Modeling Outdoor Air Pollution and the Urban Form

by

Natasha Lia Stamler

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

May 2022

Signature of Author:

Department of Mechanical Engineering
May 6, 2022

Certified by:

Leslie K. Norford
Professor of Building Technology
Thesis Supervisor

Accepted by:

Kenneth Kamrin
Associate Professor of Mechanical Engineering
Undergraduate Officer

Modeling Outdoor Air Pollution and the Urban Form

by

Natasha Lia Stamler

Submitted to the Department of Mechanical Engineering
on May 6, 2022 in Partial Fulfillment of the
Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

ABSTRACT

As cities continue to grow rapidly, air pollution is becoming an increasing health problem. However, air pollution's spatial and temporal variability make it difficult to quantify, even with field measurements. Models are thus useful to understand how pollutants interact with the built and natural environment. Computational fluid dynamics (CFD) offers the highest spatial and temporal resolution for aerial pollutant dispersion modeling within dense urban environments such as urban canyons. Open-source platforms such as OpenFOAM are valuable as they can be customized for the varying intricacies of urban airflow and are accessible to a wide audience. This thesis develops a solver for aerial pollutant transport by adding a passive-scalar transport equation to *buoyantBoussinesqPimpleFoam*, the OpenFOAM transient solver for buoyant, turbulent flow of incompressible fluids, with Reynolds Averaged Navier-Stokes (RANS) turbulence modeling. It then demonstrates that this solver can be applied to cases, such as that of an urban canyon, with geometry generated parametrically using Grasshopper, a design tool commonly used by architects and urban designers. The successful implementation of this solver could enable future integration into a streamlined Grasshopper tool that allows designers to easily evaluate the impacts of their designs on urban air pollution during the design process.

Thesis Supervisor: Leslie K. Norford
Tile: Professor of Building Technology

Table of Contents

Table of Figures

Table of Tables

1. Background

Air pollution kills seven million people around the world every year through stroke, heart disease, lung cancer, and acute respiratory infections (*Air Pollution*, n.d.). There are several different pollutants that cause human health detriments, in particular PM2.5, which is fine inhalable particles, with diameters that are generally less than 2.5 micrometers (*Particulate Matter (PM) Basics*, 2016). The relative size of a PM2.5 particle is shown in Figure 1. PM2.5 is the sixth highest risk factor for death around the world, killing over four million people each year (Larry C. Price, n.d.). These tiny particles of solids or liquids are leached into the air by dust, dirt, soot, smoke, and drops of liquid. Air pollution is a particular problem in cities because urban areas produce higher than average concentrations of air pollutants that can be traced to vehicle exhaust and industrial emissions. Urban air pollution is especially important due to its growing potential for exposure: In 2018, over 55% of the population lived in urban areas, a number that is expected to increase to over 60% by 2030 (*The World's Cities in 2018 - Data Booklet (ST/ESA/SER.A/417)*, 2018).

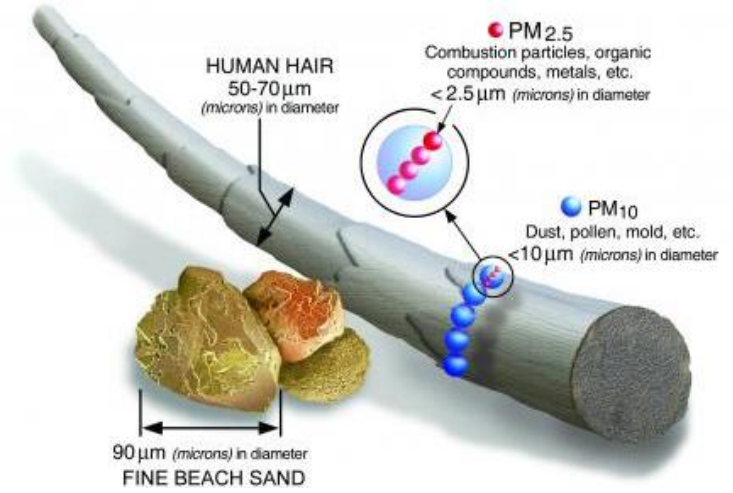


Figure 1. Size comparisons for particulate matter particles (*Particulate Matter (PM) Basics*, 2016).

However, air pollution can be difficult to quantify with field measurements due to its spatial and temporal variability (Mayer, 1999). Modeling can provide valuable insight into this variability in pollution concentration. These models can then be used to predict human impacts and to inform mitigation measures, such as changes to the built environment. Computational Fluid Dynamics (CFD) provides the highest spatial and temporal resolution for air pollution modeling. Other models, such as AEROMOD, a Gaussian Plume Model (GPM) developed by the US Environmental Protection Agency (EPA), cannot accurately compute concentrations for complex building geometries or time scales finer than an hour (Cimorelli et al., 2004). However, CFD's potential for high computational cost necessitates the development of solvers that implement efficient schemes.

CFD is increasingly being used to understand the dynamics of urban physics at scales from the meteorological to human to address problems related to health, energy, and climate (Blocken, 2015; Nakajima et al., 2018; Zheng et al., 2010). Within cities, urban (or street) canyons, canyon-like environments created on streets flanked by tall buildings on both sides, are of interest for air pollution modeling due to their prevalence and tendency to trap pollutants near the ground (Eeftens et al., 2013; Huang et al., 2021; Tauer, 2021). At this level, trapped particles increase pedestrians' exposure to pollution. At the same time, structures along the street can significantly affect pollutant concentrations, meaning the built environment plays a key role in mitigating the impacts of air pollution (Huang et al., 2021; Murena & Mele, 2016; Tauer, 2021).

2. Motivation, Scope, and Objectives

Air pollution modeling is currently largely limited to complex, government-run models, or expensive software, such as Ansys Fluent, a well-known CFD software. Open-source CFD software, such as OpenFOAM, only track wind velocity and pressure, rather than particle concentration, with their built-in solvers. However, using open-source software enables streamlined integration with design software, allowing architects and designers to consider air pollution in their designs. This is demonstrated by Eddy3D (Kastner & Dogan, 2021) and Butterfly (Chronis et al., n.d.; Maffessanti, 2019), two add-ons for Grasshopper, a parametric design software, that utilize OpenFOAM to evaluated wind flow around buildings during the design phase.

This thesis develops a solver for aerial pollutant transport building off the existing free, open-source OpenFOAM CFD software. It then demonstrates that this solver can be applied to a case of an urban canyon, with geometry generated parametrically using Grasshopper, a design tool commonly used by architects and urban designers. The successful implementation of this solver could enable future integration into a streamlined Grasshopper tool, similar to Eddy3D and Butterfly, that allows designers to easily evaluate the impacts of their designs on urban air pollution during the design process. This tool would facilitate a built environment that strives to reduce the impacts of air pollution.

3. Methods

3.1. Computational Fluid Dynamics (CFD) Simulations

Airflow speed, flow, and pollutant concentration were calculated for test cases using OpenFOAM v5 (Weller et al., 1998) in Windows 10 using blueCFD-Core 2017. Urban canyons are inherently turbulent airflow regimes; the large length scales attributed to buildings and the potential for high wind speeds can lead to high Reynolds numbers (Re) on the order of 10^7 – 10^9 (fluids become turbulent around $Re = 2 \times 10^3$). This turbulence is represented using Reynolds-averaged Navier-Stokes (RANS) turbulence modeling. RANS is used despite the higher accuracy of Large Eddy Simulation (LES) for street canyon airflow simulations (Chew & Norford, 2018; Nakajima et al., 2018) due to the computational intensity of LES that makes it infeasible for domains larger than idealized urban canyons (Elfverson & Lejon, 2021; Tauer, 2021). Similarly, most large-scale, urban simulations use RANS turbulence modeling (Toparlar et al., 2017). Computational efficiency is especially important in the case of a design tool as many simulations must be run throughout the process of designing a building or neighborhood.

3.1.1. Solver and Governing Equations

This study builds on *buoyantBoussinesqPimpleFoam*, OpenFOAM's transient solver for buoyant, turbulent flow of incompressible fluids. Using a transient model instead of a steady state one enables the handling of conditions that change with time, such as time-varying pollutant sources. Air was assumed to be an incompressible fluid despite its gaseous state due to its low speed (less than 100 m/s).

This solver uses the Boussinesq approximation for thermal convection, defined in Equation 1, due to the small temperature variations in the cases.

$$\rho_k = 1 - \beta(T - T_{ref}), \tag{1}$$

where $\rho_k = \frac{\rho}{\rho_{ref}}$ is the effective (driving) kinematic density, ρ is the fluid (air) density [kg/m³], ρ_{ref} is the fluid reference density [kg/m³], β is the thermal expansion coefficient [1/K], T is the temperature [K], and T_{ref} is the reference temperature [K], such that

$$\frac{\beta\left(T - T_{ref}\right)}{\rho_{ref}} \ll 1 \tag{2}$$

(i.e., $\rho \approx \rho_{ref}$ and $T \approx T_{ref}$). For an ideal gas, $\beta = \frac{1}{T_{ref}}$, so the Boussinesq approximation for the Navier-Stokes equation (Boussinesq equation) with gravity as a body force becomes

$$\rho \frac{\partial \boldsymbol{u}}{\partial t} + \rho(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\nabla p^2 + \mu \nabla^2 \boldsymbol{u} - \rho g \beta\left(T - T_{ref}\right)\boldsymbol{e}_y, \tag{3}$$

where $\boldsymbol{u}$ is the fluid flow velocity [$\frac{m}{s^2}$] and $\boldsymbol{e}_y$ is the unit vector in the y-direction, the direction in which gravity points. Note that **bolded variables** indicate vectors, while unbolded variables are scalars. This approximation introduces errors on the order of 1% if the temperature differences are below 15K (or °C) for air, as in the cases evaluated in this study.

This study uses the k-epsilon ($k - \varepsilon$) turbulence model to simulate mean flow characteristics (Launder & Spalding, 1974). It describes turbulence using one equation for k, the turbulent kinetic energy [m^2/s^2],

$$k = \frac{u_\tau^2}{\sqrt{C_\mu}}, \tag{4}$$

and one for ε, the turbulent kinetic energy dissipation rate [m^2/s^3], represented as `epsilon` in OpenFOAM,

$$\varepsilon(z) = \frac{u_\tau^3}{\kappa(z + z_0)}, \tag{5}$$

where κ is the dimensionless von Karman constant (0.41), z is the height [m] at which the ground-normal streamwise flow speed profile, u [m/s], is calculated, C_μ is the dimensionless turbulent viscosity constant (0.09), and z_0 is the aerodynamic roughness length [m], which defines the boundary with the roughness sublayer. z_0 varies by landscape and is taken as 0.005 m, the accepted value for unobstructed flow on unvegetated land (World Meteorological Organization, 2008).

$$u_\tau = \frac{\sqrt{\tau_w}}{\rho} \tag{6}$$

is the friction or shear velocity [m/s], where

$$\tau_w = \frac{1}{2} C_f \rho U_\infty^2 \tag{7}$$

is the wall shear stress [N/m^2], where C_f is the dimensionless skin friction coefficient, which is a function of Re dependent on the problem geometry, and U_∞ is the freestream ("far away") velocity [m/s^2].

3.1.2. Algorithm

This solver uses the PIMPLE algorithm, which combines the Pressure Implicit with Splitting of Operators (PISO) (Issa, 1986) and Semi-Implicit Method for Pressure Linked Equations (SIMPLE) (Caretto et al., 1973) algorithms. It first computes density, then velocity, then energy, and then pressure. If pressure does not converge, a new pressure is guessed until convergence. Once pressure converges, velocity and energy are recomputed until the solution converges. The PIMPLE algorithm flowchart is shown in Figure 2.

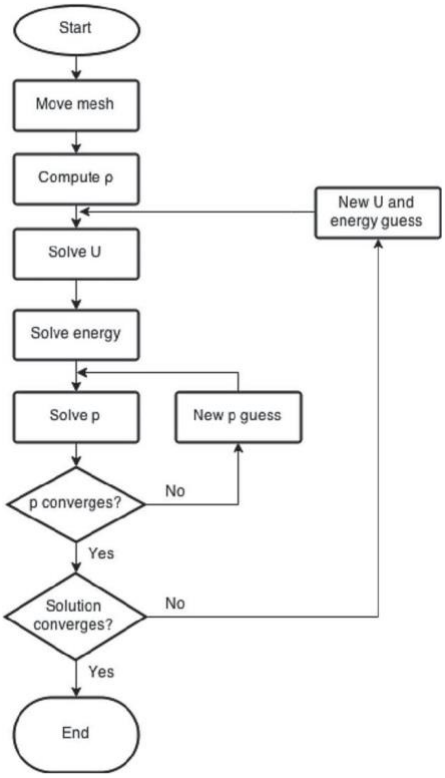


Figure 2. PIMPLE algorithm flowchart (Garcia-Alcaide et al., 2017).

The PIMPLE algorithm was modified to model pollutant transport using a passive-scalar transport equation, similar to the work of Tauer (2021), defined as

$$\frac{\partial}{\partial t} C + \nabla \cdot (\boldsymbol{\phi}\, C) - \nabla^2 \left(D_{t,t}\, C\right), \tag{8}$$

where $\boldsymbol{\phi}$ is the volumetric face-flux (flow through the cell faces) $[\frac{m^3}{s}]$, C is the pollutant concentration, and $D_{t,t}$ is the effective time-dependent turbulent mass diffusivity of the pollutant $[\frac{m^2}{s}]$, defined as

$$D_{t,t} = D_t + \frac{\nu_t}{Sc_t}, \tag{9}$$

where D_t is the mass diffusivity of the pollutant [$\frac{m^2}{s}$], ν_t is the turbulent kinematic viscosity [$\frac{m^2}{s}$], and Sc_t is the turbulent Schmidt number, a dimensionless quantity that compares the importance of advection and diffusion for mass transport. For urban environments, Sc_t should be between 0.3 and 1.2 (Longo et al., 2019; Monbureau et al., 2020; Santiago et al., 2020; Shi et al., 2019; Toja-Silva et al., 2017). Equation 9 was implemented in OpenFOAM using a scalar field defined at the cell center (`volScalarField`):

```
volScalarField DTT ("DTT", DT + turbulence->nut()/Sct);
```

Equation 8 was implemented implicitly in OpenFOAM using the finite volume method (`fvm`):

```
fvScalarMatrix ConcEqn
(
fvm::ddt(Conc)
+ fvm::div(phi, Conc)
- fvm::laplacian(DTT, Conc)
);
ConcEqn.solve();
```

4.1. Test Case: Simple Box

A box with simple boundary conditions was used to verify the solver.

4.1.1. Boundary Conditions

Table 1 lists the simple boundary conditions for this test case. The fixed downwards airflow was 1 cm/s. The fixed concentrations at the floor and ceiling were varied for different trials.

Table 1. List of boundary conditions for simple box test case.

boundary Field	α_t	$Conc$	ε	k	ν_t	p	p_{rgh}	T	u
Floor	alphatJayatilleke WallFunction	fixedValue	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	fixed Value	noSlip
Ceiling	alphatJayatilleke WallFunction	fixedValue	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	fixed Value	noSlip
FixedWalls	alphatJayatilleke WallFunction	zero Gradient	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	zero Gradient	fixed Value

α_t is the turbulent thermal diffusivity [m^2/s], represented as `alphat` in OpenFOAM; `Conc` is the pollutant concentration [kg/m^3]; ε is the turbulent kinetic energy dissipation rate [m^2/s^3], represented as `epsilon` in OpenFOAM; k is the turbulent kinetic energy [m^2/s^2]; ν_t is the turbulent viscosity [m^2/s], represented as `nut` in OpenFOAM; p is the static pressure [kg/ms^2]; p_{rgh} is the total hydrostatic pressure [kg/ms^2], represented as `p_rgh` in OpenFOAM; T is the temperature [K]; and u is the air velocity, represented as `U` in OpenFOAM [m/s^2].

`zeroGradient` applies a zero-gradient condition from the patch internal field onto the patch faces such that

$$\frac{\partial}{\partial n}\phi = 0. \tag{10}$$

`fixedFluxPressure` sets the pressure gradient to the provided value such that the flux on the boundary is that specified by the velocity boundary condition. `noSlip` fixes the velocity as zero at walls.

4.1.1.1. Wall Functions

All surfaces were modeled by wall functions to represented physical boundaries, the simplest case. Wall functions provide larger meshes near walls to accurately predict the velocity gradient across boundary layer without necessitating very fine mesh resolution near the walls. α_t was represented by the `alphatJayatillekeWallFunction` boundary condition, which describes the wall using the Jayatilleke P-function, defined in Equation 11, which accounts for the resistance to heat transfer across the viscous sublayer (Malin, 1987).

$$P = 9.24(\beta^{\frac{3}{4}} - 1)(1 + 0.28e^{-0.007\beta}), \tag{11}$$

where P is the P-function [-] and $\beta = \frac{\sigma_l}{\sigma_t}$, where σ_l and σ_t are the dimensionless laminar and turbulent Prandtl/Schmidt numbers, respectively. It follows that the dimensionless near-wall temperature (T^+) is

$$T^+ = \sigma_t(u^+ + P), \tag{12}$$

where u^+ is the dimensionless near-wall velocity defined using the universal Law of the Wall for momentum transfer, described in detail in Section 4.2.3.

ε was represented by an `epsilonWallFunction` boundary condition, which provides a wall constraint on ε for low- and high-Re turbulence models. Applying the stepwise switch (discontinuous) method to blend the ε predictions for the viscous and inertial sublayers, if $y^+ < y^+_{lam}$, then $\varepsilon = \varepsilon_{vis}$, and if $y^+ \geq y^+_{lam}$, then $\varepsilon = \varepsilon_{log}$. ε_{vis} is ε computed by the viscous sublayer assumptions [m^2/s^3], defined as

$$\varepsilon_{vis} = 2wk\frac{\nu_w}{y^2}, \tag{13}$$

where w is the cell-corner weights [-], k is the turbulent kinetic energy [m^2/s^2], ν_w is the kinematic viscosity of the fluid near the wall [m^2/s], and y is the wall-normal distance [m]. ε_{log} is ε computed by the inertial sublayer assumptions [m^2/s^3], defined as

$$\varepsilon_{log} = wC_\mu\frac{k^{\frac{3}{2}}}{\nu_{t_w}y}, \tag{14}$$

where C_μ is the empirical model constant [-] and ν_{t_w} is the turbulent viscosity near the wall [m^2/s].

k was represented by the `kqRWallFunction` boundary condition, which provides a simple wrapper around the zero-gradient condition for the cases of high Re (turbulent) flow using wall functions.

ν_t was represented by the `nutkWallFunction` boundary condition, which provides a wall constraint on ν_t based on k for low- and high-Re turbulence models, expressed as

$$\nu_t = f_{blend}\left(\nu_{t_{vis}}, \nu_{t_{log}}\right) \tag{15}$$

with

$$\nu_{t_{vis}} = 0 \tag{16}$$

$$\nu_{t_{log}} = \nu_w\left(\frac{y^+\kappa}{\ln(Ey^+)} - 1\right) \tag{17}$$

$$y^+ = C_\mu^{\frac{1}{4}} y \frac{\sqrt{k}}{\nu_w}, \tag{18}$$

where f_{blend} is a wall-function blending operator between the viscous and inertial sublayer contributions, $\nu_{t_{vis}}$ is ν_t computed by the viscous sublayer assumptions [m^2/s], $\nu_{t_{log}}$ is ν_t computed by the inertial sublayer assumptions [m^2/s], ν_w is the kinematic viscosity of fluid near wall [m^2/s], y^+ is the estimated wall-normal height of the cell center in wall units, and E is the wall roughness parameter [-].

4.1.2. Schemes

The temporal scheme (`ddtSchemes`) was a Euler implicit time scheme

$$\frac{\partial}{\partial t}(\phi) = \frac{\phi - \phi^0}{\Delta t}. \tag{19}$$

For the spatial schemes, the gradient scheme (`gradSchemes`) was least-squares, which calculates the cell gradient using least squares.

All divergence schemes (`divSchemes`) were Gauss upwind, except for `div((nuEff*dev2(T(grad(U)))))`, which was Gauss linear. Gauss upwind, defined in Equation 20, is first order and bounded. It sets the face value according to the upstream value and is equivalent to assuming that the cell values are isotropic (same in all directions) with a value that represents the average value. Its normalized variable diagram is shown in Figure 3.

$$\phi_f = \phi_c \tag{20}$$

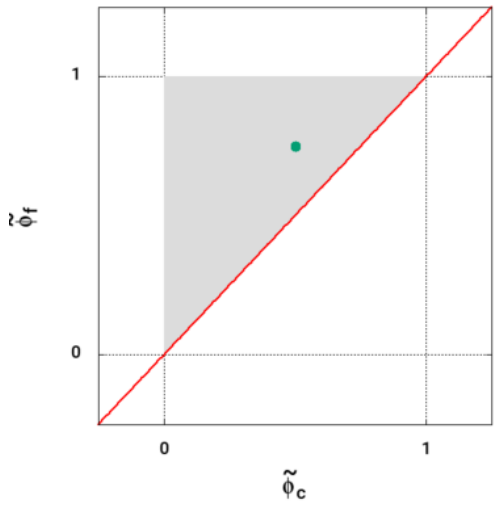


Figure 3. Normalized variable diagram for upwind divergence scheme (*Upwind Divergence Scheme*, 2017).

Gauss linear, defined in Equation 21, is second order and unbounded. It is often used for isotropic meshes due to low dissipation. Its normalized variable diagram is shown in Figure 4.

$$\phi_f = 0.5(\phi_c + \phi_d) \tag{21}$$

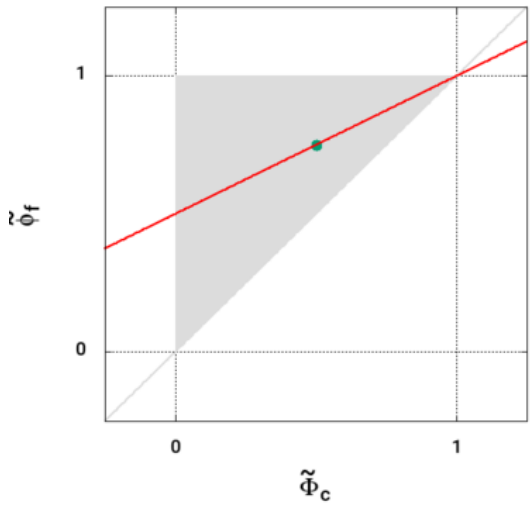


Figure 4. Normalized variable diagram for linear divergence scheme (*Linear Divergence Scheme*, 2017).

The Laplacian scheme (`laplacianSchemes`) was Gauss linear corrected, which is unbounded, second order, and conservative. The interpolation scheme (`interpolationSchemes`) was linear (central differencing). The surface-normal gradient scheme (`snGradSchemes`) was corrected, an explicit central-difference scheme with non-orthogonal correction defined as

$$\nabla_f^{\perp} Q = \alpha \frac{Q_P - Q_N}{|\boldsymbol{d}|} + \left(\hat{\boldsymbol{n}} - \alpha \hat{\boldsymbol{d}}\right) \cdot (\nabla Q)_f, \tag{22}$$

where $\alpha = \frac{1}{\cos(\theta)}$. The first term is the implicit scheme and the second is the explicit correction.

4.1.3. Computational Grid

The geometry was meshed using a coarse, uniform grid, as shown in Figure 5, to reduce computational intensity and simulation runtime.

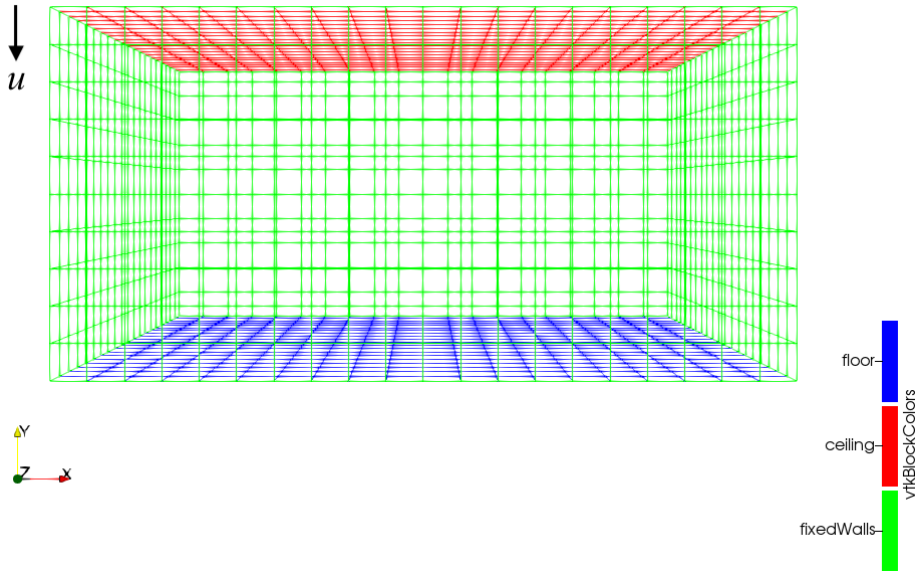
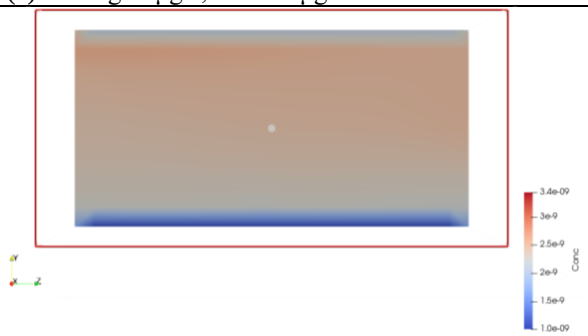
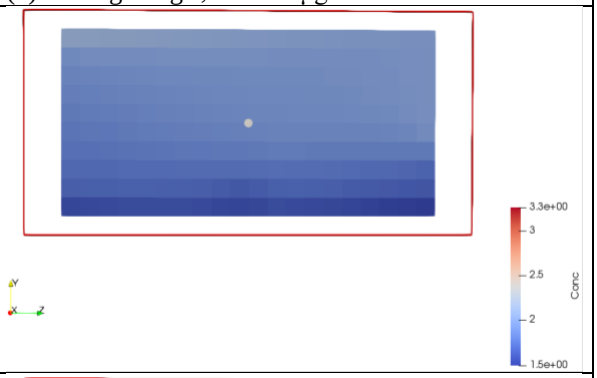
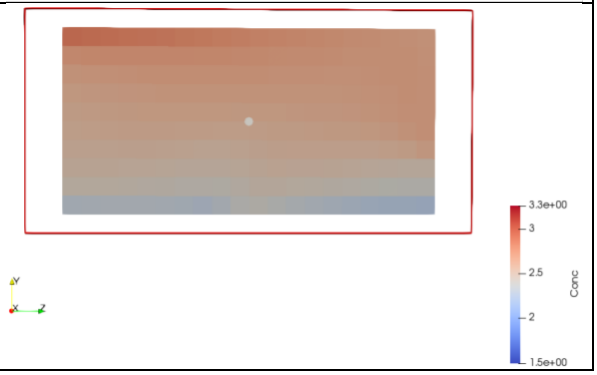


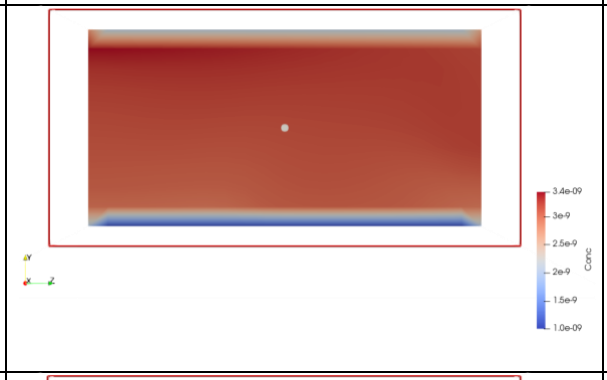
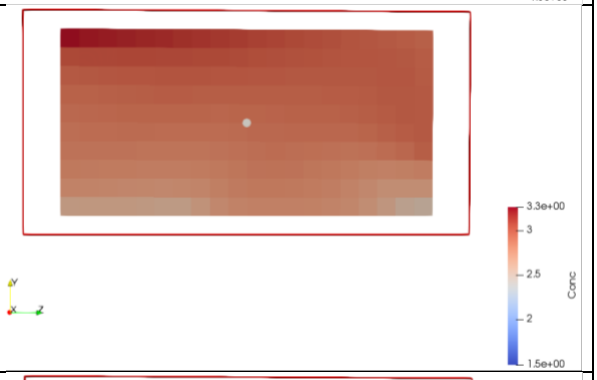
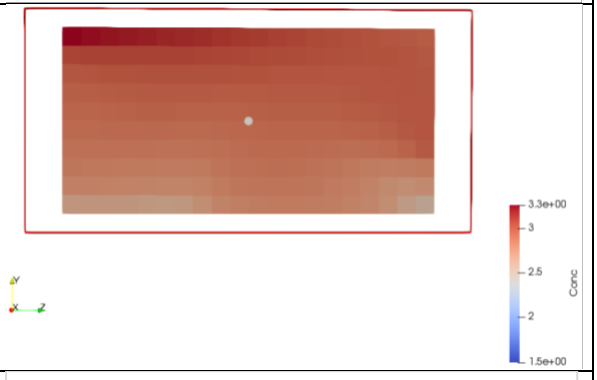
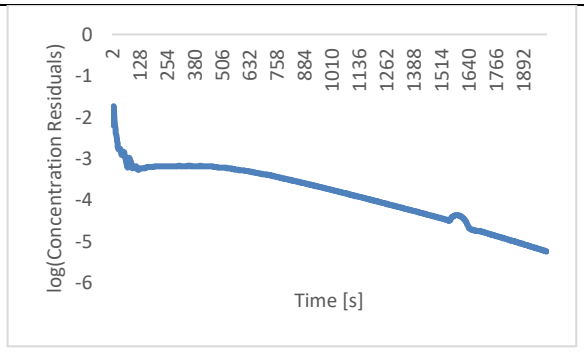
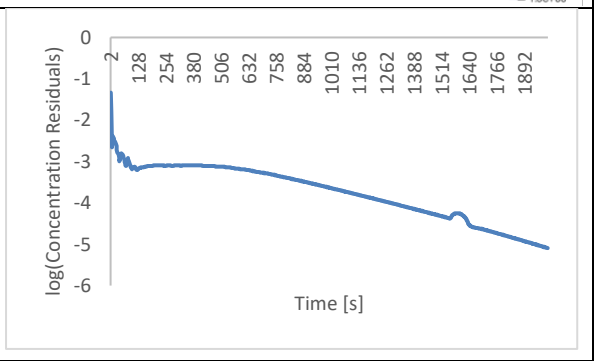
Figure 5. Meshed box in Paraview.

4.1.4. Results

The results of the test case converged for both small and large concentration gradients, as shown in Table 1. Convergence is demonstrated by the residuals approaching zero as time increases. Only the residuals for concentration are shown in Table 1, but similar results were achieved for the other variables.

Table 1. Test case results for (a) small and (b) large concentration gradient. Semi-log plots of residuals show that residuals are below 10^{-5} for both cases.

	(a) Ceiling: 2 µg/s, floor: 1 µg/s	(b) Ceiling: 2 kg/s, floor: 1 µg/s
0s		
800s		

1200s		
1600s		
2000s		
Residuals		

After the run of the test case verified the solver, the solver was applied to a more realistic case: buildings in an urban canyon.

4.2. Application: Urban Canyon

4.2.1. Parametric Urban Canyon Model

A parametric model of a symmetric urban canyon, shown in Figure 6, was modeled in Grasshopper, a visual programming environment that runs within the Rhinoceros 3D computer-aided design (CAD) application. This model allows building dimensions to be easily changed to explore their effect on the spread of air pollution in an urban canyon. It consists of two identical buildings, separated by a street, with four modifiable parameters: X, the building width; Y, the building height (or H, the canyon height); Z, the building depth (or L, the length of the street canyon); and D, the width of the street, defined as the distance between the buildings (or W, the canyon width). The left building is defined as a rectangular box with the front left point $(-\left(X+\frac{D}{2}\right), \frac{Y}{2}, 0)$ and the back right point $(-\frac{D}{2}, -\frac{Y}{2}, Z)$. The right building is defined as a rectangular box with the front left point $(\frac{D}{2}, \frac{Y}{2}, 0)$ and the back right point $(X+\frac{D}{2}, -\frac{Y}{2}, Z)$.

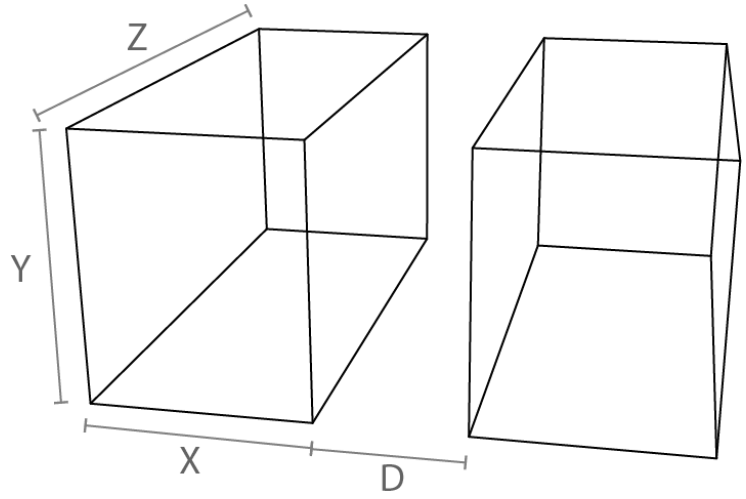


Figure 6. Model of urban canyon labeled with dimensions.

Vardoulakis et al. (2003) classify street canyons into three categories based on the aspect ratio, $\frac{Y}{D}$:

1. Regular canyon: aspect ratio ≈ 1
2. Avenue canyon: aspect ratio < 0.5
3. Deep canyon: aspect ratio ≈ 2

These three categories can be further sub-classified by the distance between two major intersections along the street, defined as the length of the street canyon (Z):

1. Short canyon: $\frac{Z}{Y} \approx 3$
2. Medium canyon: $\frac{Z}{Y} \approx 5$
3. Long canyon: $\frac{Z}{Y} \approx 7$

By creating a parametric model of urban canyon geometry, the effect of these different canyon dimensions on air pollution dispersion can be evaluated. The geometric parametrization was implemented in Grasshopper with the recipe shown in Figure 7. The two boxes were merged into a single boundary representation (BREP). This means that the boxes became a solid represented as a collection of connected surface elements, which define the boundary between interior and exterior points. This single BREP was meshed as the geometry for the CFD model.

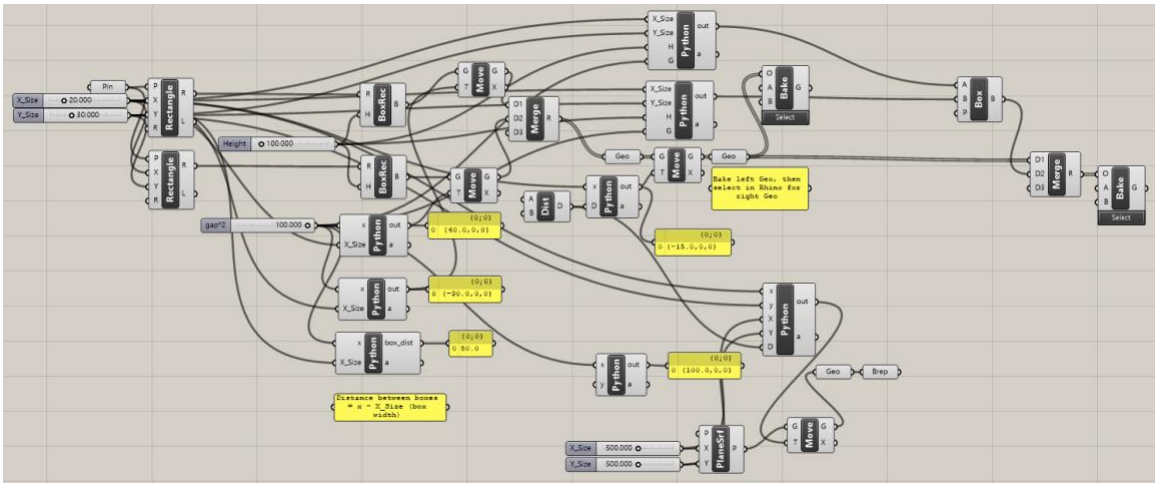


Figure 7. Grasshopper script for parametric urban canyon model.

4.2.2. Virtual Wind Tunnel

The modeled urban canyon was simulated in a wind tunnel to ensure proper boundary conditions. The dimensions of the wind tunnel were selected relative to the building height, based on the Best Practice Guidelines for the CFD Simulation of Flows in the Urban Environment from COST Action 732 (Schatzmann & Britter, 2011). As shown in Figure 8, the wind tunnel was created by drawing a buffer of 5 times the building height on all sides except for the outlet, which had a buffer of 15 times the building height, and the bottom, which was coplanar with the ground.

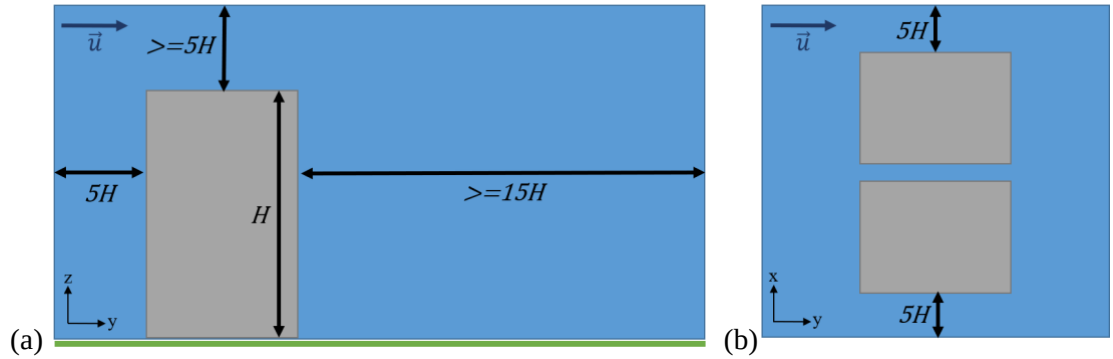
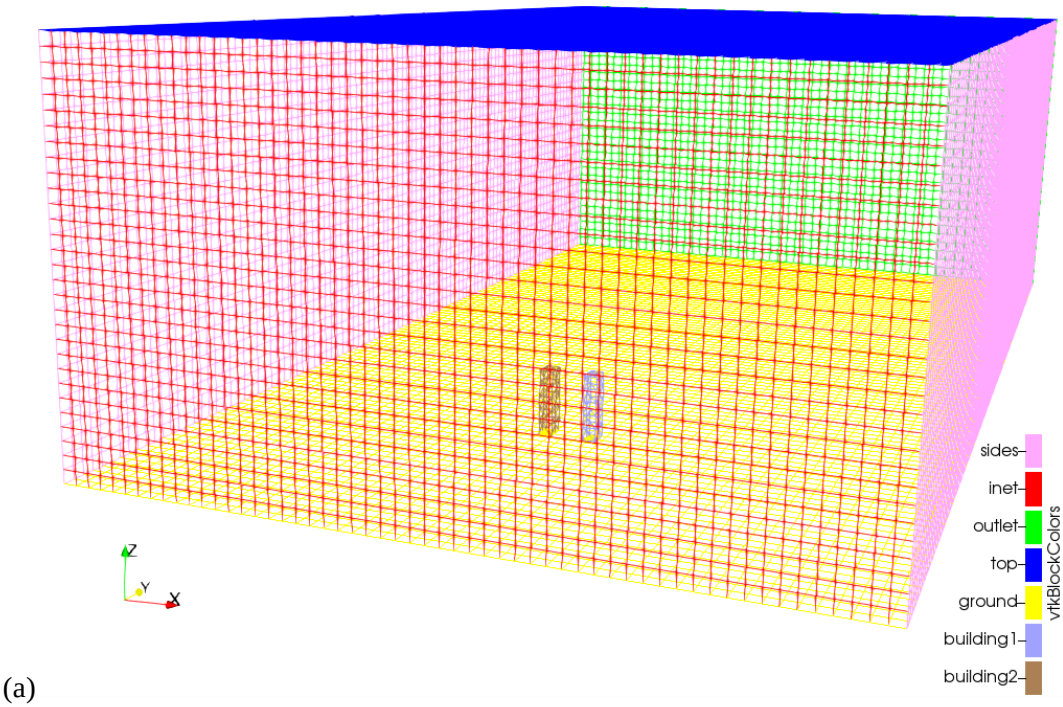


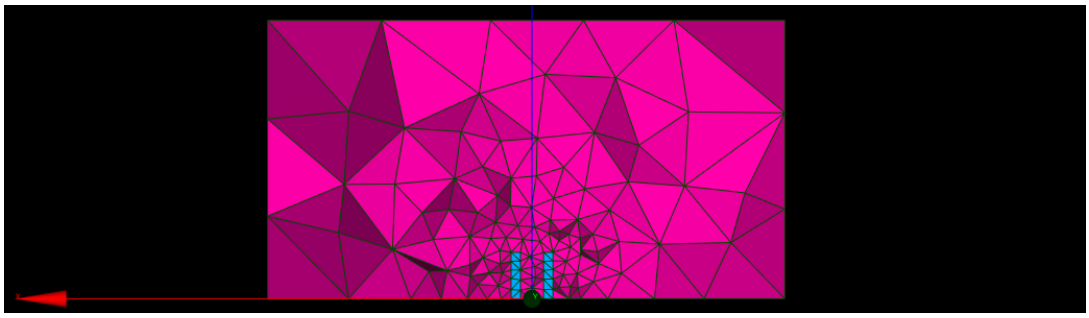
Figure 8. (a) Side and (b) top view schematics of simulated building (grey) in virtual wind tunnel (blue), not to scale. Ground highlighted in green.

4.2.3. Computational Grid

The full geometry (buildings and wind tunnel) was baked in Grasshopper, exported as a STEP (.stp) file, and imported into SALOME version 9.7.0 (*Salome Platform*, 2022) for meshing. SALOME is an open-source scientific computing environment developed by Électricité de France (EDF), Open Cascade, and the French Alternative Energies and Atomic Energy Commission (CEA) that is often used with OpenFOAM. Within SALOME, the geometry was separated from the air, as CFD only evaluates fluid flow, excluding changes to solids. Each of the boundaries (wind tunnel inlet, outlet, sides, top, ground, and buildings) were then identified to assign boundary conditions in OpenFOAM. A coarse, uniform grid, as shown in Figure 9a, was selected for the initial test cases to reduce computational intensity and simulation runtime. This mesh was then improved to a non-uniform grid in a later trial, as shown in Figure 9b.



(a)



(b)

Figure 9. (a) Meshed regions with non-uniform grid shown in Paraview and (b) slice of non-uniform mesh in SALOME.

Uniform grids may not accurately predict the velocity gradient across the boundary layer, which forms near wall surfaces. Ideally, for turbulent flows, the first cell from the wall lies within the very thin viscous sublayer. However, for complex flows and/or geometries, achieving this goal dramatically increases the computational time due to the required fineness of the mesh near the wall. A wall function was thus employed for the boundary condition near the wall to enable the use of a relatively sparser mesh near the wall, reducing the computational time. For the wall function to be appropriate, the non-dimensional distance from the wall to the first node from the wall (y^+) must be selected to ensure that the flow is simulated within the appropriate region of the turbulent boundary layer. Since a k-ε model is used for the CFD simulation under conditions without severe pressure gradients or strong non-equilibrium flows, standard (rather than non-equilibrium) wall functions were implemented. These wall functions are valid for $30 > y^+ > 300$, the fully turbulent zone, as shown in Figure 10. All cases should thus ensure $30 > y^+ > 300$. In this zone, the log-law, defined in Equation 23, holds.

$$u^+ = \frac{1}{\kappa}\ln(y^+) + C, \tag{23}$$

where u^+ is the non-dimensionalized velocity at a non-dimensionalized distance of y^+ parallel from the wall (defined in Equation 24), κ is the von Kármán constant (0.41), and C is a constant, which is approximately 5.45 for smooth walls.

$$y^+ = \frac{yu_\tau}{\nu}, \tag{24}$$

where y is the absolute distance from the wall [m], u_τ is the friction velocity, and ν is the kinematic viscosity [m^2/s].

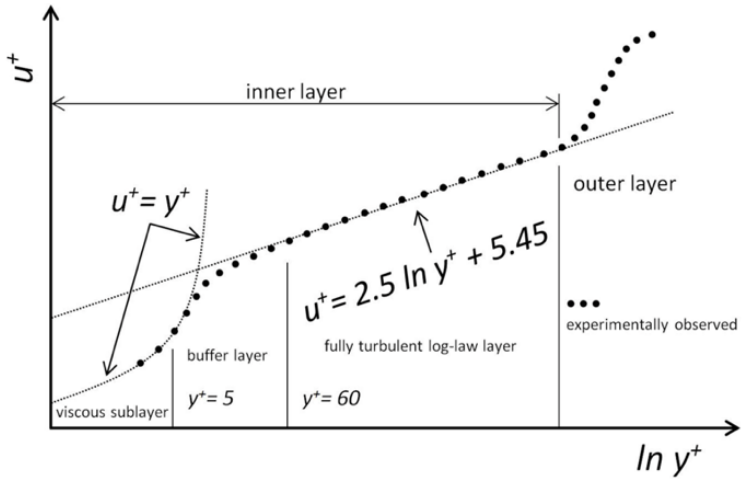


Figure 10. The Law of the Wall (Mehta et al., 2018).

4.2.4. Simplified Conditions

The urban canyon geometry was placed inside a virtual wind tunnel (i.e., a box) with the simplified boundary conditions described in Table 2, mimicking those in Table 1. Additionally, the same schemes were used as those used in the simple box test case.

Table 2. List of simplified boundary conditions for simulated wind tunnel.

boundary Field	α_t	$Conc$	ε	k	ν_t	p	p_{rgh}	T	u
Inlet	calculated	fixed Value	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	fixed Value	fixed Value
Outlet	calculated	fixed Value	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	zero Gradient	noSlip
Top	alphatJayatilleke WallFunction	zero Gradient	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	fixed Value	noSlip
Ground	alphatJayatilleke WallFunction	zero Gradient	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	fixed Value	noSlip
Sides	alphatJayatilleke WallFunction	zero Gradient	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	zero Gradient	noSlip
Buildings	alphatJayatilleke WallFunction	zero Gradient	epsilonWall Function	kqRWall Function	nutkWall Function	calculated	fixedFlux Pressure	fixed Value	noSlip

4.2.5. Results

The results of the urban canyon case with simplified boundary conditions are shown in Figure 11. Convergence is demonstrated by the residuals approaching zero as time increases. Only the residuals for concentration as shown in Figure 11c, but similar results were achieved for the other variables. While the solver did converge for this test case, the results are not especially useful due to the coarse mesh, especially around the buildings.

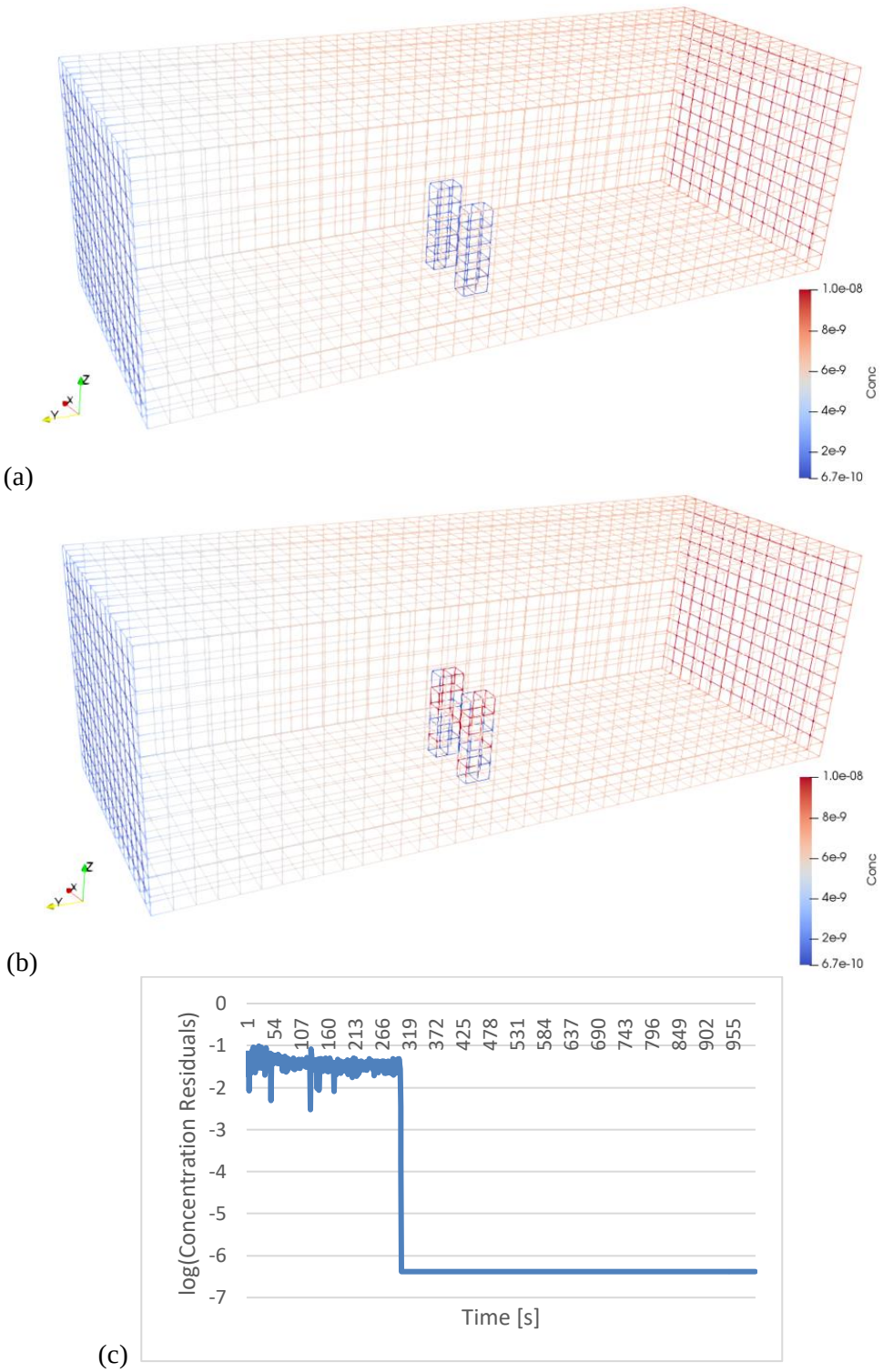


Figure 11. Results of urban canyon simulation with simplified boundary condition after (a) 400s and (b) 2000s. (c) shows the semi-log concentration residuals plot.

Future work would model an urban canyon with boundary conditions that more accurately model airflow within the atmospheric boundary layer (ABL). Future work is described in more detail in Section 5.

4.2.6. ABL Boundary Conditions

The urban canyon geometry was placed inside of a virtual wind tunnel with the boundary conditions described in Table 3.

Table 3. List of boundary conditions for simulated wind tunnel. * indicates inlet boundary conditions in the ABL.

boundary Field	α_t	$Conc$	ε	k	ν_t	p	p_{rgh}	T	u
Inlet	calculated	fixed Value	*	*	calculated	total Pressure	fixedFlux Pressure	fixedValue	*
Outlet	calculated	zero Gradient	zero Gradient	zero Gradient	calculated	zero Gradient	zero Gradient	zero Gradient	zero Gradient
Top	symmetry	symmetry	symmetry	symmetry	symmetry	symmetry	symmetry	symmetry	symmetry
Ground	alphatJayatilleke WallFunction	zero Gradient	epsilonWall Function	kqRWall Function	nutkWall Function	zero Gradient	fixedFlux Pressure	fixedValue	noSlip
Sides	symmetry	symmetry	symmetry	symmetry	symmetry	symmetry	symmetry	symmetry	symmetry
Buildings	alphatJayatilleke WallFunction	zero Gradient	epsilonWall Function	kqRWall Function	nutkWall Function	zero Gradient	fixedFlux Pressure	fixedValue	noSlip

where the inlet velocity is

$$u(z) = \frac{u_\tau}{\kappa} \ln\left(\frac{z + z_0}{z_0}\right), \tag{25}$$

where κ is the von Karman constant (0.41); z is the height [m] at which the ground-normal streamwise flow speed profile, u [m/s], is calculated; z_0 is the aerodynamic roughness length [m], which defines the boundary with the roughness sublayer; and C_μ is the dimensionless turbulent viscosity constant (0.09). z_0 varies by landscape and is taken as 0.005 m, the accepted value for unobstructed flow on unvegetated land (World Meteorological Organization, 2008).

4.2.6.1. Inlet

The inlet boundary conditions were constructed to model flow development in the atmospheric boundary layer (Richards & Hoxey, 1993), shown in Figure 12. In the urban canopy layer, air is highly turbulent with a logarithmic inlet velocity profile. The log flow profile does not extend to ground level ($z = 0$) but only a height d_0 above the ground.

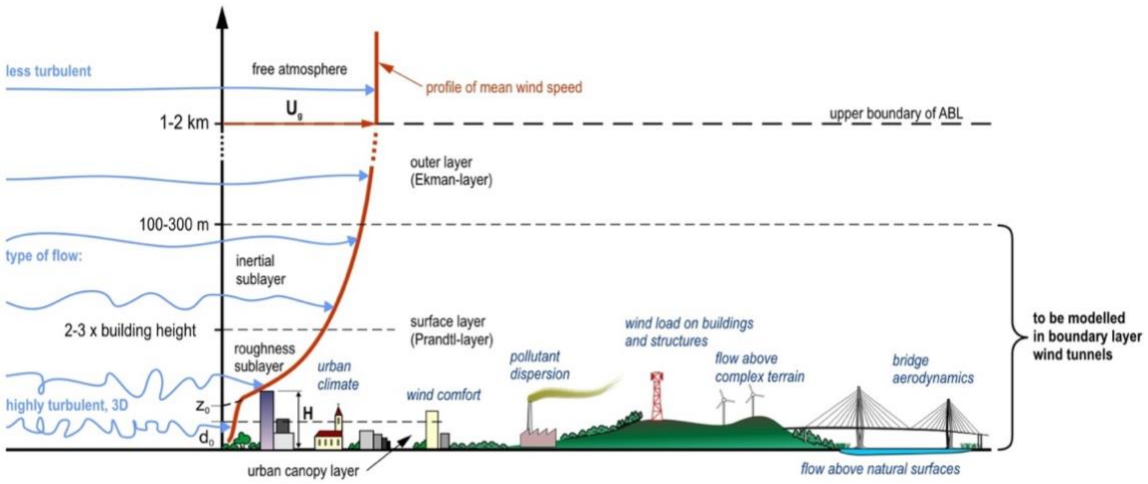


Figure 12. Properties and sublayers of the atmospheric boundary layer (ABL), including the inlet velocity profile (*Establishment of an Atmospheric Flow Laboratory*, 2016).

4.2.6.2. Top

The symmetry boundary condition was used for all variables at the top of the domain because the area of study falls within the surface boundary layer (SBL) and the top of the domain is very far away from the area of interest. Should the area of study fall within the convective boundary layer (CBL), the top of the domain would act as an outlet to model vertical motion due to buoyancy.

5. Discussion, Conclusions, and Further Work

This thesis developed an open-source CFD solver for pollutant transport, building off an existing OpenFOAM solver. This solver was then demonstrated to converge for example cases with simple boundary conditions, largely consisting of wall functions. The cases used geometry generated parametrically in Grasshopper, showing the potential for this solver to be part of a streamlined design tool, similar to Eddy3D and Butterfly, to enable designers to integrate air pollution considerations into their designs, beyond just air velocity and pressure.

Subsequently, the next step in this work would be to integrate this pollutant solver into Grasshopper components (most likely programmed in C#) compatible with Eddy3D and Butterfly. These two Grasshopper plugins already connect Rhino geometry and Grasshopper scripting to OpenFOAM through blueCFD-Core running on Windows, but only calculate wind velocity and pressure, rather than pollutant concentrations. Eddy3D provides inlet boundary conditions for both uniform and ABL flow, while Butterfly provides all other boundary conditions (wall functions, `zeroGradient`, `fixedValue`, and `calculated`) and meshing options. Integrating air pollution would involve connecting Grasshopper to the solver with pollutant transport, `buoyantBoussinesqPimpleFoamS`, instead of `simpleFoam`, and defining the necessary concentration boundary conditions.

The most significant challenge when running the OpenFOAM simulations with the pollutant solver was floating point errors (`sigFpe`), which occur when the solver must do something impossible with a floating-point number (positive or negative whole number with a decimal point), such as divide by zero. The two most important factors to consider to avoid floating point errors in CFD are boundary conditions and meshing.

Appropriate boundary conditions are necessary to ensure realistic conditions, but also to ensure convergence. Having too many boundary conditions that are being calculated, or too many Neumann-like boundary conditions, such as `symmetry` or `zeroGradient`, which require the gradient to be zero or two values to be the same without forcing an absolute number like a `fixedValue` boundary condition, could make the matrix A in the linear system being solved, $Ax = b$, close to being singular. If A is singular, it is non-invertible, so the system cannot be solved as $x = A^{-1}b$, causing the OpenFOAM solution to diverge as A becomes singular.

Meshing must be properly performed to identify all surfaces with distinct boundary conditions and exclude all non-fluid regions. Additionally, mesh sizing in all directions (space step: Δx, Δy, and Δz [m]) must be chosen with the time step (`deltaT` [s]) such that the CFL condition is satisfied. The CFL condition is important because it is a necessary condition for convergence for the partial differential equations implemented in this solver. Thus, if the CFL condition is not satisfied, the solution will diverge. In three dimensions, the CFL condition is defined as

$$C = \frac{u_x \Delta t}{\Delta x} + \frac{u_y \Delta t}{\Delta \mathrm{y}} + \frac{u_z \Delta t}{\Delta \mathrm{z}} \leq C_{max} = 1, \tag{26}$$

where C is the dimensionless Courant number, u is the velocity [m/s], Δt is the time step [s], and Δx, Δy, and Δz are the length intervals [m] in the x-, y-, and z-directions, respectively. In OpenFOAM, `deltaT` can either be defined as a constant or chosen to be runtime modifiable with a maximum Courant number (`maxCo`) of 1.

While this thesis demonstrated the applicability of the pollutant solver for simplified boundary conditions largely composed of wall functions, it has not been tested for more complex, realistic conditions, such as an inlet ABL flow velocity or symmetry boundary conditions for the wind tunnel. The geometry tested was also quite simple, composed of boxes, so future cases could include more complex geometry, such as trees or additional surrounding buildings. Further work is necessary to validate this solver, especially for conditions that better represent an urban canyon environment. Validation could include tests with different geometry and boundary conditions, finer meshing, and longer time scales. The results of these tests should then be compared to results from more established air pollution models, or from representative experiments, either at scale or in physical wind tunnels.

Acknowledgments

Thank you to Professor Les Norford for serving as my thesis advisor and providing feedback throughout this process. Thank you to Dr. Shabnam Raayai-Ardakani at the Rowland Institute at Harvard University and to Dr. Lup Wai Chew at the National University of Singapore for assistance with OpenFOAM, and to Sarah Mokhtar for assistance with Butterfly and Eddy3D. Further thanks to the 2.29/2.290: Numerical Fluid Mechanics course staff, Professor Pierre Lermusiaux and Teaching Assistants Wael Ali and Manan Doshi, for teaching me about numerical methods related to fluid mechanics simulations.

Appendix A – buoyantBoussinesqPimpleFoamS.C

Modifications to buoyantBoussinesqPimpleFoam.C are highlighted in orange.

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    buoyantBoussinesqPimpleFoamS

Description
    Modified buoyantBoussinesqPimpleFoam solver to include
    a passive scalar transport equation with
    turbulent Schmidt number.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "radiationModel.H"
#include "fvOptions.H"
#include "pimpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "postProcess.H"
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createFields.H"
    #include "createFvOptions.H"
    #include "createTimeControls.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"
    #include "initContinuityErrs.H"

    turbulence->validate();
```

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

    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "UEqn.H"
            #include "TEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                laminarTransport.correct();
                turbulence->correct();
            }
        }

        // *** Passive Scalar Transport ***
        // Create a scalar field with an effective mass diffusivity
        volScalarField DTT ("DTT", DT + turbulence->nut()/Sct);
        // Define scalar transport equation
        fvScalarMatrix ConcEqn
        (
        fvm::ddt(Conc)
        + fvm::div(phi, Conc)
        - fvm::laplacian(DTT, Conc)
        ) ;
        ConcEqn.solve();
        // *** End Passive Scalar Transport ***

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}


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

Appendix B – createFields.H

Modifications to original createFields.H are highlighted in orange.

```
Info<< "Reading thermophysical properties\n" << endl;

Info<< "Reading field T\n" << endl;
volScalarField T
(
    IOobject
    (
        "T",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

// *** Create passive scalar field ***
Info<< "Reading field Conc\n" << endl;
volScalarField Conc
(
    IOobject
    (
        "Conc",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
) ;
// *** End passive scalar field **

Info<< "Reading field p_rgh\n" << endl;
volScalarField p_rgh
(
    IOobject
    (
        "p_rgh",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
```

```
    mesh
);

#include "createPhi.H"
#include "readTransportProperties.H"

Info<< "Creating turbulence model\n" << endl;
autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, laminarTransport)
);

// Kinematic density for buoyancy force
volScalarField rhok
(
    IOobject
    (
        "rhok",
        runTime.timeName(),
        mesh
    ),
    1.0 - beta*(T - TRef)
);

// kinematic turbulent thermal conductivity m2/s
Info<< "Reading field alphat\n" << endl;
volScalarField alphat
(
    IOobject
    (
        "alphat",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


#include "readGravitationalAcceleration.H"
#include "readhRef.H"
#include "gh.H"


volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    p_rgh + rhok*gh
);

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell
```

```
(
    p,
    p_rgh,
    pimple.dict(),
    pRefCell,
    pRefValue
);

if (p_rgh.needReference())
{
    p += dimensionedScalar
    (
        "p",
        p.dimensions(),
        pRefValue - getRefCellValue(p, pRefCell)
    );
}

mesh.setFluxRequired(p_rgh.name());

#include "createMRF.H"
#include "createIncompressibleRadiationModel.H"
```

Appendix C – readTransportProperties.H

Modifications to original readTransportProperties.H are highlighted in orange.

```
singlePhaseTransportModel laminarTransport (U, phi);

// Thermal expansion coefficient [1/K]
dimensionedScalar beta
(
"beta",
dimless/dimTemperature,
laminarTransport
);

// Reference temperature [K]
dimensionedScalar TRef ("TRef", dimTemperature, laminarTransport);

// Laminar Prandtl number []
dimensionedScalar Pr("Pr", dimless, laminarTransport);

// Turbulent Prandtl number []
dimensionedScalar Prt("Prt", dimless, laminarTransport);

// Mass Diffusivity [m²/s]
dimensionedScalar DT("DT", dimLength*dimLength/dimTime, laminarTransport);

// *** Turbulent Schmidt Number [] ***
dimensionedScalar Sct("Sct", dimless, laminarTransport);
```

References

Air pollution. (n.d.). World Health Organization. Retrieved December 8, 2020, from https://www.who.int/westernpacific/health-topics/air-pollution

Blocken, B. (2015). Computational Fluid Dynamics for urban physics: Importance, scales, possibilities, limitations and ten tips and tricks towards accurate and reliable simulations. *Building and Environment*, *91*, 219–245. https://doi.org/10.1016/j.buildenv.2015.02.015

Caretto, L. S., Gosman, A. D., Patankar, S. V., & Spalding, D. B. (1973). Two calculation procedures for steady, three-dimensional flows with recirculation. In H. Cabannes & R. Temam (Eds.), *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics* (pp. 60–68). Springer. https://doi.org/10.1007/BFb0112677

Chew, L. W., & Norford, L. K. (2018). Pedestrian-level wind speed enhancement in urban street canyons with void decks. *Building and Environment*, *146*, 64–76. https://doi.org/10.1016/j.buildenv.2018.09.039

Chronis, A., Dubor, A., Cabay, E., & Roudsari, M. S. (n.d.). *Integration of CFD in Computational Design*. 10.

Cimorelli, A., Perry, S., Venkatram, A., Weil, J., Paine, R., Wilson, R., Lee, R., Peters, W., Brode, R., & Paumier, J. (2004). *AERMOD: description of model formulation, US Environmental Protection Agency*. EPA-454/R-03-004.

Eeftens, M., Beekhuizen, J., Beelen, R., Wang, M., Vermeulen, R., Brunekreef, B., Huss, A., & Hoek, G. (2013). Quantifying urban street configuration for improvements in air pollution models. *Atmospheric Environment*, *72*, 1–9. https://doi.org/10.1016/j.atmosenv.2013.02.007

Elfverson, D., & Lejon, C. (2021). Use and Scalability of OpenFOAM for Wind Fields and Pollution Dispersion with Building- and Ground-Resolving Topography. *Atmosphere*, *12*(9), 1124. https://doi.org/10.3390/atmos12091124

Establishment of an Atmospheric Flow Laboratory. (2016, August 16). Establishment of an Atmospheric Flow Laboratory. https://bmeafl.com/the-project-proposal/

Garcia-Alcaide, V. M., Pallejà Cabré, S., Castilla, R., Gamez-Montero, P., Romeu, J., Pàmies, T., Amate, J., & Milan, N. (2017). Numerical study of the aerodynamics of sound sources in a bass-reflex port. *Engineering Applications of Computational Fluid Mechanics*, *11*, 210–224. https://doi.org/10.1080/19942060.2016.1277166

Huang, Y., Lei, C., Liu, C.-H., Perez, P., Forehead, H., Kong, S., & Zhou, J. L. (2021). A review of strategies for mitigating roadside air pollution in urban street canyons. *Environmental Pollution*, *280*, 116971. https://doi.org/10.1016/j.envpol.2021.116971

Issa, R. I. (1986). Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics*, *62*(1), 40–65. https://doi.org/10.1016/0021-9991(86)90099-9

Kastner, P., & Dogan, T. (2021). Eddy3D: A toolkit for decoupled outdoor thermal comfort simulations in urban areas. *Building and Environment*, 108639. https://doi.org/10.1016/j.buildenv.2021.108639

Launder, B. E., & Spalding, D. B. (1974). The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, *3*(2), 269–289. https://doi.org/10.1016/0045-7825(74)90029-2

Longo, R., Fürst, M., Bellemans, A., Ferrarotti, M., Derudi, M., & Parente, A. (2019). CFD dispersion study based on a variable Schmidt formulation for flows around different configurations of ground-mounted buildings. *Building and Environment*, *154*, 336–347. https://doi.org/10.1016/j.buildenv.2019.02.041

Maffessanti, V. (2019). Wind and Urban Spaces. Evaluation of a CFD Parametric Framework for Early-Stage Design. *Building Simulation Applications BSA 2019*, 16. https://www.ladybug.tools/assets/pdf/Wind_and_Urban_Spaces.pdf

Malin, M. R. (1987). On the calculation of heat transfer rates in fully turbulent wall flows. *Applied Mathematical Modelling*, *11*(4), 281–284. https://doi.org/10.1016/0307-904X(87)90143-0

Mayer, H. (1999). Air pollution in cities. *Atmospheric Environment*, *33*(24), 4029–4037. https://doi.org/10.1016/S1352-2310(99)00144-2

Mehta, D., Thota Radhakrishnan, A., van Lier, J., & Clemens, F. (2018). A Wall Boundary Condition for the Simulation of a Turbulent Non-Newtonian Domestic Slurry in Pipes. *Water*, *10*(2), 124. https://doi.org/10.3390/w10020124

Monbureau, E. M., Heist, D. K., Perry, S. G., & Tang, W. (2020). Modeling lateral plume deflection in the wake of an elongated building. *Atmospheric Environment*, *234*, 117608. https://doi.org/10.1016/j.atmosenv.2020.117608

Murena, F., & Mele, B. (2016). Effect of balconies on air quality in deep street canyons. *Atmospheric Pollution Research*, *7*(6), 1004–1012. https://doi.org/10.1016/j.apr.2016.06.005

Nakajima, K., Ooka, R., & Kikumoto, H. (2018). Evaluation of k-ε Reynolds stress modeling in an idealized urban canyon using LES. *Journal of Wind Engineering and Industrial Aerodynamics*, *175*, 213–228. https://doi.org/10.1016/j.jweia.2018.01.034

Particulate Matter (PM) Basics. (2016, April 19). [Overviews and Factsheets]. US EPA. https://www.epa.gov/pm-pollution/particulate-matter-pm-basics

Richards, P. J., & Hoxey, R. P. (1993). Appropriate boundary conditions for computational wind engineering models using the k-ϵ turbulence model. *Journal of Wind Engineering and Industrial Aerodynamics*, *46–47*, 145–153. https://doi.org/10.1016/0167-6105(93)90124-7

Salome Platform. (2022). Salome Platform. https://www.salome-platform.org/

Santiago, J. L., Sanchez, B., Quaassdorff, C., de la Paz, D., Martilli, A., Martín, F., Borge, R., Rivas, E., Gómez-Moreno, F. J., Díaz, E., Artiñano, B., Yagüe, C., & Vardoulakis, S. (2020). Performance evaluation of a multiscale modelling system applied to particulate matter dispersion in a real traffic hot spot in Madrid (Spain). *Atmospheric Pollution Research*, *11*(1), 141–155. https://doi.org/10.1016/j.apr.2019.10.001

Schatzmann, M., & Britter, R. (2011). Quality assurance and improvement of micro-scale meteorological models. *International Journal of Environment and Pollution*, *44*(1/2/3/4), 139. https://doi.org/10.1504/IJEP.2011.038412

Shi, X., Sun, D. (Jian), Fu, S., Zhao, Z., & Liu, J. (2019). Assessing On-Road Emission Flow Pattern under Car-Following Induced Turbulence Using Computational Fluid Dynamics (CFD) Numerical Simulation. *Sustainability*, *11*(23), 6705. https://doi.org/10.3390/su11236705

Tauer, A. (2021). *CFD Modeling of Aerial Dispersion of Pollutants in Urban Environments* [Marquette University]. https://epublications.marquette.edu/cgi/viewcontent.cgi?article=1660&context=theses_open

The World's Cities in 2018—Data Booklet (ST/ESA/SER.A/417). (2018). United Nations, Department of Economic and Social Affairs, Population Division. https://digitallibrary.un.org/record/3799524?ln=en

Toja-Silva, F., Chen, J., Hachinger, S., & Hase, F. (2017). CFD simulation of CO2 dispersion from urban thermal power plant: Analysis of turbulent Schmidt number and comparison with Gaussian plume model and measurements. *Journal of Wind Engineering and Industrial Aerodynamics*, *169*, 177–193. https://doi.org/10.1016/j.jweia.2017.07.015

Toparlar, Y., Blocken, B., Maiheu, B., & van Heijst, G. J. F. (2017). A review on the CFD analysis of urban microclimate. *Renewable and Sustainable Energy Reviews*, *80*, 1613–1640. https://doi.org/10.1016/j.rser.2017.05.248

Upwind divergence scheme. (2017). OpenFOAM: User Guide V2112. https://www.openfoam.com/documentation/guides/latest/doc/guide-schemes-divergence-upwind.html

Vardoulakis, S., Fisher, B. E. A., Pericleous, K., & Gonzalez-Flesca, N. (2003). Modelling air quality in street canyons: A review. *Atmospheric Environment*, *37*(2), 155–182. https://doi.org/10.1016/S1352-2310(02)00857-9

Weller, H. G., Tabor, G., Jasak, H., & Fureby, C. (1998). A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in Physics*, *12*(6), 620. https://doi.org/10.1063/1.168744

World Meteorological Organization. (2008). *Guide to Meteorological Instruments and Methods of Observation (WMO-No. 8)* (Seventh edition). World Meteorological Organization. https://www.posmet.ufv.br/wp-content/uploads/2016/09/MET-474-WMO-Guide.pdf

Zheng, M. H., Guo, Y. R., Ai, X. Q., Qin, T., Wang, Q., & Xu, J. M. (2010). Coupling GIS with CFD modeling to simulate urban pollutant dispersion. *2010 International Conference on Mechanic Automation and Control Engineering*, 1785–1788. https://doi.org/10.1109/MACE.2010.5536018