



**An Extended Report on
'Incorporating Three Weeks Open Source Software Lab
Module in CFD and Fluids Courses'**

Compiled by

Zahra Mansouri

Supervised By

Dr. R. Panneer Selvam

July 2021

**Department of Civil Engineering, College of Engineering
University of Arkansas**

2021 ASEE MIDWEST SECTION VIRTUAL CONFERENCE



**Organized By
American Society of Engineering Education**

Teaching Modeling Turbulent Flow Around Building Using LES Turbulence Method and Open-source Software OpenFOAM

Zahra Mansouri, Sumit Verma, and R. Panneer Selvam

Department of Civil Engineering, BELL 4190 University of Arkansas, Fayetteville, AR 72701, USA

Abstract:

In our earlier work (<https://github.com/rpsuark/ASEE21-OpenFOAM-Introduction>), it was reasoned that open-source software *OpenFOAM* would be a cost effective and more accessible alternative for teaching *Computational Fluid Dynamics* (CFD) than commercial software for teaching and learning purposes. It models wind flow around a building for smooth flow, whereas extreme winds, which tend to be irregular, can cause various structural failures of buildings. This kind of irregular wind flows are called turbulent flows. To avoid these devastating structural damages, engineers should consider turbulent wind effects while evaluating wind loads from CFD model. Not only that work models wind flow around a building without considering wind turbulence, but also the important aspects that makes a numerical model closer to real world scenario are lacking such as the correct wind velocity profile. The wind velocity near the ground should be approximated by the logarithmic velocity profile, which is zero close to the ground due to the friction and increases with height. Similarly, to capture high velocity gradient near the ground, the standard wall function can be implemented at walls instead of mesh refinement. Therefore, it is important to incorporate *Large Eddy Simulation* (LES) as turbulence model, log profile for inlet velocity, and wall function in the CFD model to get a better estimate of wind loads to design buildings better and safer. Thus, in this contribution, an additional three-week class module is provided for ‘CFD for Wind Engineering’ class which includes hands-on material on modeling turbulent wind flow around building using LES, log profile and wall function. The current study’s case files and the extended report are provided at <https://github.com/rpsuark/ASEE21-OpenFOAM-LES>.

Keywords

Computational Fluid Dynamics, OpenFOAM, Large Eddy Simulation, Wall Function, Logarithmic Velocity Profile.

1. Introduction

1.1. Suggested Course Module in CFD for Industrial Application Purposes

Computational Fluid Dynamics (CFD) and wind engineering are some of the most important courses taught in universities which provides exposure to the students about the potential catastrophic damages that can be brought by severe winds. These courses are basically driven with the motive to train and educate students to compute the wind velocities and pressures on building so that with better estimate of wind loads, the structures and buildings could be designed better. At the University of Arkansas, CFD and *Computational Wind Engineering* (i.e., CWE 563) are taught by introducing a lab sessions consisting of real-world problem solving using research codes adapted for teaching purposes. Such method of course instruction and delivery were found to be

greatly beneficial to students as it introduced the theoretical concepts and provided decent exposure to real-world problem solving. However, due to complexity, developing new code is impossible for every engineer in the industry, hence using software is more likely. Due to commercial software are so expensive, it was reasoned by Verma et al. [1] also that open-source software *OpenFOAM* would be a cost effective and more accessible alternative than commercial software for teaching learning purposes. Verma et al. [1] proposed a similar instruction methodology by incorporating open-source software programs *OpenFOAM* and *ParaView*. However, as the contribution was primarily meant for introduction of *OpenFOAM* and *ParaView* for teaching CFD and Fluid courses, so, the problem considered in [1] was relatively simple and lacked several important aspects required to model a realistic wind flow case around a building that adequately resembles a real-world wind flow scenario. For instance, the case file from Verma et. al. [1] is limited to laminar flow; laminar flow is smooth and regular flow and occurs in a very slow-moving fluid. As the flow speed increases, the flow tends to be more unstable and irregular. Most flows in nature are categorized in this type of flow which is also called turbulent flow.

The disastrous failure of structures caused by strong winds is mainly due to underestimation of peak wind pressures while designing the building components. Strong winds are highly turbulent and so if turbulence is not well accounted for in the numerical model then, the computed wind loads (in the form of pressure coefficients) would not be adequate for building design purposes are more likely to fail during severe storms/winds events. For instance, the maximum peak pressure coefficient obtained from ASCE 7-16 is -3 for a low-rise building. However, field measurements (of turbulent winds in nature) have reported that the maximum peak pressure coefficients on a low-rise building can be even lower than -8. Thus, it is necessary to consider correct peak pressures in building design. As conducting field measurements for designing purposes is expensive and time-consuming, CFD can be an economical tool for engineers to estimate accurate wind pressures on buildings.

Thus, in this work, a three-weeks lab component of course module is proposed as an extension to 2 weeks module proposed by [1] by incorporating the important aspects (i.e., *Large Eddy Simulation* (LES), wall function and log profile explained in detail in following sections), which are important to capture the real physics of wind flow around a building. In the proposed module, first a brief introduction to LES, log profile and wall function is included followed by the implementation of LES for wind flow around the building and then implementation of wall function in the wall boundaries. Finally, the procedure of implementing logarithmic velocity profile at inlet is discussed after which some flow visualizations are included. The relevant *OpenFOAM* implementation case files including an extended report and description to obtain various visualizations included in this work are provided as hands-on material at [2].

1.2. Turbulence

The plot shown in Fig. 1. is for a turbulent flow in which the velocity is recorded in time at a particular point in space. In Fig.1, the instantaneous velocity is given by $u = U + u'$, where U is the time-averaged velocity and u' is the velocity fluctuation over time. As the variation is time does not follow or repeat in a periodic manner, so, such flows are random and chaotic and are called turbulent flows.

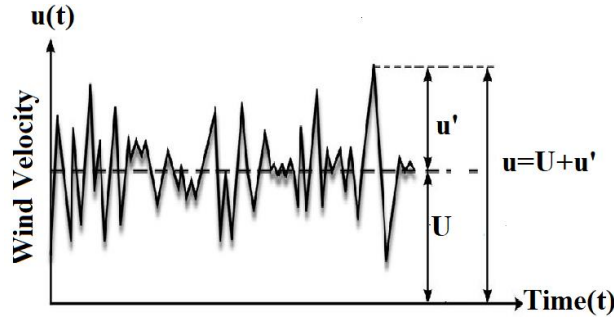


Fig. 1. Wind velocity in time at one point

The dimensionless parameter, *Reynolds Number* (Re) is defined by Eqn.1. and it helps to distinguish between the laminar and turbulent flow.

$$Re = \frac{UH}{\nu} \quad (1)$$

In Eqn.1, U (m/s) is the free stream velocity, ν is kinematic fluid viscosity (m^2/s), and H (m) is the building height. The flow with Re higher than $Re = 5 \times 10^5$ at boundary layer normally is turbulent.

1.3. Turbulent Modeling and Large Eddy Simulation Method

Turbulent flow is irregular and due to this irregularity, the turbulent flow appears difficult to be expressed as a function of space and time. To model turbulent flow, we should use turbulence modeling. Turbulence modeling is a mathematical approximation to model the physical behavior of turbulent flows. The *Navier Stokes* (NS) equations for incompressible flow is provided at Eqn. 2 and Eqn.3 in tensorial notation. [3]

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\nu \frac{\partial u_i}{\partial x_j} \right) \quad (2)$$

$$\frac{\partial u_i}{\partial x_i} = 0 \quad (3)$$

If we average the NS equation in time or space and consider $u_i = U_i + u'_i$ we have:

$$U_j \frac{\partial U_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j} \left((\nu + \nu_t) \frac{\partial U_i}{\partial x_j} \right) \quad (4)$$

$$\frac{\partial U_i}{\partial x_i} = 0 \quad (5)$$

The details of derivation can be found from [3, 4]. Turbulence modeling methods tries to approximate the equivalent viscosity or solve directly NS equations to consider turbulence [5, 7]. Turbulence modeling can be categorized into three methods [7, 8]:

1. *Reynolds-averaged Navier–Stokes* (RANS) based models: In this model, eddy viscosity similar to molecular viscosity is assumed to approximate turbulence quantities. The example of this method is two-equation models such as k-ε. In the k-ε model, in addition to the time-averaged NS equations, transport equations for k and ε must be solved. Since the time averaging equations are used, time dependent effects cannot be captured.
2. *Direct numerical simulation* (DNS) solves the equations for all eddies. With the available computer resources, it is very difficult to apply for practical problems.
3. *Large eddy simulation* (LES): With LES, larger eddies that can be captured by mesh are calculated directly, whereas the smaller eddies are modeled by *Subgrid Scale Stress* (SGS) model which is similar to RANS methods. Here, the *Smagorinsky–Lilly* SGS model developed by Smagorinsky [7] is used for the SGS model and it estimates the equivalent viscosity for turbulence of smaller eddies by Eqn.6.

$$\nu_{SGS} = \sqrt{\frac{C_k^3}{C_\epsilon}} \Delta^2 \sqrt{2\overline{S_{ij}} \cdot \overline{S_{ij}}}, \quad \overline{S_{ij}} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (6).$$

Where, Δ is the grid size and C is constant. This method assumes that the energy production and dissipation are in equilibrium for small scales.

These three turbulence modeling methods are shown also in Fig. 2 with respect to eddy size variation in the wind spectrum. In Fig. 2, eddy is shown as circular vortex. As it can be seen, all eddy sizes are resolved in DNS whereas effect of all the eddies of different length scales are modeled in RANS. However, in LES, large eddies are resolved and the effect of smaller eddies are modeled.

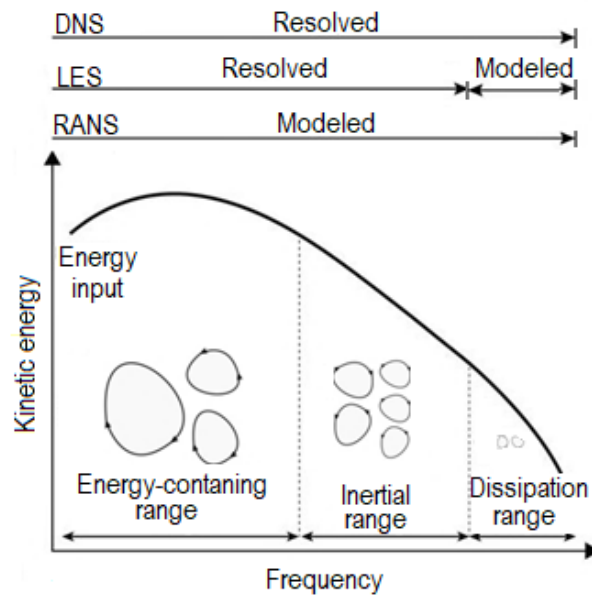


Fig. 2. Comparing different turbulence models with respect to eddy size variation in the wind spectrum.

1.4. Large Eddy Simulation Method Selection Reason

The complexity of turbulence makes it impossible to consider a single turbulence model as a universal model for modeling any turbulent flows. Thus, RANS turbulence models should be considered as engineering approximations rather than a scientific law. Turbulence is locally

dynamic in some flows, and time averaging completely removes these turbulence features. LES approximates these local flow variables. Because of this, DNS and LES have got attention in recent decades. Whereas RANS models deal with time-averaged fields, transient eddies are not calculated even if they are larger than the mesh size. On the other hand, in LES, any eddies larger than mesh size are calculated. For instance, as an analogical example, RANS calculates approximately how many people are passing the crosswalk. Whereas, in LES, we can capture each person's movement at each moment.

Although DNS and LES has enhanced capability in predicting the unsteadiness in the flow field, DNS can only be used for low Reynolds number flows and simple geometries. The flow details provided by DNS are not required for design purposes. Space-averaged quantities are appropriate for engineering applications. Highly resolved flow field obtained from LES rather than modeled effect of RANS k- ϵ could be of greater interest in the industry in the future.

1.5. Boundary Layer Velocity Profile

Additionally, the wind flow near the ground due to friction is zero, and as the height increases, velocity increases till the height that these obstructions cannot affect wind flow anymore. There are two approximations for this velocity profile, one of them is named logarithmic velocity profile. Verma et al. [1] stated that a region in space called computational domain has to be considered around a building for computing wind flow around a building. However, the shortcoming of the model [1] is that uniform inflow is introduced at the inlet. If uniform flow is considered, a large region needs to be modeled on the upstream side of building to develop the logarithmic velocity profile before the flow approaches the building. This would increase the computational time and subsequently the computational cost. Hence, to optimize, a logarithmic velocity profile can be selected for the inlet, which saves both computational time and cost and at the same time resembles a closer reflection to real world atmospheric flows.

1.6. Standard Wall Function

Furthermore, the velocity gradient close to the wall is very high near the ground due to ground friction. Therefore, to solve flow correctly, we need to refine the mesh near the walls; however, mesh refinement close to the walls increases the computational time and cost tremendously. An alternative round-about technique to obtain realistic flows without refining mesh close to the ground is to use the standard wall functions [7]. However, Verma et. al. [1] did not either consider law of the wall or mesh refinement near the walls. Thus, in this work, wall functions are used in the wall boundary faces to capture steep velocity gradients near to the walls.

1.7. Objectives

Following teaching module components will be provided as a hands-on material for students:

1. *OpenFOAM* as part of the CFD course to consider turbulence in the flow.
2. Applying standard wall function to use coarser mesh and save computational time and cost.
3. Applying logarithmic velocity profile at the inlet to model boundary layer wind.

4. Plotting mean, maximum and minimum peak pressure on a building by retrieving time series data from *OpenFOAM* and performing statistical analysis.

2. Problem Statement (Turbulent Wind Flow Around the Building)

The wind loads required for structural design of structures can be obtained using numerical (CFD) models. In the current study, the turbulent wind around a cubical building (i.e., *Silsoe* building) is modeled using *OpenFOAM*. To capture the turbulence effects in wind flow, LES with *Smagorinsky–Lilly* SGS model is used as a turbulence modeling method. A region around the building called computational domain shown in Fig.3 must be considered to compute wind flow around a building. The flow characteristics used as initial conditions are provided in Table 1.

To estimate wind load accurately, numerical model should be able to capture the real flow physics. In the real boundary layer, the velocity is zero close to the ground due to the friction and increases with height, hence, the wind velocity near the ground is approximated by the logarithmic velocity profile. Similarly, the standard wall function is implemented at walls to capture high velocity gradient near the ground.

Table. 1. Field data for the *Silsoe* building [9].

Test Characteristics	Explanation	Full Scale
H	Building height	6 m
U_H	Flow velocity at building height	9.52 m/s
z_0	Roughness length	0.1 m

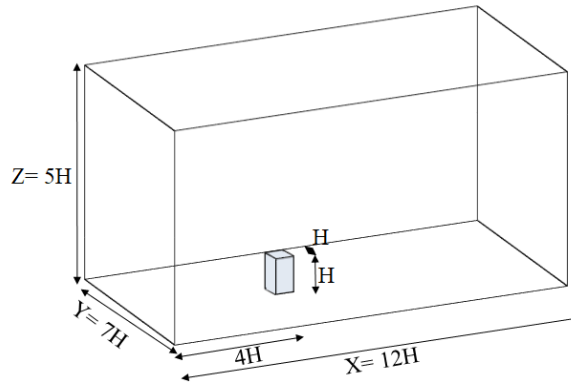


Fig. 3. 3D view of computational domain

2.1. Numerical Setup

In this study, the turbulent wind flow around the *Silsoe* building with a dimension of $1H$ is modeled, where ‘ H ’ is the height of the building. For the computational domain, enough region should be considered around the building. The size of the computational domain should be large enough so that the influence of the cube is not felt much by the outside boundary. Size of the computational domain is $12H$ in X -direction, $7H$ in Y -direction, and $5H$ in Z -direction. In this region, governing equations (i.e., 3D incompressible Navier–Stokes (NS) equation) are solved to obtain flow field details around the building. Mesh and geometry generations are explained by

Verma et al. [1]. The grid spacing size is $0.1H$ in each of X, Y, and Z-direction as shown in Fig 4. The total number of cells in the mesh of computational domain is 419,000.

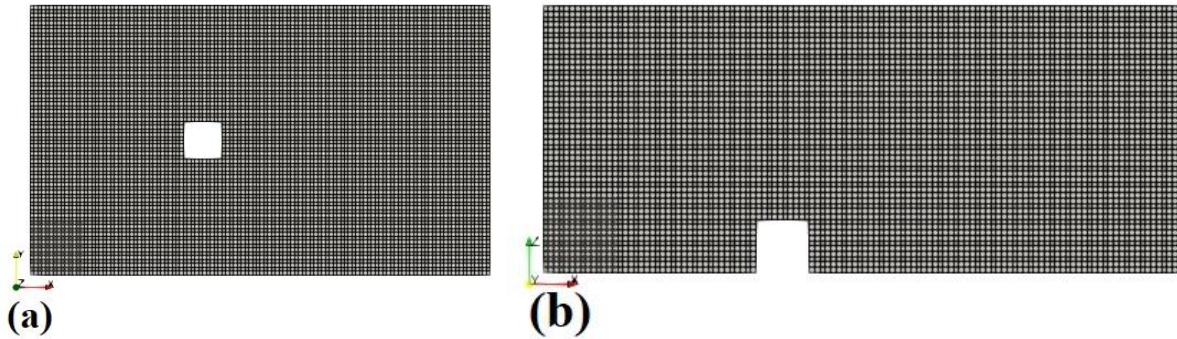


Fig. 4. Mesh of computational domain in (a) XZ-plane at $y = 3.5$ (b) XY-plane at $z = 0.5$.

2.1.1. Boundary Condition

Setting correct boundary conditions is important to obtain a real physical model. These boundary conditions and initial amounts are used to solve governing equations numerically. The boundary conditions are indicated for all boundary faces in Fig.5. The symmetric boundary conditions are implemented on the sidewalls, and the outflow boundary condition is specified at the outlet similar to what is used by Verma et. al. [1]. However, the inlet and wall boundary conditions are chosen differently for this study due to capture the wind flow physics around a building better in CFD model.

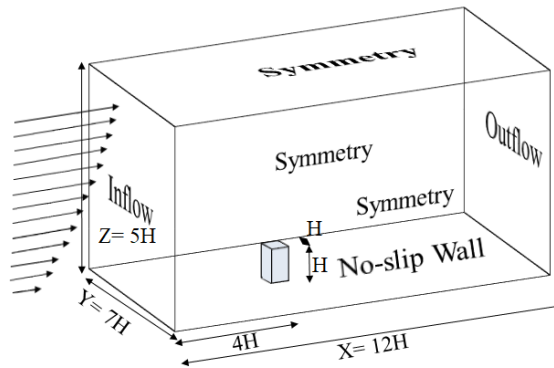


Fig. 5. Domain dimensions and domain boundary conditions.

2.1.1.1. Wind Profile Velocity at Inlet:

Ground obstructions close to the ground surface slows down the winds. Due to ground obstructions, wind velocity changes from zero at the ground surface to a maximum at a certain height from the ground where airflow is no longer affected by the obstructions. The height at which the airflow becomes unaffected by the ground obstructions is a function of ground roughness as it can be seen in Fig.6. The variation of velocity as a function of height taking ground roughness into

account is given by a well-known and accepted velocity profile approximation called the logarithmic velocity profile defined by Eqn.7.

$$U(z) = \frac{u^*}{\kappa} \ln\left(\frac{z}{z_0}\right) \quad (\text{m/s}) \quad \text{and} \quad V = W = 0 \quad (7)$$

Where U , V , and W are streamwise, spanwise, and ground-normal flow speed (m/s) respectively. u^* is friction velocity (m/s) and $\kappa = 0.41$ is von *Kármán* constant. z and z_0 are height and roughness length (m) respectively [10]. The roughness length, z_0 , is the height above the ground where the flow velocity is zero. As in this work, non-dimensional form of NS equations is considered, the nondimensional roughness is considered as $z_0 = 0.1/6 = 0.016 \approx 0.01$ in the numerical model.

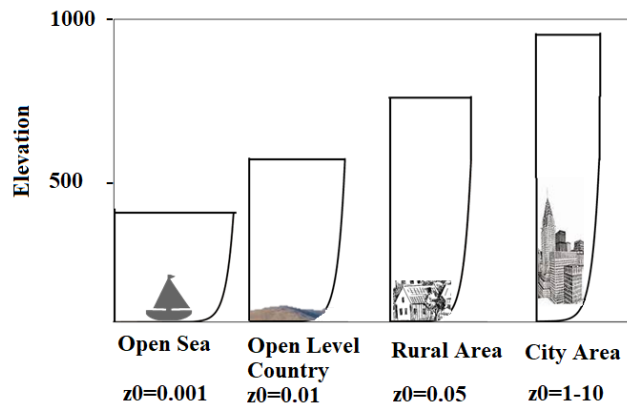


Fig.6. Variation of velocity as a function of height for different roughness and terrain exposure condition.

2.1.1.2. Wall Functions for Near Wall Treatment

Wind velocity close to the ground due to friction is zero, and as the height increases, velocity increases. This velocity gradient close to the wall is very high and the flow near the wall can be categorized into three different sublayers (i.e., viscous sublayer, inner region, and buffer layer) as shown in Fig.7. In the viscous sublayer, flows are almost smooth and laminar close to the wall, and the viscosity determines the flow behavior. Whereas, far from the wall, flows are fully irregular and turbulent in inner layer. In between, there is a layer is called buffer layer in which the effects of viscosity and turbulence are both important. To capture this steep flow gradient close to the wall, a very fine mesh resolution is required. However, it needs considerable computational resources in complex geometries or three-dimensional flows. To decrease computational costs without compromising the accuracy of the solution, empirical equations are used to satisfy the physics in the near-wall region. These formulas are called wall functions [7]. In *OpenFOAM* different wall functions are provided. In the current study, the *nutkWallFunction* is implemented at walls. The implementation of the wall function is explained in section 3.6.

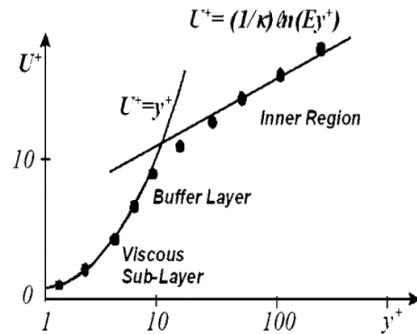


Fig. 7. Velocity in the near wall region.

3. *OpenFOAM* Implementation

As stated in the introduction, case files from Verma et al. [1] provided in [11] needs to be modified for a realistic model. In this section step by step, each file that needs to be modified is explained in details. The modified case file name is *BuildingLES* provided in [2]. In this file, LES is used for turbulence modeling, the logarithmic velocity is used for boundary condition at the inlet, and wall function is implemented at walls.

3.1. System Folder

Four files are placed in this folder: *blockMesh*, *ControlDict*, *fvSchemes* and *fvSolution*. Running time parameters (i.e., start/end time and time step) are set in *ControlDict*. The numerical schemes for terms are set in *fvSchemes* file. In *fvSolution* file, equation solvers, convergence tolerances, relaxation factors and other algorithm controls are set. Using *blockMesh* file and *blockMesh* command the mesh files (i.e., *polyMesh* files located at *constant* folder) will be created.

Firstly, a proper solver for turbulent flow should be determined. *pimpleFoam*, *pisoFoam*, and *simpleFoam* are solvers for turbulent flow provided in *OpenFOAM*. *SimpleFoam* is steady, whereas *pimpleFoam* and *pisoFoam* are transient. For coupling velocity and pressure, *pimpleFoam* uses the merged the PISO-Simple algorithm and *pisoFoam* uses the PISO algorithm. For this work, *pisoFoam* is chosen and is included in the *ControlDict* file.

In *ControlDict* file inside the system folder, time step (Δt) is taken as 0.025, total time=50, and the output files for visualization are written every 1 time units.

3.2. Constant Folder

Two files (*transportProperties*, *turbulenceProperties*) and a folder (*polyMesh*) are placed in this folder. *PolyMesh* files are created when the '*blockMesh*' command is used before running the case file. In *transportProperties* file, the properties of the fluid are included such as viscosity of the fluid. In this work, air is considered.

In *turbulenceProperties* file, the turbulence model is specified, and its parameters are set. In *polyMesh* folder, all the data for the mesh (i.e., points, edges, faces and so on) is located. This folder can be created in different ways such as manually, using and transforming the data from

other software to *OpenFOAM* such as *Salome*, or using *OpenFOAM* native meshing tools such as *blockMesh* and *snappyHexMesh*. The *boundary* file also is located in this folder where boundary names are defined.

It should be noted that for applying LES we use *turbulenceProperties* file.

3.3.“0” Folder: Boundary Condition

The boundary conditions are included in this folder. Depending on the turbulence model we are working with, different boundary condition files will need to be included in addition to typical boundary conditions such velocity, pressure and so on. The boundary conditions will be set for each boundary face and the structure will be the following:

```
Name_of_the boundary face
{
type Type of boundary face
value Value
}
```

The boundary condition type will be set depending on the wall definition. For each wall definition there are some values available that can be found in *OpenFOAM* User Guide [\[12\]](#).

3.4. Applying LES

To apply LES with *Smagorinsky* SGS model for turbulence modeling, the following lines are included in *turbulenceProperties* file.

```
simulationType LES;
LES
{
  LESModel      Smagorinsky;
  turbulence     on;
  printCoeffs   on;
  delta         cubeRootVol;
  cubeRootVolCoeffs
  {
    deltaCoeff   1;
  }
  SmagorinskyCoeffs
  {
    Ck           0.094;
    Ce           1.048;
  }
}
```

3.5. Applying Nonuniform Velocity at Inlet

Using the *codedFixedValue* we can define logarithmic velocity profile as the inlet boundary condition which is a characteristic of wind boundary layer. Hence, we should use the following code as the inlet boundary condition in *U* files.

```
inlet
{
type codedFixedValue;
redirectType velocity_inlet;
code
#{
scalar Ustar=0.089;
scalar k=0.41;
scalar z0=0.01;
fixedValueFvPatchVectorField myPatch(*this);
forAll(this->patch().Cf(),i)
{
myPatch[i]=vector(Ustar/k*(Foam::log((this->patch().Cf()[i].z())/z0)),0,0);
}
operator==(myPatch);
#};
value $internalField;
}
```

3.6. Applying Wall Function

To apply wall function, in the nut file under '0' folder, the wall function should be specified for walls boundary condition as follows:

```
Wall
{
type      nutkWallFunction;
value     uniform 0;
}
```

4. Visualization of Results

The post processing step of every numerical modeling are visualization plots in which we can find the details of flow field characteristics. Data files are stored each limited number of time steps setting the write interval inside *ControlDict* file. For current work, simulation was run for a total time of 50 units and the data files were written by solver every 40 time steps. We can use *ParaView* in *Windows* or *Linux* system. To open result files in *ParaView*, the *ControlDict* file should be chosen firstly, and then the *OpenFOAMReader* should be chosen and applied.

Pressure contour and velocity contour plots of wind flow around the building are plotted at the 50 time units in Fig 8 (a) and (b). The way of plotting is explained by [11]. There is a red colored region upstream of the building shows the stagnation region. The positive pressure due to wind force is applied on the windward side of building.

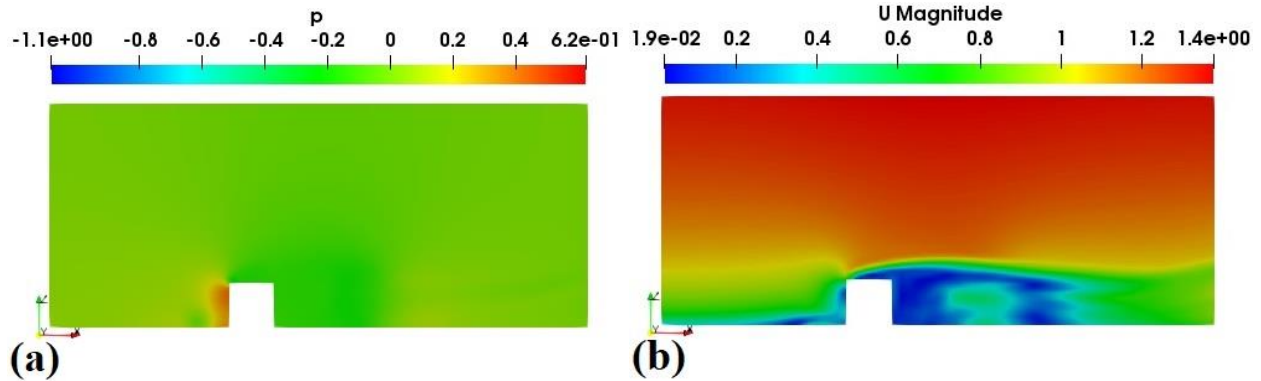


Fig. 8. Contour plots in computational domain through XZ-plane at $y = 3.5$ (a) Pressure contour (b) Velocity contour at last time step.

The pressure contour and velocity contour plots in the XY-plane at roof height of building (i.e. $z = 1.0$) are plotted respectively at 50 time units in Fig.9. (a) and (b). Here, we can also see the stagnation region on windward side of the building which has high pressure values. On the leeward side of building we can see the separation flow due to velocity is very low there.

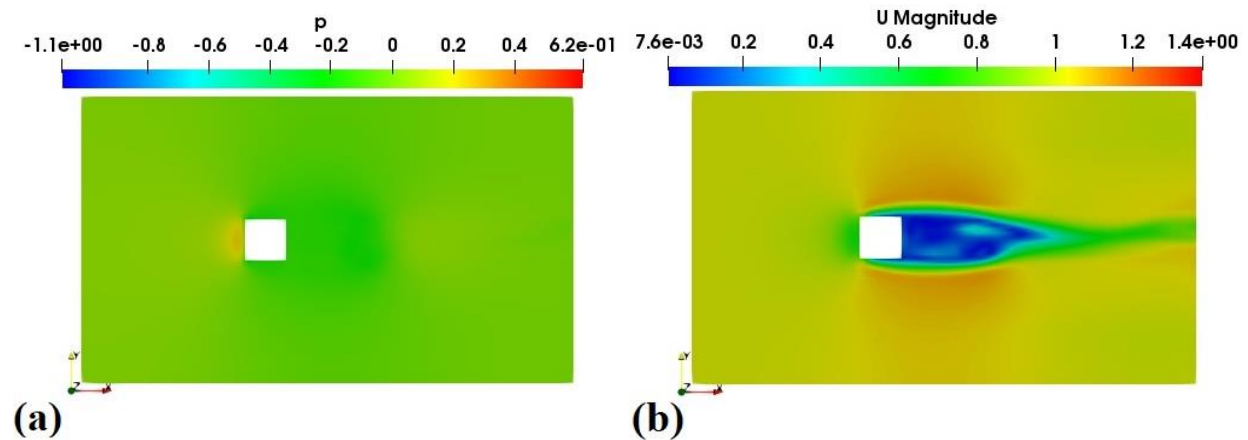


Fig. 9. Contour plots in computational domain through XY-plane at $z = 1.0$ (a) Pressure contour (b) Velocity contour at last time step.

The velocity profile at the inlet is plotted in Fig.10 in the YZ-plane which clearly shows the logarithmic profile defined at the inlet. The way of plotting is explained at report [2].

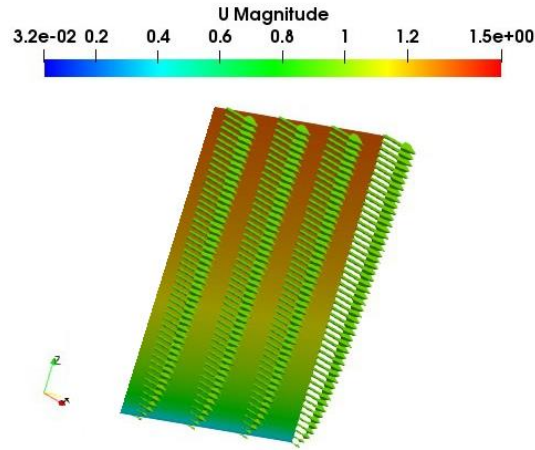


Fig.10. Velocity profile at the part of the inlet.

4.1. Pressure Calculation:

To calculate pressure statistics the following steps can be followed:

1. Writing pressure at each point located along the building centerline by adding following lines in the *ControlDict* file.

```
probes
{
    type          probes;
    libs          ("libsampling.so");
    writeControl   timeStep;
    writeInterval  1;
    fields
    (U p );
    probeLocations
    (
        (4 3.5 0.1)
        .....
        (5 3.5 0.1)
    );
}
```

2. Removing the first 10 time units pressure coefficients.
3. Calculating the maximum, minimum, and average of pressure coefficients amount at each point based on every time steps from 10 to 50 time units by using excel functions.

Using the above-mentioned steps, 9 points at the windward and leeward of building and 11 points at the roof are considered. The distance between every two consecutive point is $H/10$. The final plot is shown in Fig. 11. As it can be seen, the maximum positive pressure coefficient occurs on the windward face of building, whereas the minimum occurs at the roof edge.

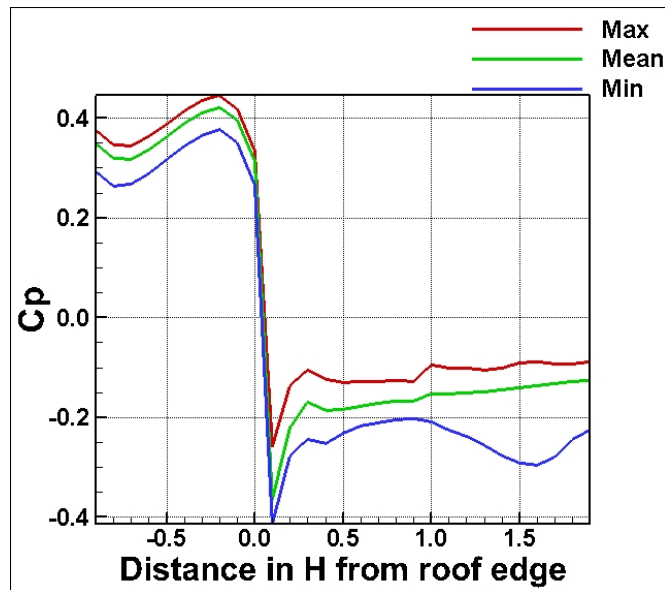


Fig. 11. Mean, maximum, and minimum C_p plot along the centerline of building the grid spacing size of $H/10$.

In Fig. 12, the pressure coefficient is plotted over time at roof edge. According to the Fig. 12, the minimum pressure coefficient could be around -0.4.

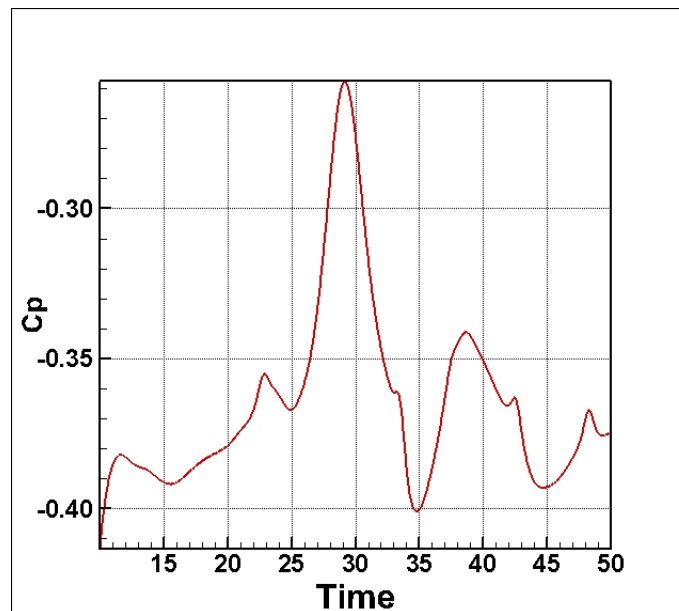


Fig. 12. C_p plot at point near the roof edge at (4.1, 3.5, 1) over time.

Conclusion:

Solving realistic problems in CWE is a demand for engineers to be able to design real structures. Using theoretical and numerical methods to solve these problems saves a lot of cost compared to experimental studies. Furthermore, numerical results give more details of flow field such as

detailed velocity and pressure at every point in the computational domain. Engineers and researchers can use CFD to help them for structural designing and obtaining wind loads. Each engineer should know fluid mechanics and CFD to be able to analyze and solve complex real fluid related problems to have a successful design.

In this contribution, a teaching method including open-source software packages such as *OpenFOAM* for teaching CFD and CWE is suggested because open-source software *OpenFOAM* would be a cost effective and more accessible alternative than commercial software for teaching learning purposes. In this method, it is tried to bring as an example of realistic wind related problem, flow around the building with real physical condition. To model real turbulent wind flow, LES as a turbulence model is used for turbulence modeling. Furthermore, to save computation time and cost and make numerical model closer to physical boundary layer, the logarithmic velocity profile is used at the inlet, and standard wall function is implemented at walls. Open-source software packages such as *OpenFOAM* & *ParaView* is the advantageous method in comparing to commercial software for teaching these courses. In addition, the ability to customize it makes the open source software more flexible to teach wide range of computational fluid dynamics problems. Therefore, using this work and the case file and extended report provided at <https://github.com/rpsuark/ASEE21-OpenFOAM-LES> by the authors, students can develop a good understanding about the case set up in *OpenFOAM* [2].

Acknowledgment

The authors acknowledge the support received from National Science Foundation (NSF) under award number CMMI-1762999. Ms. Zahra Mansouri acknowledges the financial support from the James T. Womble Professorship from the University of Arkansas.

References

- [1] Verma, S., Mansouri, Z., and Selvam, R.P. Teaching Building Aerodynamics Using Open-Source Software OpenFOAM in Proc. 2021 ASEE Midwest Section Virtual Conference, Sep. 13-15, 2021.
- [2] Z. Mansouri, S. Verma and R.P. Selvam, Teaching modeling turbulent flow around building using LES turbulence method and open-source software OpenFOAM- Extended report. 2021. <https://github.com/rpsuark/ASEE21-OpenFOAM-LES>.
- [3] Versteeg, H. K., & Malalasekera, W. An introduction to computational fluid dynamics: the finite volume method. Pearson education. 2007.
- [4] Davidson, P. A. Turbulence: an introduction for scientists and engineers. Oxford university press. 2015.
- [5] Pope, S. B. Turbulent Flows. Cambridge University Press. 2000.
- [6] Sagaut, P. Large Eddy Simulation for Incompressible Flows (Third ed.). Springer. 2006.
- [7] Wilcox, David C. Turbulence modeling for CFD. Canada, CA: DCW industries, 1998.
- [8] Smagorinsky J. General circulation experiments with the primitive equations: I. The basic experiment. Monthly weather review. 1963.

- [9] Mooneghi, M. A., P. Irwin, and A. G. Chowdhury. Partial turbulence simulation method for predicting peak wind loads on small structures and building appurtenances. J. Wind. 2016.
- [10] Dyrbye, C., & Hansen, S. O. Wind loads on structures. 1997.
- [11] R. P. Selvam. ASEE21-OpenFOAM-Introduction, github.com
<https://github.com/rpsuark/ASEE21-OpenFOAM-Introduction>
- [12] OpenFOAM: User Guide v2012, openfoam.com
<https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-derived-wall.html>

Biographical information

Zahra Mansouri completed her bachelor's degree in Civil Engineering from University of Shahrekord and her master's degree in Hydraulic Engineering from K. N. Toosi University of Technology. She had 3 years of industrial experience in designing structures, including preparation of hydraulic calculations, structural design calculations, specifications, and sketches for hydraulic and residential structures. Currently, she is a Ph.D. student and graduate research assistant at University of Arkansas, Civil Engineering Department.

Sumit Verma completed his bachelor's degree in Civil Engineering from the Institute of Engineering, Pulchowk Campus, Tribhuvan University, Nepal in 2015. Following his graduation, he worked as a Civil Engineer for an engineering consulting firm in Nepal before starting his graduate studies at the University of Arkansas from Fall of 2018. Currently, he is in the 3rd year of his Ph.D. program working as graduate research assistant in the Department of Civil Engineering at University of Arkansas.

R. Panneer Selvam is currently the University Professor and Womble Professor of Computational Mechanics and Nanotechnology Modeling in the Department of Civil Engineering, University of Arkansas. He has been teaching finite element methods, computational fluid dynamics (CFD), CFD for wind engineering, structural dynamics and structural loading. He has published a book on, 'Structural Dynamics and Loading' for undergraduate senior students. Currently he is writing a book on CFD for Wind Engineering to be published by Wiley in December 2021. He can be reached by email: rps@uark.edu.

Appendix 1. Case Files Explanation in Detail- 0 Folder: Boundary Condition

In the folder 0 you can enter your initial amount for velocity and pressure and nut.

 nut  p  U

The boundary conditions are included in this folder. Depending on the turbulence model we are working with, different boundary condition files will need to be included in addition to typical boundary conditions such as velocity, pressure and so on. It should be noted that the pressure is not the actual pressure, it is divided by density.

The boundary conditions will be set for each boundary face and the structure will be the following:

```
Name_of_the boundary face
{
type Type of boundary face
value Value
}
```

In **p** file you can see:

1. First lines are a header.
2. Dimension line is the exponent of seven SI units [0 2 -2 0 0 0]. For examples, for p, the first number is the exponent of Kg, the second number is the exponent of meter (i.e., m²), the third number is the exponent of second (i.e., s²).
3. In the next line, all domain pressure are initialized to zero.
4. Then boundary conditions are specified for every boundary face.
 - Inlet → zeroGradient
 - outlet → pressure fixed to zero.
 - Wall → zeroGradient
 - top → symmetryPlane
 - front → symmetryPlane
 - back → symmetryPlane

```

/*-----*- C++ -*-----*\
=====|
\\      / F i e l d | OpenFOAM: The Open Source CFD Toolbox
\\      / O p e r a t i o n | Website: https://openfoam.org
\\      / A n d | Version: 7
\\//    M a n i p u l a t i o n |

\*-----*-*/

FoamFile{
    version      2.0;
    format       ascii;
    class        volScalarField;
    object       p;}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -2 0 0 0 0];
internalField    uniform 0;
boundaryField
{
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value            uniform 0;
    }
    top
    {
        type            symmetryPlane;
    }
    Wall
    {
        type            zeroGradient;
    }
    front
    {
        type            symmetryPlane;
    }
    back
    {
        type            symmetryPlane;
    }
}

// *****

```


In **U** you can see:

5. First lines are a header.
6. Dimension line is the exponent of seven SI unites [0 1 -1 0 0 0 0]. For examples, for **U**, the first number is the exponent of Kg, the second number is the exponent of meter (i.e., m¹), the third number is the exponent of second (i.e., s⁻¹).
7. In the next line, all domain velocity are initialized to vector of zero. (0 0 0)
8. Then boundary conditions are specified for every boundary face.
 - Inlet→ Using the *codedFixedValue* we can define logarithmic velocity profile as the inlet boundary condition which is a characteristic of wind boundary layer. Hence, we should use the following code as the inlet boundary condition in **U** files.

$$U(z) = \frac{u^*}{\kappa} \ln\left(\frac{z}{z_0}\right) \text{ (m/s) and } V = W = 0$$

```
type codedFixedValue;

redirectType velocity_inlet;
code
#{
  scalar Ustar=0.089;
  scalar k=0.41;
  scalar z0=0.01;
  fixedValueFvPatchVectorField myPatch(*this);
  forAll(this->patch().Cf(), i)
  {
    myPatch[i]=vector(Ustar/k*(Foam::log((this->patch().Cf()[i].z())/z0)),0,0);
  }
  operator==(myPatch);
  #};
value $internalField;
}
```

- outlet→ zeroGradient.
- Wall → noslip
- top→ symmetryPlane
- front→ symmetryPlane
- back→ symmetryPlane

```
/*-----* C++ -*-----*\
=====
\\      / F i e l d      |
\\      / O p e r a t i o n | OpenFOAM: The Open Source CFD Toolbox
\\      / A n d           | Website:  https://openfoam.org
  \\/    M a n i p u l a t i o n | Version:  7
\*-----*/

FoamFile
{
    version      2.0;
    format       ascii;
    class        volVectorField;
```

```

        object          U;}
// *****

dimensions          [0 1 -1 0 0 0 0];
internalField        uniform (0 0 0);
boundaryField
{
    inlet
    {
        type codedFixedValue;
        redirectType velocity_inlet;
        code
        #{
            scalar Ustar=0.089;
            scalar k=0.41;
            scalar z0=0.01;
            fixedValueFvPatchVectorField myPatch(*this);
            forAll(this->patch().Cf(),i)
            {
                myPatch[i]=vector(Ustar/k*(Foam::log((this->patch().Cf()[i].z())/z0)),0,0);
            }
            operator==(myPatch);
        #};
        value $internalField;
    }

    outlet
    {
        type          zeroGradient;
    }
    Wall
    {
        type          noSlip;
    }
    top
    {
        type          symmetryPlane;
    }
    front
    {
        type          symmetryPlane;
    }
    back
    {
        type          symmetryPlane;
    }
}
// *****

```

In **nut** file you can see:

9. First lines are a header.
10. Dimension line is the exponent of seven SI unites [0 2 -1 0 0 0 0]. For examples, the first number is the exponent of Kg, the second number is the exponent of meter (i.e., m^2), the third number is the exponent of second (i.e., s^{-1}).
11. In the next line, for all domain nut are initialized to zero.
12. Then boundary conditions are specified for every boundary face. And *nutkWallFunction* is implemented at walls.
 - Inlet→ zeroGradient
 - outlet→ pressure fixed to zero.
 - Wall → **Wall**

```

{
    type                nutkWallFunction;
    value               uniform 0;
}
•
• top→ symmetryPlane
• front→ symmetryPlane
• back→ symmetryPlane

/*-----*- C++ -*-----*\
=====
\\      /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox
\\      /  O p e r a t i o n | Website:  https://openfoam.org
\\      /  A n d           | Version:   7
\\      /  M a n i p u l a t i o n |
\*-----*/
FoamFile{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "1";
    object       nut;}
// *****

dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    inlet
    {
        type                zeroGradient;
    }
    outlet

```

```
{
    type          zeroGradient;
}
top
{
    type          symmetryPlane;
}
front
{
    type          symmetryPlane;
}
back
{
    type          symmetryPlane;
}
Wall
{
    type          nutkWallFunction;
    value         uniform 0;
}
}
//***** //
```

Appendix 2. Case Files Explanation in Detail- Constant Folder

Two files (*transportProperties*, *turbulenceProperties*) and a folder (*polyMesh*) are placed in this folder. *PolyMesh* files are created when the ‘*blockMesh*’ command is used before running the case file. In *transportProperties* file, the properties of the fluid are included such as viscosity of the fluid. In this work, air is considered.

 *transportProperties*  *turbulenceProperties*

In *transportProperties* you can input the amount for ν = kinematic viscosity .

Dimension line is the exponent of seven SI unites [0 2 -1 0 0 0 0]. For examples, the first number is the exponent of Kg, the second number is the exponent of meter (i.e., m^2), the third number is the exponent of second (i.e., s^{-1}). And amount of ν is $1.5e-5$.

```

/*-----*-- C++  *-----*\
=====
\\      /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox
\\      /  O p e r a t i o n | Website:  https://openfoam.org
\\      /  A n d           | Version:   7
  \\    /  M a n i p u l a t i o n |
\*-----*--*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       transportProperties;
}
// * * * * *
transportModel  Newtonian;
nu              [0 2 -1 0 0 0 0] 1.5e-05;
// *****

```

In *turbulenceProperties* file, the turbulence model is specified, and its parameters are set. In *polyMesh* folder, all the data for the mesh (i.e., points, edges, faces and so on) is located. This folder can be created in different ways such as manually, using and transforming the data from other software to *OpenFOAM* such as *Salome*, or using *OpenFOAM* native meshing tools such as *blockMesh* and *snappyHexMesh*. The *boundary* file also is located in this folder where boundary names are defined.

It should be noted that for applying LES we use *turbulenceProperties* file. To apply LES with *Smagorinsky* SGS model for turbulence modeling, the following lines are included in *turbulenceProperties* file.

```




simulationType LES;
LES

```

{

Appendix 3. Case Files Explanation in Detail- System Folder

Four files are placed in this folder: *blockMesh*, *ControlDict*, *fvSchemes* and *fvSolution*. Running time parameters (i.e., start/end time and time step) are set in *ControlDict*. The numerical schemes for terms are set in *fvSchemes* file. In *fvSolution* file, equation solvers, convergence tolerances, relaxation factors and other algorithm controls are set. Using *blockMesh* file and *blockMesh* command the mesh files (i.e., *polyMesh* files located at *constant* folder) will be created.

 *blockMeshDict*  *controlDict*  *fvSchemes*  *fvSolution*

blockMesh is explained in detail by [1].

Firstly, a proper solver for turbulent flow should be determined. *pimpleFoam*, *pisoFoam*, and *simpleFoam* are solvers for turbulent flow provided in *OpenFOAM*. *SimpleFoam* is steady, whereas *pimpleFoam* and *pisoFoam* are transient. For coupling velocity and pressure, *pimpleFoam* uses the merged the Piso-Simple algorithm and *pisoFoam* uses the Piso algorithm. For this work, *pisoFoam* is chosen and is included in the *ContolDict* file.

In *ContolDict* file inside the system folder, time step (dt) is taken as 0.025, total time=50, and the output files for visualization are written every 1 time units.

In *ContolDict* you can see:

StartFrom: latestTime is the time that we did run before for example 10 second and we want the run will continue from that time.

startTime: 0

endTime is total time that we want to run our simulation. In this case it is 50.

deltaT=0.025 the amount of timestep

writeControl: timeStep – The results of run will be write with the number of timestep we will define.

writeIntervals: 40 results be written in each 40 timesteps. In this case, it will be at every $40 \times 0.025 = 1$ time units.

Using the below code, you let that the program used the code for velocity inlet:

InfoSwitches

{

writePrecision 6;

writeJobInfo 0;

// Allow case-supplied C++ code (#codeStream, codedFixedValue)

allowSystemOperations 1;


```
}
```

Using the following lines, the program writes P and U at the specific points that determined at probes. These probes are located at the centerline of building.

```
functions
```

```
{
    probes
    {
        type      probes;
        libs      ("libsampling.so");
        writeControl    timeStep;
        writeInterval    1;

        fields
        (
            U p
        );

        probeLocations
        (
            (4 3.5 0.1)
            (4 3.5 0.2)
            (4 3.5 0.3)
            (4 3.5 0.4)
            (4 3.5 0.5)
            (4 3.5 0.6)
            (4 3.5 0.7)
            (4 3.5 0.8)
            (4 3.5 0.9)
            (4 3.5 1)
            (4.1 3.5 1)
            (4.2 3.5 1)
            (4.3 3.5 1)
            (4.4 3.5 1)
            (4.5 3.5 1)
            (4.6 3.5 1)
            (4.7 3.5 1)
            (4.8 3.5 1)
            (4.9 3.5 1)
            (5 3.5 1)
            (5 3.5 0.9)
        )
    }
}
```

```

(5 3.5 0.8)
(5 3.5 0.7)
(5 3.5 0.6)
(5 3.5 0.5)
(5 3.5 0.4)
(5 3.5 0.3)
(5 3.5 0.2)
(5 3.5 0.1)
);

}
}

/*-----*- C++ -*-----
=====
\\      /  F i e l d      |  OpenFOAM: The Open Source CFD Toolbox
\\      /  O p e r a t i o n      |  Website:  https://openfoam.org
\\      /  A n d      |  Version:  7
\\//      M a n i p u l a t i o n      |
\\*-----*-*/
FoamFile
{
    version      2.0;
    format      ascii;
    class      dictionary;
    location    "system";
    object      controlDict;
}
// * * * * *

application      pisoFoam;
startFrom        startTime;
startTime        0;
stopAt           endTime;
endTime          50;
deltaT           0.25e-1;
writeControl     timeStep;
writeInterval    40;
purgeWrite       0;
writeFormat      ascii;
writePrecision   6;
writeCompression off;
timeFormat       general;
timePrecision    6;
runTimeModifiable true;
InfoSwitches

```

```

{
    writePrecision 6;
    writeJobInfo 0;
    // Allow case-supplied C++ code (#codeStream, codedFixedValue)
    allowSystemOperations 1;
}
functions
{
    probes
    {
        type                probes;
        libs                ("libsampling.so");
        writeControl        timeStep;
        writeInterval       1;

        fields
        (
            U p
        );

        probeLocations
        (
            (4 3.5 0.1)
            (4 3.5 0.2)
            (4 3.5 0.3)
            (4 3.5 0.4)
            (4 3.5 0.5)
            (4 3.5 0.6)
            (4 3.5 0.7)
            (4 3.5 0.8)
            (4 3.5 0.9)
            (4 3.5 1)
            (4.1 3.5 1)
            (4.2 3.5 1)
            (4.3 3.5 1)
            (4.4 3.5 1)
            (4.5 3.5 1)
            (4.6 3.5 1)
            (4.7 3.5 1)
            (4.8 3.5 1)
            (4.9 3.5 1)
            (5 3.5 1)
            (5 3.5 0.9)
            (5 3.5 0.8)
            (5 3.5 0.7)
            (5 3.5 0.6)
            (5 3.5 0.5)
            (5 3.5 0.4)
        )
    }
}

```

```

        (5 3.5 0.3)
        (5 3.5 0.2)
        (5 3.5 0.1)
    );

}

}
// *****

```

In fvSchemes and fvSolution you can define different schemes, solution methods, tolerance amount and so on. The PISO algorithm is chosen for coupling velocity and pressure. To solve pressure the GAMG is chosen. For velocity solver, symGaussSeidel is chosen.

```

/*-----*- C++ -*-----*\
=====
\\      /  F ield      | OpenFOAM: The Open Source CFD Toolbox
\\      /  O peration  | Website:  https://openfoam.org
\\      /  A nd        | Version:  7
\\      /  M anipulation |
\-----*-/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       fvSolution;
}
// * * * * *

solvers
{
    p
    {
        solver      GAMG;
        smoother     GaussSeidel;
        tolerance    1e-5;
        relTol       0.1;
    }

    pFinal
    {
        $p;
        tolerance    1e-5;
    }
}

```

```

        relTol            0;
    };

    "(U|k|omega|epsilon)"
    {
        solver            smoothSolver;
        smoother          symGaussSeidel;
        tolerance         1e-6;
        relTol            0;
    }
}

PISO
{
    nCorrectors          2;
    nNonOrthogonalCorrectors 0;
}

// *****

/*-----*- C++ -*-----*\
=====|
\\      /  F ield      | OpenFOAM: The Open Source CFD Toolbox
\\      /  O peration  | Website:  https://openfoam.org
\\      /  A nd        | Version:  6
\\//     M anipulation |
\*-----*-
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       fvSchemes;
}
// *****

ddtSchemes
{
    default      Euler;
}

gradSchemes
{
    default      Gauss linear;

    limited      cellLimited Gauss linear 1;

```

```

    grad(U)          $limited;
    grad(k)          $limited;
    grad(epsilon)    $limited;
}

divSchemes
{
    default          none;

    div(phi,U)       bounded Gauss linearUpwind limited;

    turbulence       bounded Gauss limitedLinear 1;
    div(phi,k)       $turbulence;
    div(phi,epsilon) $turbulence;

    div((nuEff*dev2(T(grad(U)))) Gauss linear;
}

laplacianSchemes
{
    default          Gauss linear corrected;
}

interpolationSchemes
{
    default          linear;
}

snGradSchemes
{
    default          corrected;
}

wallDist
{
    method meshWave;
}

// ***** //
```


Appendix 4. Source code location and running steps

Source code:

Furthermore, you can see the source code from the following directory:

Computer → Opt → OpenFoam7 → Application → incompressible → pisoFoam

For running case file:

For running the case file, the following steps should be used.

1. opening a Terminal in the case file folder
2. using *blockMesh* command
3. using *pisoFoam* command

Appendix 5. Postprocessing and visualization

ParaView and velocity profile at inlet


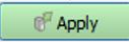
Open → choose *ControlDict* → press ok → choose *OpenFoamReader* → press *Apply*

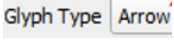
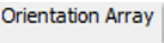
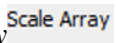
You can see your mesh with Surface With edges.

You can choose P and U to see them in different time.

you can save Animation in video format and picture format in file → save animation.

Velocity vector at inlet:

Using Verma et al [1] you can change the color and choose your inlet. After that, *Glyph* icon  should be chosen. Then, press *Apply* .

Now, some of the default field values pointed out by the arrow pointers are customized. *Glyph Type* is defined to *Arrow* . The orientation of velocity vector will be based on the orientation of velocity vector “U”,  *U*. However, for *Scale Array*  the field value will be changed to “U”. Under masking, *Every Nth point*, should be set to 10. Following these steps, users can obtain a velocity vector plot at inlet as shown in Fig. 13 below.

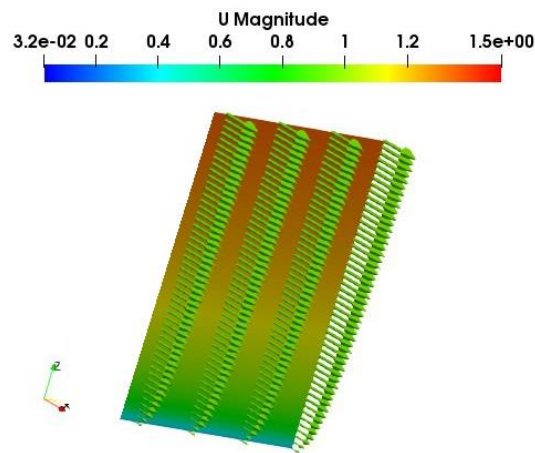


Fig.13. Velocity profile at the part of the inlet.

Pressure Calculation:

To calculate pressure statistics the following steps can be followed:

1. Writing pressure at each point located along the building centerline by adding following lines in the *ControlDict* file.

```
probes
{
type      probes;
libs      ("libsampling.so");
```

```

writeControl   timeStep;
writeInterval  1;
fields
(U p );
probeLocations
(
    (4 3.5 0.1)
    (4 3.5 0.2)
    (4 3.5 0.3)
    (4 3.5 0.4)
    (4 3.5 0.5)
    (4 3.5 0.6)
    (4 3.5 0.7)
    (4 3.5 0.8)
    (4 3.5 0.9)
    (4 3.5 1)
    (4.1 3.5 1)
    (4.2 3.5 1)
    (4.3 3.5 1)
    (4.4 3.5 1)
    (4.5 3.5 1)
    (4.6 3.5 1)
    (4.7 3.5 1)
    (4.8 3.5 1)
    (4.9 3.5 1)
    (5 3.5 1)
    (5 3.5 0.9)
    (5 3.5 0.8)
    (5 3.5 0.7)
    (5 3.5 0.6)
    (5 3.5 0.5)
    (5 3.5 0.4)
    (5 3.5 0.3)
    (5 3.5 0.2)
    (5 3.5 0.1));
}

```

2. Removing the first 10 time units pressure coefficients.
3. Calculating the maximum, minimum, and average of pressure coefficients amount at each point based on every time steps from 10 to 50 time units by using excel functions.

Using the above-mentioned steps, 9 points at the windward and leeward of building and 11 points at the roof are considered. The distance between every two consecutive point is $H/10$. The final plot is shown in Fig. 11. As it can be seen, the maximum positive pressure coefficient occurs on the windward face of building, whereas the minimum occurs at the roof edge.

Using excel file, the maximum amount, minimum, and average amount at each point over time are determined and plotted as shown in Fig.14.

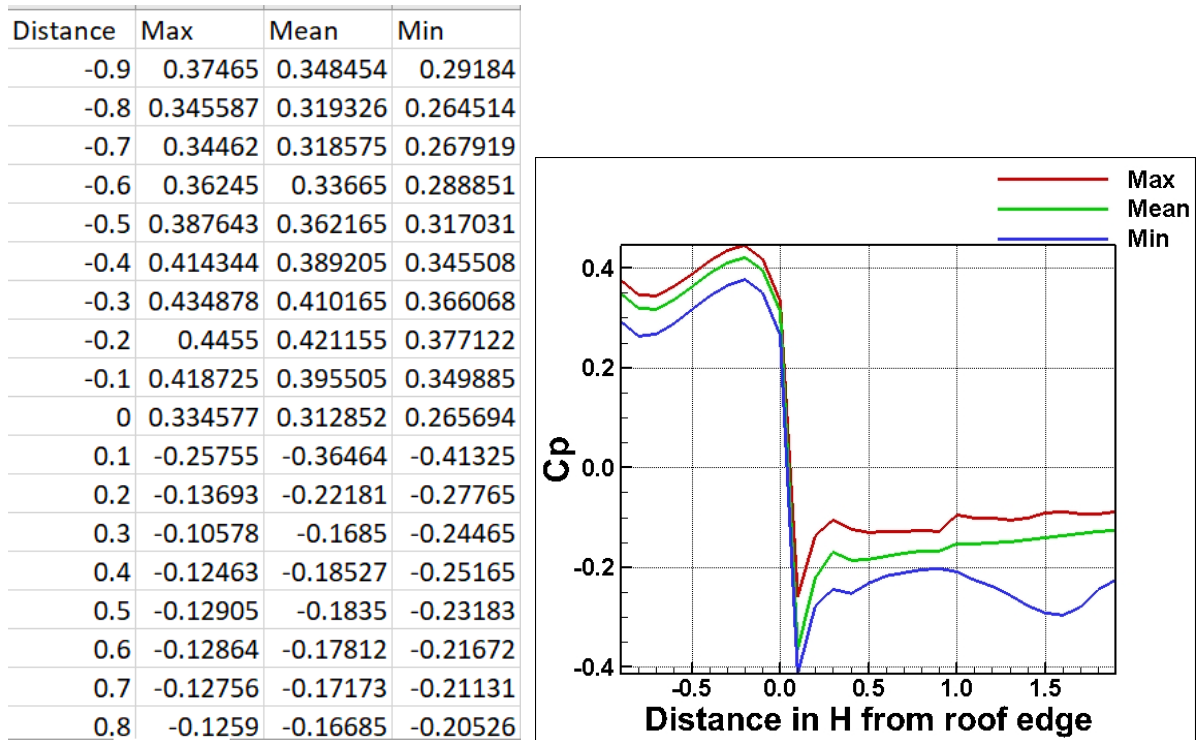


Fig. 14. Mean, maximum, and minimum Cp plot along the centerline of building the grid spacing size of H/10.