 (Governing equations). However, these need to be supplied with the consistent boundary conditions on the Ω boundary, $\partial\Omega$. As depicted in Figure 3, the computational domain boundary is split into (i) *inlet* ($\partial\Omega_{\mathrm{i}}$), (ii) *outlet* ($\partial\Omega_{\mathrm{o}}$), (iii) *ground* ($\partial\Omega_{\mathrm{g}}$), (iv) *sides* ($\partial\Omega_{\mathrm{s}}$), and (v) *top* ($\partial\Omega_{\mathrm{t}}$).



$\partial\Omega_{\mathrm{i}}$
$\partial\Omega_{\mathrm{o}}$
$\partial\Omega_{\mathrm{g}}$
$\partial\Omega_{\mathrm{s}}$
$\partial\Omega_{\mathrm{t}}$

Figure 3: Boundaries of the computational domain.

Used boundary conditions are utilized from [1, 2] and summarized in Table 1. Starting with flow variables, uniform Dirichlet (*fixedValue*), and OpenFOAM inletOutlet boundary conditions are prescribed for velocity ($\boldsymbol{u}$) at Ω_{i}, and Ω_{o}, respectively. At Ω_{g}, we standardly assume *noSlip* boundary condition, and finally the sides of the geometry (Ω_{s} and Ω_{t}) are treated as the symmetry (*slip*). This setting is supplemented with OpenFOAM *totalPressure* boundary condition for pressure (p) at Ω_{o}, and zeroGradient (*zeroGrad*) elsewhere.

As described in the later subsection, $k-\varepsilon$ turbulence model is used to estimate unresolved eddy dissipation. The turbulence model requires specification of the boundary conditions for turbulent kinetic energy (k), rate of dissipation of the turbulent kinetic energy (ε), and the turbulent viscosity (ν_{t}). At Ω_{i}, we use OpenFOAM turbulentIntensityKineticEnergyInlet (*turbIntKinEngInlet*), and turbulentMixingLengthDissipationRateInlet (*turbMixDissRateInlet*) boundary conditions, which calculate the inlet k and ε, respectively, based on specified value of the inlet turbulence intensity I_{i}. At the Ω_{g}, standard wall functions are utilized *kqRWallFunction*, *epsWallFunction*, and *nutkWallFunction*, in order for k, ε, and ν_{t}. And finally, *zeroGrad* boundary condition is prescribed elsewhere.

Scalar transport equation describing the spreading of the pollution y_{P} is supplied with the *fixedValue* boundary condition at Ω_{g}, defining the source of the pollution on the main road, and *zeroGrad* boundary condition elsewhere.

Table 1: Used boundary conditions.

variable	Ω_i	Ω_o	Ω_g	Ω_s	Ω_t
$\boldsymbol{u}$	fixedValue	inletOutlet	noSlip	slip	slip
p	zeroGrad	totalPressure	zeroGrad	zeroGrad	zeroGrad
k	turbIntKinEngInlet	zeroGrad	kqRWallFunction	zeroGrad	zeroGrad
ε	turbMixDissRateInlet	zeroGrad	epsWallFunction	zeroGrad	zeroGrad
ν_t	fixedValue	zeroGrad	nutkWallFunction	zeroGrad	zeroGrad
y_P	zeroGrad	zeroGrad	fixedValue	zeroGrad	zeroGrad

2.2 Governing equations

The description of the chosen governing equations is split into two parts, (i) air-flow governing equations, and (ii) the pollution scalar transport.

Flow governing equations Steady state isothermal incompressible flow of the Newtonian gas is assumed. Adopting Boussinesq hypothesis, and noting the mean time velocity, $\boldsymbol{u} = (u_x, u_y, u_z)^\intercal$, and the mean time turbulent pressure, $\tilde{p}$, the flow can be described by Reynolds-averaged Navier-Stokes (RANS) equations in a form,

$$\nabla \cdot (\boldsymbol{u}\boldsymbol{u}^\intercal) - \nabla \cdot [(\nu + \nu_\mathrm{t})(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^\intercal)] = -\nabla \tilde{p}\,, \tag{1}$$

$$\nabla \cdot (\boldsymbol{u}) = 0 \tag{2}$$

where ν is the gas kinematic viscosity, and ν_t is the turbulent eddy viscosity.

In general, there exist numerous various approaches of ν_t estimation. In the present work, the OpenFOAM variant of $k-\varepsilon$ turbulence model is used to calculate the turbulent eddy viscosity as,

$$\nu_\mathrm{t} = \frac{k}{\varepsilon}\,, \tag{3}$$

where k, and ε are the turbulence kinetic energy, and its rate of the dissipation, respectively. For more information on the calculation of the k and ε, see e.g. [3, 4].

Pollution transport governing equation Assuming constant mass density, the transient dispersion of the polluting gas in the carrier gas can be estimated employing standard scalar variable transport equation in a form,

$$\frac{\partial (y_\mathrm{P})}{\partial t} + \nabla \cdot (\boldsymbol{u}\, y_\mathrm{P}) - \nabla \cdot ((D\,+\,D_\mathrm{t})\nabla y_\mathrm{P}) = 0\,, \tag{4}$$

where $\boldsymbol{u}$ is the mean-time velocity obtain from solution of (1,2), y_P is the molar fraction of the polluting gas, t is time, and D and D_t are in order molar and turbulent diffusivities of the polluting gas. Remark that no source of the pollution is assumed on the right hand side of the equation (4), as the source of the pollution is ensured by proper application of the boundary conditions.

Furthermore, the turbulent diffusivity, D_t, is calculated using Schmidt number (Sc) defined as,

$$\mathrm{Sc} = \frac{\nu + \nu_\mathrm{t}}{D + D_t}\,, \tag{5}$$

where $\nu + \nu_\mathrm{t}$ are the gas kinematic and the turbulent viscosities, respectively. Note that a standard assumption of the constant Schmidt number is adopted [5], and ν_t is estimated from from (3).

2.3 Model parameters

The above described mathematical model has several tunable parameters that can be split into two groups, (i) thermophysical parameters of the flowing gas, and (ii) boundary conditions

The flow of the air at the common temperature of the $T = 25\,°\mathrm{C}$ has been assumed. The value of the gas kinematic viscosity has been calculated from Sutherland equation [6] as $\nu = 1.53 \cdot 10^{-5}\,\mathrm{m}^2\,\mathrm{s}^{-1}$, and the value of the pollution molar diffusivity from Fuller equation [7] as $D = 1.72 \cdot 10^{-5}\,\mathrm{m}^2\,\mathrm{s}^{-1}$. As in [5], the Schmidt number has been assumed constant and equal $\mathrm{Sc} = 0.9$.

To complete the description of the prepared model, it remains to specify the inlet values of the velocity, and the turbulence intensity, and the value of the pollution molar fraction on the ground of the main road.finish

3 Computational environment

NCHC servers use Singularity container platform [8] and Slurm workload manager [9] to run user-defined tasks. Thus,

(i) custom singularity container, which includes necessary packages installed and compiled inside, is prepared, and

(ii) Slurm task (which runs within the container) is prepared and run at the computational server.

3.1 Preparation of the singularity container

Assuming you have super-user permission and singularity installed on local (see step-by-step guide in [10]), a preparation of the singularity container image from docker ubuntu:latest release can be done as follows:

1. Navigate outside the home directory and work here, e.g.:

   ```
   mkdir /tmp/
   mkdir /tmp/test
   cd /tmp/test/
   ```

2. New container (`./ubuntu`) can be built from ubuntu docker repository using:

   ```
   sudo singularity build --sandbox ./ubuntu docker://ubuntu:latest
   ```

   Note: `--sandbox` flag allows to write into container later.

Shell inside container can be opened using:

```
sudo singularity shell ./ubuntu --writable
```

where `--writable` flag again allows to write into container and install packages here.

Openfoam.org/v10 and other used packages can be installed inside the container as:

```
apt update
apt install python3 python3-pip wget vim software-properties-common
    python3-tk    pip3 install matplotlib
sh -c "wget -O - https://dl.openfoam.org/gpg.key >
    /etc/apt/trusted.gpg.d/openfoam.asc"
add-apt-repository http://dl.openfoam.org/ubuntu
apt update
apt install openfoam10
```

Compilation of the custom solver inside container is done as follows:

1. Source OpenFOAM in the container shell:

   ```
   . /opt/openfoam10/etc/bashrc
   ```

2. Navigate to solver folder, e.g.:

   ```
   cd /tmp/pollutionFoam
   ```

3. Compile solver executing:

   ```
   wmake
   ```

When everything is installed, the `.sif` container file can be built from prepared `/tmp/test/ubuntu` directory using:

```
sudo singularity build /tmp/test/ubuntu.sif /tmp/test/ubuntu/
```

Following the above listed guideline, singularity container image `ubuntu.sif` is created. This can be uploaded to NCHC servers and used as described in following subsection.

3.2 Preparation of Slurm control script and running the tasks

The slurm control script uses bash and the example (`taiw3SlurmRunV2.sh`) can look as follows:

```
#!/bin/bash
#SBATCH -J testOF10                    # name of the task
#SBATCH -p ct56                        # partition (see sinfo)
#SBATCH -t 12:00:00                    # needed time to run
#SBATCH --account=GOV109092            # computing resource wallet ID
#SBATCH --ntasks=12                    # (-n) Number of MPI processes

# -- Slurm script to run simulations on Taiwania 3

# -- load singularity
module load libs/singularity

# -- run python control
cd ../00_pyCodes
python3 -u caseManagerV1.py
```

The created Slurm job can be added to the front by:

```
sbatch taiw3SlurmRunV2.sh
```

Note that the setup and the control of the OpenFOAM simulation is ensured by the custom developed OpenFOAMCase class described in the next subsection.

3.3 Developed OpenFOAMCase python class

To easily control the OpenFOAM simulations, the custom python class OpenFOAMCase has been developed. This class allows to copy OpenFOAM base folder to specified directory, easily modify dictionaries and run the OpenFOAM utilities.

The class is stored in `01_codes/OF_cases/00_pyCodes/OF_caseClass.py`, and in the same folder also the python control script that is run by Slurm job can be found (`caseManagerV1.py`), whose shorten example version is given also here:

```
# -- Python script to create and manage OpenFoam cases

# -- imports
from OF_caseClass import OpenFOAMCase

# -- paths
bsCsDir = "../01_baseCaseV2"        # base case directory
finCsDir = "../ZZ_cases/testRunV1"  # where to copy base directory and work
singularityFl = "~/Singularity/ubuntu3.sif"       # singularity file

# -- dictionaries parameters
DOMAIN_SIZE_X = 500*11
DOMAIN_SIZE_Y = 270*2
REF_LEVEL_VELKY_BOX = 2
endTime1 = 500
pRelax1 = 0.005
defDivSch = "bounded Gauss SFCD"

# -- create OpenFOAMCase object and copy base directory to working directory
testCase = OpenFOAMCase()
testCase.loadOFCaseFromBaseCase(bsCsDir)
testCase.changeOFCaseDir(finCsDir)
testCase.copyBaseCase()

# -- replace the parameters in the dictionaries
testCase.replace(
    [
        [
            "system/blockMeshDict",                     # in file
            ["DOMAIN_SIZE_X", "DOMAIN_SIZE_Y"],         # list of what to replace
            [str(DOMAIN_SIZE_X), str(DOMAIN_SIZE_Y)]  # list of by to replace
        ],
        [
            "system/snappyHexMeshDict",
            ["REF_LEVEL_VELKY_BOX"],
            [str(REF_LEVEL_VELKY_BOX)]
        ],
    ]
)

# -- set the parameters in the dictionaries
testCase.setParameter(
    [
        [
            "system/fvSolution",                        # in file
            "p ",                                       # set parameter
            str(pRelax1),                               # to value
            "fields"                                    # in subdictionary
        ],
        ["system/fvSchemes", "default", defDivSch, "divSchemes"],
        ["system/controlDict", "endTime", str(endTime1), ""],
    ]
)

# -- run command in the working directory using prepared singularity container
testCase.runCommands (
    [
        # -- change permissions
        "chmod 775 -R ./*",
        # -- run ''geometry'' bash script in the working directory
        "singularity exec -H %s/%s %s bash ./geometry"
            % (hk1.whereIStart, hk1.dir, singularityFl),
        # -- run ''simulationFlow'' script in the working directory
        "singularity exec -H %s/%s %s bash ./simulationFlow"
            % (hk1.whereIStart, hk1.dir, singularityFl),
    ]
)

... to be continued with the pollution simulation
```

The above python class simplifies the control of the OpenFOAM simulations, the only thing that remains unanswered is the form of the `geometry`, `simulationFlow`, and other bash scripts. The example is given below:

```
#!/bin/sh

# -- script to create geometry of the case

# -- run from here
cd ${0%/*} || exit 1

# -- source OpenFOAM installed in the singularity container
. /opt/openfoam10/etc/bashrc
. $WM_PROJECT_DIR/bin/tools/RunFunctions
```

```
# -- copy the initial guess and boundary conditions
cp -r 0.org 0

# -- blockMesh
runApplication blockMesh

# -- creation of the .OpenFOAM file for paraview
paraFoam -touch

# -- decomposePar and snappyHexMesh
runApplication decomposePar -copyZero
runParallel snappyHexMesh -overwrite
```

This tutorial demonstrates the usage of the developed framework to control OpenFOAM simulations on the NCHC clusters. Note that all the control scripts are part of the public repository `https://github.com/hlavatytomas/NARLabsInternship.git`.

4 Results

The resulting stream-wise velocity component contours on the y-normal slice, and the pressure contours on the ground computational domain boundary ($\partial\Omega_g$), are presented in Figure 4.

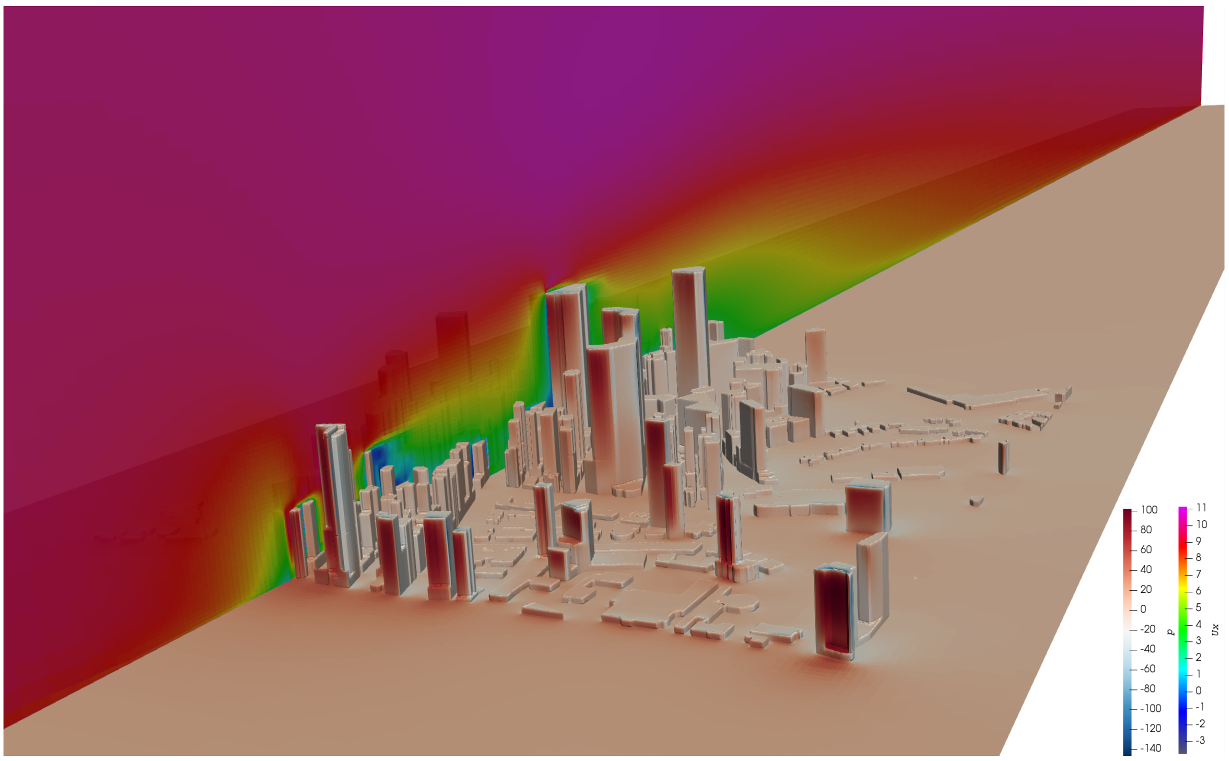


Figure 4: Stream-vise velocity component (u_x), and pressure (p) contours on y-normal slice, and the ground boundary, respectively.

5 Conclusions

And conclusion here

Nomenclature

D	molar diffusivity, $\mathrm{m^2\,s^{-1}}$
D_t	turbulent molar diffusivity, $\mathrm{m^2\,s^{-1}}$
k	turbulent kinetic energy, $\mathrm{m^2\,s^{-2}}$
L	length, m
p	pressure, Pa
$\boldsymbol{u} = (u_x, u_y, u_z)^\intercal$	velocity, $(\mathrm{m\,s^{-1}}, \mathrm{m\,s^{-1}}, \mathrm{m\,s^{-1}})$
Sc	Schmidt number, 1
T	temperature, 1
y_P	molar fraction of the polluting gas, 1

Greek letters

ε	turbulent kinetic energy dissipation rate, $\mathrm{m^2\,s^{-3}}$
ν	kinematic viscosity, $\mathrm{m^2\,s^{-1}}$
ν_t	turbulent kinematic viscosity, $\mathrm{m^2\,s^{-1}}$
$\Omega = (x, y, z)^\intercal$	computational domain, $(\mathrm{m}, \mathrm{m}, \mathrm{m})$
$\partial\Omega$	computational domain boundary, $(\mathrm{m}, \mathrm{m}, \mathrm{m})$

Subscripts and superscripts

d	downwind
g	ground
h	height
s	sides
o	outlet
t	top
u	upwind

References

[1] Mariya Pantusheva et al. “Air Pollution Dispersion Modelling in Urban Environment Using CFD: A Systematic Review”. In: *Atmosphere* 13.10 (2022). DOI: `10.3390/atmos13101640`.

[2] Rakesh Kadaverugu et al. “Improving accuracy in simulation of urban wind flows by dynamic downscaling WRF with OpenFOAM”. In: *Urban Climate* 38 (2021), p. 100912. DOI: `https://doi.org/10.1016/j.uclim.2021.100912`.

[3] F. Moukalled, M. Darwish and L. Mangani. *The finite volume method in computational fluid dynamics: an advanced introduction with OpenFOAM and Matlab*. 1st ed. Berlin, Germany: Springer-Verlag, 2016. ISBN: 978-3-319-16874-6.

[4] B.E. Launder and D.B. Spalding. “The numerical computation of turbulent flows”. In: *Comp. Meth. App. Mech. Eng.* 3 (1974), pp. 269–289.

[5] Jong-Jin Baik, Jae-Jin Kim and Harindra J. S. Fernando. “A CFD Model for Simulating Urban Flow and Dispersion”. In: *Journal of Applied Meteorology* 42.11 (2003), pp. 1636–1648. DOI: `10.1175/1520-0450(2003)042<1636:ACMFSU>2.0.CO;2`.

[6] W. Sutherland. “The viscosity of gases and molecular force”. In: *Phil. Mag. S. 5* 36 (1893), pp. 507–531.

[7] E. N. Fuller, P. D. Schettler and J. C. Giddings. “New method for prediction of binary gas-phase diffusion coefficients”. In: *Ind. Eng. Chem. Res.* 58.5 (1966), pp. 18–27.

[8] SingularityCE Project Contributors. *SingularityCE User Guide*. `https://docs.sylabs.io/guides/main/user-guide.pdf`. Accessed: 20-2-2023.

[9] Andy B. Yoo, Morris A. Jette and Mark Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: *Job Scheduling Strategies for Parallel Processing*. 2003.

[10] SingularityCE Project Contributors. *SingularityCE Installation*. `https://docs.sylabs.io/guides/3.0/user-guide/installation.html`. Accessed: 20-2-2023.