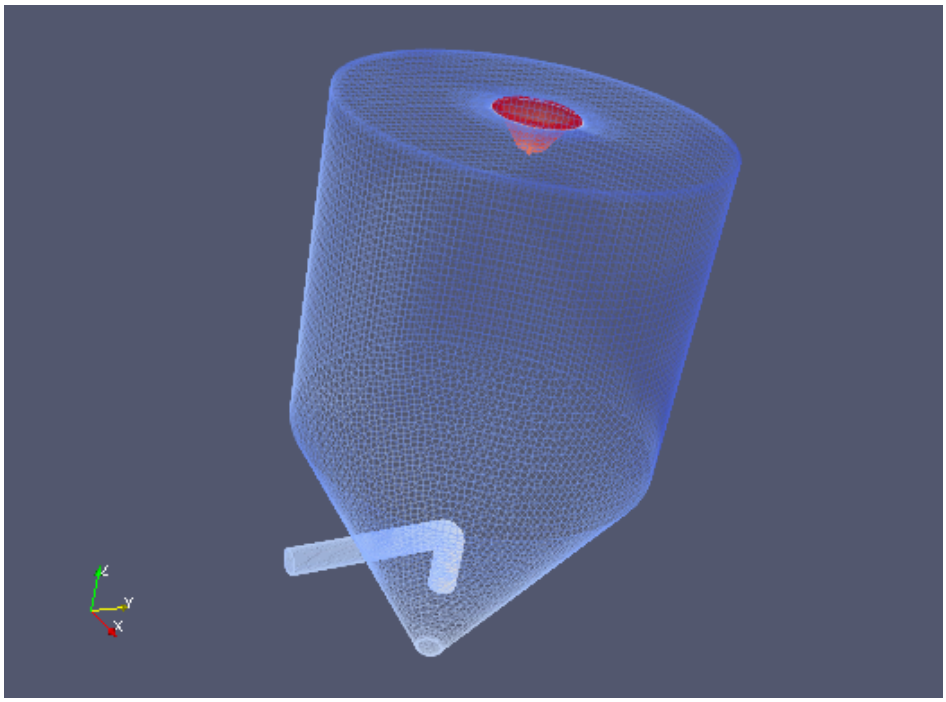
CdL in Energy and Aeronautical Engineering

Modeling of Spray Drying using Computational Fluid Dynamics with an Open-Source software

Supervisor: Jan Oscar Pralits
Co-supervisors: Matteo Colli, Joel Guerrero
Student: Stefano Pastorino

September 2016



Abstract

The work presented in this thesis is part of a newly started project aimed to develop a numerical model of a spray dryer to be used in the framework of food processing research at the department of civil, chemical and environmental engineering at the university of Genoa. In particular, it is the aim to use open source software, freely available and without license costs. The numerical platform chosen is OpenFOAM which has a large variety of physical models and good capacity to perform large scale simulations. A large part of the work has been dedicated to carefully fine tune the numerical model with existing experimental data from the literature in order to obtain a trustworthy numerical platform. Three-dimensional simulations of the turbulent flow, accounting for variable temperature fields, liquid drops as well as solid particles, with input data from experiments conducted at the chemical engineering department, have been performed on a 16 processor work station, and validated with experimental data from the literature. The work presents a large step towards a working numerical platform for spray dryer simulations using a software free-of-charge and future challenges are outlined for the continuation of the project.

Contents

1 Introduction

The topic of this thesis concerns the area of spray drying which is the transformation of droplets to dried particles; the decrease in the moisture content is obtained by feeding the droplets as a spray into a hot drying chamber. Despite spray dryers are used in a wide range of industries like food manufactures, chemical and pharmaceutical industry and other product processes, they are still designed by virtue of pilot experiments which are expensive and time consuming to measure directly. Therefore, it is currently thus important to develop a computational model for theoretical investigation allowing to predict the gas flow pattern and the particle history such as velocity, temperature, humidity, particle size and residence time. In this respect advanced two- and three-dimensional spray drying models based on Computational Fluid Dynamics (CFD) have been proposed by several academic and industrial groups [1–18] over the last few years. This approach permits to analyse problems that involve fluid flows in complex geometry through numerical methods and algorithms applied to solve equations such as conservation of mass, momentum and energy. The aim is nowadays to improve computational models to determine the size and the morphology of individual particles as accurately as possible in order to predict the quality of the final product.

For example, an important bench mark in recent literature results GEA Niro DRYNETICS^{TM}. The idea of this modern spray drying tool is to provide the solution by incorporating experimental data into the CFD software. These data collected concern single droplets of a feed to determine its actual drying properties and they are then transferred to the CFD software with the help of appropriate mathematical models, in order to simulate the drying process with a considerable accuracy. This has all been possible due to an experimental apparatus named DRYING KINETICS ANALYZER^{TM} (DKA) and based on the principle of ultrasonic levitation [36]. Therefore investigations of the influence of different spray drying parameters being relevant for the particle morphology formation are achievable by using DRYNETICS^{TM} and this also permits to obtain suitable data used for optimizing the design of new spray dryers. All this is possible thanks to the coupling between the results of DKA measurements and the commercial FLUENT^{TM} CFD software and it is very important in order to be able to modify the process parameters or the equipment design before the spray dryer is produced.

2 Description of a Spray Dryer

Spray drying is a well established method for converting liquid feed materials into a dry powder form. As mentioned in the chapter above, spray dryer is an important step to control the final product quality, in fact it usually comes at the end-point of the processing line. It has some advantages such as, rapid drying rates, a wide range of operating temperatures, short residence times and the ability to control the particle size distribution; all this characteristics depend on the spray dryer form anyway. The two main designs of spray dryers can be identified in this regard: short-form and tall-form as shown in Figure 2.1. Short-form dryers are characterized by a restrained aspect ratio meaning that the height-diameter ratio is of around 2:1 while tall-form dryers have a height-diameter ratio greater than 5:1. In the latter case dryers have less complex

flow patterns than short-form dryers, but they are afflicted by an higher percentage of particles impacting on the cylindrical wall which is a negative effect on the final product quality.

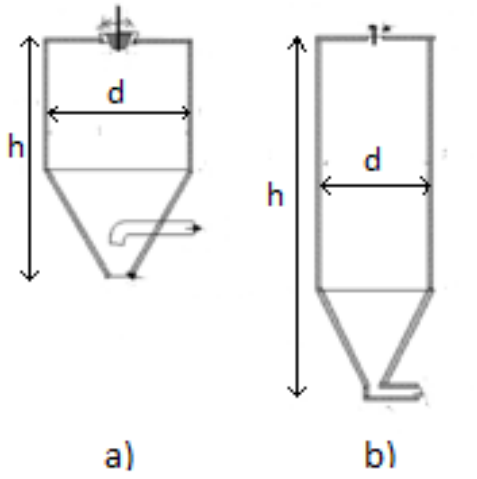


Figure 2.1: Spray dryer short-form a) and tall-form b) [23].

Spray drying involves four stages of operation for obtaining dried particles as shown in Figure 2.2:

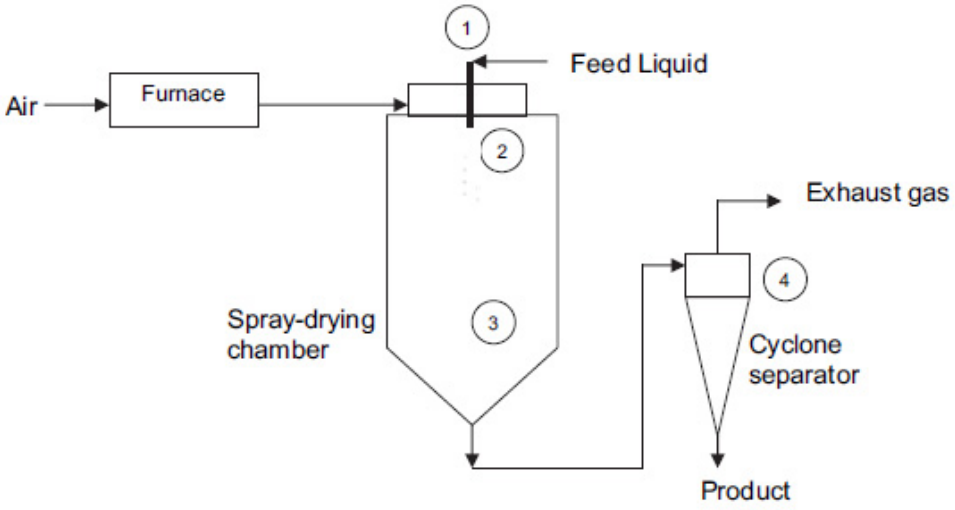


Figure 2.2: The process stages of spray drying [52].

1. atomisation of liquid feed into a spray chamber;
2. contact between the spray and the drying agent;
3. moisture evaporation;
4. separation of dried particles from air flow;

Atomisation is a process used to disperse the liquid or slurry into a controlled drop size spray. An atomizer nozzle can take on many forms so that its choice is very important to achieve a suitable quality of the product. The different types of atomizer are *centrifugal* or *rotary atomizer* for which the size of droplets produced from the nozzle varies directly with feed rate and feed viscosity and inversely with wheel speed and wheel diameter, *pressure nozzle atomizer* in which droplet size varies directly again with feed rate and feed viscosity, but inversely with pressure. Finally, atomization can be obtained also through the *two-fluid nozzle atomizer*, meaning that a shear field created by compressed air, atomizes the liquid and produces a wide range of droplet sizes. [20,21].

During spray-air contact, the hot drying gas (air in most cases) can be blown in the same direction as the sprayed liquid - *co-current flow* - or it can be against the flow from the atomizer - *counter-current flow* -. The *co-current* flow is recommended for the drying of heat-sensitive materials because the temperature of the dried particles at the outlet is slightly lower than the exhaust air (drying medium) temperature. In fact with this configuration the spray is in contact with the hot inlet air as soon as it enter the dryer causing a very high rate of evaporation; as the content of moisture decreases, the air temperature decreases and the droplets temperature is kept low allowing a less thermal degradation of the products as well as a rapid spray evaporation. In the *counter-current* arrangement, on the other hand, the spray inlet correspond to the drying medium outlet and this arouses a final product temperature higher than the exhaust drying agent temperature and that is why this configuration is used for non-heat-sensitive products only.

Regarding the last stage, the separation of dried particles from air flow can be done in different ways depending on the operating conditions such as particle size, shape, bulk density and powder outlet position. For example, dried particles can be picked up at the base of the dryer and absorbed by a cyclone separator or a screw conveyor. Other equipment useful to collect the dry powder are bag filters and electrostatic precipitators [22].

As previously said, different spray dryer's models can be found in the literature, in fact many models have been developed by various authors through the years by using payment CFD software. After an accurate literature reading, some models have been chosen as references for this work as it is briefly illustrated in table 2.1.

Table 2.1: reference models

Authors	Year of Publication	Spray Dryer as the specific topic	Model Geometry	Software	Model Frame	Computation of turbulent flow	Including evaporation
F.G. Kieviet [3]	1997	yes	2D	CFX	Eulerian-Lagrangian	not specified	yes [3]
L. Huang, K. Kumar, A.S. Mujumdar [7]	2006	yes	3D	Fluent	Eulerian-Lagrangian	not specified	yes [7]
C. Anandharamakrishnan[24]	2008	yes	3D	Fluent	Eulerian-Lagrangian	discrete eddy concept [24]	yes [24]
M.Mezhericher, A. Levy, I. Borde [17]	2010	yes	2D	Fluent	Eulerian-Lagrangian	not specified	yes [17]
S. N. Saleh	2010	yes	3D	Fluent	Eulerian-Lagrangian	stochastic method [5]	yes [10]
D. F. Fletcher, B. Guo, D. J. E. Harvie, T.A.G.Langrish, J. J. Nijdam, J. Williams [27]	2006	yes	3D	CFX	Eulerian-Lagrangian	stochastic method, Gaussian distribution [27]	yes [46],[47]

3 A brief introduction to Computational Fluid Dynamics (CFD)

Computational fluid dynamics (CFD) is a branch of fluid mechanics that uses numerical analysis and algorithms to solve and analyze problems that involve fluid flows. Computers are used to perform the calculations required to simulate the interaction of liquids and gases with surfaces defined by boundary conditions. Numerical methods are used to give predictions of typical parameter of the represented phenomenon such as velocity, temperature and pressure profiles inside the system by solving, for example, equations for the conservation of mass, momentum and energy. This approach is very interest because it means that pilot measurements, that are very difficult and expensive to obtain in large-scale, can be replaced with this accurate predictions. That is the reason why by the 1990s, there has been a significant increase in computing power and a considerable development in software and solutions.

CFD is also a powerful tool used to show , for example, the flow behaviour of fluid with 3D images [19], that is very important in order to fully understand the behaviour of the phenomenon.

Whatever the represented phenomenon is, the same basic procedure is followed:

- During preprocessing
  - The geometry (physical bounds) of the problem is defined.
  - The volume occupied by the fluid is divided into discrete cells (the mesh) in which the governing equations will be solved.
  - The physical modeling is defined – for example, the equations of motion + enthalpy + radiation + species conservation - and a suitable method to discretize it, is choosen.
  - Boundary conditions are defined. This involves specifying the fluid behaviour and properties at the boundaries of the problem. For transient problems, the initial conditions are also defined.
- The simulation is started and the equations are solved iteratively as a steady-state or transient
- Finally a postprocessor is used for the analysis and visualization of the resulting solution

An important item above-mentioned is discretize governing equations. The equations are PDEs (Partial Differential Equations) meaning that they are a combination of differential terms (rates of change) that describe a conservation principle. Without loss in generality, all physical processes can be described by PDEs. Now, the CFD process requires the discretization of the governing PDEs, i.e. the derivation of equivalent algebraic relations that should faithfully represent the original PDEs. This is done by transforming each differential term into an approximate algebraic relation. Some of the discretization methods used are:

- Finite Difference Method (FDM)

- Finite Element Method (FEM)
- Finite Volume Method (FVM)

In particular as regards the finite difference method, governing equations are written in differential form and the domain is structured with a grid; the partial derivatives are replaced by approximations in terms of node values of the functions and one algebraic equation per grid node is written so as to have a linear algebraic equation system. The FDM is finally applied to the structured grid.

Instead, as concerns the finite element method, the solution domain is subdivided into a finite number of elements, the governing equation is solved for each element and then overall solution is obtained by "assembly". Lastly, equations are multiplied by a weight function before integrated over the entire domain.

Finally, the most versatile discretization techniques used in CFD, and also for this thesis, is the finite volume method. Based on the control volume formulation of analytical fluid dynamics, the first step in the FVM is to divide the domain into a number of control volumes (aka cells, elements) where the variable of interest is located at the centroid of the control volume. The next step is to integrate the differential form of the governing equations (very similar to the control volume approach) over each control volume. Interpolation profiles are then assumed in order to describe the variation of the concerned variable between cell centroids. The resulting equation is called the discretized or discretization equation. In this manner, the discretization equation expresses the conservation principle for the variable inside the control volume. The most compelling feature of the FVM is that the resulting solution satisfies the conservation of quantities such as mass, momentum, energy, and species. An example of finite volume formulation is shown below: the computational domain $\boldsymbol{V}_h$ is divided into non-overlapping cells or finite volumes $\boldsymbol{V}_r$, $r = 1, ..., N$

$$V_h = U_r V_r \tag{3.1}$$

Usually these cells are polygons (triangles, quadrilaterals) in 2-D and polyhedra (tetrahedron, hexahedron, prims, etc) in 3-D. Then the cell average value over the cell $\boldsymbol{V}_r$ is introduced:

$$u_r = \frac{1}{|V_r|} \int_{V_r} u_x dx \tag{3.2}$$

which is the basic unknown quantity in the finite volume method. Next step is to define $\boldsymbol{N}(r)$ as the set of cells which share a common face with $\boldsymbol{V}_r$ and then write the integral conservation law for cell $\boldsymbol{V}_r$:

$$|V_r| \frac{du_r}{dt} + \sum_{s \in N(r)} \int_{V_r \cap V_s} f_i n_i ds = 0 \tag{3.3}$$

It now remains to approximate the flux integral and this can be achieved using for example gaussian quadrature; taking $\boldsymbol{p}$ gaussian points:

$$\int_{V_r \cap V_s} f_i n_i ds = \Delta s_{rs} \sum_{m=1}^{p} \omega_m F_{rs}^m \tag{3.4}$$

where ω_m are the gaussian weights and $\boldsymbol{F}$ is an approximation to $\boldsymbol{f}_i \boldsymbol{n}_i$.

3.1 Reference frames

The commonly used modelling frames for a flow in a spray dryer are the Eulerian-Eulerian and the Eulerian-Lagrangian methods, useful to represent the two-phase flow (gas and droplets or gas and particles). The Eulerian-Eulerian method treats the particle phase as a continuum, so that there are two Eulerian phases, one for the gas and another for droplets, in fact this approach develops its conservation equations on a control volume basis and in a similar form as that for the fluid phase. On the other hand, the Eulerian-Lagrangian method considers particles as a discrete phase and tracks the path of each individual particle (from their injection point until they escape the domain), after calculating the gas field with the Eulerian approach [27]. To choose a suitable method for a certain problem depends heavly on the objective and characteristics of the problem under examination. For example, the advantages of the Eulerian-Eulerian approach are that turbulence can be modelled quite simply, it is usually cheap regarding computational demands and it is recommended when the high void fraction of the flow becomes a dominating flow controlling parameter. However, the Eulerian method can be expensive when a separate set of transport equations is solved for each particle size [25, 26]. In the case of spray dryer the Eulerian-Lagrangian approach is usually used: the exchange of mass, momentum and energy are calculated along the particle trajectories and then, if the gas and particle solutions are coupled, these terms are added to the respectively source terms of the Navier-Stokes equations of the gas flow. And therefore, it can better represent the behaviour and residence times of single particles and can potentially approximate mass and heat transfer more accurately. The Eulerian-Lagrangian model has also the advantage of being computationally cheaper than the Eulerian-Eulerian method for a large range of particle sizes, but on the other hand, the approach can be expensive if a large number of particles have to be tracked (droplets or particles should not exceed 10% by volume of the mixture contained in the spray dryer) [28]. As regards the literature, opposing views can be found: according to the authors Mostafa and Mongia [25], the Eulerian approach performs better than Lagrangian method. Nijdam et al. [27] instead, found that both Eulerian and Lagrangian approaches equally predict turbulent droplets dispersion and agglomeration of sprays for a wide range of droplet and gas flows. Finally, Lagrangian models are preferred because of their wider range of applicability.

3.2 Governing equations

The governing equations of fluid mechanics and heat transfer (i.e.,fluid dynamics) are described below. The fundamental equations of fluid dynamics are based on the following universal laws of conservation:

- Conservation of Mass
- Conservation of Momentum
- Conservation of Energy

The equation that results from applying the Conservation of Mass law to a fluid flow is called the continuity equation. The Conservation of Momentum law represents the

Newton's Second Law. When this law is applied to a fluid flow, it yields a vector equation known as the momentum equation. The Conservation of Energy law is identical to the First Law of Thermodynamics, and the resulting fluid dynamic equation is named the energy equation. In addition to the equations developed from these universal laws, it is necessary to establish relationships between fluid properties in order to close the system of equations. An example of such a relationship is the equation of state, which relates the thermodynamic variables pressure p, density p, and temperature T.

Historically, there have been two different approaches taken to derive the equations of fluid dynamics: the phenomenological approach and the kinetic theory approach. In the phenomenological approach, certain relations between stress and rate of strain and heat flux and temperature gradient are postulated, and the fluid dynamic equations are then developed from the conservation laws. The required constants of proportionality between stress and rate of strain and heat flux and temperature gradient (which are called transport coefficients) must be determined experimentally in this approach. In the kinetic theory approach (also called the mathematical theory of nonuniform gases), the fluid dynamic equations are obtained with the transport coefficients defined in terms of certain integral relations, which involve the dynamics of colliding particles. The drawback to this approach is that the interparticle forces must be specified in order to evaluate the collision integrals. Thus a mathematical uncertainty takes the place of the experimental uncertainty of the phenomenological approach. These two approaches will yield the same fluid dynamic equations if equivalent assumptions are made during their derivations.

The derivation of the fundamental equations of fluid dynamics will not be presented here. The derivation of the equations using the phenomenological approach is thoroughly treated by Schlichting (1968), and the kinetic theory approach is described in detail by Hirschfelder et al. (1954). The fundamental equations given initially in this chapter were derived for a uniform, homogeneous fluid without mass diffusion or finite-rate chemical reactions. In order to include these later effects it is necessary to consider extra relations, called the species continuity equations, and to add terms to the energy equation to account for diffusion.

3.3 Continuity equation

The Conservation of Mass law applied to a fluid passing through an infinitesimal, fixed control volume (Fig. 3.3.1) yields the following equation of continuity:

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \boldsymbol{V}) = 0 \tag{3.3.5}$$

where ρ is the fluid density and $\boldsymbol{V}$ is the fluid velocity. The first term in this equation represents the rate of increase of the density in the control volume, and the second term represents the rate of mass flux passing out of the control surface (which surrounds the control volume) per unit volume.

It is convenient to use the substantial derivative

$$\frac{D()}{Dt} = \frac{\partial()}{\partial t} + \boldsymbol{V} \cdot \boldsymbol{\nabla}() \tag{3.3.6}$$

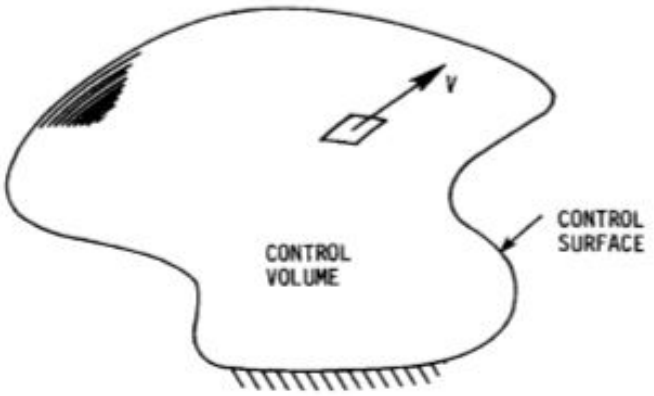


Figure 3.3.1: Control volume for Eulerian approach.

to change eq. (3.3.1) into the form

$$\frac{D\rho}{Dt} + \rho(\boldsymbol{\nabla} \cdot \boldsymbol{V}) = 0 \tag{3.3.7}$$

Equation (3.3.1) was derived using the Eulerian approach. In this approach, a fixed control volume is utilized, and the changes to the fluid are recorded as the fluid passes through the control volume. In the alternative Lagrangian approach, the changes to the properties of a fluid element are recorded by an observer moving with the fluid element (see 3.1). The Eulerian viewpoint is commonly used in fluid mechanics.

For a Cartesian coordinate system, where u,u,w represent the x,y,z components of the velocity vector, eq. (3.3.1) becomes:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} = 0 \tag{3.3.8}$$

A flow in which the density of each fluid element remains constant is called *incompressible*. Mathematically, this implies that

$$\frac{D\rho}{Dt} = 0 \tag{3.3.9}$$

which reduces Eq. (3.3.3) to

$$\boldsymbol{\nabla} \cdot \boldsymbol{V} = 0 \tag{3.3.10}$$

or for the Cartesian coordinate system

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{3.3.11}$$

For steady air flows with speed V$<$ 100 m/s or M $<$ 0.3 the assumption of incompressibility is a good approximation.

3.4 Momentum equation

Newton's Second Law applied to a fluid passing through an infinitesimal, fixed control volume yields the following momentum equation:

$$\frac{\partial(\rho \boldsymbol{V})}{\partial t} + \boldsymbol{\nabla} \cdot \rho \boldsymbol{V}\boldsymbol{V} = \rho \boldsymbol{f} + \boldsymbol{\nabla} \cdot \boldsymbol{\Pi}_{ij} \tag{3.4.1}$$

The first term in this equation represents the rate of increase of momentum per unit volume in the control volume. The second term represents the rate of momentum lost by convection (per unit volume) through the control surface. Note that $\rho \boldsymbol{V}\boldsymbol{V}$ is a tensor, so that $\boldsymbol{\nabla} \cdot \rho \boldsymbol{V}\boldsymbol{V}$ is not a simple divergence. This term can be expanded, however as

$$\boldsymbol{\nabla} \cdot \rho \boldsymbol{V}\boldsymbol{V} = \rho \boldsymbol{V} \cdot \boldsymbol{\nabla} \boldsymbol{V} + \boldsymbol{V}(\boldsymbol{\nabla} \cdot \rho \boldsymbol{V}) \tag{3.4.2}$$

When this expression for $\boldsymbol{\nabla} \cdot \rho \boldsymbol{V}\boldsymbol{V}$ is substituted into eq. (3.4.1), and the resulting equation is simplified using the continuity equation, the momentum equation reduces to:

$$\rho \frac{D\boldsymbol{V}}{Dt} = \rho \boldsymbol{f} + \boldsymbol{\nabla} \cdot \boldsymbol{\Pi}_{ij} \tag{3.4.3}$$

The first term on the right-hand side of eq. (3.4.3) is the body force per unit volume. Body forces act at a distance and apply to the entire mass of the fluid. The most common body force is the gravitational force. In this case, the force per unit mass ($\boldsymbol{f}$) equals the acceleration of gravity vector $\boldsymbol{g}$:

$$\rho \boldsymbol{f} = \rho \boldsymbol{g} \tag{3.4.4}$$

The second term on the right-hand side of eq. (3.4.3) represents the surface forces per unit volume. These forces are applied by the external stresses on the fluid element. The stresses consist of normal stresses and shearing stresses and are represented by the components of the stress tensor $\boldsymbol{\Pi}_{ij}$

The momentum equation given above is quite general and is applicable to both continuum and noncontinuum flows. It is only when approximate expressions are inserted for the shear-stress tensor that eq. (3.4.1) loses its generality. For all gases that can be treated as a continuum, and most liquids, it has been observed that the stress at a point is linearly dependent on the rates of strain (deformation) of the fluid. A fluid that behaves in this manner is called a Newtonian fluid. With this assumption, it is possible to derive (Schlichting, 1968) a general deformation law that relates the stress tensor to the pressure and velocity components. In compact tensor notation, this relation becomes:

$$\boldsymbol{\Pi}_{ij} = -p\delta_{ij} + \mu\Big(\frac{\partial u_i}{\partial x_j}) + \frac{\partial u_j}{\partial x_i}\Big) + \delta_{ij}\mu' \frac{\partial u_k}{\partial x_k} \qquad i, j, k = 1, 2, 3 \tag{3.4.5}$$

where δ_{ij} is the Kronecker delta function ($\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$); u_1, u_2, u_3 represent the three components of the velocity vector $\boldsymbol{V}$;x$_1$, x_2, x_3 represent the three components of the position vector; μ is the coefficient of viscosity (dynamic viscosity), and μ' is the second coefficient of viscosity. The two coefficients of viscosity are related to the coefficient of bulk viscosity K by the expression:

$$K = \frac{2}{3}\mu + \mu' \tag{3.4.6}$$

In general, it is believed that K is negligible except in the study of the structure of shock waves and in the absorption and attenuation of acoustic waves. For this reason, we will ignore bulk viscosity for the remainder of the text. With $K = 0$, the second coefficient of viscosity becomes:

$$\mu' = -\frac{2}{3}\mu \tag{3.4.7}$$

and the stress tensor may be written as

$$\boldsymbol{\Pi}_{ij} = -p\delta_{ij} + \mu\Big[\Big(\frac{\partial u_i}{\partial x_j}) + \frac{\partial u_j}{\partial x_i}\Big) - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\Big] \qquad i,j,k = 1,2,3 \tag{3.4.8}$$

The stress tensor

$$\boldsymbol{\Pi}_{ij} = -p\delta_{ij} + \boldsymbol{\tau}_{ij} \tag{3.4.9}$$

where $\boldsymbol{\tau}_{ij}$ represents the viscous stress tensor given by:

$$\boldsymbol{\tau}_{ij} = \mu\Big[\Big(\frac{\partial u_i}{\partial x_j}) + \frac{\partial u_j}{\partial x_i}\Big) - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\Big] \qquad i,j,k = 1,2,3 \tag{3.4.10}$$

Upon substituting eq. (3.4.8) into eq. (3.4.3) the famous *Navier-Stokes equation* is obtained:

$$\rho\frac{D\boldsymbol{V}}{Dt} = \rho\boldsymbol{f} - \boldsymbol{\nabla}p + \frac{\partial}{\partial x_j}\Big(\mu\Big[\Big(\frac{\partial u_i}{\partial x_j}) + \frac{\partial u_j}{\partial x_i}\Big) - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\Big]\Big) \qquad i,j,k = 1,2,3 \tag{3.4.11}$$

For a Cartesian coordinate system, eq. (3.4.11) can be separated into the following three scalar Navier-Stokes equations:

$$\begin{aligned}
\rho\frac{Du}{Dt} &= \rho\boldsymbol{f_x} - \frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\Big[\frac{2}{3}\mu\Big(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} - \frac{\partial w}{\partial z}\Big)\Big] + \frac{\partial}{\partial y}\Big[\mu\Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big)\Big] + \frac{\partial}{\partial z}\Big[\mu\Big(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\Big)\Big] \\
\rho\frac{Dv}{Dt} &= \rho\boldsymbol{f_y} - \frac{\partial p}{\partial y} + \frac{\partial}{\partial x}\Big[\mu\Big(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\Big)\Big] + \frac{\partial}{\partial y}\Big[\frac{2}{3}\mu\Big(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x} - \frac{\partial w}{\partial z}\Big)\Big] + \frac{\partial}{\partial z}\Big[\mu\Big(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\Big)\Big] \\
\rho\frac{Dw}{Dt} &= \rho\boldsymbol{f_z} - \frac{\partial p}{\partial z} + \frac{\partial}{\partial x}\Big[\mu\Big(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\Big)\Big] + \frac{\partial}{\partial y}\Big[\mu\Big(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\Big)\Big] + \frac{\partial}{\partial z}\Big[\frac{2}{3}\mu\Big(2\frac{\partial w}{\partial z} - \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\Big)\Big]
\end{aligned} \tag{3.4.12}$$

The Navier-Stokes equations form the basis upon which the entire science of viscous flow theory has been developed. Strictly speaking, the term Navier- Stokes equations refers to the components of the viscous momentum equation [eq. (3.4.11)]. However, it is common practice to include the continuity equation and the energy equation in the set of equations referred to as the Navier-Stokes equations.

If the flow is incompressible and the coefficient of viscosity (μ) is assumed constant, eq. (3.4.11) will reduce to the much simpler form:

$$\rho\frac{D\boldsymbol{V}}{Dt} = \rho\boldsymbol{f} - \boldsymbol{\nabla}p + \mu\nabla^2\boldsymbol{V} \tag{3.4.13}$$

It should be remembered that eq. (3.4.13) is derived by assuming a constant viscosity, which may be a poor approximation for the nonisothermal flow of a liquid whose viscosity is highly temperature dependent. On the other hand, the viscosity of gases is only moderately temperature dependent, and eq. (3.4.13) is a good approximation for the incompressible flow of a gas.

3.5 Energy equation

The First Law of Thermodynamics applied to a fluid passing through an infinitesimal, fixed control volume yields the following energy equation:

$$\frac{\partial E_t}{\partial t} + \boldsymbol{\nabla} \cdot E_t \boldsymbol{V} = \frac{\partial Q}{\partial t} - \boldsymbol{\nabla} \cdot \boldsymbol{q} + \rho \boldsymbol{f} \cdot \boldsymbol{V} + \boldsymbol{\nabla} \cdot (\boldsymbol{\Pi_{ij}} \cdot \boldsymbol{V}) \tag{3.5.1}$$

where E_t is the total energy per unit volume.

The first term on the left-hand side of eq. (3.5.1) represents the rate of increase of E, in the control volume, while the second term represents the rate of total energy lost by convection (per unit volume) through the control surface. The first term on the right-hand side of eq. (3.5.1) is the rate of heat produced per unit volume by external agencies, while the second term ($\boldsymbol{\nabla} \cdot \boldsymbol{q}$) is the rate of heat loss by conduction (per unit volume) through the control surface. Fourier's law for heat transfer by conduction will be assumed, so that the heat transfer $\boldsymbol{q}$ can be expressed as:

$$\boldsymbol{q} = -k \boldsymbol{\nabla} T \tag{3.5.2}$$

where k is the coefficient of thermal conductivity and T is the temperature. The third term on the right-hand side of eq. (3.5.1) represents the work done on the control volume (per unit volume) by the body forces, while the fourth term represents the work done on the control volume (per unit volume) by the surface forces. It should be obvious that eq. (3.5.1) is simply the First Law of Thermodynamics applied to the control volume. That is, the increase of energy in the system is equal to heat added to the system plus the work done on the system.

For a Cartesian coordinate system, eq. (3.5.1) becomes:

$$\begin{aligned}
&\frac{\partial E_t}{\partial t} - \frac{\partial Q}{\partial t} - \rho(f_x u + f_y v + f_z w) + \frac{\partial}{\partial x}(E_t u + pu - u\tau_{xx} - v\tau xy - w\tau_{xz} + q_x) + \\
&+ \frac{\partial}{\partial y}(E_t v + pv - u\tau_{xy} - v\tau yy - w\tau_{zy} + q_y) + \\
&+ \frac{\partial}{\partial z}(E_t w + pw - u\tau_{xz} - v\tau yz - w\tau_{zz} + q_z) = 0
\end{aligned} \tag{3.5.3}$$

which is in conservation-law form.

Using the continuity equation, the left-hand side of eq. (3.5.1) can be replaced by the following expression:

$$\rho \frac{D(E_t/\rho)}{Dt} = \frac{\partial E_t}{\partial t} + \boldsymbol{\nabla} \cdot E_t \boldsymbol{V} \tag{3.5.4}$$

which is equivalent to:

$$\rho \frac{D(E_t/\rho)}{Dt} = \rho \frac{De}{Dt} + \rho \frac{D(V^2/2)}{Dt} \tag{3.5.5}$$

if only internal energy (e) and kinetic energy ($V^2/2$) are considered significant as terms of the total energy (E_t). Forming the scalar dot product of eq. (3.4.3) with the velocity vector $\boldsymbol{V}$ allows one to obtain:

$$\rho \frac{D\boldsymbol{V}}{Dt} \cdot \boldsymbol{V} = \rho \boldsymbol{f} \cdot \boldsymbol{V} - \boldsymbol{\nabla} p \cdot \boldsymbol{V} + (\boldsymbol{\nabla} \cdot \tau_{ij}) \cdot \boldsymbol{V} \tag{3.5.6}$$

Now if eqs. (3.5.4), (3.5.5), and (3.5.6) are combined and substituted into eq. (3.5.1), a useful variation of the original energy equation is obtained:

$$\rho\frac{De}{Dt} + p(\boldsymbol{\nabla}\cdot\boldsymbol{V}) = \frac{\partial Q}{\partial t} - \boldsymbol{\nabla}\cdot\boldsymbol{q} + \boldsymbol{\nabla}\cdot(\tau_{ij}\cdot\boldsymbol{V}) - (\boldsymbol{\nabla}\cdot\tau_{ij})\cdot\boldsymbol{V} \tag{3.5.7}$$

The last two terms in this equation can be combined into a single term, since

$$\tau_{ij}\frac{\partial u_i}{\partial x_j} = \boldsymbol{\nabla}\cdot(\tau_{ij}\cdot\boldsymbol{V}) - (\boldsymbol{\nabla}\cdot\tau_{ij})\cdot\boldsymbol{V} \tag{3.5.8}$$

This term is customarily called the *dissipation function* Φ and represents the rate at which mechanical energy is expended in the process of deformation of the fluid due to viscosity. After inserting the dissipation function, eq. (3.5.7) becomes:

$$\rho\frac{De}{Dt} + p(\boldsymbol{\nabla}\cdot\boldsymbol{V}) = \frac{\partial Q}{\partial t} - \boldsymbol{\nabla}\cdot\boldsymbol{q} + \Phi \tag{3.5.9}$$

Using the definition of enthalpy,

$$h = e + \frac{p}{\rho} \tag{3.5.10}$$

and the continuity equation, eq. (3.5.9) can be rewritten as:

$$\rho\frac{Dh}{Dt} = \frac{Dp}{Dt} + \frac{\partial Q}{\partial t} - \boldsymbol{\nabla}\cdot\boldsymbol{q} + \Phi \tag{3.5.11}$$

For a Cartesian coordinate system, the dissipation function, which is always positive if μ' = -(2/3)μ, becomes:

$$\begin{aligned}\Phi = \mu\Big[2\Big(\frac{\partial u}{\partial x}\Big)^2 + 2\Big(\frac{\partial v}{\partial y}\Big)^2 + 2\Big(\frac{\partial w}{\partial z}\Big)^2 + \Big(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\Big)^2 + \\ + \Big(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial z}\Big)^2 + \Big(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\Big)^2 - \frac{2}{3}\Big(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\Big)^2\Big]\end{aligned} \tag{3.5.12}$$

If the flow is incompressible, and if the coefficient of thermal conductivity is assumed constant, eq. (3.5.9) reduces to:

$$\rho\frac{De}{Dt} = \frac{\partial Q}{\partial t} + k\nabla^2 T + \Phi \tag{3.5.13}$$

3.6 State equation

In order to close the system of fluid dynamic equations it is necessary to establish relations between the thermodynamic variables (p, ρ, T, e, h) as well as to relate the transport properties (μ, k) to the thermodynamic variables. For example, consider a compressible flow without external heat addition or body forces and use eq. (3.3.4) for the continuity equation, eqs. (3.4.12) for the three momentum equations, and eq. (3.5.3) for the energy equation. These five scalar equations contain seven unknowns p, ρ, T, e, u, v, w, provided that the transport coefficients μ, k can be related to the thermodynamic properties in the list of unknowns. It is obvious that two additional equations are required to close the system. These two additional equations can be

obtained by determining relations that exist between the thermodynamic variables. Relations of this type are known as equations of state. According to the *state principle* of thermodynamics, the local thermodynamic state is fixed by any two independent thermodynamic variables, provided that the chemical composition of the fluid is not changing owing to diffusion or finite-rate chemical reactions. Thus for the present example, if we choose e and ρ as the two independent variables, then equations of state of the form

$$p = p(e, \rho) \qquad T = T(e, \rho) \tag{3.6.1}$$

are required.

For most problems in gas dynamics, it is possible to assume a *perfect gas*. A perfect gas is defined as a gas whose intermolecular forces are negligible. A perfect gas obeys the perfect gas equation of state

$$p = \rho RT \tag{3.6.2}$$

where R is the gas constant. The intermolecular forces become important under conditions of high pressure and relatively low temperature. For these conditions, the gas no longer obeys the perfect gas equation of state. Therefore real gas effects, which is not modelled by the ideal gas law, can be taken into account by using for example the Peng-Robinson equation of state:

$$p = \frac{RT}{V_m - b} - \frac{a\alpha}{V_m^2 - 2bV_m - b^2} \tag{3.6.3}$$

$$a = \frac{0,457235R^2T_c^2}{p_c}; \qquad b = \frac{0,077796RT_c}{p_c}; \qquad T_r = \frac{T}{T_c}$$

$$\alpha = \left((1 + k(1 - \sqrt{T_r})\right)^2; \qquad k = 0,37464 + 1,54226\omega_a - 0,26992{\omega_a}^2$$

V_m is the volume of 1 mole of gas, also known as molar volume, ω_a is an acentric factor, p_c and T_c are the critical pressure and temperature and R is the universal gas constant [41].

3.7 Turbulence models

Turbulence modeling is a key issue in most CFD simulations. Virtually all engineering applications are turbulent and hence require a turbulence model. These turbulence models used in most commercial CFD packages are based on the splitting up of instantaneous quantities into a time-averaged and a fluctuating part by a process known as Reynolds decomposition.

The most common type of turbulence models are the two equations turbulence models, meaning that they include two extra transport equations to represent the turbulent properties of the flow, and this is a fundamental aspect in order to represent turbulence effects such as convection and diffusion. In two equations turbulence models the first transported variable is the turbulent kinetic energy k while the second transported variable depends on the model chosen: the choice commonly falls on the turbulent dissipation ϵ or on the specific dissipation ω depending on the scale of turbulence that could thus be lenght-scale or time-scale.

As concerns the turbulent dissipation ϵ, four turbulence models are usually used for simulating sprays: (i) standard k-ϵ [29, 30] (ii) RNG k- ϵ, [31] (iii) realizable k- ϵ, [32] (iv) Reynolds Stress Model (RSM) [33]. The standard k- ϵ model focuses on the mechanisms that affect the turbulent kinetic energy and it is the most common one due to its simple implementation, accuracy and robustness (it has stable calculations that converge relatively easily). However, this model has also some disadvantages such as a poor prediction for swirling and rotating flows as well as it presents a simplistic ϵ equation and it valid only for fully developed turbulent flows. The transport of the turbulence kinetic energy k and its dissipation rate ϵ is given as follows:

$$\frac{\partial(\rho\kappa)}{\partial t} + \nabla \cdot (\rho\kappa\vec{v}) = \nabla \cdot [(\mu + \frac{\mu_t}{\sigma_k})\nabla k] + G_k - \rho\epsilon \tag{3.1}$$

$$\frac{\partial(\rho\epsilon)}{\partial t} + \nabla \cdot (\rho\epsilon\vec{v}) = \nabla \cdot [(\mu + \frac{\mu_t}{\sigma_\epsilon})\nabla \epsilon] + C_{1\epsilon}\frac{\epsilon}{k}(G_k) - C_{2\epsilon}\rho\frac{\epsilon^2}{k} \tag{3.2}$$

Where G_k is the generation of kinetic energy due to the mean velocity gradients. The quantities σ_k and σ_ϵ are the turbulent Prandtl numbers for κ and ϵ respectively and $C_{1\epsilon}$, $C_{2\epsilon}$ are constant. The turbulent (or eddy) viscosity μ_t is calculated from κ and ϵ as follows:

$$\mu_t = \rho C_\mu \frac{\epsilon^2}{k} \tag{3.3}$$

For calculating an approximate solution of fluid flow equations, the equations have to be made discrete. For this, the flow domain is divided into number of control volumes. This is called a grid and at each grid cell approximate solutions for the Navier-Stokes and the continuity equations are calculated.

The RNG k- ϵ turbulence model one of the improvement model based on the standard k- ϵ model and is derived from the instantaneous Navier-Stokes equations, using a mathematical technique called "renormalization group" (RNG). Some additional terms and functions are added in the transport equations for k and ϵ; in particural, the form is similar to the standard k- ϵ equations, but it includes the effect of swirl on turbulence, differential formula for effective viscosity and a new term for interaction between turbulence dissipation and mean shear in ϵ equation. These improvements ensure a suitable prediction for transitional flows and mass and wall heat transfer.

In the realizable k- ϵ model, the word "realizable" underlines that it can satisfy certain mathematical constraints on the normal stresses, consistent with the physics of turbulent flows. This turbulence model has the same turbulent kinetic energy equation as the standard k- ϵ model while it presents an improved ϵ equation. It improves therefore prediction for flows involving rotation, recirculation, boundary layers under strong adverse pressure gradients and strong streamline curvature [19, 34].

The Reynolds Stress Model has likewise the same general form as the instantaneous Navier-Stokes equations, with the velocities and other solution variables ensemble-averaged (or time-averaged). In the RSM the eddy viscosity approach has been discarded and the Reynolds stresses are directly computed and the exact Reynolds stress transport equation accounts for the directional effects of the Reynolds stress fields. The

RSM is the most performing model for problems where anisotropy of turbulence has a dominant effect on the mean flow such as for highly swirling flows [35].

Another frequently used turbulence model is the k-ω model. In particular it is used as a closure for the Reynolds-averaged Navier–Stokes equations (RANS equations) and for the unsteady Reynolds-averaged Navier–Stokes equations (URANS equations).The model attempts to predict turbulence by two partial differential equations for two variables, k and ω, with the first variable, as mentioned before, being the turbulence kinetic energy (k) while the second (ω) is the specific rate of dissipation (referred to the turbulence kinetic energy k into internal thermal energy). Mathematically, this turbulence model presents a modified version of the k equation compared to the k- ϵ model and a transport equation for ω. The k-ω model has more difficulty converging and is quite sensitive to the initial guess at the solution; hence, the k- ϵ model is often used first to find an initial condition for solving the k-ω model. The k-ω model is also useful in many cases where the k- ϵ model is not accurate, such as internal flows, separated flows, jets and flows that exhibit strong curvature.

Especially, for the purpose of this thesis it's very important to analyze the shear stress transport (SST) k-ω turbulence model. The SST k-ω turbulence model is a two-equation eddy-viscosity model which use has become very common. The use of a k-ω formulation in the inner parts of the boundary layer makes the model directly usable all the way down to the wall through the viscous sub-layer, hence the SST k-ω model can be used as a Low-Re turbulence model without any extra damping functions. The SST formulation also switches to a k-ϵ behaviour in the free-stream and thereby avoids the common k-ω problem that the model is too sensitive to the inlet free-stream turbulence properties. In literature many authors agree that the SST k-ω model has good behaviour in adverse pressure gradients and separating flow, but it produces a bit too large turbulence levels in regions with large normal strain, like stagnation regions and regions with strong acceleration. This tendency is much less pronounced than with a normal k-ϵ model though [50]. The turbulence kinetic energy equation and its dissipation rate equation for the SST k-ω model are represented respectively in eq. 3.4 and 3.5 .

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = P_k - \beta^* k\omega + \frac{\partial}{\partial x_j}\Big[(\nu + \sigma_k \nu_T)\frac{\partial k}{\partial x_j}\Big] \tag{3.4}$$

$$\frac{\partial \omega}{\partial t} + U_j \frac{\partial \omega}{\partial x_j} = \alpha S^2 - \beta \omega^2 + \frac{\partial}{\partial x_j}\Big[(\nu + \sigma_\omega \nu_T)\frac{\partial \omega}{\partial x_j}\Big] + 2(1 - F_1)\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} \tag{3.5}$$

Where ν_T is the kinematic eddy viscosity, S is the mean rate of strain, and β, β^*, σ_ω, $\sigma_{\omega 2}$, σ_k and α are constants. F_1 is known as the blending function. It has a value of one in the free stream and is zero in the boundary layer. This is how the k-ω model is activated in the boundary layer and turned off in the free stream. More detailed explanations regarding model constants and derivation of the transport equations can be found in [37]. The k-ω SST is reported to perform well for both under-expanded jets and heat transfer applications, see [38], [39] and [40].

3.8 Heat transfer

In general, three different modes of heat transfer exist: conduction, convection and radiation. Radiation is neglected in this case.

Heat transfer problems can be either steady state or transient. Solutions to steady state problems only varies with location, while transient problems also varies with time. Steady state problems are therefore easier to solve, since all derivatives with respect to time is equal to zero.

3.8.1 Conduction

Heat transfer due to conduction takes place in solids and quiescent fluids. The heat is transferred by diffusion and collisions between particles, without any mass flow [42]. Heat flows from a high- to a low-temperature region due to the temperature gradient between those regions [43]. The heat transfer rate varies, depending on the material, geometry, and the temperature gradient. Fourier's law states the relationship between the heat flow and the temperature gradient, here shown for a one-dimensional problem.

$$\dot{Q}_{cond} = -k_c A \frac{dT}{dx} \tag{3.8.1}$$

where $\dot{Q}_{cond}$ is the heat flux, k_c is the thermal conductivity, which is a measure of a materials ability to transfer heat by conduction. A is the cross-sectional area, and $\frac{dT}{dx}$ is the temperature gradient. The negative sign indicates that the heat flows in the opposite direction of the temperature gradient.

Materials with the atoms closely spaced, such as solids, generally have the highest thermal conductivity. Gases and vapors have the lowest conductivity due to greater distance between the atoms [44].

The thermal conductivity is also temperature dependent. In many pure metals, it tends to decrease with increasing temperature. In gases however, the opposite is true. Higher temperatures results in greater thermal conductivity. For anisotropic materials, k_c also varies with orientation.

Another important material property is the thermal diffusivity, α. It is defined as the thermal conductivity divided by density, ρ, times specific heat capacity, C_p [43].

$$\alpha = \frac{k_c}{\rho C_p} \tag{3.8.2}$$

In transient problems, the thermal diffusivity is a measure of how quickly the heat is conducted through the material. The quantity ρC_p is often referred to as the volumetric heat capacity. Thus, the thermal conductivity is a measure of a materials ability to conduct heat relative to its volumetric heat capacity. The distribution of heat due to conduction is described by a parabolic partial differential equation, also known as the heat equation. For an isotropic material without internal heat generation, the one-dimensional heat equation becomes:

$$\frac{\partial T}{\partial t} = \frac{k_c}{\rho C_p}\left(\frac{\partial^2 T}{\partial t^2}\right) = \alpha\left(\frac{\partial^2 T}{\partial x^2}\right) \tag{3.8.3}$$

3.8.2 Convection

In the presence of bulk fluid motion, heat is transferred trough a fluid by convection. It's possible to distinguish between forced and natural convection. When the flow is initiated by the buoyancy effect, we have natural convection. If the fluid motion is caused by external means, such as a pump, we have forced convection [45]. It is also usual to classify convection as either internal or external, depending on whether the flow occurs over a plate or inside a pipe. Convection is the most complicated heat transfer mode, and the rate of heat transfer depends on several fluid properties such as: dynamic viscosity μ, thermal conductivity k_c, density ρ, specific heat capacity C_p and fluid velocity. Other important variables are: geometry, surface roughness, and whether the flow is turbulent or laminar. However, despite the complexity, the rate of heat transfer due to convection is proportional to the temperature difference. This is expressed in Newton's law of cooling:

$$\dot{Q}_{conv} = h_c A_s (T_s - T_\infty) \tag{3.8.4}$$

where h_c is the convective heat transfer coefficient, A_s is the surface area with heat transfer, T_s is the surface temperature and T_∞ is the temperature in the fluid sufficiently far from the surface. Even though this expression looks relatively simple, the convective heat transfer coefficient is difficult to determine, since it depends on many of the above-mentioned fluid properties.

3.9 URANS

All simulations tested in this thesis have been run in URANS mode, meaning that Unsteady Reynolds-averaged Navier-Stokes equations have been solved. Let's give a brief definition of URANS equations starting from Reynolds-averaged Navier-Stokes equations (RANS). The RANS equations are time-averaged equations of motion for fluid flow; the idea behind the equations is Reynolds decomposition, whereby an instantaneous quantity is decomposed into its time-averaged and fluctuating quantities. The RANS equations are primarily used to describe turbulent flows and they can be used with approximations based on knowledge of the properties of flow turbulence to give approximate time-averaged solutions to the Navier–Stokes equations (eq. 3.4.13). For a stationary, incompressible Newtonian fluid, the RANS equations can be written in Einstein notation as follows:

$$\rho \bar{u_j} \frac{\partial \bar{u_i}}{\partial x_j} = \rho \bar{f_i} + \frac{\partial}{\partial x_j} \Big[-\bar{p}\delta_{ij} + \mu \Big(\frac{\partial \bar{u_i}}{\partial x_j} + \frac{\partial \bar{u_j}}{\bar{x_i}} \Big) - \rho \overline{u_i' u_j'} \Big] \tag{3.9.1}$$

The left hand side of this equation represents the change in mean momentum of fluid element owing to the unsteadiness in the mean flow and the convection by the mean flow. This change is balanced by the mean body force, the isotropic stress owing to the mean pressure field, the viscous stresses, and apparent stress ($-\rho\overline{u_i' u_j'}$) owing to the fluctuating velocity field, generally referred to as the Reynolds stress. This nonlinear Reynolds stress term requires additional modeling to close the RANS equation for solving, and has led to the creation of many different turbulence models. The time-average operator $\bar{\cdot}$ is a Reynolds operator.

The URANS equations are the usual RANS equations as in eq. 3.9.1, but the transient term $\partial\bar{u_i}/\partial t$ is retained during computations. So, for URANS equations, eq. 3.9.1 becomes:

$$\rho\Big(\frac{\partial\bar{u_i}}{\partial t}+\bar{u_j}\frac{\partial\bar{u_i}}{\partial x_j}\Big)=\rho\bar{f_i}+\frac{\partial}{\partial x_j}\Big[-\bar{p}\delta_{ij}+\mu\Big(\frac{\partial\bar{u_i}}{\partial x_j}+\frac{\partial\bar{u_j}}{\bar{x_i}}\Big)-\rho\overline{u_i'u_j'}\Big] \tag{3.9.2}$$

As will be more accurately seen in chapter 8.3, a one-way coupling between continuum and dispersed phase will be applied. Mathematically, it means that no body forces appear in the momentum equation, so $\bar{f_i}$ is imposed equal to 0 and eq. 3.9.2 becomes:

$$\rho\Big(\frac{\partial\bar{u_i}}{\partial t}+\bar{u_j}\frac{\partial\bar{u_i}}{\partial x_j}\Big)=\frac{\partial}{\partial x_j}\Big[-\bar{p}\delta_{ij}+\mu\Big(\frac{\partial\bar{u_i}}{\partial x_j}+\frac{\partial\bar{u_j}}{\bar{x_i}}\Big)-\rho\overline{u_i'u_j'}\Big] \tag{3.9.3}$$

3.10 Coupling solution with particles

As mentioned in sub-chap. 3.1, Eulerian-Lagrangian method is very useful when simulations of two-phase flows with a continuous and a dispersed phase are carried out: the Eulerian approach calculates the continuum field, instead the Lagrangian approach tracks the path of each single particle by integrating the force balance (eq. 3.10.2) [57]. In the set of equations that solve particle field (location and velocity) heat and mass transfer between particles are neglected, while particles are assumed as spherical. Moreover, the rotational motion of particles is neglected, while the translational motion is calculated based on:

$$\frac{d\bar{x_p}}{dt}=\bar{u_p} \tag{3.10.1}$$

$$m_p\frac{d\bar{u_p}}{dt}=\bar{F_D}+\bar{F_B}+\bar{F_G} \tag{3.10.2}$$

where $\bar{x_p}$ is the position vector of the particle, $\bar{u_p}$ is its velocity and m_p its mass. In the right hand side of eq. 3.10.2 forces with more influence on particle trajectories are represented; in particular F_D is the drag force, F_B is the buoyancy force and F_G is the gravitational force.
The drag force is implemented as follows:

$$\bar{F_D}=\frac{3}{4}\frac{\rho}{\rho_p}\frac{m_p}{d_p}\cdot C_D(\bar{u}-\bar{u_p})\big|\bar{u}-\bar{u_p}\big| \tag{3.10.3}$$

where $\bar{U}$ and ρ are respectively the fluid velocity and density, ρ_p is the particle density and d_p is the particle diameter. The drag coefficient C_D depends on the flow regime and, in this respect, when spherical particles are considered OpenFOAM uses a modified empirical relation:

$$C_D=\begin{cases}\frac{24}{Re_p}\Big(1+\frac{1}{6}Re_p^{2/3}\Big) & Re_p\leq 1000\\ \\ 0.424 & Re_p\geq 1000\end{cases}$$

where the Reynolds number of the particle is defined as follows:

$$Re_p=\frac{\rho d_p(\bar{u_p}-\bar{u})}{\mu}$$

Finally, the bouyancy and the gravitational forces are calculated together as shown in eq. 3.10.4.

$$\bar{F_B} + \bar{F_G} = \frac{(\rho_p - \rho)\pi d_p^3}{6}\bar{g} \tag{3.10.4}$$

where $\bar{g}$ is the gravitational acceleration.

3.11 Heat and mass transfer between continuum and dispersed phase

The heat and mass transfer between the particles and the hot gas is represented by eq. 3.11.1.

$$m_p c_p \frac{dT_p}{dt} = hA_p(T_g - T_p) + h_{fg}\frac{dm_p}{dt} \tag{3.11.1}$$

in which m_p is the mass particle, c_p is the particle heat capacity, T_p is the particle temperature, A_p is the particle surface area, h_{fg} is the latent heat and h is the heat transfer coefficient at the gas-particle interface. This latter parameter is extrapolated from the Ranz-Marshall equation (3.11.2), referring to spherical particles [58].

$$Nu = \frac{hd_p}{k} = 2 + 0.6(Re_p)^{1/2}(Pr)^{1/3} \tag{3.11.2}$$

where Nu is the Nusselt number, k is the thermal conductivity of the gas, d_p is the particle diameter, Re_p is the Reynolds number (see chapter 3.10) and Pr is the Prandtl number, calculated with the following empirical correlation:

$$Pr = \frac{\mu c_p}{k}$$

where μ is the kinematic viscosity of the gas.
Droplet evaporation causes a mass transfer between the two phases. The mass transfer rate is given by eq. 3.11.3.

$$\frac{dm_p}{dt} = -k_c A_p(Y_s - Y_g) \tag{3.11.3}$$

where Y_s is the saturation umidity, Y_g is the gas humidity and k_c is the mass transfer coefficient obtained from the Sherwood number:

$$Sh = \frac{k_c d_p}{D_{i,m}} = 2 + 0.6(Re_p)^{1/2}(Sc)^{1/3}$$

in which $D_{i,m}$ is the diffusion coefficient of the ith-species in the mixture (gas phase) and S_c is the Schmidt number, defined as:

$$Sc = \frac{\mu}{\rho D_{i,m}}$$

where ρ is the gas density.

The time-change of the droplet diameter is calculated including in eq. 3.11.3 the density variation, determined by an incompressible, polynomial equations of state:

$$\rho_p = \sum_{i=0}^{N-1} a_i T_p^i \tag{3.11.4}$$

where a_i are polynomial coefficients.

4 A short description of OpenFOAM

OpenFOAM (for "Open source Field Operation And Manipulation") is a C++ toolbox for the development of customized numerical solvers, and pre-/post-processing utilities for the solution of continuum mechanics problems, including computational fluid dynamics (CFD) based on the finite volume method (FVM). The code is released as free and open source software under the GNU General Public License. OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to acoustics, solid mechanics and electromagnetics [48]. In fact OpenFOAM is a collection of approximately 250 applications built upon a collection of over 100 software libraries (modules) and each application performs a specific task within a CFD workflow.

More deeply, the applications fall into two categories: *solvers*, that are each designed to solve a specific problem in continuum mechanics; and *utilities*, that are designed to perform tasks that involve data manipulation. The OpenFOAM distribution contains numerous solvers and utilities covering a wide range of problems such as incompressible or compressible flow, multiphase flow, heat transfer and buoyancy-driven flows, particle-tracking flows, combustion, stress analysis of solids etc [49].

One of the strengths of OpenFOAM is that new solvers and utilities can be created by its users with some pre-requisite knowledge of the underlying method, physics and programming techniques involved.

OpenFOAM is supplied with pre- and post-processing environments. The interface to the pre- and post-processing are themselves OpenFOAM utilities, thereby ensuring consistent data handling across all environments.

The overall structure of OpenFOAM is shown in Figure 4.1 .

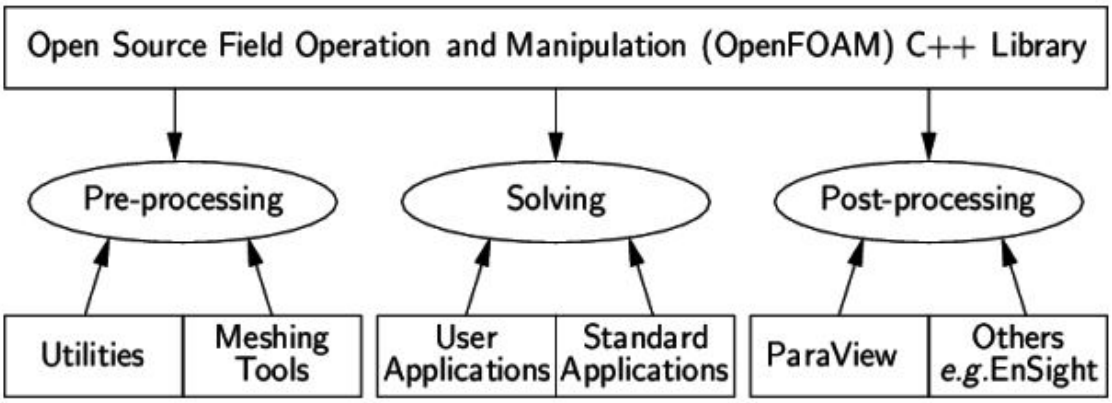


Figure 4.1: Overview of OpenFOAM structure [48].

4.1 Solvers and utilities structure in OpenFOAM

As mentioned above, the *applications* directory contained into the installation folder of OpenFOAM, includes *solvers* and *utilities*. The generic file ***solverName.C*** is located in *solverName* folder and, as can be seen in Fig.4.1.1, it contains the *source code*, instead the variable declarations, mimicking equations and the initialization commands of the solution are located into the *header files*, having extension ***.H*** .

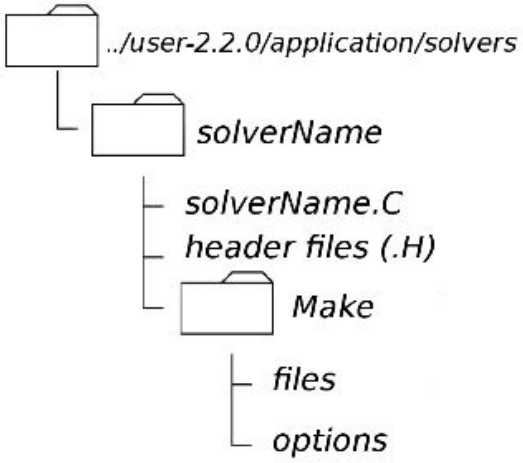


Figure 4.1.1: Overview of a solver directory structure [49].

The *Make* directory contains files in which the name of the solver is specified as well as the destination and the output directory and also the list of all source files used, while the directories of files and libraries recalled from the solver are located in the *options* folder.

The same applies to the directory that contains a generic *utilityName*, as can be seen in the figure below:

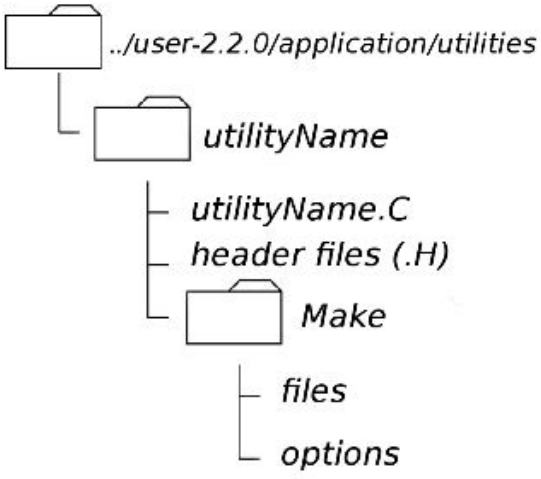


Figure 4.1.2: Overview of an utility directory structure [49].

4.2 File structure of OpenFOAM cases

The basic directory structure for an OpenFOAM case, that contains the minimum set of files required to run an application, is shown in Figure 4.2.1 and described as follows:

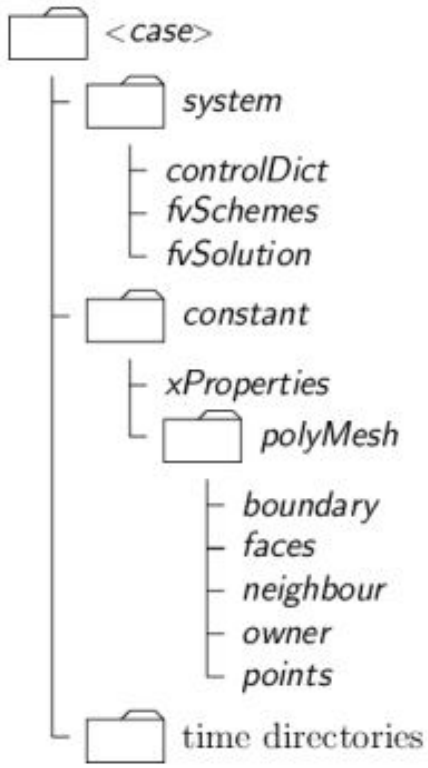


Figure 4.2.1: Case directory structure [50].

The ***system*** directory is to set parameters associated with the solution procedure itself. It contains at least the following 3 files: *controlDict* where run control parameters are set including start/end time, time step and parameters for data output; *fvSchemes* where discretisation schemes used in the solution may be selected at run-time, and *fvSolution* where the equation solvers, tolerances and other algorithm controls are set for the run.

As regards the ***constant*** directory, it contains a full description of the case mesh in a subdirectory polyMesh and files specifying physical properties for the application concerned.

Finally, individual files of data for particular fields can be found into the ***time directories*** . The data can be initial values and boundary conditions that the user must specify to define the problem or results written to file by OpenFOAM. It's important to note that the OpenFOAM fields must always be initialised, even when the solution does not strictly require it, as in steady-state problems. The name of each time directory is based on the simulated time at which the data is written; without getting into specifics, it is sufficient to say that since we usually start our simulations at time $t = 0$, the initial conditions are usually stored in a directory named *0*.

4.3 Basic input/output file format

OpenFOAM needs to read a range of data structures such as strings, scalars, vectors, tensors, lists and fields. The input/output (I/O) format of files is designed to be extremely flexible to enable the user to modify the I/O in OpenFOAM applications as easily as possible. The I/O follows a simple set of rules that make the files extremely easy to understand, in contrast to many software packages whose file format may not only be difficult to understand intuitively but also not be published anywhere.

The format follows some of the general principles of C++ source code:

- Files have free form, with no particular meaning assigned to any column and no need to indicate continuation across lines.
- Lines have no particular meaning except to a // comment delimiter which makes OpenFOAM ignore any text that follows it until the end of line.
- A comment over multiple lines is done by enclosing the text between /* and */ delimiters.

Furthermore OpenFOAM uses dictionaries as the most common means of specifying data. A dictionary is an entity that contains a set of data entries that can be retrieved by the I/O by means of keywords. Most OpenFOAM data files are themselves dictionaries containing a set of keyword entries. Dictionaries provide the means for organising entries into logical categories and can be specified hierarchically so that any dictionary can itself contain one or more dictionary entries. The format for a dictionary is to specify the dictionary name followed the entries enclosed in curly braces as follows:

```
<dictionaryName>
{
    … keyword entries …
}
```

Another characteristic to underscore is that all data files that are read and written by OpenFOAM begin with a dictionary named FoamFile containing a standard set of keyword entries, listed in Table below:

[H]

Keyword	Description	Entry
`version`	I/O format version	`2.0`
`format`	Data format	`ascii /binary`
`location`	Path to the file, in "…"	(optional)
`class`	OpenFOAM class constructed from the data file concerned	typically `dictionary` or a field, *e.g.*`volVectorField`
`object`	Filename	*e.g.*`controlDict`

The table provides brief descriptions of each entry, which is probably sufficient for most entries with the notable exception of class. The class entry is the name of the

C++ class in the OpenFOAM library that will be constructed from the data in the file. Without knowledge of the underlying code which calls the file to be read, and knowledge of the OpenFOAM classes, the user will probably be unable to surmise the class entry correctly. However, most data files with simple keyword entries are read into an internal dictionary class and therefore the class entry is dictionary in those cases.

4.4 OpenFOAM programming language

The success of verbal language and mathematics is based on efficiency, especially in expressing abstract concepts. For example, in fluid flow, we use the term "velocity field", which has meaning without any reference to the nature of the flow or any specific velocity data. The term encapsulates the idea of movement with direction and magnitude and relates to other physical properties. In mathematics, we can represent velocity field by a single symbol, *e.g.* $\boldsymbol{U}$, and express certain concepts using symbols, *e.g.* "the field of velocity magnitude" by $|\boldsymbol{U}|$. The advantage of mathematics over verbal language is its greater efficiency, making it possible to express complex concepts with extreme clarity [49].

The problems that we wish to solve in continuum mechanics are not presented in terms of intrinsic entities, or types, known to a computer, *e.g.* bits, bytes, integers. They are usually presented first in verbal language, then as partial differential equations in 3 dimensions of space and time. The equations contain the following concepts: scalars, vectors, tensors, and fields thereof; tensor algebra; tensor calculus; dimensional units. The solution to these equations involves discretisation procedures, matrices, solvers, and solution algorithms.

As mentioned above, OpenFOAM is based on C++ programming language that is an OOP (Object-Oriented Programming), meaning that it provides the mechanism — *classes* — to declare types and associated operations that are part of the verbal and mathematical languages used in science and engineering. Our velocity field introduced earlier can be represented in programming code by the symbol $\boldsymbol{U}$ and "the field of velocity magnitude" can be mag($\boldsymbol{U}$). The velocity is a vector field for which there should exist, in an object-oriented code, a *vectorField* class. The velocity field U would then be an instance, or *object*, of the *vectorField* class; hence the term object-oriented.

The clarity of having objects in programming that represent physical objects and abstract entities should not be underestimated. The class structure concentrates code development to contained regions of the code, *i.e.* the classes themselves, thereby making the code easier to manage. New classes can be derived or inherit properties from other classes, *e.g.* the *vectorField* can be derived from a *vector* class and a *Field* class. C++ provides the mechanism of *template classes* such that the template class *Field<Type>* can represent a field of any *<Type>, e.g. scalar, vector, tensor.* The general features of the template class are passed on to any class created from the template. Templating and inheritance reduce duplication of code and create class hierarchies that impose an overall structure on the code.

A central theme of the OpenFOAM design is that the solver applications, written using the OpenFOAM classes, have a syntax that closely resembles the partial

differential equations being solved. For example the equation

$$\frac{\partial(\rho \boldsymbol{U})}{\partial t} + \nabla \cdot (\phi \boldsymbol{U}) - \nabla \cdot \mu \nabla \boldsymbol{U} = -\nabla p \tag{4.4.1}$$

is represented by the code

```
solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
    ==
  - fvc::grad(p)
);
```

This and other requirements demand that the principal programming language of OpenFOAM has object-oriented features such as inheritance, template classes, virtual functions and operator overloading. These features are not available in many languages that purport to be object-orientated but actually have very limited object-orientated capability, such as FORTRAN-90. C++, however, possesses all these features while having the additional advantage that it is widely used with a standard specification so that reliable compilers are available that produce efficient executables. It is therefore the primary language of OpenFOAM.

The four principal characteristics of an OOP are:

- Abstraction: it is a powerful methodology to manage complex systems. Abstraction is managed by well-defined objects and their hierarchical classification.
- Encapsulation: the idea of encapsulation is to keep classes separated and prevent them from having tightly coupled with each other. Encapsulation binds the data with the code that manipulates it and it keeps the data and the code safe from external interference.
- Inheritance: it is the mechanism by which an object acquires the some/all properties of another object and it supports the concept of hierarchical classification.
- Polymorphism: it is usefull to process objects differently based on their data type, in other words it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object). This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.

4.5 Different types of boundaries in OpenFOAM

In this sub-section we discuss the way in which boundaries are treated in OpenFOAM and this will be very important in order to better understand the boundary conditions used in this thesis (see subchapter 6.2 and chapter 8). The subject of boundaries is a little involved because their role in modelling is not simply that of a geometric entity but an integral part of the solution and numerics through boundary conditions or inter-boundary 'connections' [53].

We first need to consider that, for the purpose of applying boundary conditions, a boundary is generally broken up into a set of *patches*. One patch may include one or more enclosed areas of the boundary surface which do not necessarily need to be physically connected.

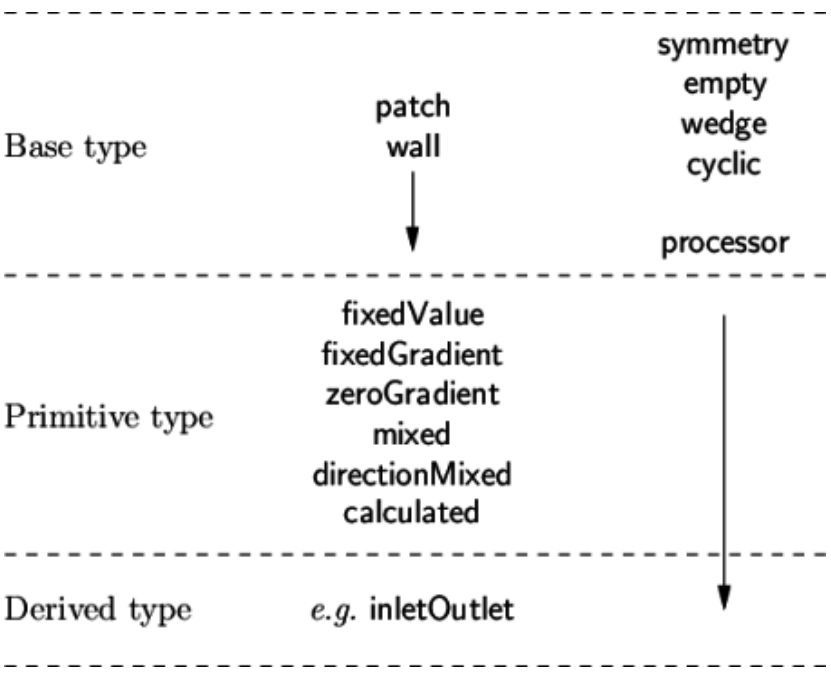


Figure 4.5.1: Patch attributes [53].

There are three attributes associated with a patch that are described below in their natural order and Figure 4.5.1 shows the names of different patch types introduced at each level of the hierarchy.

- Base type: the type of patch described purely in terms of geometry or a data 'communication link'.
- Primitive type: the base numerical patch condition assigned to a field variable on the patch.
- Derived type: a complex patch condition, derived from the primitive type, assigned to a field variable on the patch.

4.5.1 Base types

The base and geometric types are described below.

- patch: the basic patch type for a patch condition that contains no geometric or topological information about the mesh (with the exception of wall), e.g. an inlet or an outlet.

- wall: There are instances where a patch that coincides with a wall needs to be identifiable as such, particularly where specialist modelling is applied at wall boundaries. A good example is wall turbulence modelling where a wall must be specified with a *wall* patch type, so that the distance from the wall to the cell centres next to the wall are stored as part of the patch.

- symmetryPlane: for symmetry plane.

- empty: While OpenFOAM always generates geometries in 3 dimensions, it can be instructed to solve in 2 (or 1) dimensions by specifying a special *empty* condition on each patch whose plane is normal to the 3rd (and 2nd) dimension for which no solution is required.

- wedge: For 2 dimensional axi-symmetric cases, e.g. a cylinder, the geometry is specified as a wedge of small angle (e.g. < 5) and 1 cell thick running along the plane of symmetry, straddling one of the coordinate planes. The axi-symmetric wedge planes must be specified as separate patches of *wedge* type.

- cyclic: Enables two patches to be treated as if they are physically connected; used for repeated geometries, e.g. heat exchanger tube bundles. One *cyclic* patch is linked to another through a *neighbourPatch* keyword in the *boundary* file. Each pair of connecting faces must have similar area to within a tolerance given by the *matchTolerance* keyword in the *boundary* file. Faces do not need to be of the same orientation.

- processor: If a code is being run in parallel, on a number of processors, then the mesh must be divided up so that each processor computes on roughly the same number of cells. The boundaries between the different parts of the mesh are called *processor* boundaries.

4.5.2 Primitive types

The primitive types referring to an hypothetical field ϕ are listed below.

- fixedValue: the value of ϕ is specified.

- fixedGradient: the normal gradient of ϕ is specified.

- zeroGradient: the normal gradient of ϕ is zero.

- calculated: the boundary field ϕ derived from other fields.

- mixed: mixed *fixedValue/fixedGradient* condition depending on the value in *valueFraction*.

- directionMixed: a *mixed* condition with tensorial *valueFraction*, e.g. for different levels of mixing in normal and tangential directions.

4.5.3 Derived types

There are numerous derived types of boundary conditions depending on the primitive types from which they derive. For example, many complex conditions are derived from *fixedValue*, where the value is calculated by a function of other patch fields, time, geometric information, etc. Some other conditions derived from *mixed/directionMixed* switch between *fixedValue* and *fixedGradient* (usually *zeroGradient*).

In particular, the derived types conditions regarding the inlet and the outlet are very useful for this thesis.

5 Parallel computing

In CFD computers are used to perform the calculations required to simulate the interaction of liquids and gases with surfaces defined by boundary conditions. To be honest workstations replaced normal computers in order to achieve better and more accurate solutions; in fact the workstation offers higher performance than mainstream personal computers, especially with respect to CPU and graphics, memory capacity, and multitasking capability. These characteristics make it suitable for technical or scientific applications and, in particular, workstations were optimized for the visualization and manipulation of different types of complex data such as 3D mechanical design, engineering simulation (e.g. computational fluid dynamics), animation and rendering of images, and mathematical plots.

It's convenient to give a brief definition of some fundamental physical components and concepts of a workstation so as to easily understand its potential performance. Since the advent of multi-core technology, such as dual-cores and quad-cores, the term *processor* has been used to describe a logical execution unit or a physical chip. A multi-core chip may have several cores. With the advent of multi-core technology, the term *processor* has become context-sensitive, and it is largely ambiguous when describing large multi-core systems. Essentially a core comprises a logical execution unit containing an L1 cache and functional units.

Another component of the workstation is the *chip* or *CPU chip* refers to the actual integrated circuit (IC) on a computer; a chip mainly refers to execution unit that can be a single core technology or a multicore technology.

Then, the socket refers to a physical connector on a computer motherboard that accepts a single physical chip. Many motherboards can have multiple sockets that can in turn accept multi-core chips.

A *process* is an independent program running on a computer; it has a full stack of memory associated for its own use, and does not depend on another process for execution. MPI (Message Passing Interface) processes are true processes because they can run on independent machines or the same machine. A concept linked to the *process* is the *thread*: it is essentially a process that does not have a full stack of memory associated for it. The thread is tied to a parent process, and is merely an offshoot of execution. Typically thread processes must run on the same computer, but can execute simultaneously on separate cores of the same node.

In particular, two identical workstations "**4U Dual Socket Intel - 8 bays SAS/ SATA - 1200W**" (Figure 5.1) have been used for simulations run in this thesis.

The workstations have been named *Amarok* and *Shabang* by the Department of Civil, Chemical and Environmental Engineering in University of Genoa.
An overview on the specifics of the workstations employed by this thesis can be seen in the list below:

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	32
On-line CPU(s) list:	0-31
Thread(s) per core:	2
Core(s) per socket:	8
Socket(s):	2
NUMA node(s):	2
Vendor ID:	GenuineIntel
CPU family:	6
Model:	62
Model name:	Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz
Stepping:	4
CPU MHz:	1200.000
BogoMIPS:	4005.09
Virtualization:	VT-x
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	20480K
NUMA node0 CPU(s):	0-7,16-23
NUMA node1 CPU(s):	8-15,24-31

In summary, it can be possible to note that workstations employed for this thesis use a total number of cores available after hyper threading (virtual cores) equal to 32, a number of threads per core (hyper threading) amounting to 2, a number of cores per socket (physical processor) equivalent to 8 and a number of sockets (physical processors) equal to 2. So, remembering that:

$$\textit{Number of physical cores} = \textit{Number of cores per socket X Number of sockets}$$

in this case the number of physical cores is 16.

A common adopted technique in CFD area is *parallel computing*; it is a very efficient procedure which allows to solve larger and more complex problems (scale-up) and drastically reduce time simulation and therefore costs simulation. Traditionally, computer software has been written for serial computation. To solve a problem, an algorithm is constructed and implemented as a serial stream of instructions. These instructions are executed on a central processing unit on one computer. Only one instruction may execute at a time—after that instruction is finished, the next one is executed.

Figure 5.1: Workstation used for this thesis.

Parallel computing, on the other hand, uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be different and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of them.

In summary, finding the solution with the *parallel computing* means that firstly, a problem is broken into discrete parts that can be solved concurrently and each part is further broken down to a series of instructions. Then, these instructions from each part execute simultaneously on different processors and an overall control/coordination mechanism is employed [51].

5.1 Running in parallel with OpenFOAM

To take full advantage of the hardware, it is necessary to use the maximum number of physical cores when running in parallel, while using the maximum number of virtual cores, OpenFOAM will run but it will be slower in comparison to running with the maximum number of physical cores (or even less cores).

To run operationally in parallel with OpenFOAM it is necessary to follow some specific step [54]:

- Decompose the domain: *decomposePar* utility is used and it is based on the *decomposeParDict* which is a dictionary located in the *system* directory. (Example in Figure 5.1.1)

- Distribute the jobs among the processors or computing nodes: OpenFOAM uses the public domain OpenMPI implementation of the standard message passing interface (MPI). By using MPI, each processor runs a copy of the solver on a separate part of the decomposed domain.

- Reconstruct the domain: *reconstructPar* utility is used to obtain the final result. (Example in Figure 5.1.2)

The method of parallel computing used by OpenFOAM is known as domain decomposition, in which the geometry and associated fields are broken into pieces and distributed among different processors. The main goal is to break up the domain with minimal effort but in such a way to guarantee a fairly economic solution; in other words, the purpose is to minimize the inter-processors communication and the processor workload. In the *decomposeParDict* dictionary the user must set the number of domains in which the case should be decomposed. It is indicated with ***NP*** and, usually, it corresponds to the number of physical cores available. Then a set of subdirectories will be created, one for each processor: the directories are named *processorN* where $N = 0, 1, 2, 3$, and so on. Each directory contains the decomposed fields, namely the mesh information, boundary and initial conditions and the solution for that processor.

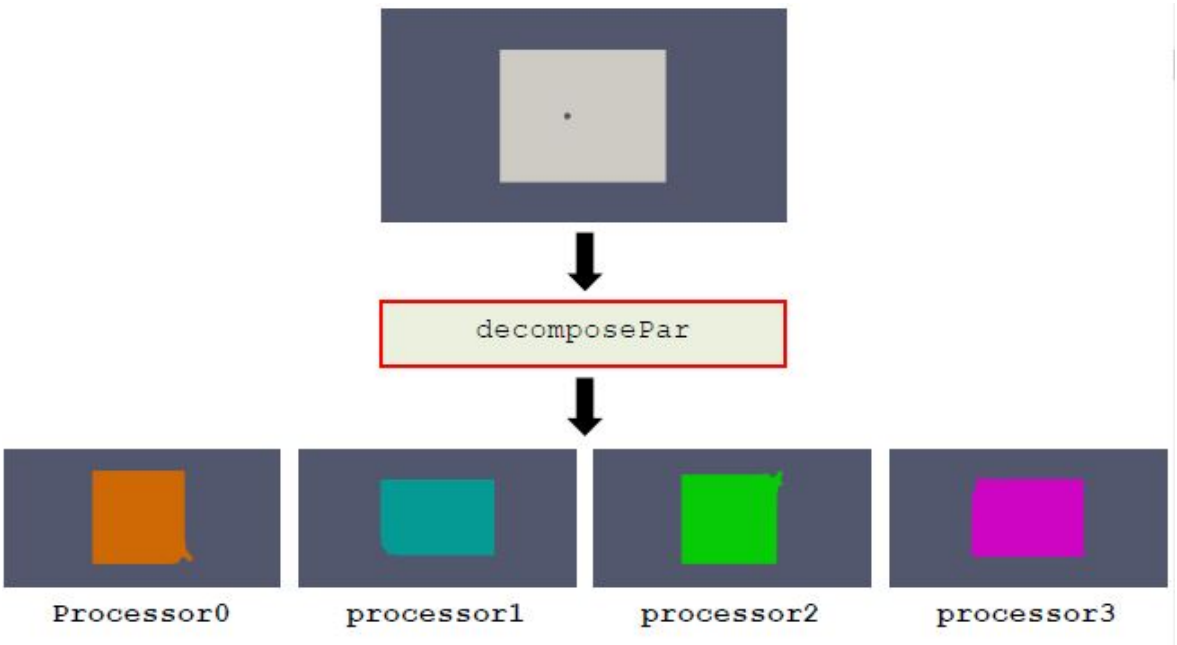


Figure 5.1.1: Decomposition - Subdomains [54].

When the case is reconstructed, all the information contained in the decomposed case are glued together.

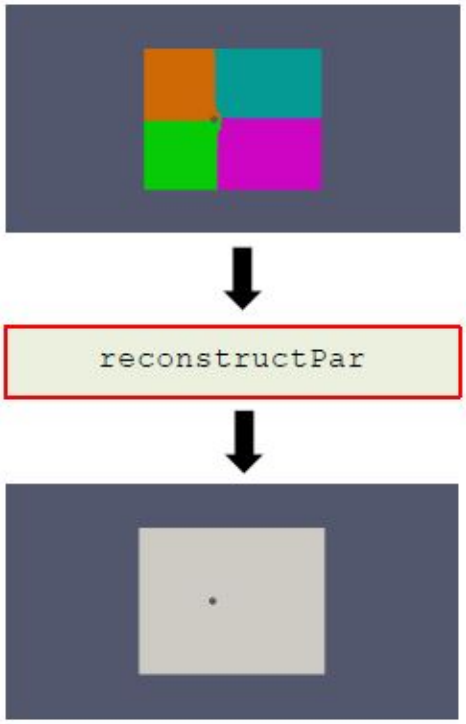


Figure 5.1.2: Reconstruction [54].

In this thesis, solutions have been achieved by decomposing the domain represented by the spray dryer and the associated field using 16 processors (maximum number of physical cores) or 32 processors (maximum number of physical and virtual cores) based on need to run one or two different simulations at the same time.

6 The case study

The choice of the spray dryer's geometry to take as basis for the *case study* has been meticulous and complicated by the fact that in the literature information about the project details (in particular details about the design of the inlet and the outlet of the spray dryer) are not reported. Therefore, even though all the articles considered have taken as reference the same experimental set-up, it has been impossible to reconstruct it adequately.

6.1 Original geometry and parameters

Most of the information has been obtained from Anandharamakrishnan[24], Huang [7] and Kieviet's geometry [4] represented in figures 6.1.1, 6.1.2 and 6.1.3. As can be seen, this spray dryer is characterized by a short-form geometry (see chapter 2), in fact the dimensions of the main body (cylinder and cone parts) are well known.

Besides the geometry, inlet data-setup regarding inlet air turbulence condition, liquid spray and chamber wall conditions have been extrapolated from reference articles [3], [7], [24]. This is very important in order to have suitable guidelines that will allow

to compare final results with references; the complete setup is provided in detail in table 6.1 .

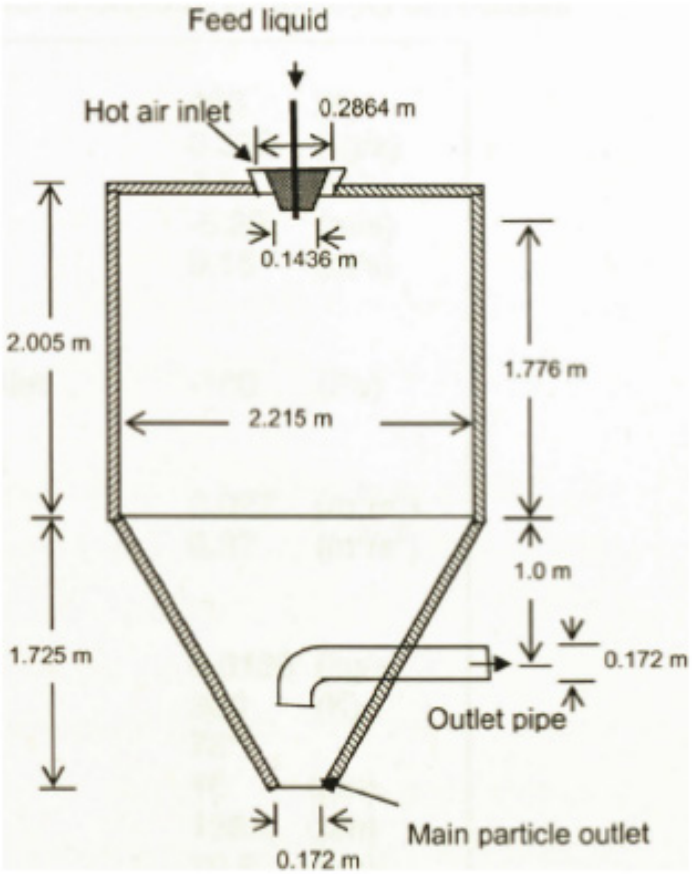


Figure 6.1.1: Anandharamakrishnan's geometry [24].

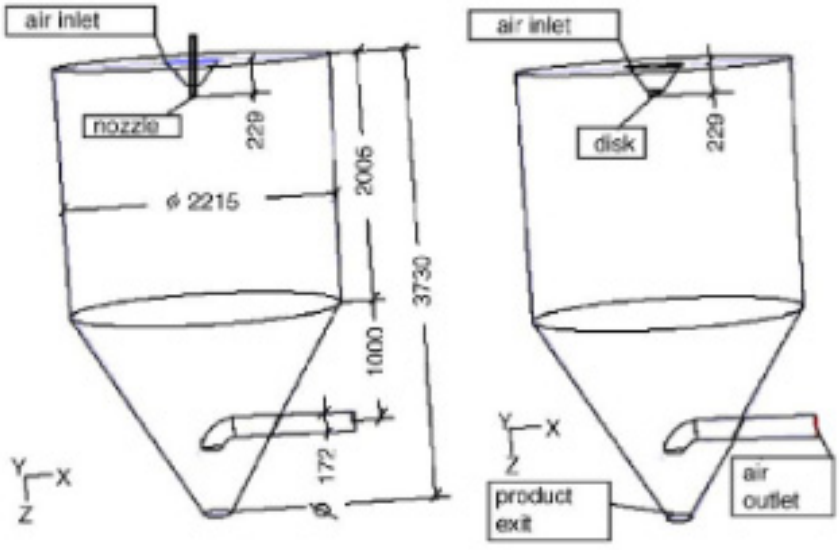


Figure 6.1.2: Huang's geometry [7].

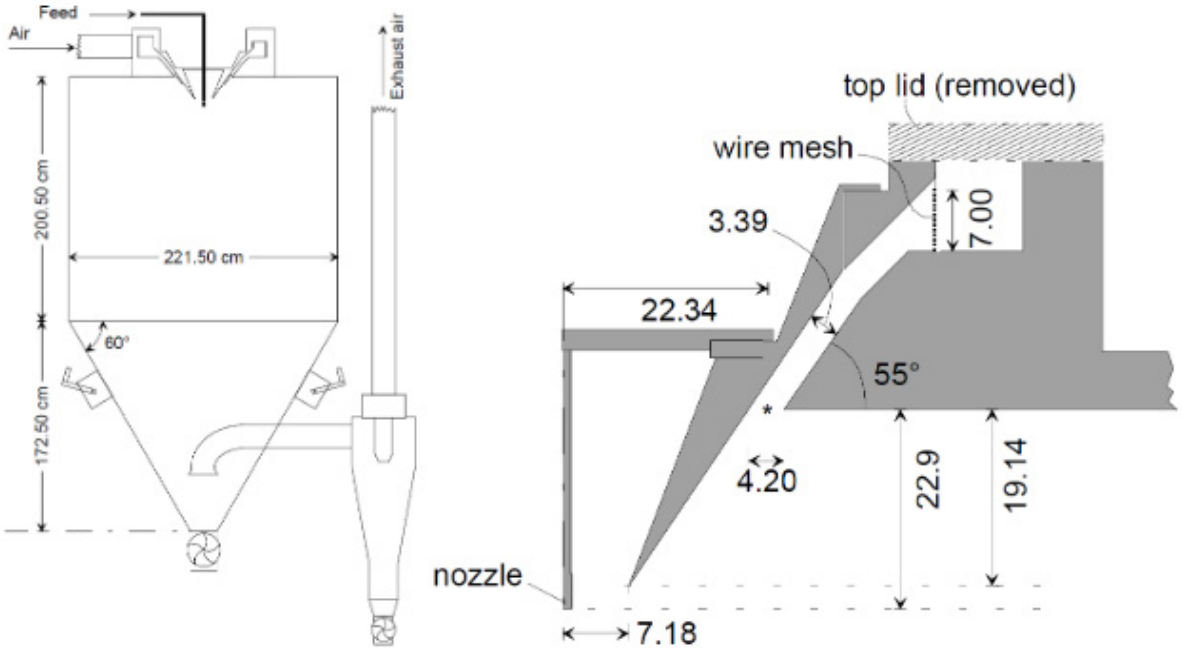


Figure 6.1.3: Kieviet's geometry [4].

Inlet Air		
Air inlet temperature	468	(K)
Air mass flow rate	0.336	(kg/s)
Air axial velocity	7.5	(m/s)
Air radial velocity	-5.25	(m/s)
Air total velocity	9.15	(m/s)
Outlet Condition		
Outflow and reference at outlet	-100	(Pa)
Turbulence inlet condition		
Turbulence k-value	0.027	(m^2/s^2)
Turbulence ϵ-value	0.37	(m^2/s^3)
Liquid spray from nozzle		
Liquid feed rate (spray rate)	0.0139	(kg/s)
Feed Temperature	300	(K)
Spray angle	76	(deg)
Minimum droplet diameter	10	(μm)
Maximum droplet diameter	138.0	(μm)
Average droplet diameter	70.5	(μm)
Droplet velocity at nozzle exit	59	(m/s)
Rosin-Rammler parameter	2.05	

Chamber wall conditions		
Chamber wall thickness	0.002	(m)
Wall material	Steel	
Overall wall-heat transfer coefficient	3.5	(W/m^2K)
Air temperature outside wall	300	(K)
Interaction between wall and droplet	Escape	

Table 6.1: Boundary Condition in references [24].

6.2 The modified geometry

As mentioned above, in the literature there is limited information about the details of the atomiser and, subsequently, about all surfaces constituting the inlet, as well as for the outlet pipe. That is the reason why different configurations have been carefully studied so as to find the appropriate geometry able to reproduce the inlet data-setup. Before looking at the selected configuration in more detail, it's necessary to take an overview of all the patches forming the spray dryer.

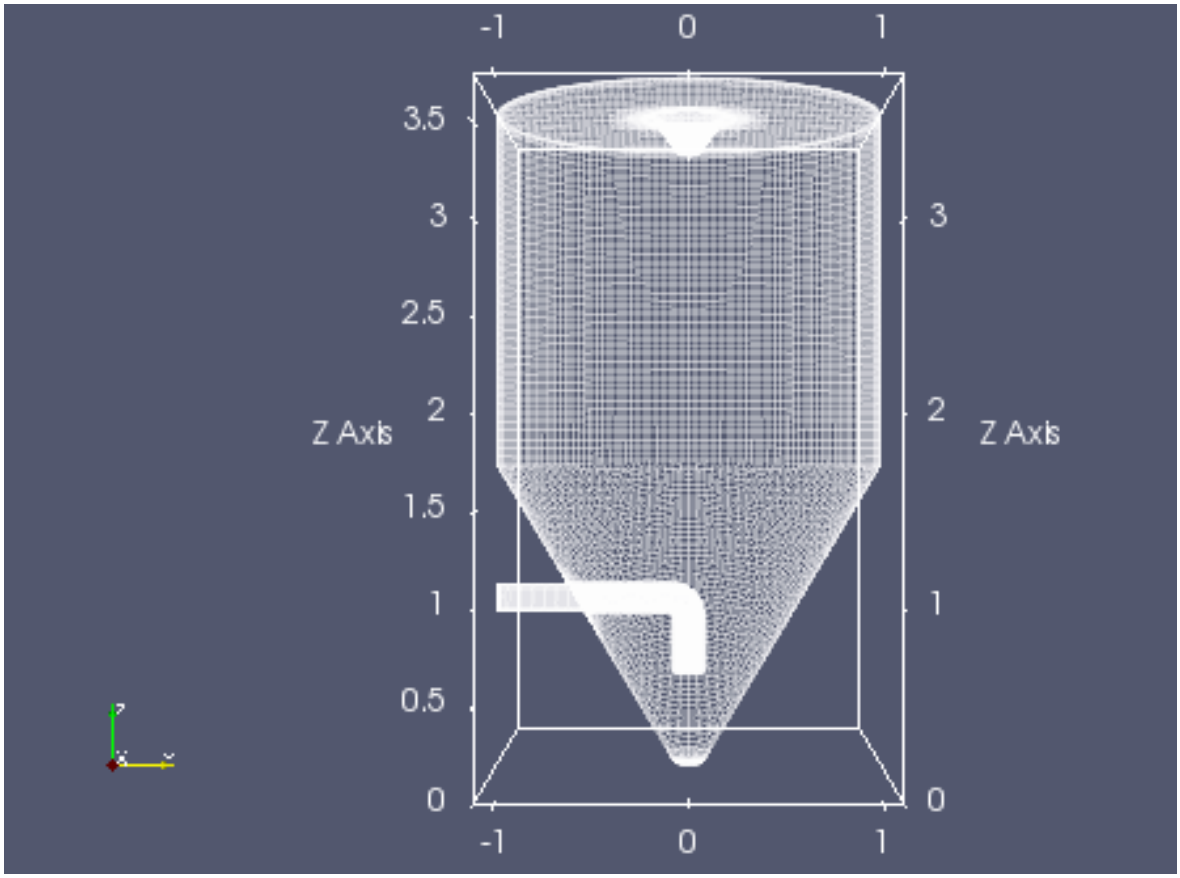


Figure 6.2.1: Overview of the whole geometry.

Single patches constituting the whole geometry are briefly described in the following list and represented in figures below. Details about boundary conditions will be reported later.

- type **wall**:
  - *Wall*: it's the side surface of the chamber of the spray dryer;
  - *WallCone*: it's the side surface of the cone in the lower part of the spray dryer;
  - *WallTop*: it's the top surface of the cylinder;
  - *Pipe*: it's the pipe through the exhausted air goes away;
  - *PipeTurn*: it's the last part of the pipe, that *turns* from the horizontal direction to the bottom;
  - *AtomizerInletCone*: it's the part of the truncated cone in front of the inlet;
  - *AtomizerBot*: it's the base of the truncated cone;
  - *AtomizerCone*: it's the truncated cone that is between the *AtomizerInletCone* and the *AtomizerBot*. From one of its bases the sprayer comes out;
  - *AtomizerPipe*: it's the little pipe of the sprayer;
  - *AtomizerInletFood*: it's the sprayer in itself, it is set as a wall in this first step;
- type **inlet**:
  - *AtomizerInletAir*: it's the air inlet;
  - *InletPipe*: it's the pipe inlet, air is apired in;
- type **outlet**:
  - *OutletCone*: where to heavy food particles are collected;
  - *OutletPipe*: to the cyclon separator, air is aspired out;

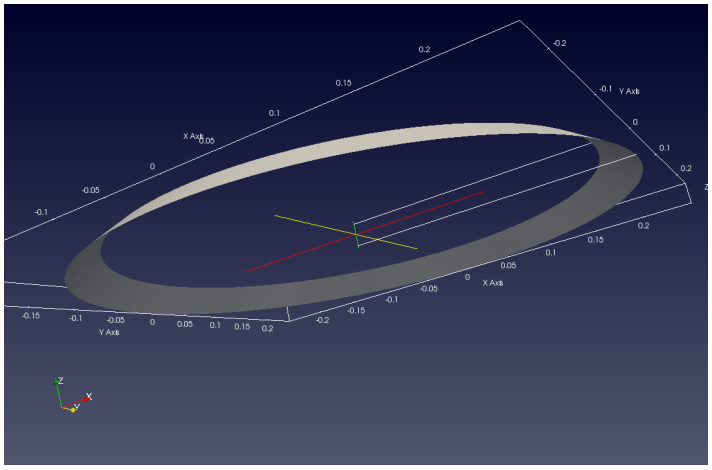
Dimensions of each patch are reported in the table below:

Patch	**Dimension (m)**
Wall $r = 1.1075$	$h = 2.005$
WallCone $r_1 = 1.107$ $r_2 = 0.086$	$h = 1.725$
Pipe distance from **WallTop**= 3.005	$D = 0.172$
PipeTurn	$r_1 = r_2 = 0.086$

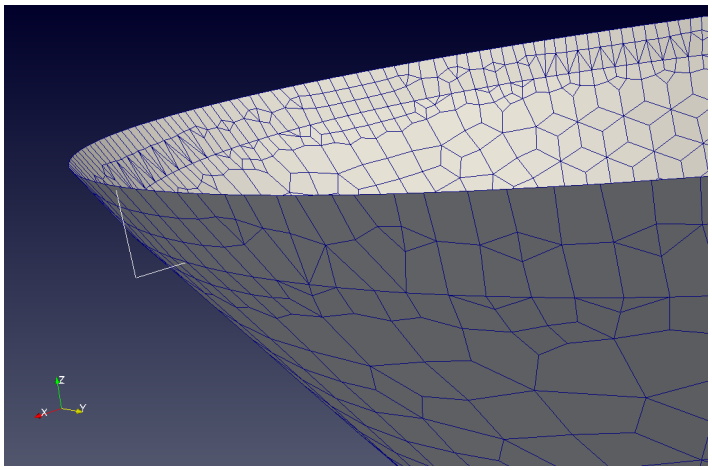
AtomizerCone	$h = 0.1914$
$r_1 = 0.2058$	
$h = 0.1914$	
AtomizerInletAir	$h = 0.021$
$r_1 = 0.2059$	
$r_2 = 0.2179$	
AtomizerInletCone	$h = 0.021$
$r_1 = 0.2179$	
$r_2 = 0.2479$	

Table 6.2: Dimensions of patch.

Minor an major radius reported in table 6.2 (r_1 and r_2) for both *AtomizerInletAir* and *AtomizerInletCone* are better illustrated in Figure 6.2.2.

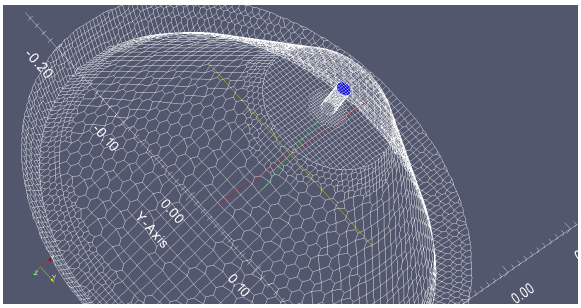


(a) The two radii of the *AtomizerInletAir*.

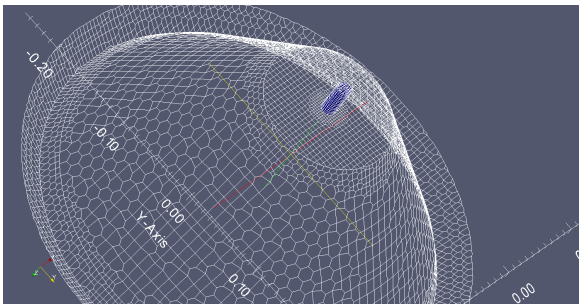


(b) Difference between the two radii of the *AtomizerInletCone*.

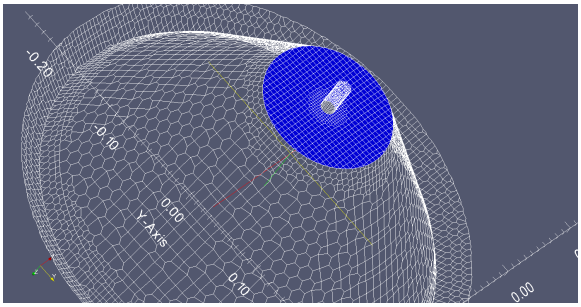
Figure 6.2.2: Radii of the two atomizer patches.



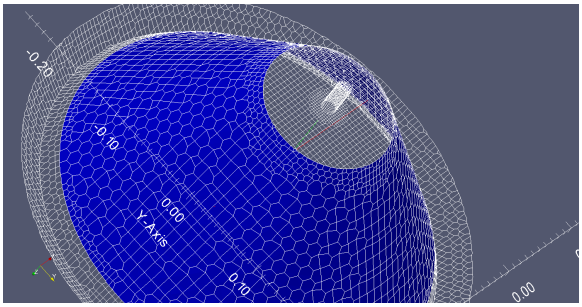
(a) AtomizerInletFood



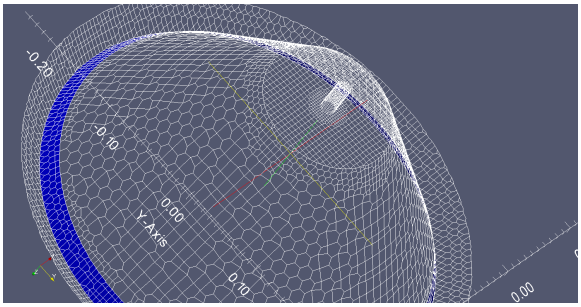
(b) AtomizerPipe



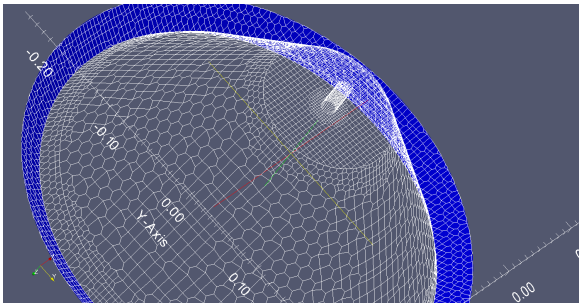
(c) AtomizerBot



(d) AtomizerCone

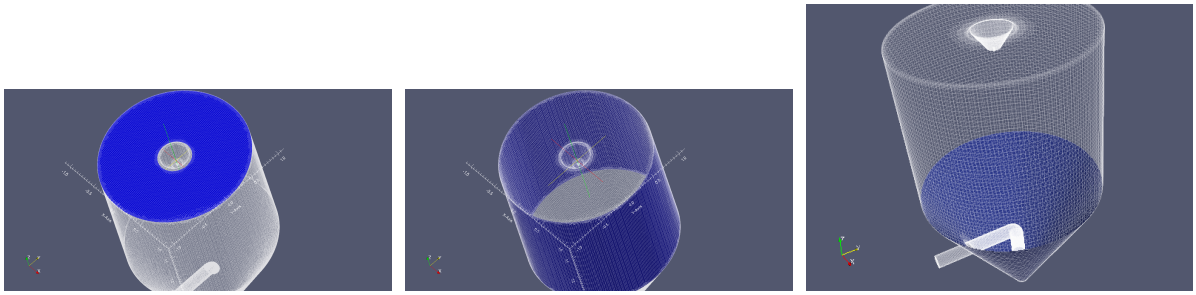


(e) AtomizerInletCone



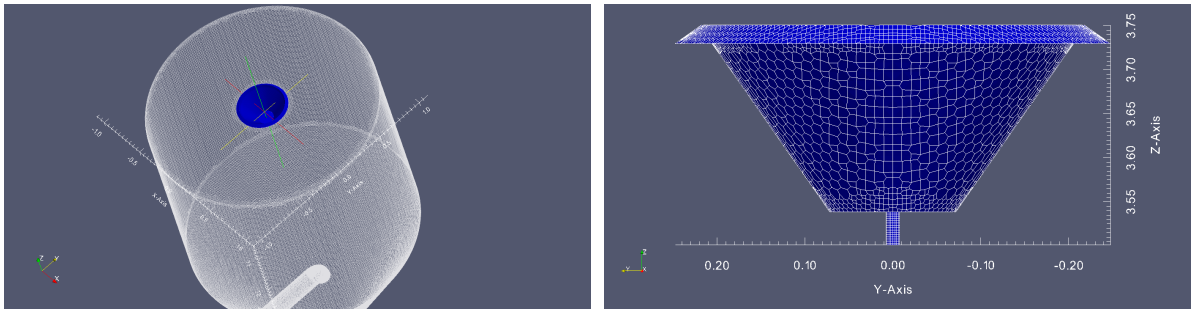
(f) AtomizerInletAir

Figure 6.2.3: Different parts of the atomizer.



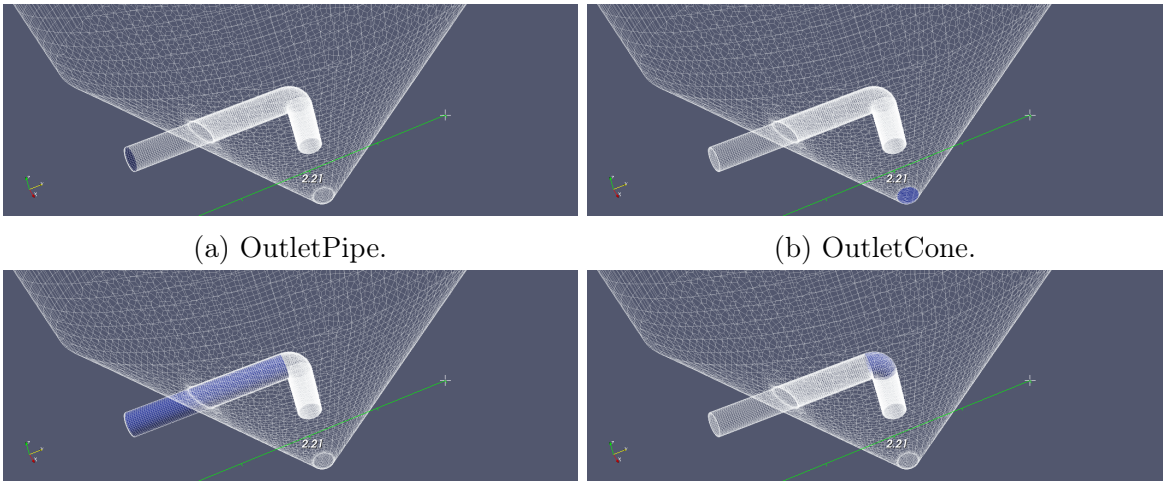
(a) WallTop (b) Wall (c) WallCone

Figure 6.2.4: External wall.



(a) Position (b) Overview of the atomizer

Figure 6.2.5: Atomizer.



(a) OutletPipe. (b) OutletCone.

(c) OutputPipe length (d) PipeTurn

Figure 6.2.6: Pipe for the exhausted air and outlet surfaces.

As stated above, some parts of the geometry have been varied in the most appropriate way because of the absence of reliable data in literature; let's now proceed to the description of the modified geometry, subject of this thesis, by detailing the step process.

As concerns the atomiser, different configurations have been studied. In the literature take as a base reference [3], [7], [24], the atomizer is an annulus and the velocity has been described via its components in polar coordinates. In the *case study*, the lateral surface of a truncated cone has been chosen to use as inlet. The surface has normal direction, parallel to the velocity vector. In addition, it has been maintained the same extension of the surface of the inlet in the literature, so as to obtain the same average velocity by imposing the same airflow as inlet condition.

Another component that has been varied in the geometry under consideration is the vertical part of the pipe. This component is the first piece of duct that absorbs particles evolving into the spray dryer and it has been named *PipeDown* as shown in Figure 6.2.7 (highlighted in blue):

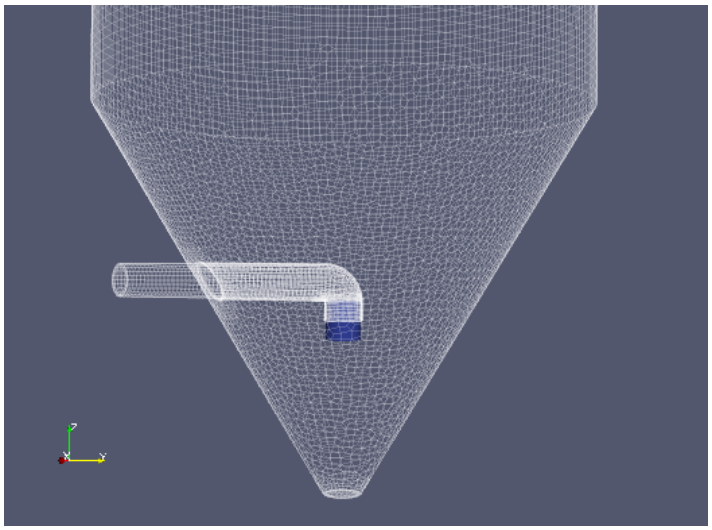


Figure 6.2.7: PipeDown.

In particular, three different configurations of the *PipeDown* have been tested in order to choose the one which was able to approach literature data with most successful. Three different simulations have been run as the length of the *PipeDown* changes, as can be seen in Figure 6.2.8. These lengths have been chosen so as to try a wide range of cases:

case A) $L_{PipeDown} = 0.5D$

case C) $L_{PipeDown} = 2D$

case D) $L_{PipeDown} = 0.3m$

where $D = 0.172$ m is the pipe diameter.

Finally, the choice fell on the case C) for which $L_{PipeDown} = 2D = 0.344$ m. The reason of this selection will be better explained in chapter 9.

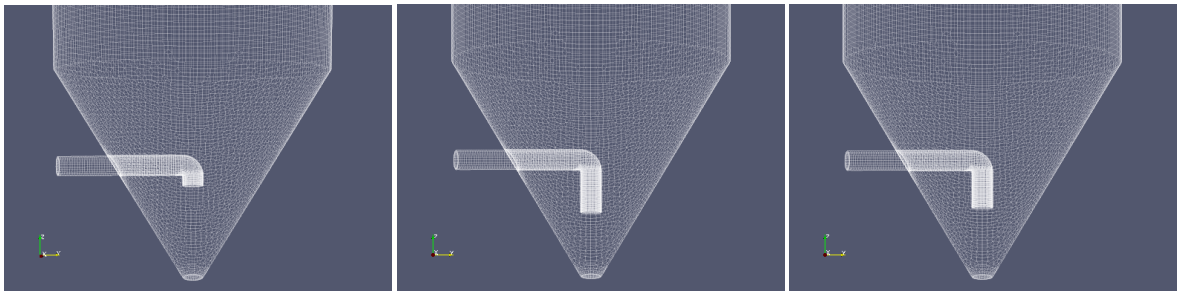


Figure 6.2.8: Different configurations of the PipeDown. case A) on the left - case C) in the middle - case D) on the right

Again with regard to the pipe, its output length aside from the spray dryer chamber (Figure 6.8 - (c)) was not exactly specified in reference models, so it has been decided to impose it as the same length of the cylinder radius $r_1 = 1.107$ m, as shown in Figure 6.2.9:

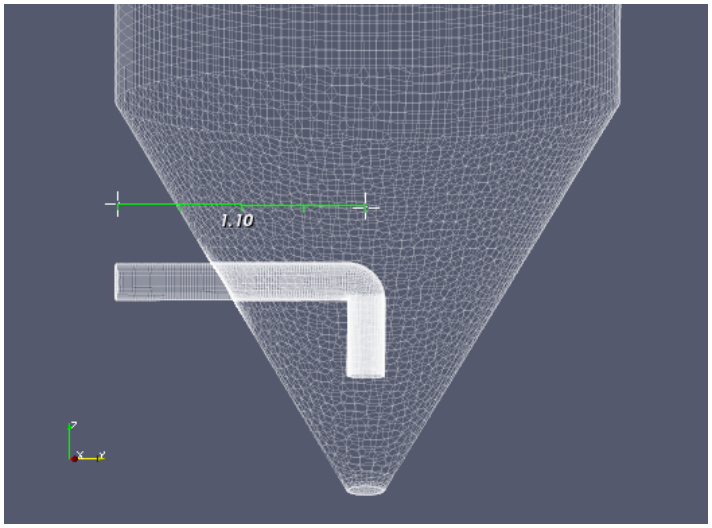
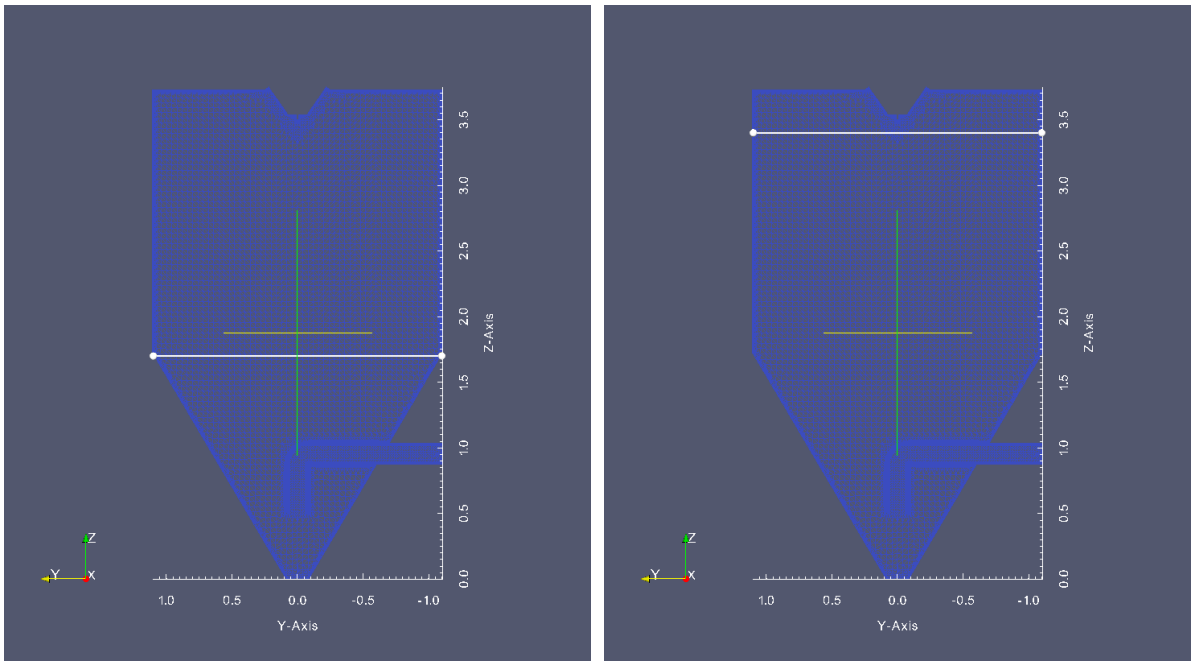


Figure 6.2.9: Pipe output length.

6.3 Reference values of velocity and temperature

Besides reference geometry, it has been very important to find the corresponding reference values of velocity and temperature in order to compare results of the *case study* with the literature . In particular, results reported by Anandharamakrishnan [24] have been taken as baseline for this thesis. These results have been calculated in two different cross sections of the spray dryer as shown in Figure 6.3.1:

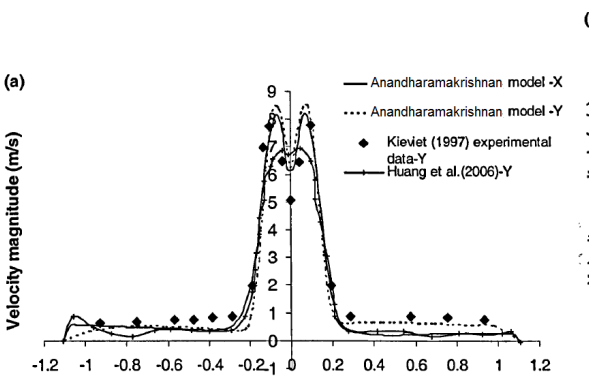
- Near Outlet: cross section 1.7 m away from the *OutletCone*
- Near Inlet: cross section 3.4 m away from the *OutletCone*



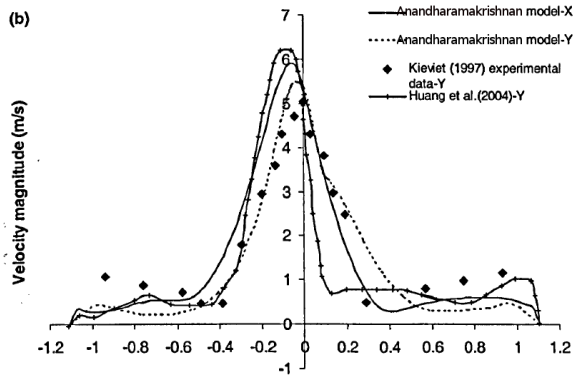
(a) Near Outlet (b) Near Inlet

Figure 6.3.1: Reference sections

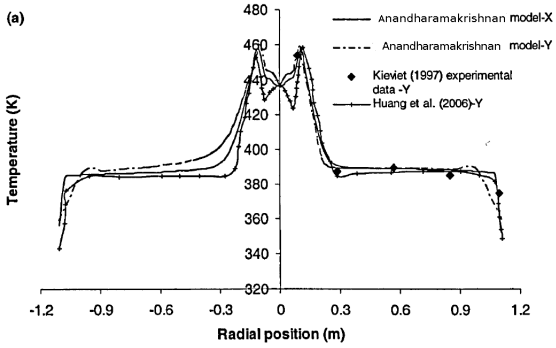
Anandharamakrishnan's references are now represented below; notice that in Figure 6.3.2 are reported results belonging to different models, but all obtained with the same geometry; in particular Kieviet's experimental measurements [3], Huang's simulation prediction [7] and different simulation predictions from two distinct Anandharamakrishnan's models [24] can be distinguished.



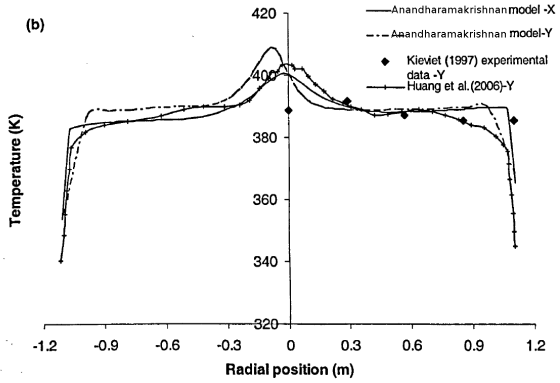
(a) Velocity profile Near Inlet.



(b) Velocity profile Near Outlet.



(c) Temperature profile Near Inlet.



(d) Temperature profile Near Outlet.

Figure 6.3.2: Results reported by Anandharamakrishnan [24].

One of the key points of this thesis will be the comparison between the results of the *case study* and the Kieviet's experimental measurements, so as to verify the validity of the model developed in this thesis; it will be also essential to compare the current results with those of Huang and Anandharamakrishnan in order to take into account the limits of the CFD code compared to the real case.

7 Mesh generation

In this chapter one of the most important step of this thesis, as concerns the accuracy of the solution, is presented.
Mesh generation consists in dividing the physical domain into a finite number of discrete regions called control volumes or cells in which the solution is sought. The mesh can be internal or external, in this case the study regards internal aerodynamics, so that the only internal volume of the spray dryer geometry has been meshed.
Moreover, the choice fell on an unstructured mesh, meaning that it only requires as input the element size on the lines and surfaces that define the geometry. Unstructured meshes are generally hybrid meshes composed of different types of cell; in particular, meshes used in this work are mainly composed of hexahedral cells, and to a lesser extent tetrahedral cells, pyramids, prisms and polyhedra cells.
As regards the mesh quality, no single standard benchmark or metric can effectively evaluate it, but users can rely on suggested best practices by assessing certain mesh properties that can jointly give a sense of mesh quality [54]. These mesh quality metrics are:

- Orthogonality
- Skewness
- Aspect Ratio

As shown in Figure 7.1, mesh orthogonality is the angular deviation of the vector S (located at the face center f) from the vector d connecting the two cell centers **P** and **N**. In very practical terms mesh orthogonality affects the gradient of the face center f, it adds diffusion to the solution and it mainly affects the diffusive terms.

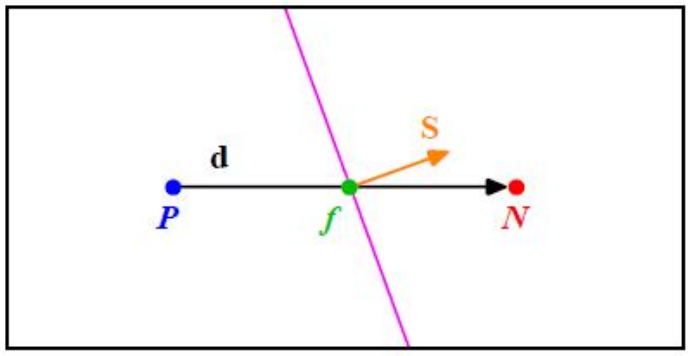


Figure 7.1: Mesh orthogonality [54].

In accordance with Figure 7.2, skewness is the deviation of the vector d that connects the two cells **P** and **N**, from the face center f. The deviation vector is represented with Δ and f_i is the point where the vector d intersects the face f. Skewness also adds diffusion to the solution, and besides it affects the interpolation of the cell centered quantities to the face center f and it influences the convective terms.

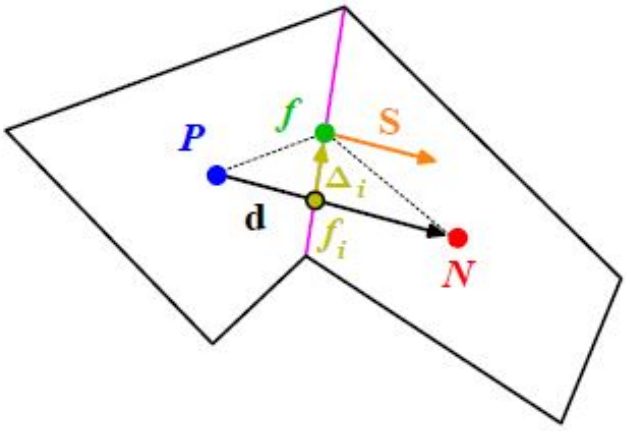


Figure 7.2: Mesh skewness [54].

As concerns the aspect ration AR, it is the ratio between the longest side Δx and the shortest side Δy referring to Figure 7.3. Large AR are suitable if gradients in the largest direction are small, while high AR smear gradients.

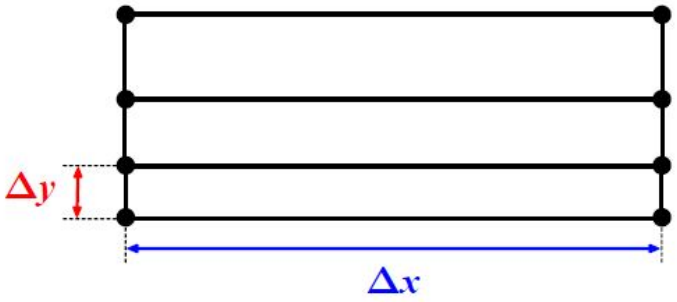


Figure 7.3: Mesh aspect ratio [54].

Another important aspect of the mesh is its refinement level near the wall depending on the turbulence near the wall and therefore on the *"Law of the wall"* (Figure 7.4). This law states that the average velocity of a turbulent flow at a certain point is proportional to the logarithm of the distance from that point to the "wall", or the boundary of the fluid region. The *"Law of the wall"* can be analytically expressed as:

$$u^+ = \frac{1}{k} \ln y^+ + C^+ \tag{7.0.1}$$

where u^+ is a dimensionless velocity and it represents the velocity u parallel to the wall as a function of y (distance from the wall), divided by the friction velocity u_τ.
y^+ is the wall coordinate expressed as the distance y to the wall, made dimensionless with the friction velocity u_τ and kinematic viscosity ν:

$$y^+ = \frac{y u_\tau}{\nu} \tag{7.0.2}$$

where the friction velocity is expressed as a function of the fluid density and the wall shear stress τ_w:

$$u_\tau = \left(\frac{\tau_w}{\rho}\right)^{1/2} \tag{7.0.3}$$

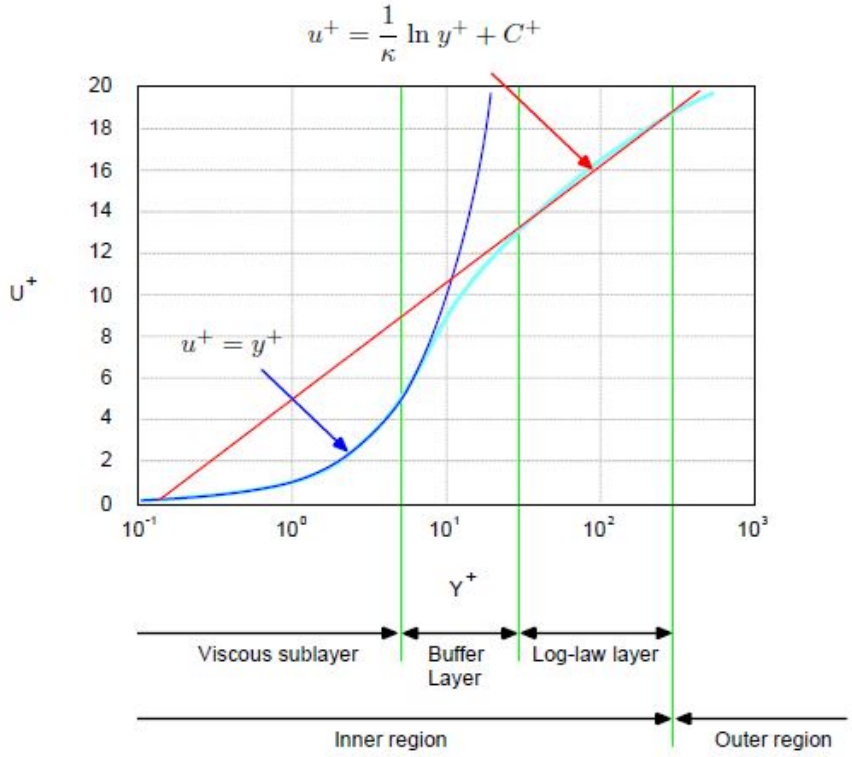


Figure 7.4: Law of the wall [54].

Different layers shown in 7.4 are better illustrated in Figure 7.5 .

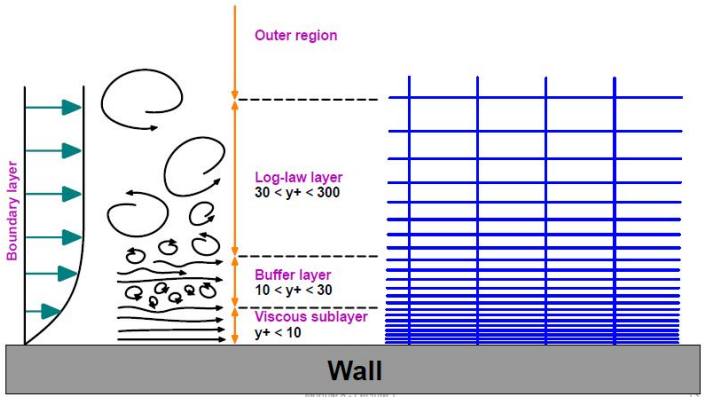


Figure 7.5: Wall layers [54].

In CFD it is usual to employ a different near-wall treatment according to the available range of y^+. In particular if:

- $y^+ \leq 6$ the boundary layer is resolved.
- $30 \leq y^+ \leq 300$ wall functions are used.

Resolving the viscous sublayer involves the full resolution of the boundary layer and therefore requires a high refinement level near the wall, while using wall functions

means that the boundary layer is modelled using a log-law wall function and a lower refinement level is enough. This is efficiently shown in Figure 7.6.

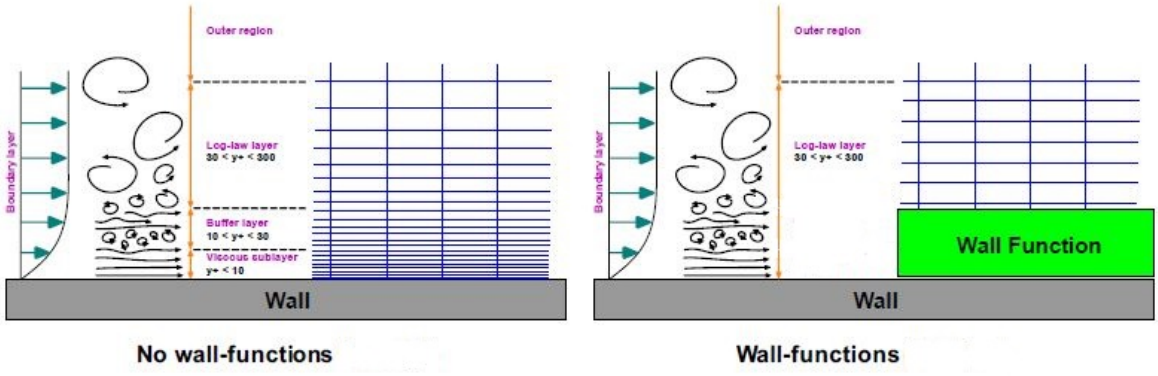


Figure 7.6: Different near-wall treatments [54].

As it will be seen in chapter 8, wall functions have been chosen for tested cases because in drying process internal aerodynamics are more relevant than the forces or the heat transfer on the walls.

As regards the mesh generation, OpenFoam provides the right tools: *blockMesh* and *snappyHexMesh* utilities have been used to generate the mesh. The *snappyHexMesh* utility generates 3-dimensional meshes containing hexahedra (hex) and split-hexahedra (split-hex) automatically from triangulated surface geometries, or tri-surfaces, in Stereolithography (STL) or Wavefront Object (OBJ) format. The mesh approximately conforms to the surface by iteratively refining a starting mesh and morphing the resulting split-hex mesh to the surface. The specification of mesh refinement level is very flexible and the surface handling is robust with a pre-specified final mesh quality [56]. In order to run *snappyHexMesh* a background hex mesh which defines the extent of the computational domain and a base level mesh density is required. This is achieved by using the *blockMesh* utility.
In Figure 7.7 it is possible to see the final step of the mesh generation. High refinement levels have been employed where more accuracy is required as shown in Figure 7.8. In fact the *Inlet* and the external and internal surface of the *Pipe* are the most sensitive and turbulent regions of the spray dryer.

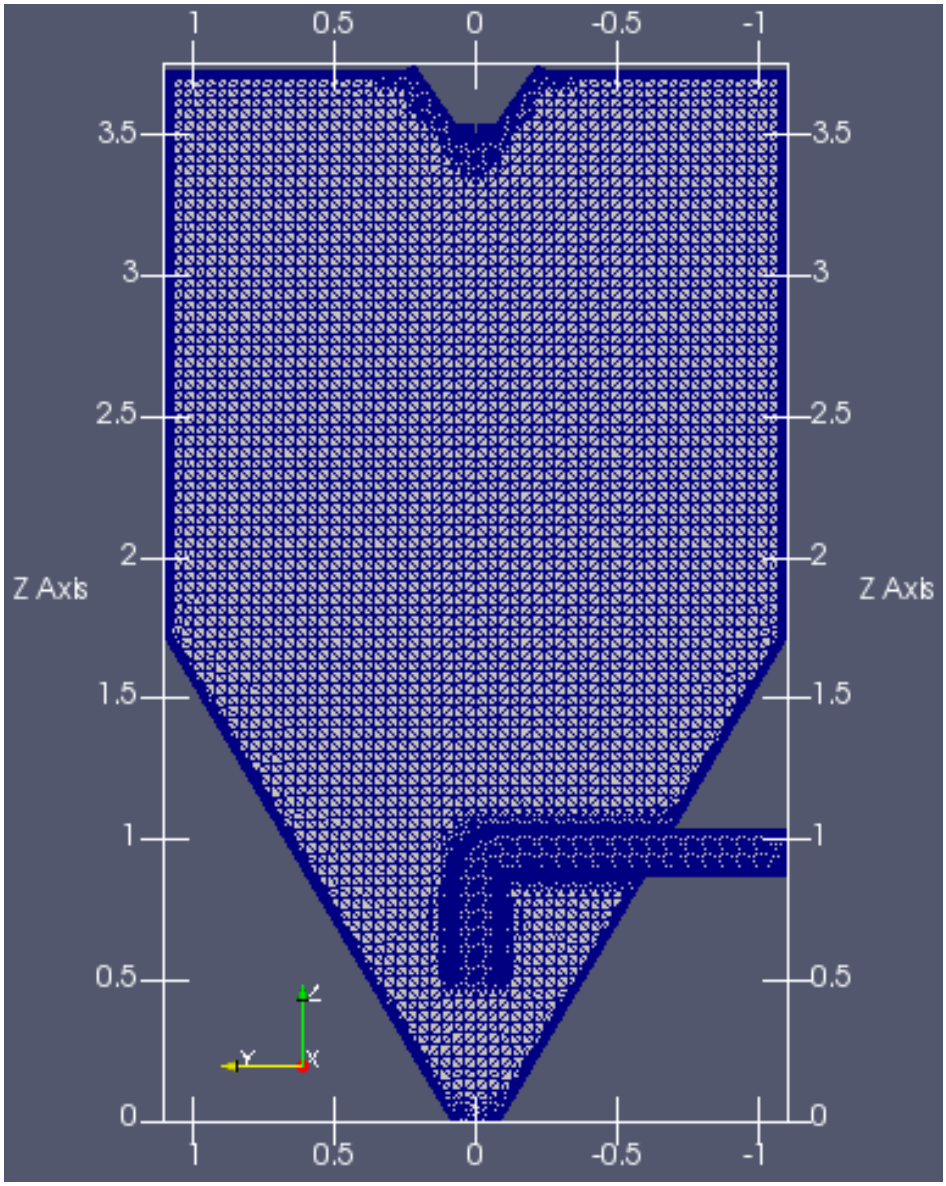
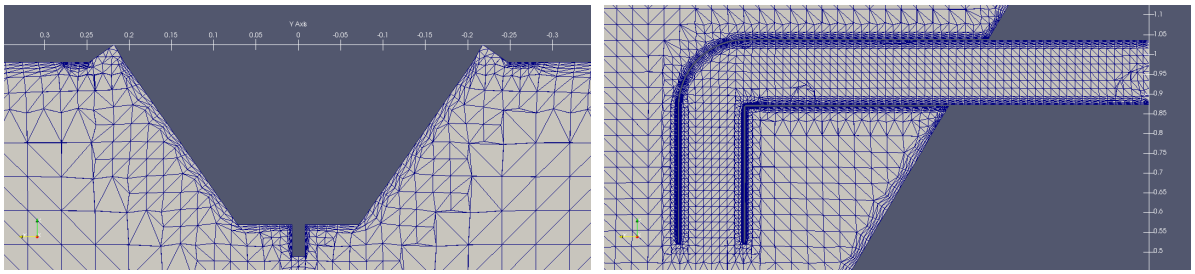


Figure 7.7: snappyHexMesh refinement.



(a) Inlet refinement (b) Pipe refinement

Figure 7.8: Refinements overview.

7.1 Different cases

As previously said in chapter 6.2, three different configurations of the *PipeDown* have been tested (see Figure 6.2.7), so likewise meshes have been realized. Figure 7.1.1 shows the pipe refinement detail of the three different meshes as the lenght of the *PipeDown* changes.

As for the specific details of the mesh, in table 7.1.1 and in table 7.1.2 are respectively illustrated the mesh statistics and the value of the mesh properties for the three different cases.

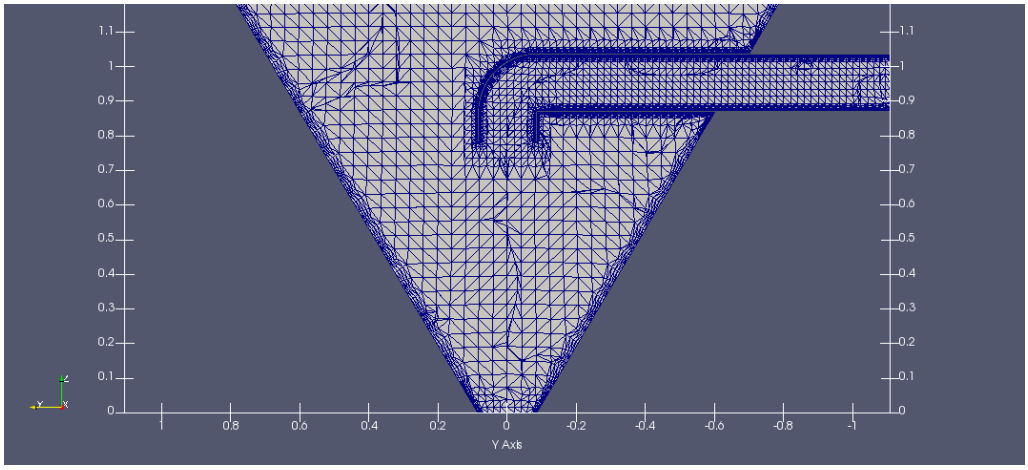
	case A)	case C)	case D)
points	296'161	318'957	314'361
faces	859'861	921'221	908'811
internal faces	828'191	886'058	874'331
total cells	281'975	301'260	297'354
hexaedra	268'543	287'219	283'506
prisms	2'946	2'858	2'834
pyramids	3'056	2'907	2'932
tetrahedra	3'148	2'990	3'016
polyhedra	4'282	5'286	5'066

Table 7.1.1: Mesh statistics.

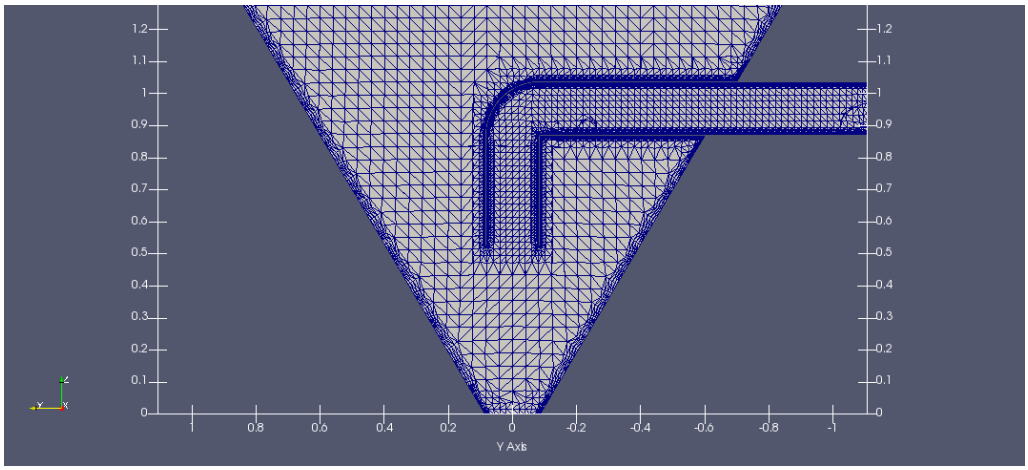
		case A)	case C)	case D)
non-Orthogonality	max	71.2438	71.3278	83.4707
	average	7.03507	6.98172	7.01623
max Aspect Ratio		13.3881	13.6445	34.5057
max Skewness		2.46379	2.48323	3.23615

Table 7.1.2: Mesh quality metrics.

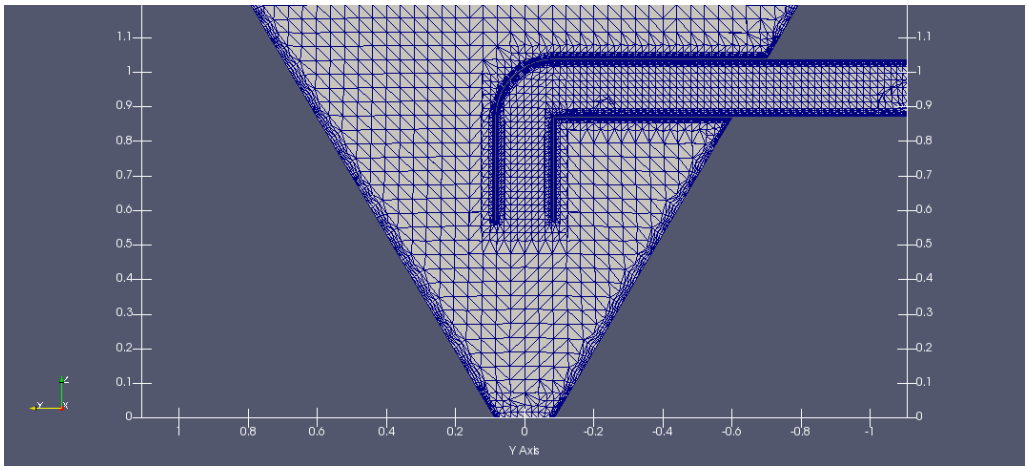
Returning to the earlier remarks about the *Law of the wall*, it is not possible to know a priori the y^+ value; in fact, preliminar simulations have been run for a few time-steps to get an estimate of it. The implementation of wall functions is then justified by y^+ values calculated for certain patches of each different simulation, as can be seen in table 7.1.3 (values reported in this table have been evaluated respectively for each latest time of simulation).



(a) case A)



(b) case C)



(c) case D)

Figure 7.1.1: Different mesh refinements varying the lenght of the *PipeDown*.

	case A)			case C)			case D)		
	min	max	average	min	max	average	min	max	average
AtomizerInletFood	65.474	103.365	82.014	98.541	109.704	103.192	73.112	103.540	81.096
Atomizer	0.875	98.451	43.292	7.916	85.407	41.517	1.415	96.430	39.629
Vessel	0.025	251.799	23.838	2.732	103.871	27.656	0.007	116.372	28.987
Pipe	0.737	299.722	82.951	2.629	277.487	60.570	0.132	259.172	63.238

Table 7.1.3: y^+ values for different cases.

8 Numerical schemes and boundary conditions

In this chapter numerical schemes and boundary conditions used for this thesis are presented. Firstly, a brief introduction to numerical schemes generally used in OpenFOAM are described in order to easily understand those chosen for this work as the case studies change; same goes for boundary conditions, already introduced in chapter 4.5.

Numerical methods are at the heart of the CFD process. Researchers dedicate their attention to two fundamental aspects in CFD: physical modeling and numerics. In physical modeling, we seek a set of equations or mathematical relations that allow to close the governing equations. On the other hand, the focus in numerics is to devise efficient, robust, and reliable algorithms for the solution of PDEs.

Numerical schemes have to be set for terms that are calculated during a simulation, such as derivatives in equations. In this regard, in OpenFOAM the schemes are specified in the *fvSchemes* dictionary in the *system* directory. The terms that must typically be assigned a numerical scheme in *fvSchemes* range from derivatives, e.g. gradient ∇, to interpolations of values from one set of points to another. The aim in OpenFOAM is to offer an unrestricted choice to the user, starting with the choice of discretisation practice which is generally standard Gaussian finite volume integration. Gaussian integration is based on summing values on cell faces, which must be interpolated from cell centres. The set of terms, for which numerical schemes must be specified, are subdivided within the *fvSchemes* dictionary into the categories below.

- ddtSchemes: schemes for first and second time derivatives , e.g. $\frac{\partial}{\partial t}$, $\frac{\partial^2}{\partial t^2}$
  - steadyState: sets time derivatives to zero.
  - Euler: transient, first order implicit, bounded.
  - backward: transient, second order implicit, potentially unbounded.
  - CrankNicolson: transient, second order implicit, bounded,
  - localEuler: pseudo transient for accelerating a solution to steady-state using local-time stepping; first order implicit.
- gradSchemes: schemes for gradient discretization, ∇
  - Gauss linear: the Gauss entry specifies the standard finite volume discretisation of Gaussian integration which requires the interpolation of values from cell centres to face centres. The interpolation scheme is then given by the linear entry, meaning linear interpolation or central differencing.
  - cellLimited Gauss linear: cellLimited scheme limits the gradient such that when cell values are extrapolated to faces using the calculated gradient, the face values do not fall outside the bounds of values in surrounding cells. A limiting coefficient is specified after the underlying scheme for which 1 guarantees boundedness and 0 applies no limiting.
- divSchemes: schemes for divergence discretization, $\nabla\cdot$
  - Gauss linear: second order, unbounded.

  - Gauss linearUpwind: second order, upwind-biased, unbounded (but much less so than linear), that requires discretisation of the velocity gradient to be specified.
  - Gauss limitedLinear: linear scheme that limits towards upwind in regions of rapidly changing gradient; requires a coefficient, where 1 is strongest limiting, tending towards linear as the coefficient tends to 0.
  - Gauss upwind: first-order bounded, generally too inaccurate to be recommended.

- snGradSchemes: schemes for gradient discretization in direction perpendicular to the boundary
  - corrected: orthogonality requires a regular mesh, typically aligned with the Cartesian co-ordinate system, which does not normally occur in meshes for real world, engineering geometries. Therefore, to maintain second-order accuracy, an explicit non-orthogonal correction can be added to the orthogonal component, known as the corrected scheme. The correction increases in size as the non-orthonality, the angle α between the cell-cell vector and face normal vector, increases.
  - limited corrected: as α tends towards 90°, e.g. beyond 70°, the explicit correction can be so large to cause a solution to go unstable. The solution can be stabilised by applying the limited scheme to the correction which requires a coefficient ψ, where ψ=0 corresponds to uncorrected and ψ=1 corresponds to corrected.

- laplacianSchemes: schemes for Laplace operator discretization, ∇^2
  - Gauss linear corrected: the Gauss scheme is the only choice of discretization and requires a selection of both an interpolation scheme for the diffusion coefficient and a surface normal gradient scheme.
  - Gauss linear limited corrected

- interpolationSchemes: schemes for interpolating from cell centers to cell faces
  - linear

- fluxRequired: quantities for whom the flux is computed

- wallDist: distance to wall calculation, where required.

OpenFOAM includes a vast number of discretisation schemes, those just listed are typically recommended for real-world, engineering applications.

Another important file is the *fvSolution* utility in which are stored parameters for solving system of linear equations and parameters for numerical method. The most important parameters contained in *fvSolution* are listed below arrenged for sections:

- solvers: this section defines which linear system solvers are used to compute quantities (partial differential equations are transformed to system of linear equations)

  - solver: it defines which linear solver is used choosing between GAMG (generalised geometric-algebraic multi-grid), smoothSolver (solver that uses a smoother), PCG/PBiCG (reconditioned (bi-)conjugate gradient, with PCG for symmetric matrices, PBiCG for asymmetric matrices) and diagonal (diagonal solver for explicit systems)
  - preconditioner: it defines preconditioner of linear system solver
  - relTol: it defines relative tolerance of linear system solver (order of residual decrease for each iteration)
  - tolerance: it defines absolute tolerance of linear system solver (initial residual is checked)

- relaxationFactors: relaxation factors for individual quantities are defined here
- PISO, SIMPLE or PIMPLE: in this section algorithm and convergence criteria are defined
  - nNonOrthogonalCorrectors: it defines the number of non-orthogonal correctors in order to have a suitable convergence on non-orthogonal meshes (0 for orthogonal mesh and 20 for strongly non-orthogonal mesh)

To finally reach the setup simulation involving air flow and particles together, different cases have been tested starting from a simplified configuration regarding the only flow evolution, introducing ever more parameters so as to efficiently approach to the real case.

For reasons of simplification, single patches presented in sub-chapter 6.2, have been regrouped for each case as follows; it is also indicated the corresponding base type of boundaries (see paragraph 4.5.1) for every patch.

- AtomizerInletAir - type inlet
- AtomizerInletFood - type wall
- Atomizer: it includes *AtomizerCone*, *AtomizerInletAir*, *AtomizerBot* and *AtomizerPipe* - type wall
- Vessel: it includes *WallTop*, *Wall* and *WallCone* - type wall
- Pipe: it includes *OutputPipe length*, *PipeTurn*, and *PipeDown* - type wall
- OutletPipe - type outlet
- OutletCone - type outlet

As regards the turbulence model implemented, after a preliminary comparison with the *kEpsilon* model, the *kOmegaSST* has been chosen as the model used for all different configurations tested in this thesis because it allows to combine the advantages of both *kEpsilon* and *kOmega* models. For more informations about turbulence models, see chapter 3.7.

8.1 Case with only the flow

Once the suitable setup geometry has been found (see sub-chapter 6.2), the first set of simulations has been run. In this case the only air flow (incompressible flow) has been investigated in order to search a fully developed solution, or rather a steady or statistically steady solution which can be used as basis for following simulations.

OpenFOAM does not have a generic solver applicable to all cases, in fact users must choose a specific solver for a class of problems to solve. The solver *pimpleFoam* has been chosen as the most suitable for this case; it is a large time-step transient solver for incompressible flow, and its name reflects the algorithm used, namely PIMPLE algorithm, which is an hybrid between SIMPLE and PISO algorithms [48]. PIMPLE allows transient solutions of higher *Courant numbers*, it iterates and under-relaxes the solution of pressure-velocity coupling within a time step and PIMPLE-loop ends by reaching either a static number of iterations or a residual limit ϵ.

Let's give a brief definition of the *Courant number*: in mathematics, the Courant-Friedrichs-Lewy (CFL) condition is a necessary condition for convergence while solving certain partial differential equations numerically by the method of finite differences. It arises in the numerical analysis of explicit time integration schemes, when these are used for the numerical solution. As a consequence, the time step must be less than a certain time in many explicit time-marching computer simulations, otherwise the simulation will produce incorrect results. The principle behind the condition is that, for example, if a wave is moving across a discrete spatial grid and its amplitude at discrete time steps of equal duration must be computed, then this duration must be less than the time for the wave to travel to adjacent grid points. As a corollary, when the grid point separation is reduced, the upper limit for the time step also decreases. That is why the *Courant number* is introduced; for a general *n-dimensional* case:

$$C = \triangle t \sum_{i=1}^{n} \frac{u_{xi}}{\triangle x_i} \leq C_{max} \tag{8.1.1}$$

where $\triangle t$ is the time step and $\triangle x_i$ is the spatial variable. The value of C_{max} changes with the method used to solve the discretised equation, especially depending on whether the method is explicit or implicit. If an explicit (time-marching) solver is used then typically C_{max}=1. Implicit (matrix) solvers are usually less sensitive to numerical instability and so larger values of C_{max} may be tolerated.

URANS model (see sub-chap. 3.9) allows to use a higher *Courant numbers*, because with this model it is possible to neglect small-scales turbulence and, therefore, it is possible to use an high value of the time step (eq 8.1.1). In fact the value used in this case has been $C_{max} = 2$, so as to have a faster simulation without losing accuracy.

In this configuration, the calculated flow physical variables are the turbulent kinetic energy k, the specific dissipation ω, the vorticity z, the pressure p, the velocity magnitude U and its components U_x U_y U_z and the turbulent viscosity ν_T (that is not a physical property). The temperature and therefore the heat transfer are neglected in this case.

The boundary conditions and the initial conditions for these quantities are shown in table 8.1.1.

	walls	inlet	outlet
ρ-normalized pressure - $\{$p*$\}$	zeroGradient	zeroGradient	_ zeroGradient for OutletCone _ fixedValue for OutletPipe: uniform -100 $[\frac{m^2}{s^2}]$
velocity - $\{\vec{U}\}$	fixedValue uniform (0 0 0) $[\frac{m}{s}]$	flowRateInletVelocity volumetricFlowRate 0.445 $[\frac{m^3}{s}]$	_ zeroGradient for OutletPipe _ fixedValue for OutletCone: uniform (0 0 0) $[\frac{m}{s}]$
vorticity - $\{\vec{z}\}$	calculated initial value uniform (0 0 0) $[\frac{1}{s}]$	calculated initial value uniform (0 0 0) $[\frac{1}{s}]$	calculated initial value uniform (0 0 0) $[\frac{1}{s}]$
turbulent kinetic energy - $\{$k$\}$	kqRWallFunction initial value: uniform 0.027 $[\frac{m^2}{s^2}]$	fixedValue initial value: uniform 0.027 $[\frac{m^2}{s^2}]$	zeroGradient
specific dissipation - $\{\omega\}$	omegaWallFunction initial value: uniform 4.28 $[\frac{1}{s}]$	fixedValue initial value: uniform 4.28 $[\frac{1}{s}]$	zeroGradient
turbulent viscosity - $\{\nu_T\}$	nutkWallFunction initial value: uniform 0 $[\frac{m^2}{s}]$	calculated initial value: uniform 0 $[\frac{m^2}{s}]$	calculated initial value: uniform 0 $[\frac{m^2}{s}]$

Table 8.1.1: Initial and boundary conditions - flow only

The primitive types boundary conditions such as *zeroGradient*, *fixedValue* and *corrected* are already described in paragraph 4.5.2 . The last two conditions are very intuitive while for the first one is reminded that it extrapolates the quantity to the patch from the nearest cell value; the meaning is, the quantity is developed in space and its gradient is equal to zero in direction perpendicular to the patch (perpendicular to the boundary). Translated into matematical expression, for example for the ρ-normalized pressure, it results as follows:

$$\frac{\partial p*}{\partial n} = 0 \tag{8.1.2}$$

As concerns k, ω and ν_T, in 8.1.1 there are also reported *kqRWallFunction*, *omegaWallFunction* and *nutkWallFunction* that are respectively the wall functions for each variable regarding the SST k-ω turbulent model. These wall functions impose a wall-value to each variable; for example the value of the specific dissipation ω becomes:

$$\omega_{wall} = 10\frac{6\nu}{\beta y^2} \tag{8.1.3}$$

where $\beta = 0.075$, ν is the kinematic viscosity and y is the distance to the first cell center normal to the wall.

Regarding the numerical schemes, part of the *fvSchemes* utility is reported in A. Same goes to the *fvSolution* utility reported in B.

8.2 Implementation of the temperature

In this chapter the examination of the case of the air flow with the implementation of the temperature is presented. Temperature is one of the most important variable in spray drying process, it affects the resultant particles in a relevant way, particularly in relation to the moisture content. The idea has been to run this case first without heat transfer and then putting in practice it by varying therefore the boundary conditions.

The implementation of the temperature led to change the solver used; the choice fell on *buoyantPimpleFoam*, a transient solver for buoyant, turbulent flow of compressible fluids for ventilation and heat-transfer, that uses PIMPLE algorithm [48].

In the two cases represented below, a $C_{max} = 2$ is also imposed.

8.2.1 Flow without heat transfer

In this configuration, the physical quantities that have been calculated are the temperature T, the turbulent diffusity α_T, the turbulent kinetic energy k, the specific dissipation ω, the turbulent viscosity ν_T, the vorticity z, the pressure p, the velocity magnitude U and its components U_x U_y U_z.

Initial and boundary conditions for these parameters are reported in table 8.2.1. Please note that the boundary condition for temperature without heat transfer should be *zeroGradient* but in this case a *wallHeatTransfer* condition has been imposed in order to gain confidence with the heat transfer phenomenon. The *wallHeatTransfer* condition provides an enthalpy condition for wall heat transfer and it requires as input the wall temperature T_{inf} and the thermal diffusivity α_{wall}. This latter parameter has been imposed as very low value so as to neglect the heat transfer (see table 8.2.1) and so, walls can be considered as adiabatic surfaces.

Another new boundary condition that has been set is *fixedFluxPressure* for dynamic pressure: this boundary condition adjusts the pressure gradient such that the flux on the boundary is that specified by the velocity boundary condition. The predicted flux to be compensated by the pressure gradient is evaluated as $(\phi - \phi_{H/A})$, both of which are looked-up from the database, as is the pressure diffusivity used to calculate the gradient using:

$$\nabla(p) = \frac{\phi_{H/A} - \phi}{|Sf|\, D_p} \tag{8.2.1}$$

in which ϕ is the flux, D_p is the pressure diffusity and Sf is the patch face area $[m^2]$.

The values of the air inlet temperature ($T = 468K$) and of the air temperature outside walls ($T = 300K$) have been extrapolated from literature as shown in table 6.1 .

Numerical schemes concerning this case are reported in appendices C and D.

	walls	inlet	outlet
temperature - $\{$T$\}$	_ zeroGradient for the Pipe _ wallHeatTransfer for other walls initial value: uniform 300 $[K]$ $T_{inf} = 300$[K] α_{wall}=0.001 $[\frac{W}{m^2}]$	fixedValue value: uniform 468 $[K]$	_ zeroGradient for the OutletPipe _ wallHeatTransfer for OutletCone initial value: uniform 300 [K] $T_{inf} = 300[K]$ α_{wall}=0.001 $[\frac{W}{m^2}]$
thermal diffusity - $\{\alpha_T\}$	alphatWallFunction initial value: uniform 0 $[\frac{kg}{ms}]$ Pr_T=0.85	calculated initial value: uniform 0 $[\frac{kg}{ms}]$	zeroGradient
pressure - $\{$p$\}$	calculated initial value: uniform 101325 $[\frac{kg}{ms^2}]$	calculated initial value: uniform 101325 $[\frac{kg}{ms^2}]$	zeroGradient
dynamic pressure - $\{\ p_{din}\}$	fixedFluxPressure initial value: uniform 101325 $[\frac{kg}{ms^2}]$	zeroGradient	_ fixedFluxPressure for OutletCone initial value: uniform 101325 $[Pa]$ _ fixedValue for OutletPipe uniform 101225 $[Pa]$
velocity - $\{\vec{U}\}$	fixedValue uniform (0 0 0) $[\frac{m}{s}]$	flowRateInletVelocity volumetricFlowRate 0.445 $[\frac{m^3}{s}]$	_ zeroGradient for OutletPipe _ fixedValue for OutletCone value: uniform (0 0 0) $[\frac{m}{s}]$
vorticity - $\{\vec{z}\}$	calculated initial value uniform (0 0 0) $[\frac{1}{s}]$	calculated initial value uniform (0 0 0) $[\frac{1}{s}]$	calculated initial value uniform (0 0 0) $[\frac{1}{s}]$
turbulent kinetic energy - $\{$k$\}$	kqRWallFunction initial value: uniform 0.027 $[\frac{m^2}{s^2}]$	fixedValue initial value: uniform 0.027 $[\frac{m^2}{s^2}]$	zeroGradient
specific dissipation - $\{\omega\}$	omegaWallFunction initial value: uniform 4.28 $[\frac{1}{s}]$	fixedValue initial value: uniform 4.28 $[\frac{1}{s}]$	zeroGradient
turbulent viscosity - $\{\mu_T\}$	mutkWallFunction initial value: uniform 0 $[\frac{kg}{ms}]$	calculated initial value: uniform 0 $[\frac{kg}{ms}]$	calculated initial value: uniform 0 $[\frac{kg}{ms}]$

Table 8.2.1: Initial and boundary conditions - temperature without heat transfer

8.2.2 Flow with heat transfer

Implementing the heat transfer, initial and boundary conditions are the same reported in table 8.2.1 for the previous case, except for the temperature of the *Vessel* wall (see list in chap.8 pag. 48, and Figure 6.2.2). For this patch the *externalWallHeatFluxTemperature* condition has been imposed; this BC supplies a heat flux condition for temperature on an external wall. The idea is to calculate the heat flux q $[W/m^2]$ by fixing the heat transfer coefficient h $[W/m^2/K]$ and the ambient temperature T_a $[K]$ as shown in eq. 8.2.2.

$$q = h(T_a - T_w) \tag{8.2.2}$$

where T_w is the temperature on the wall, while the heat transfer coefficient and the ambient temperature have been extrapolated from literature: (see also table 6.1)

$h = 3.5$ $[W/m^2/K]$
$T_a = 300$ $[K]$

As regards the numerical schemes, the *fvSchemes* and the *fvSolution* utilities are the same of the case without the heat transfer, respectively shown in appendices C and D.

8.3 Implementation of the particles

Finally, the last implementation step is presented. Particles are injected and computational calculations start from the average field of a developed flow (calculated from a previous simulation 8.2.2). In this case air flow and particles evolve together in the spray dryer, and that is the reason why the choice fell on the *reactingParcelFoam* solver that is a transient PIMPLE solver for compressible, laminar or turbulent flow with reacting multiphase Lagrangian parcels [48].

This kind of solver uses the thermophysical model library by reading the *thermophysicalProperties* dictionary. Thermophysical models are concerned with energy, heat and physical properties and they are constructed in OpenFOAM as a pressure-temperature (p-T) system from which other properties are computed. There is one compulsory dictionary entry called *thermoType* which specifies the package of thermophysical modelling that is used in the simulation. OpenFOAM includes a large set of pre-compiled combinations of modelling, built within the code using C++ templates. This coding approach assembles thermophysical modelling packages beginning with the equation of state and then adding more layers of thermophysical modelling that derive properties from the previous layer(s) [55]. The *thermoType* used in this thesis is described in the following list:

- heRhoThermo: it is a thermophysical model for reacting mixture, based on ρ.
- reactingMixture: it specifies the mixture composition. It is used for mixtures with variable composition and so, it requires the thermophysical models coefficients to be specified for each specie within sub-dictionaries named after each specie.
- polynomial: it is a transport model evaluating dynamic viscosity μ, thermal conductivity k and thermal diffusivity α. Polynomial model calculates μ and k

as a function of temperature T from a polynomial of any order N, e.g.:

$$\mu = \sum_{i=0}^{N-1} a_i T^i \tag{8.3.1}$$

where a_i are polynomial coefficients.

- hPolynomial: it is a thermodynamic model evaluating the specific heat C_p from which other properties are derived. hPolynomial model calculates C_p as a function of temperature by a polynomial of any order N, e.g.:

$$C_p = \sum_{i=0}^{N-1} a_i T^i \tag{8.3.2}$$

- sensibleEnthalpy: it selects the energy variable (enthalpy in this case). The word sensible means that in the (sensible) energy heat of formation is not included.

- icoPolynomial: it is an incompressible, polynomial equation of state, e.g.:

$$\rho = \sum_{i=0}^{N-1} a_i T^i \tag{8.3.3}$$

- specie: the basic thermophysical properties are specified for each species from input data. Data entries must contain the name of the specie as the keyword, e.g. *O2* or *H2O* mixture, followed by sub-dictionaries of coefficients, including specie: it contains, i.e., number of moles of the specie and molecular weight in units of g/mol.

As concerns the interactions such as the collisions between the particles in the dispersed phase, they are not taken into account in any cases tested in this thesis. Moreover, the coupling between the continuum phase and the dispersed phase (see equations in chapter 3.10) can be treated differently depending on the nature of the flow and on the nature of the particles. The effect of the dispersed phase on the continuum phase has been assumed not significant according to a preliminary study on the *Stokes number* that will be briefly presented.
Before that, let's summarise what has been stated: a one-way coupling has been considered sufficient because particle-particle interactions and the effect of the dispersed phase on the fluid flow are negligible. In this case, no additional body forces $\bar{f}_i$ appear in the momentum equation of the continuum phase (eq. 3.9.1), so the term $\bar{f}_i$ is imposed equal to 0 as is shown in eq. 3.9.2.
The *Stokes number* is a dimensionless number characterising the behavior of particles suspended in a fluid flow and it is defined as the ratio of the characteristic time of a particle (or droplet) named *response time* τ_p, to a characteristic time scale of the surrounding flow τ_a:

$$St = \frac{\tau_p}{\tau_a} \tag{8.3.4}$$

The particle *response time* τ_p is a crucial parameter in evaluating the particles trajectories that depends on the particle mass, the fluid viscosity and the particle Reynolds number as can be seen in eq. 8.3.5:

$$\tau_p = \frac{\rho_p d_p^2}{18\mu_a(1 + 0.15Re_p^{2/3})} \tag{8.3.5}$$

where ρ_p is the particle density, μ_a is the fluid dynamic viscosity and the particle Reynolds number is defined as:

$$Re_p = \frac{d_p|\boldsymbol{v_p} - \boldsymbol{v_a}|}{\nu_a} \tag{8.3.6}$$

in which d_p is the particle diameter, ν_a is the fluid cinematic viscosity and $|\boldsymbol{v_p} - \boldsymbol{v_a}|$ is the magnitude of relative particle-to-air velocity. Instead, the characteristic time scale of the surrounding flow τ_a is defined as follows:

$$\tau_a = \left(\frac{\nu_a}{\epsilon}\right)^{1/2} \tag{8.3.7}$$

where ϵ is the turbulent dissipation.
A particle with a low *Stokes number* follows fluid streamlines (perfect advection), while a particle with a large *Stokes number* is dominated by its inertia and it continues along its initial trajectory.
The *Stokes number* has been evaluated on three different sections of the spray dryer: cross section *Near Outlet* (Figure 6.3.1 - case a), cross section *Near Inlet* (Figure 6.3.1 - case b) and the *Inlet Pipe* (Inlet of *PipeDown* shown in Figure 6.2.7 - case C).
An approximate, but reliable evaluation of this parameter has been possible on the basis of numerical results obtained from previous simulations (8.2.2) and thanks to experimental measurements regarding dried particles (olive pomace extract mixed to maltodextrin) provided by the Material Engineering Laboratory of DICCA - University of Genoa. These measurements have been carried out using a **Mini Spray Dryer B-290**.
Variables have been considered in order to have a suitable range of the *Stokes number* for each sections, estimating the minimum and the maximum value as reported in table below:

	Near Inlet	Near Outlet	InletPipe
St_{min}	0.01	5.98e-06	8.86e-04
St_{max}	4.34	2.35	21.52

Table 8.3.1: Evaluation of Stokes number.

As can be seen from table 8.3.1, the range of the *Stokes number* is restrained and very close to 1 meaning that it is possible to neglect the effect of the dispersed phase on the continuum phase.

Concerning the *Courant number*, the C_{max} is imposed equal to 2, as was the case with the only flow (sub-chap.8.1).

An important issue in the particles implementation is clearly represented by the injection model. The injection model can deeply influence the drying process, in fact injection settings have been entirely extrapolated from the literature (see table 6.1) in order to have comparable results. The model chose is, therefore, the *ConeInjection-Model* so as to use all the available informations; in fact the *ConeInjectionModel* is a multi-point injection model in which users specifies the time of start of injection, the list of injector positions and directions (along injection axes), parcel velocities, inner and outer spray cone angles, parcel diameters obtained by distribution model (Rosin Rammler, see table 6.1) and number of parcel to inject per injector (one in this case, see Figure 6.2.4 - case a and b). The latter information is the only variable. Please, note that parcel means a group of particles, but in this thesis they are imposed as the same.

Once the fundamental aspects such as the *ThermoType* equatioins, the coupling between the two phases, and the injection model was imposed, different cases have been tested by varying the dispersed phase (liquid or solid), the number of injected particle per second and the wall interaction model for *Vessel* and *Pipe* patches (see list in chap. 8 - pag. 48).
In particular, treated cases are illustrated in table 8.3.2 :

	dispersed phase	wall interaction model	number of injected particles per second
case 1	liquid	rebound	2'000
case 2	liquid	stick	2'000
case 3	liquid	rebound	20'000
case 4	liquid	stick	20'000
case 5	solid	rebound	2'000
case 6	solid	stick	2'000

Table 8.3.2: Different treated cases for particles implementation.

where:

- stick: it assigns zero velocity to the particles that impact on the wall.
- rebound: it specifies elasticity and restitution coefficients.

and the dispersed phases that have been tested are water droplets (liquid) and olive pomace extract mixed to maltodextrin particles (solid). As concerns the initial and boundary conditions of the physical quantity calculated, it is possible to consult the previous case about the implementation of the temperature with heat transfer (8.2.2) because conditions imposed are the same. In this case, the thermophysical model needs more informations about species at stake because of the polynomial equations used (see appendix E).

Numerical Schemes are illustrated in appendices F and G.

9 Results

The main results of this work are now presented. First, the convergence criteria are explained showing some numerical examples and then, the simulation results are illustrated for each case (as outlined in chap. 8) as well as a comparison between the different cases.

9.1 Convergence study

In CFD analysis, the convergence of the iterative solution is defined by looking at *Residual* values because they directly quantifies the error in the solution of the system of equations. The residual measures the local error of a conserved variable in each control volume. Therefore, every cell in the model will have its own residual value for each of the equations being solved. In an iterative numerical solution, the residual will never be exactly zero. However, the lower the residual value is, the more numerically accurate the solution. In particular, residual reference values of best practice have been set in different cases studied in this thesis, as can be seen in table 9.1.1. When the solver reaches these values, it stops iterations and it moves to the next instant.

	pressure	velocity	turbulent kinetic energy	turbulent specific dissipation
residual value	1e-06	1e-08	1e-08	1e-08

Table 9.1.1: Residual values for different physical quantities.

An example of the evolution of the residuals is reported in Figure 9.1.1. This figure, referring to the *case C* (see Figure 6.2.7), reports the initial values of the residuals in order to have a precautionary approach because initial values are also higher values.

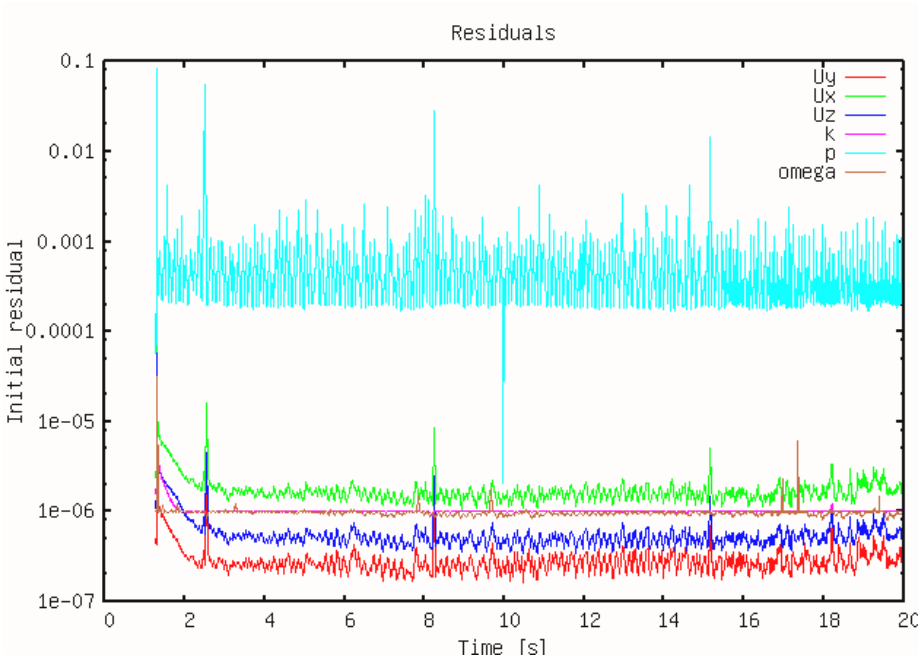
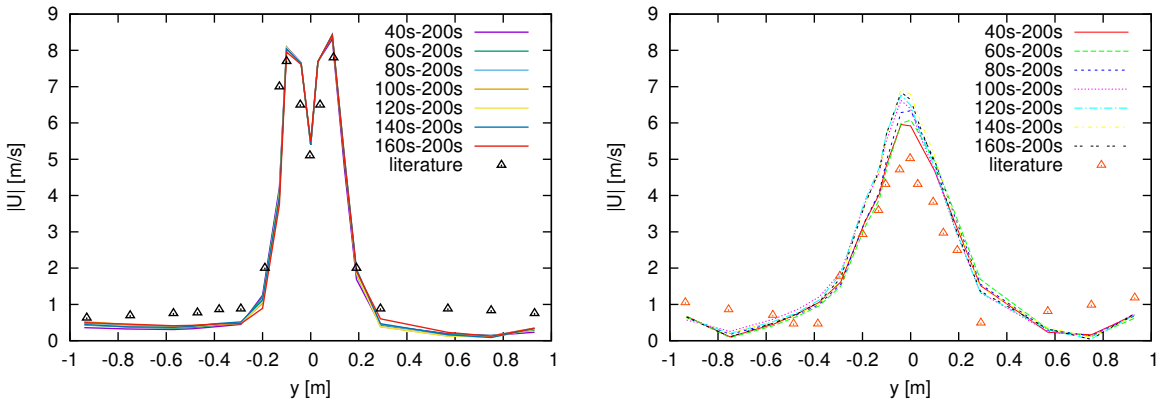


Figure 9.1.1: Residual values for case C.

Moreover, simulations have been found to be time-dependent having no periodic state response. That is the reason why analyses are performed over a statistical steady-state regime, reached after an initial transient stage. The statistical steady-state regime has been confirmed by the convergence analysis shown in Figure 9.1.2; the magnitude velocity plot has been calculated as the time window changes until a suitable overlap has been found between different profiles. This kind of analysis has been done for velocity magnitude profiles calculated in two different sections (Near Outlet and Near Inlet as can be seen in Figure 6.3.1) in order to have more accurate results.



(a) Near Inlet

(b) Near Outlet

Figure 9.1.2: Velocity magnitude on different time windows.

As can be seen in Figure 9.1.2, the statistical steady-state regime for the *case C* has been reached after 200 seconds, while for other cases (see Figure 6.2.7) convergence has been reached earlier: 20 seconds for *case A* and 30 seconds for *case D*.

As stated above, no periodic state response has been found. To verify that, a spectral analysis has been implemented. As it can be seen in Figure 9.1.3, referring to the *case C*, no dominant frequency has been found.

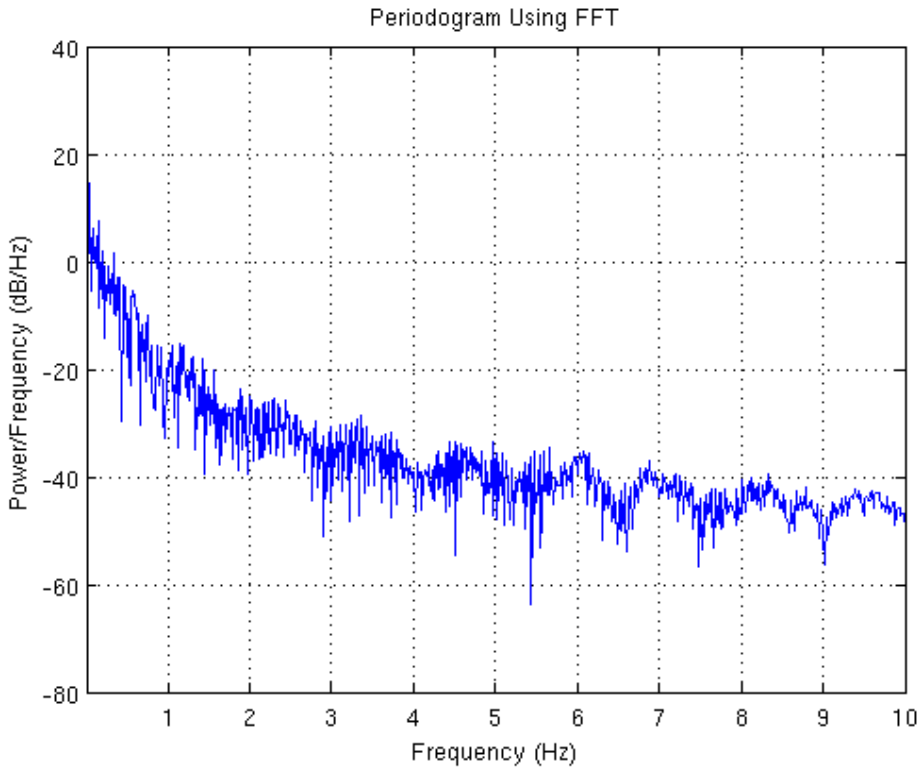
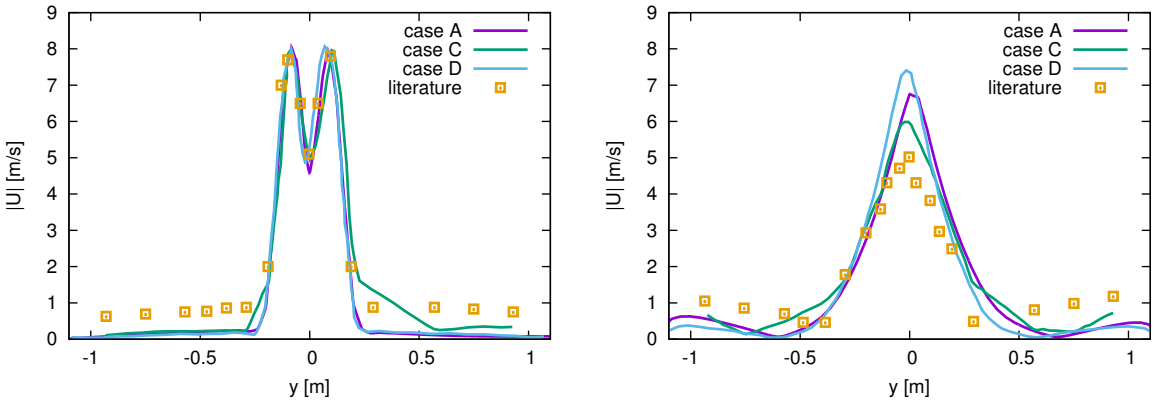


Figure 9.1.3: Spectral analysis for case C.

9.2 Varying the geometry

Different simulations have been run varying the geometry as reported in chapter 6. In particular different configurations of the air inlet and of the vertical part of the pipe (see Figure 6.2.7) have been studied because of the absence of reliable data in the literature.
As concerns the angle of the air inlet, results are not presented because a variation of this parameter does not affect the results.
On the contrary, interesting results emerge varying the length of the *PipeDown* (see Figure 6.2.6). In Figure 9.2.1 the magnitude velocity profile is reported for each case; each plot has been calculated for the respective time convergence.
As regards the inlet (a), no relevant differences are noted, while for the outlet (b) the *case C* was found to be the configuration that better approaches to the literature data. That is the reason why *case C* has been chosen as basis for the cases in which temperature variations and particles are accounted for.



(a) Near Inlet (b) Near Outlet

Figure 9.2.1: Velocity magnitude profiles for different cases.

A detailed analysis of *case C* is found in Figure 9.2.2, where different colour plots have been taken into account for each variable of the problem in order to better understand the internal aerodynamics and thermodynamics. As shown in Figure 9.2.2, the colour plots have been evaluated in a *y-z plane* sectioning the spray dryer.

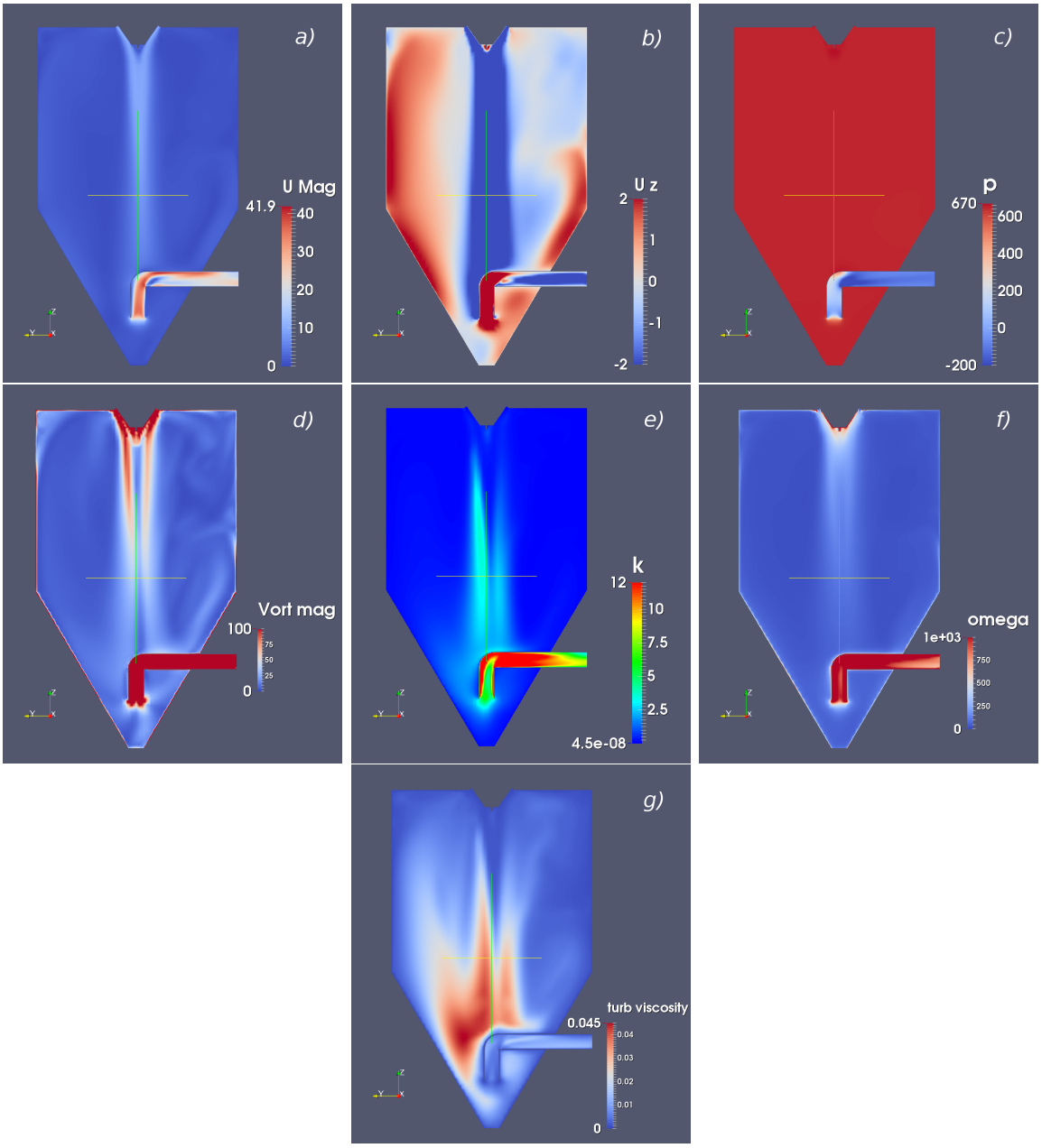
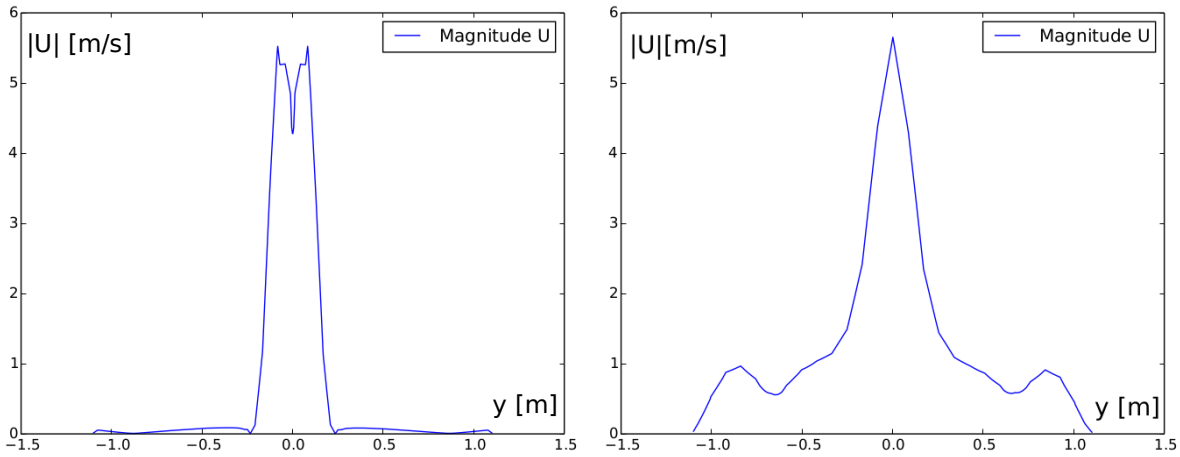


Figure 9.2.2: Overview of the results of each calculated variable for case C. a) the magnitude of the velocity U - b) z velocity component U_z ; it is the most relevant one - c) pressure p - d) the magnitude of the vorticity Z - e) the turbulent kinetic energy k - f) the specific turbulent dissipation ω - g) the turbulent viscosity ν_T

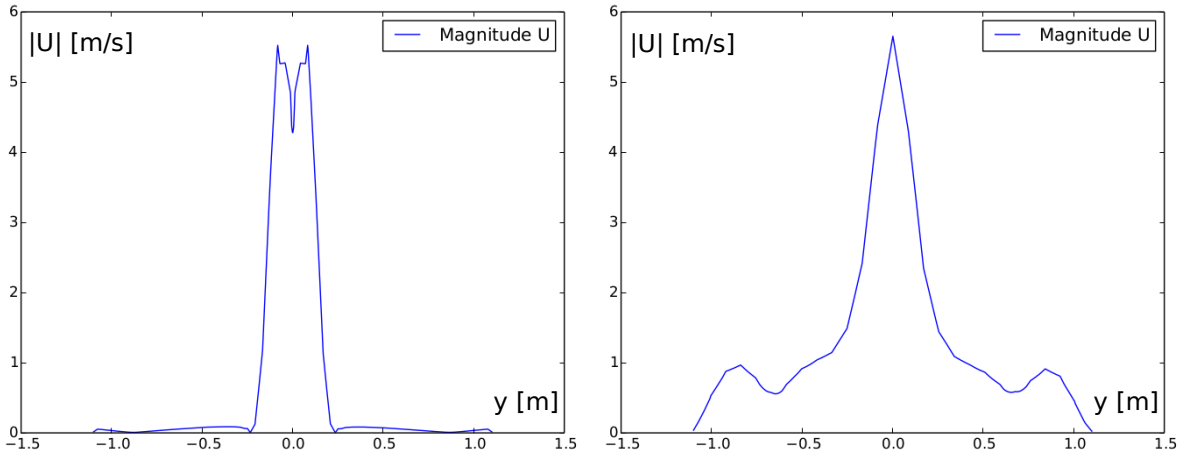
9.3 Turbulence models

At the beginning of this work, a preliminar study on turbulence models has been done in order to appreciate the most performing one. The k-ϵ model used in the literature [3], [7], [24], has been compared to the SSTk-ω model on equal terms (same geometry and same mesh). From Figures 9.3.1 and 9.3.2 it is possible to deduce that the two turbulent models return the same values for both the inlet and the outlet sections. Finally, the choice fell on the SSTk-ω because it puts together the advantages of the k-ω and k-ϵ models (see chapter 3.7).



(a) Near Inlet (b) Near Outlet

Figure 9.3.1: Velocity magnitude profiles calculated with SSTk-ω model.

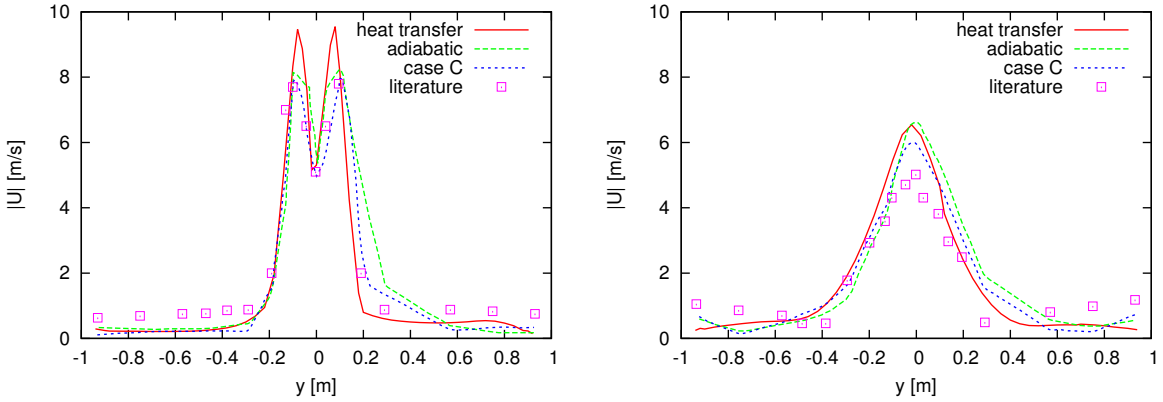


(a) Near Inlet (b) Near Outlet

Figure 9.3.2: Velocity magnitude profiles calculated with k-ϵ model.

9.4 Implementation of the temperature

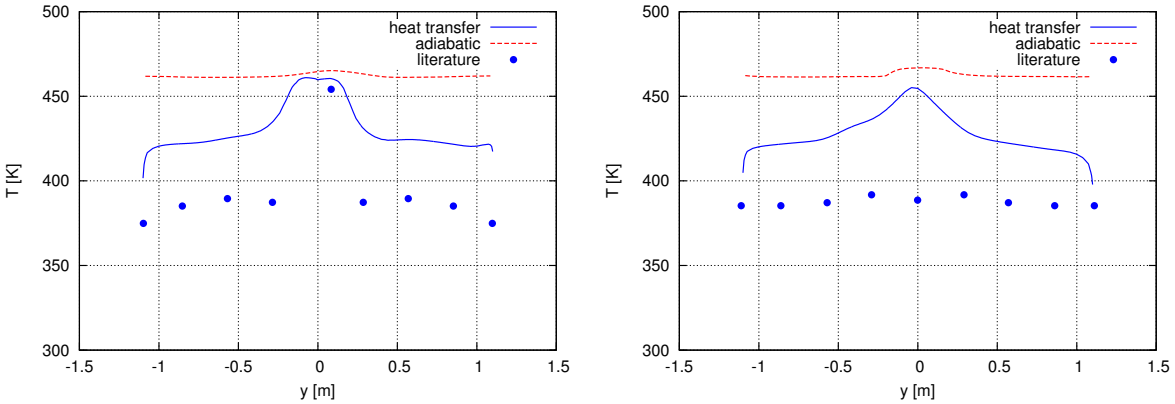
In this chapter results of the temperature implementation are presented. The first step has been to add only the temperature variable to the developed solution of the *Case C*, considering therefore an adiabatic case. Then, also the heat transfer on the external walls has been implemented. The last cases are compared to the *case C* (only the flow) and to the literature data as shown in Figure 9.4.1.



(a) Near Inlet (b) Near Outlet

Figure 9.4.1: Velocity magnitude profiles for different cases.

Moreover, Figure 9.4.2 shows the effect of the heat transfer on the temperature profile evaluated near inlet and near outlet.



(a) Near Inlet (b) Near Outlet

Figure 9.4.2: Temperature profiles for different cases.

In addition to the velocity and temperature fields, also a spectral analysis has been conducted. Neither of the two temperature simulations have a dominant frequency (Figures 9.4.3 and 9.4.4) in accordance to what was verified for the *case C*.

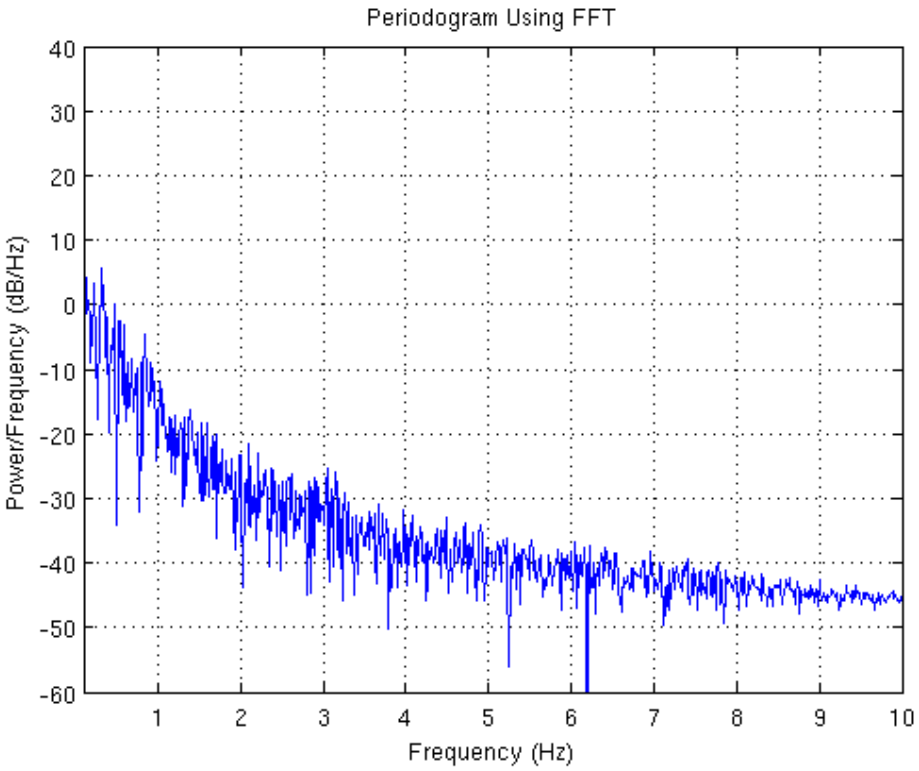


Figure 9.4.3: Spectral analysis for the adiabatic case.

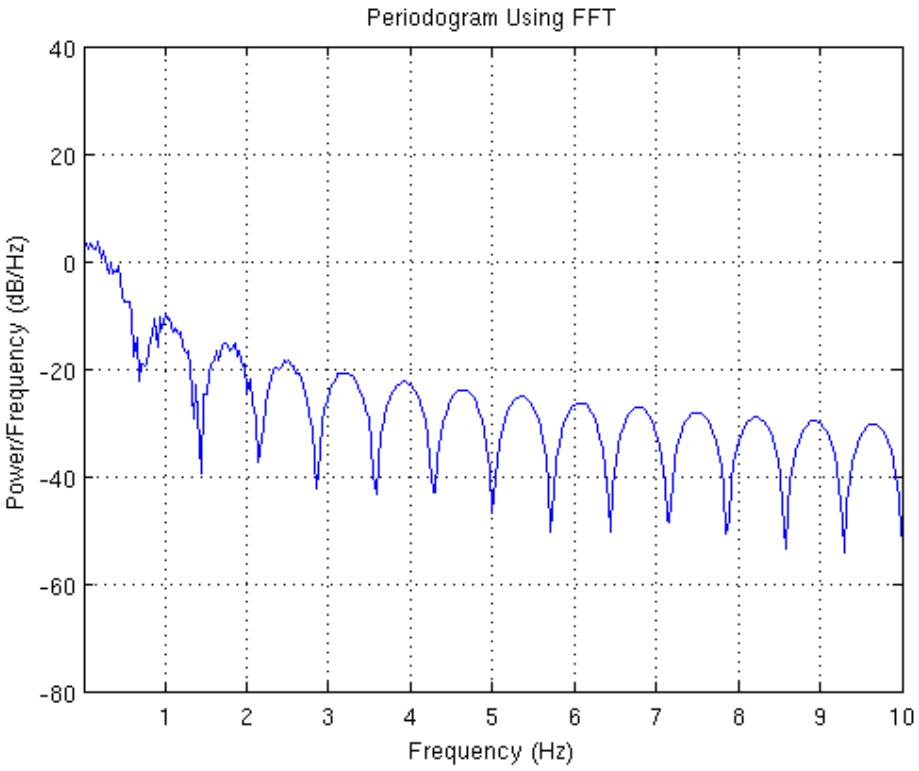


Figure 9.4.4: Spectral analysis for the case with heat transfer.

Colour plots for both adiabatic and heat transfer cases are now reported. The implementation of the temperature justifies also an increase in problem variables. The temperature field and the turbulent diffusity α_T are added compared to Figure 9.2.2 (that represents the case with only the flow). Moreover, temperature and velocity have not been calculated as instantaneous fields, but the mean fields have been considered.

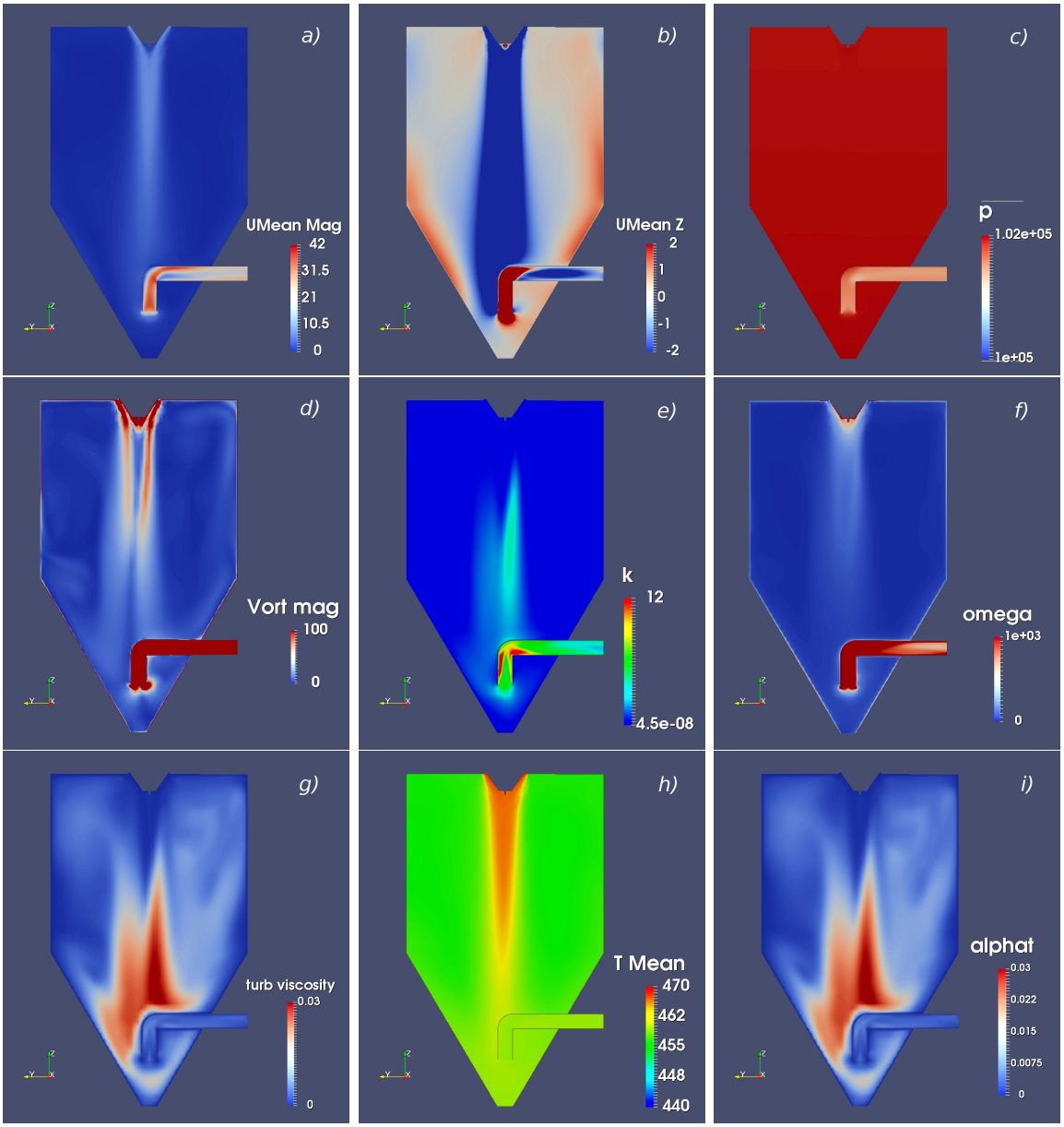


Figure 9.4.5: Color plots the adiabatic case. Please, note that in picture *i)* "alphat" means the turbulent diffusity α_T, and also in picture *f)* "omega" means the specific turbulent dissipation ω. Regarding the other figures see caption of Figure 9.2.2.

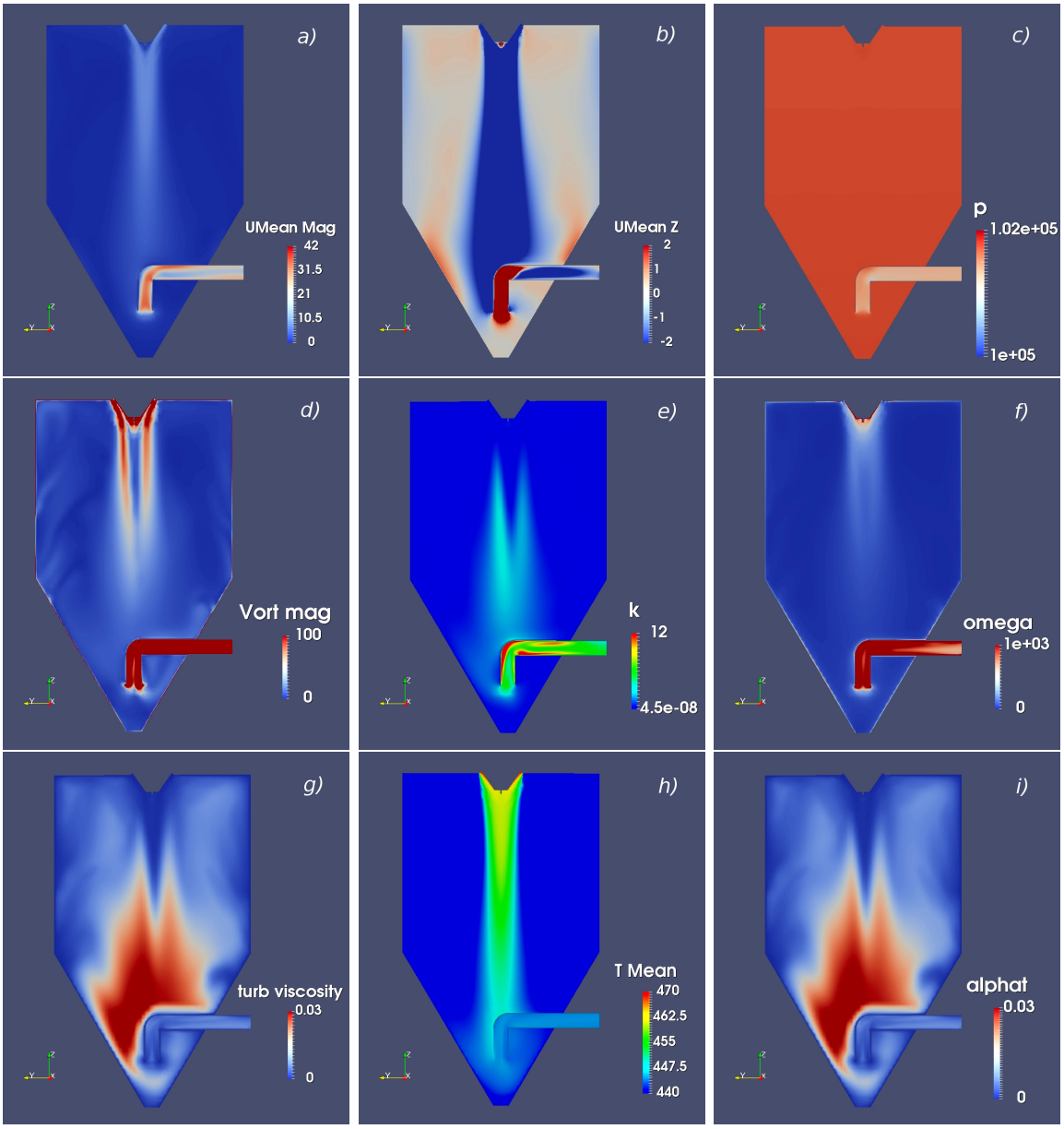


Figure 9.4.6: Color plots for the case with heat transfer. Please, note that in picture *i)* "alphat" means the turbulent diffusity α_T, and also in picture *f)* "omega" means the specific turbulent dissipation ω. Regarding the other figures see caption of Figure 9.2.2.

9.5 Implementation of the particles

The implementation of the particles has involved different studied configurations obtained by varying the dispersed phase (liquid or solid), the number of injected particle per second and the wall interaction model for *Vessel* and *Pipe* patches (see table 8.3.2). In particular, cases 2, 4 and 5 have been chosen as the most realistic cases between those listed in table 8.3.2, and their main results are now represented.

9.5.1 Liquid case

Two different cases have been studied considering the same wall interaction model (droplets stick to the walls) and considering also water as the dispersed phase, while the number of injected particles per second has been changed. First, results for 2'000 particles per second are presented and then those for 20'000 particles per second.

Let's start with the case 2'000 part/s.
Figure 9.5.1 represents the number of evaporated droplets normalized by the total number of the injected droplets, as a function of the evaporated particle age. It is possible to note that all droplets evaporate before t=0.8 seconds.

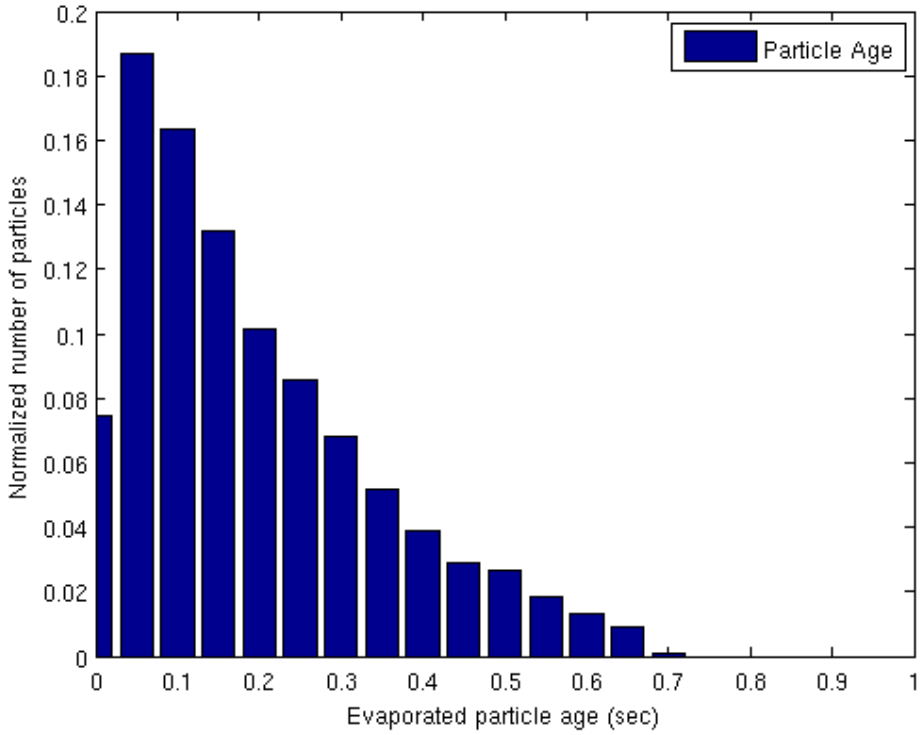


Figure 9.5.1: Liquid case - 2'000 part/sec - Distribution of the evaporated particles age normalized by the total number of evaporated particles.

In Figure 9.5.2, the particle age is presented as the droplet diameter changes. In particular, the total range of the droplet diameters has been divided in six classes. It is evident that, in the spray dryer, droplets with a bigger diameter survive longer than the others.

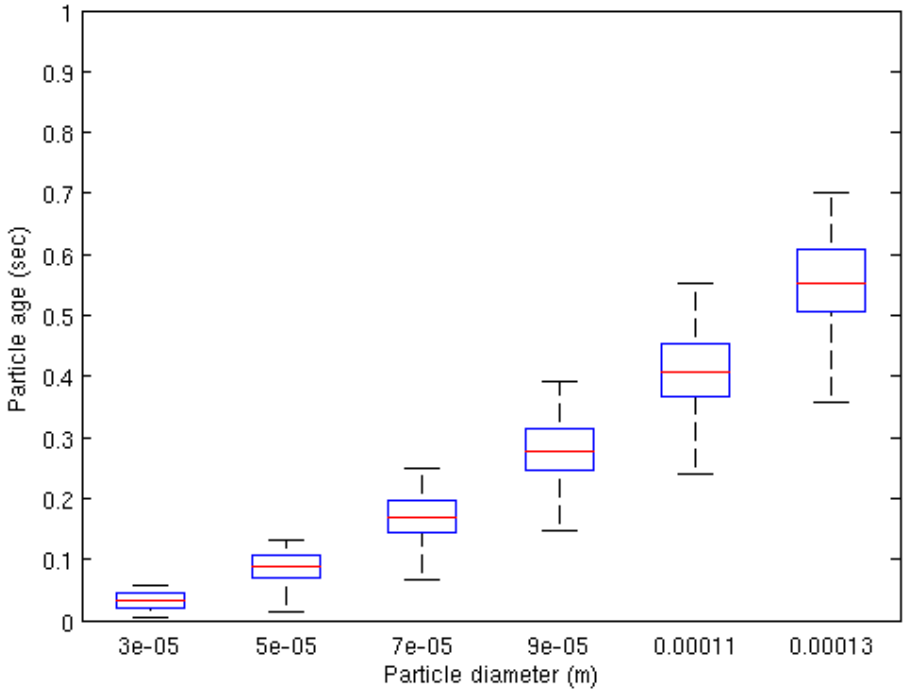


Figure 9.5.2: Liquid case - 2'000 part/sec - Particle age divided in diameter classes.

Figure 9.5.3 represents the same results obtained in the histogram of Figure 9.5.1, but in a more directly and intuitive way, adding also information about droplet positions. In fact, the spatial distribution of the droplets at the moment of the evaporation is represented as the droplet age changes. As stated above, it is possible to note that evaporation has a relevant effect and water droplets are all evaporated at t=0.8s.

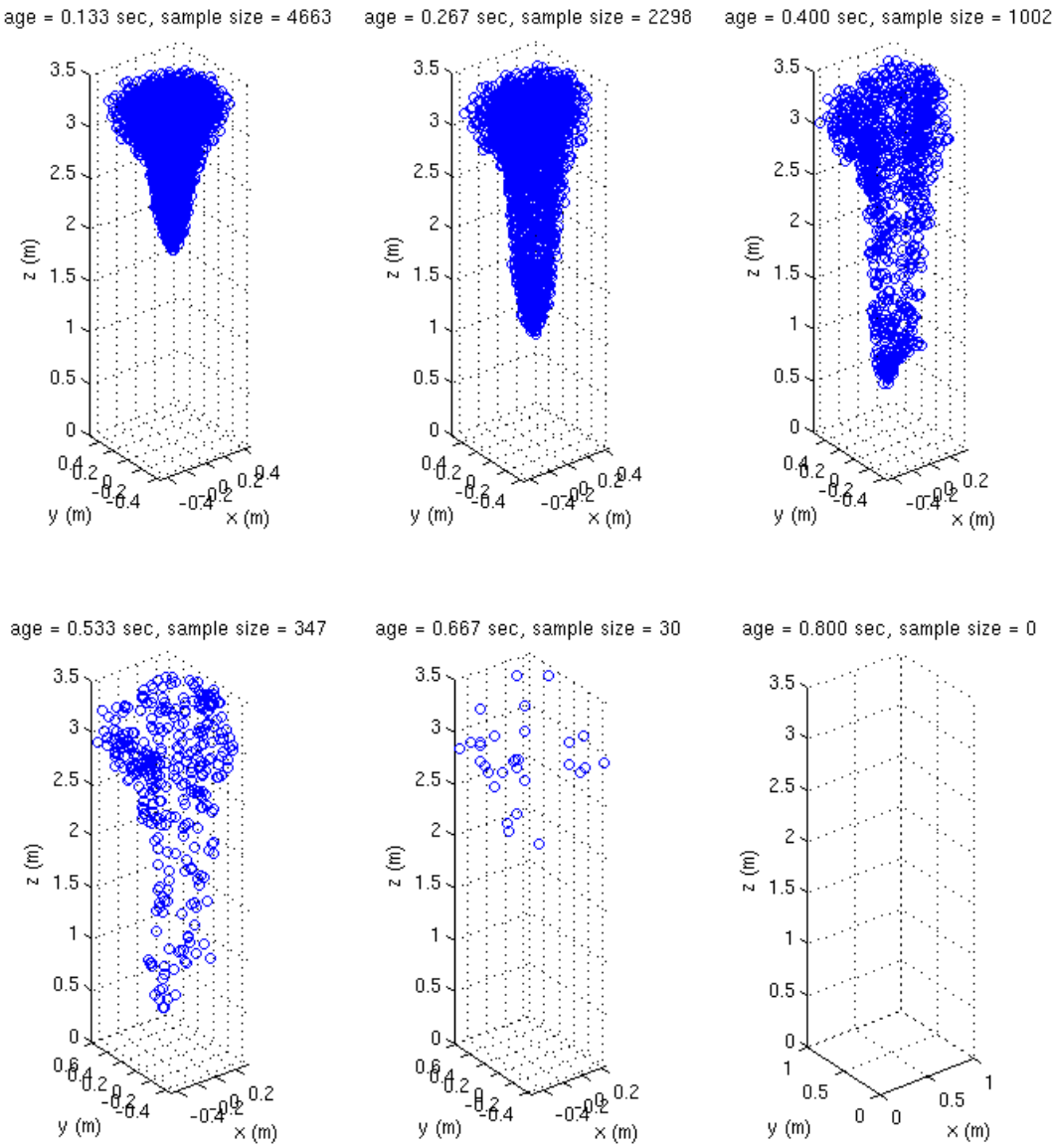


Figure 9.5.3: Liquid case - 2'000 part/sec - Spatial distribution of the evaporated particles.

The same analysis has been made for the case with 20'000 part/sec and similar results have been obtained.

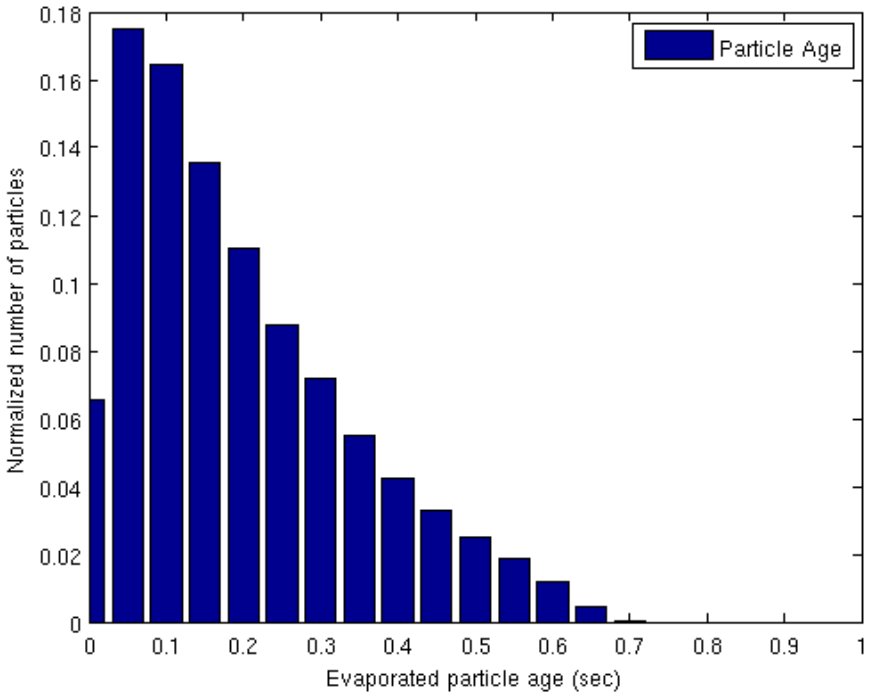


Figure 9.5.4: Liquid case - 20'000 part/sec - Distribution of the evaporated particles age normalized by the total number of evaporated particles.

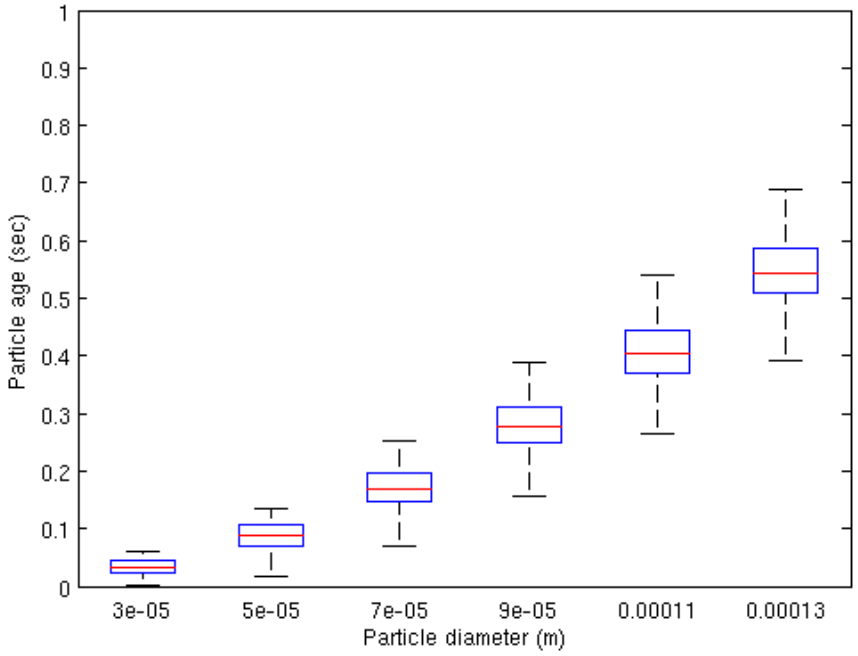


Figure 9.5.5: Liquid case - 20'000 part/sec - Particle age divided in diameter classes.

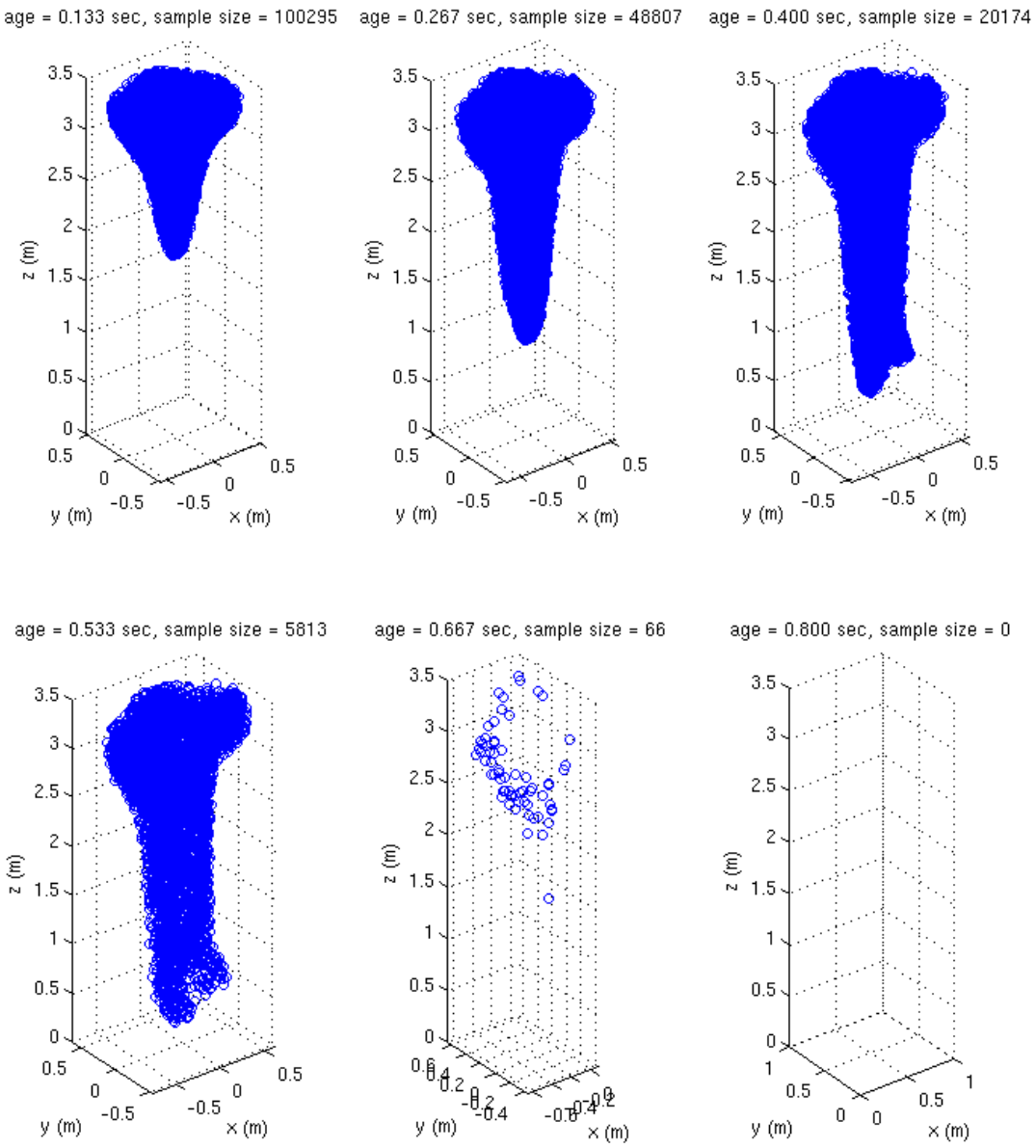


Figure 9.5.6: Liquid case - 20'000 part/sec - Spatial distribution of the evaporated particles.

Taking into account Figures 9.5.4, 9.5.5 and 9.5.6, it is possible to conclude that the increase of the injected particles number does not really affect the evaporation phase.

9.5.2 Solid case

Results for the solid case are now presented. As concerns the nature of the particle under consideration, olive pomace extract mixed to maltodextrin has been chosen as dispersed phase and its physical properties have been obtained thanks to experimental measurements provided by the Material Engineering Laboratory of DICCA - University of Genoa. As it will be seen in chapter 9.6, the different density ($\rho_{solid} \simeq 100 kg/m^3$ evaluated at the reference inlet value of the temperature) affects the results in a relevant way. The *rebound* condition has been chosen as the most realistic wall interaction model; moreover, the injection model has been set in order to constantly add 2'000 part/sec.
Contrary to the liquid case, in Figure 9.5.7 is represented the number of the collected particles normalized by the sample size of the injected particles, as a function of the collected particle age. The reason is that the heat transfer model is neglected in the solid case, so particles do not increase temperature and do not evaporate. Therefore, a larger number of particles can reach the outlet.

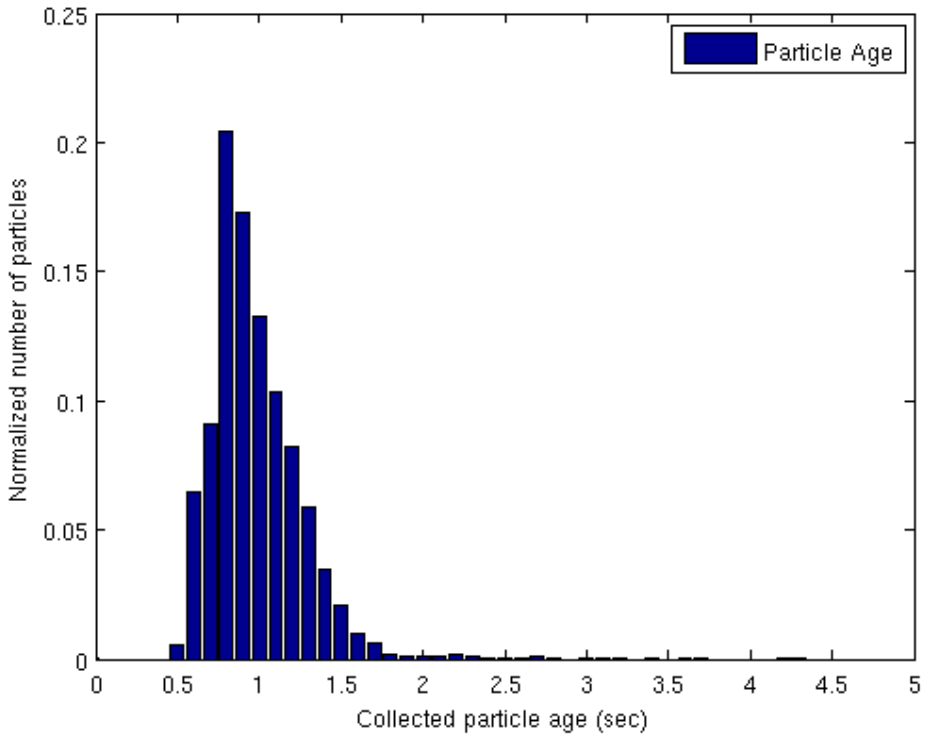


Figure 9.5.7: Solid case - 20'000 part/sec - Distribution of collected particles age normalized by the total number of collected particles.

Figure 9.5.8 represents the injected and the collected particles distributions, both respectively normalized by the sum of the injected or collected particles. Indeed, in doing so, the two cases are not comparable, but this figure is useful to find the peak of the range of the particle diameters for both injected and collected particles.

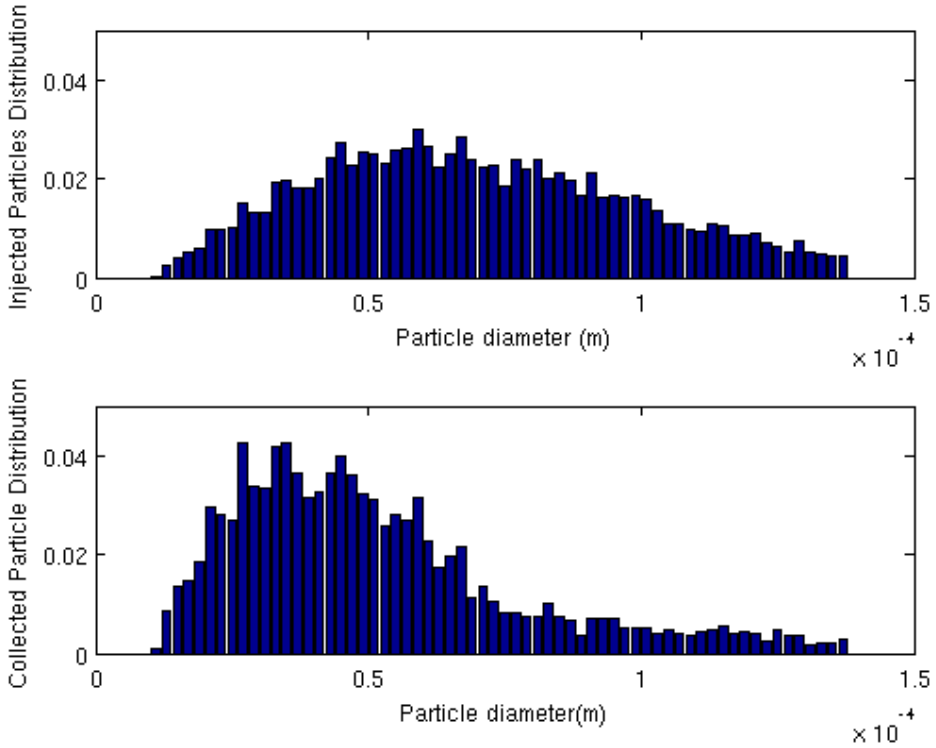


Figure 9.5.8: Solid case - 20'000 part/sec - Injected particles distribution normalized by the sum of the injected particles - on the top. Collected particles distribution normalized by the sum of the collected particles - on the bottom.

Instead, the comparison between the injected and the collected particles distributions is illustrated in Figure 9.5.9. In fact, the number of the injected and collected particles are now both normalized by the sum of the injected particles. Different time windows with same size (1 sec) have been represented in order to look for a steady state of the collected particles. It is possible to note a relevant variation of the collected particles profile (highlighted in green in Figure 9.5.9) between t=4 sec and t=6 sec, while starting from this latter instant the collected particles profile is more or less constant, confirming a steady behavior. That is the reason why the simulation has been stopped at t=10 sec. As concerns the injected particles, the distribution (highlighted in blue in Figure 9.5.9) is the same in all the time windows; minor variations are a consequence of the random injection model.

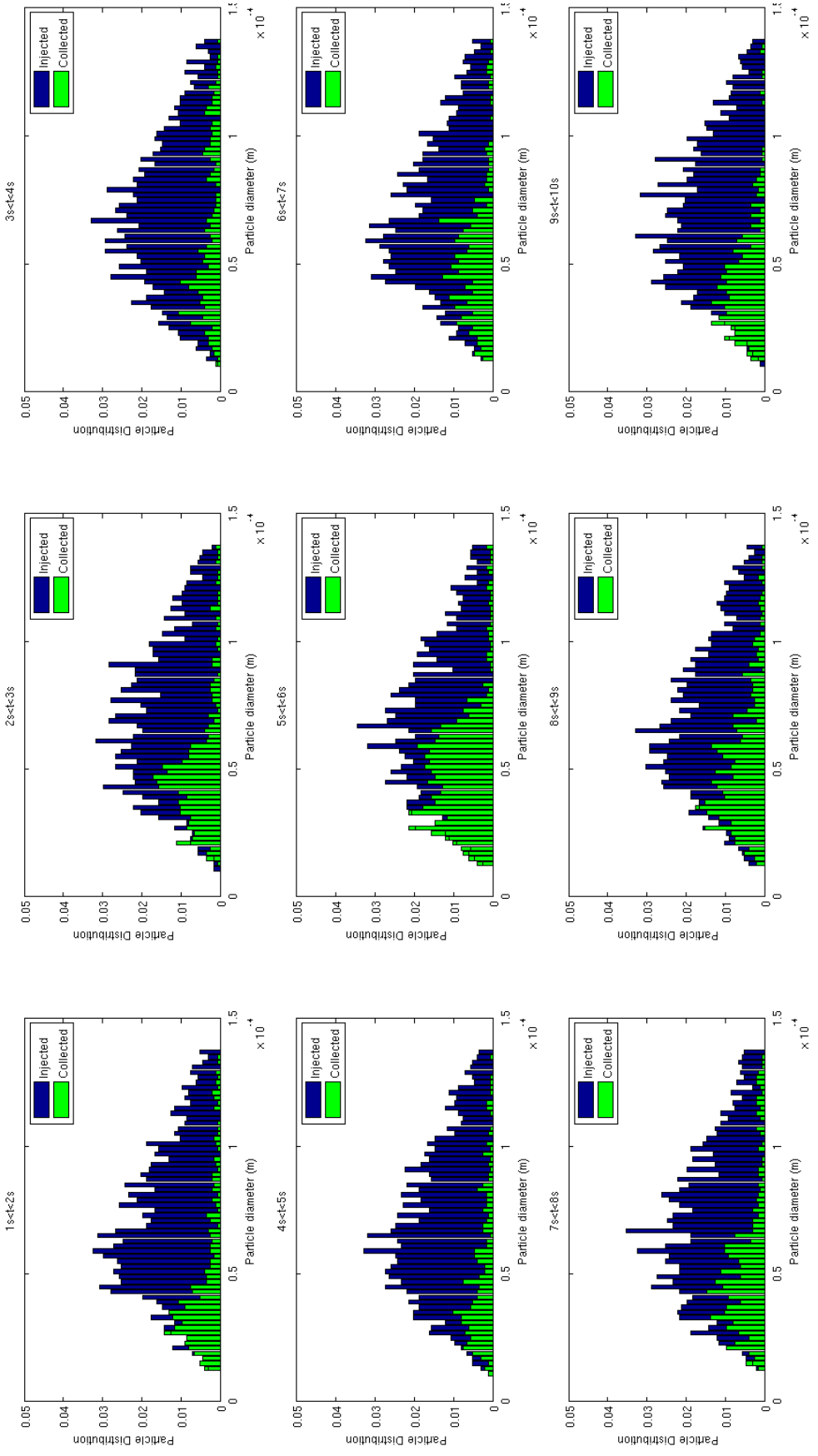


Figure 9.5.9: Solid case - 20'000 part/sec - Particles distributions normalized by the sum of the injected particles.

Finally, to better understand the behavior and the trend of the particles in time for the solid case, a set of figures are now presented. In Figure 9.5.10 particle tracks are shown for the total range of diameters (from 10 μm to 137 μm) evaluating them for different instants. The same images have also been taken for four different classes of diameters in order to appreciate the influence of the particle dimension on the particle trajectories as it is shown in Figures 9.5.11, 9.5.12, 9.5.13 and 9.5.14.

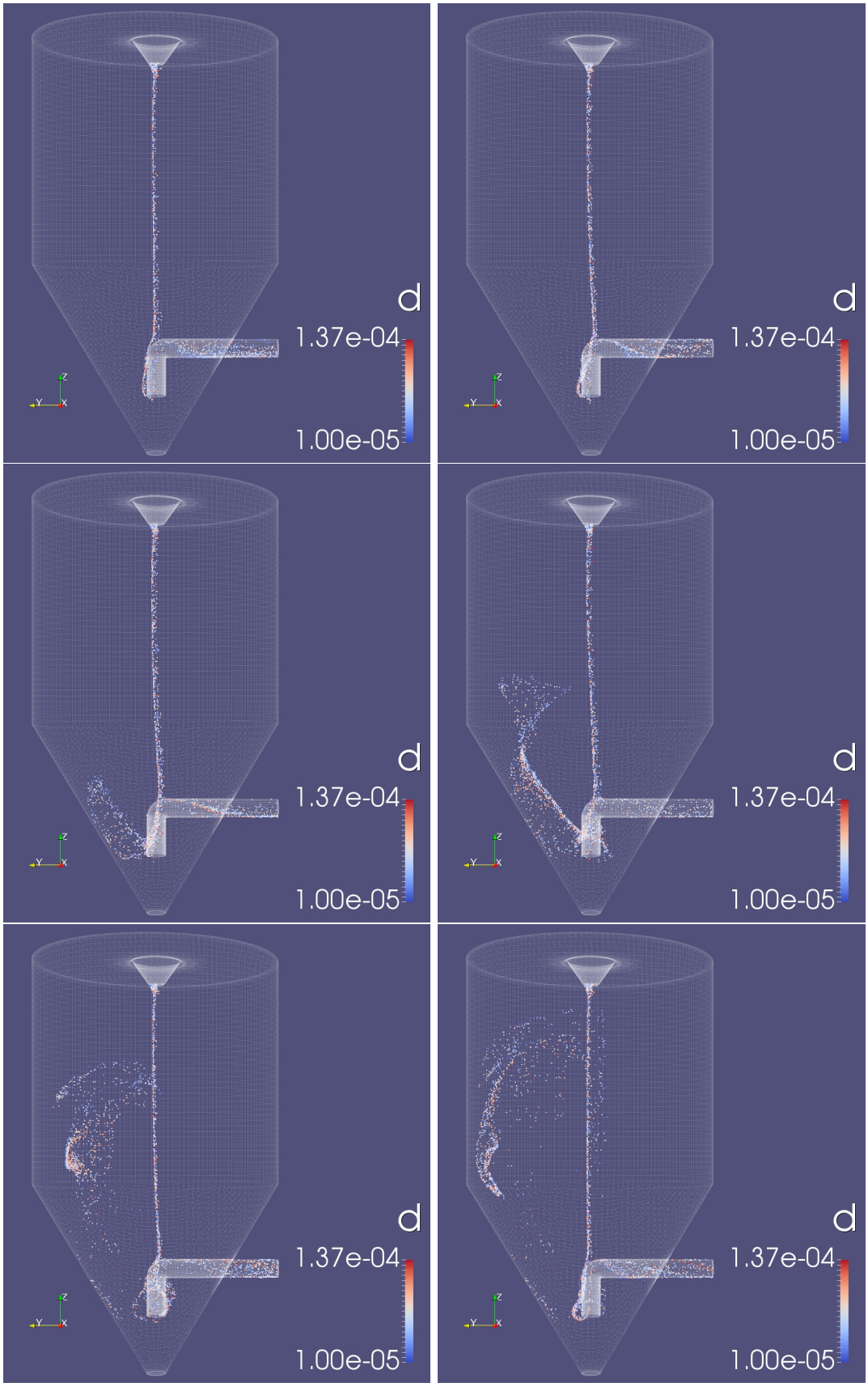


Figure 9.5.10: Total range of particles as a function of time: 1.5s on the top left - 2.5 on the top right - 3.5s on the middle left - 4.5s on the middle right - 5.5s on the bottom left - 6.5s on the bottom right.

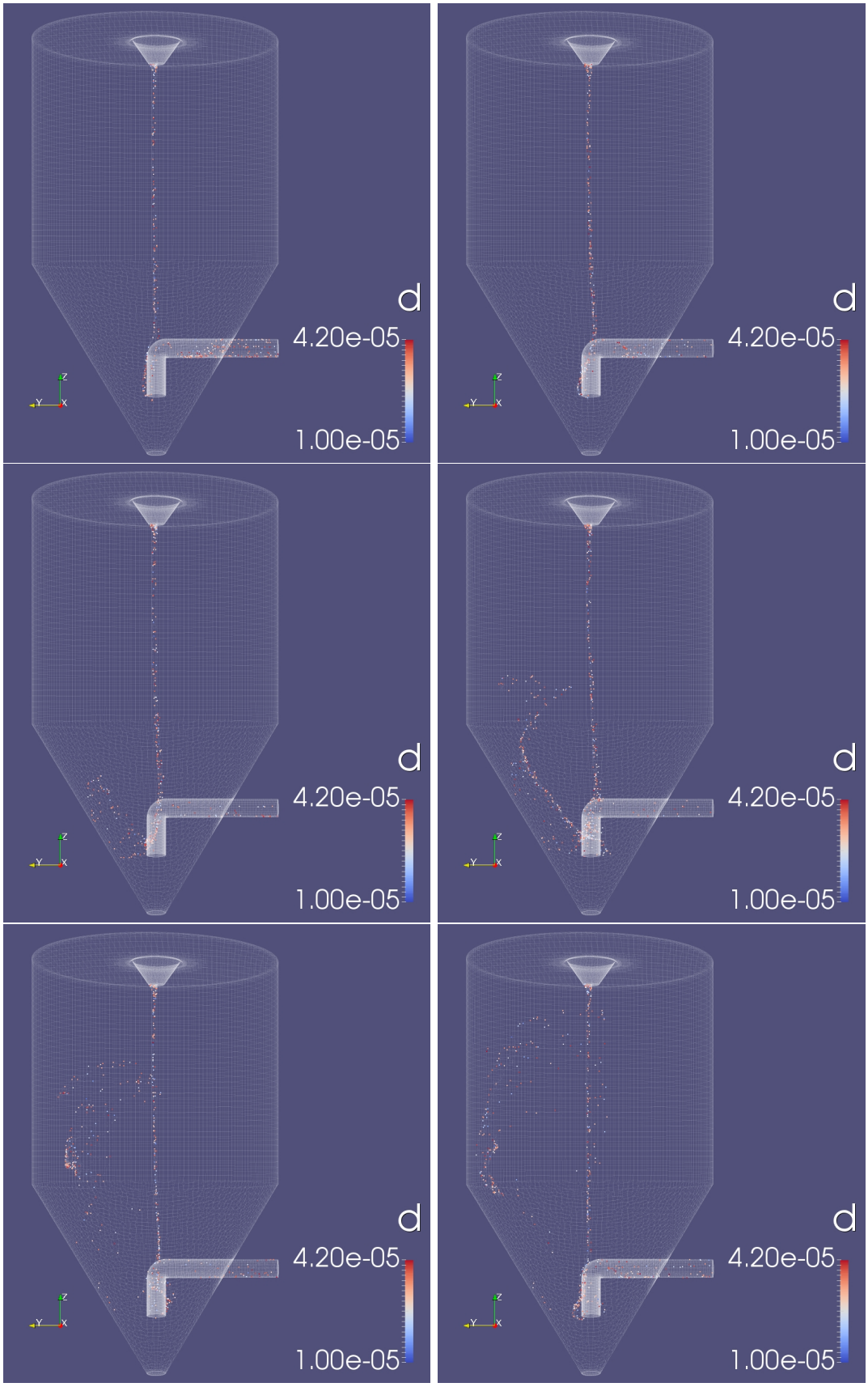


Figure 9.5.11: First class of particle diameters as a function of time: 1.5s on the top left - 2.5 on the top right - 3.5s on the middle left - 4.5s on the middle right - 5.5s on the bottom left - 6.5s on the bottom right.

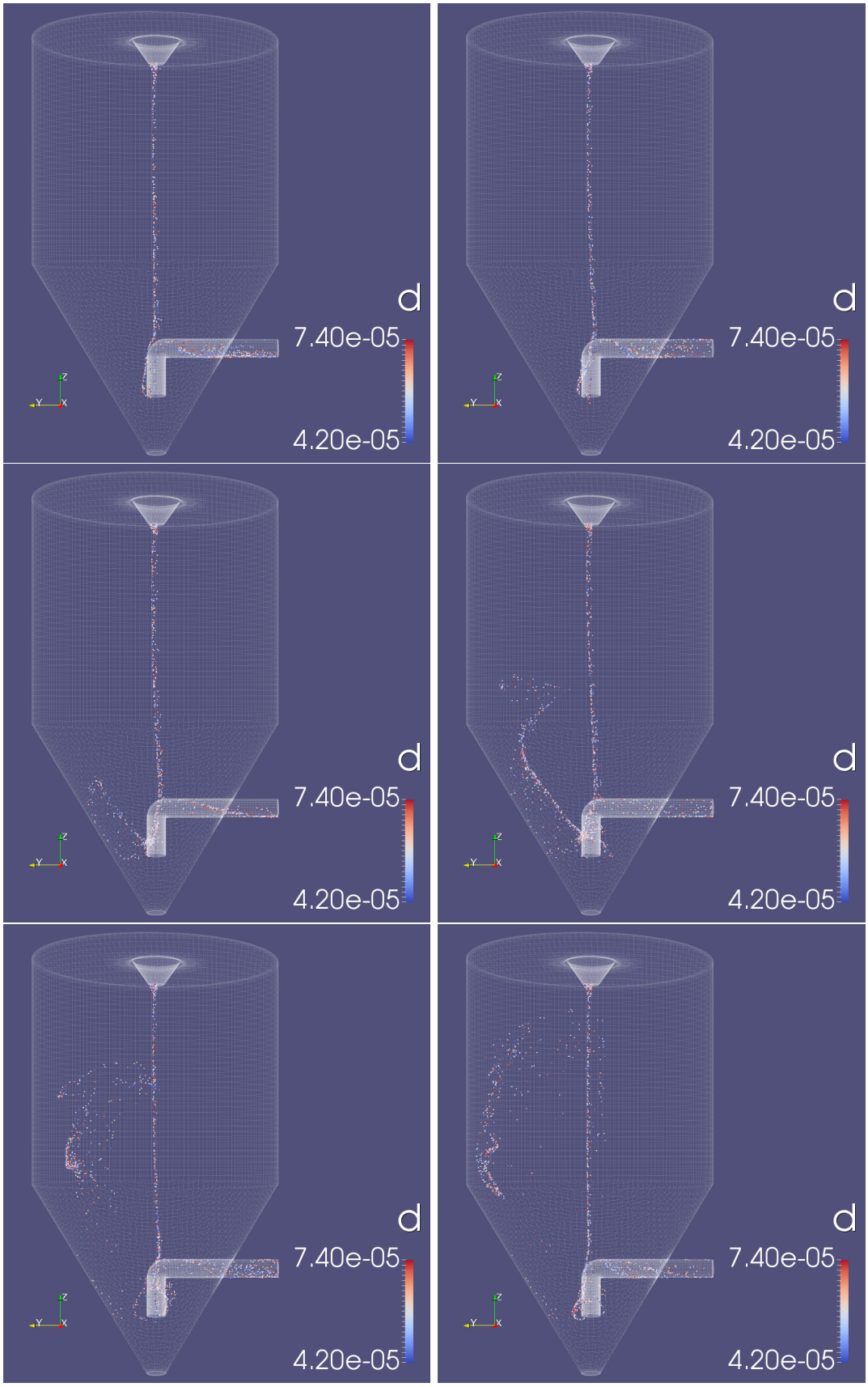


Figure 9.5.12: Second class of particle diameters as a function of time: 1.5s on the top left - 2.5 on the top right - 3.5s on the middle left - 4.5s on the middle right - 5.5s on the bottom left - 6.5s on the bottom right.

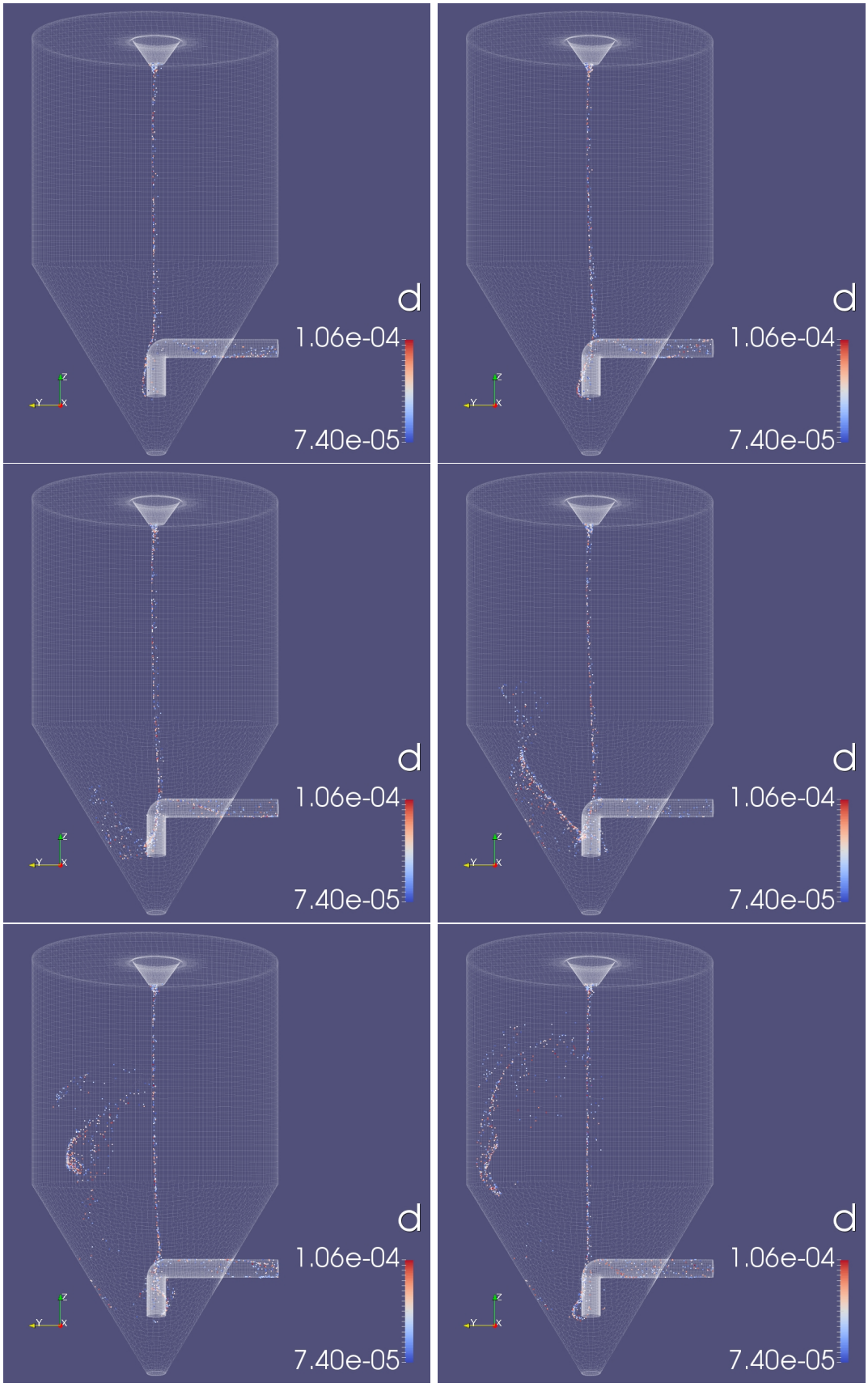


Figure 9.5.13: Third class of particle diameters as a function of time: 1.5s on the top left - 2.5 on the top right - 3.5s on the middle left - 4.5s on the middle right - 5.5s on the bottom left - 6.5s on the bottom right.

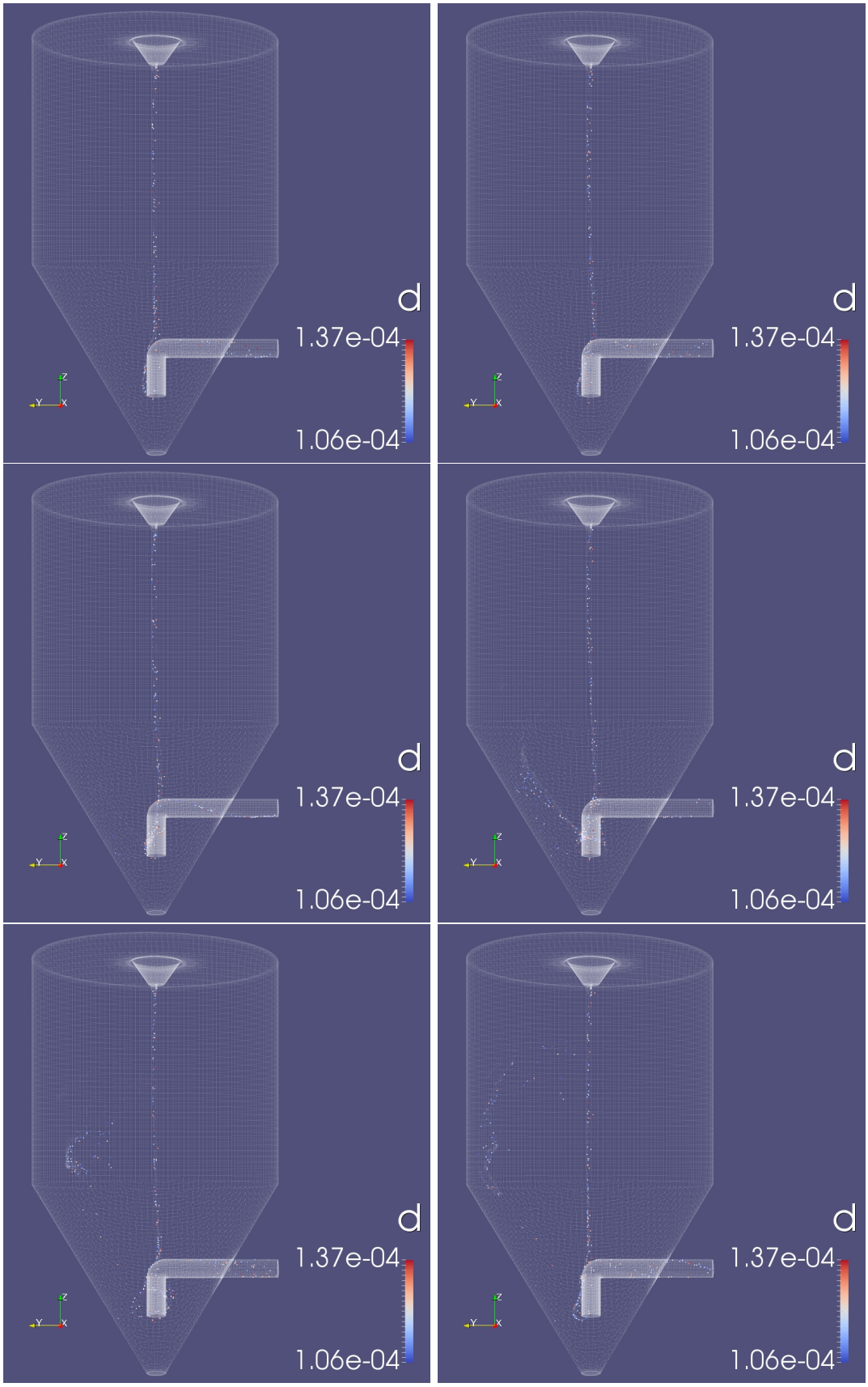


Figure 9.5.14: Fourth class of particle diameters as a function of time: 1.5s on the top left - 2.5 on the top right - 3.5s on the middle left - 4.5s on the middle right - 5.5s on the bottom left - 6.5s on the bottom right.

9.5.3 Comparison between liquid and solid case.

A brief comparison between the liquid and the solid phase is now presented. The main aspect that can be compared is the trend of the droplets and particles positions, that are affected in a relevant way by the particles density, as mentioned in chapter 9.5.2.

In Figure 9.5.15, it is possible to note how a lower density (solid case) reduces the particles injected inertia, causing a vertical tight trajectory from the atomizer to the pipe, where particles rebound and start recirculating. On the contrary the higher density of the H2O allows the droplets to spread out following the conic form imposed by the injection model (see table 6.1). That is why the left column of Figure 9.5.15 (liquid case) shows a larger scatter of the particles in the spray dryer volume.

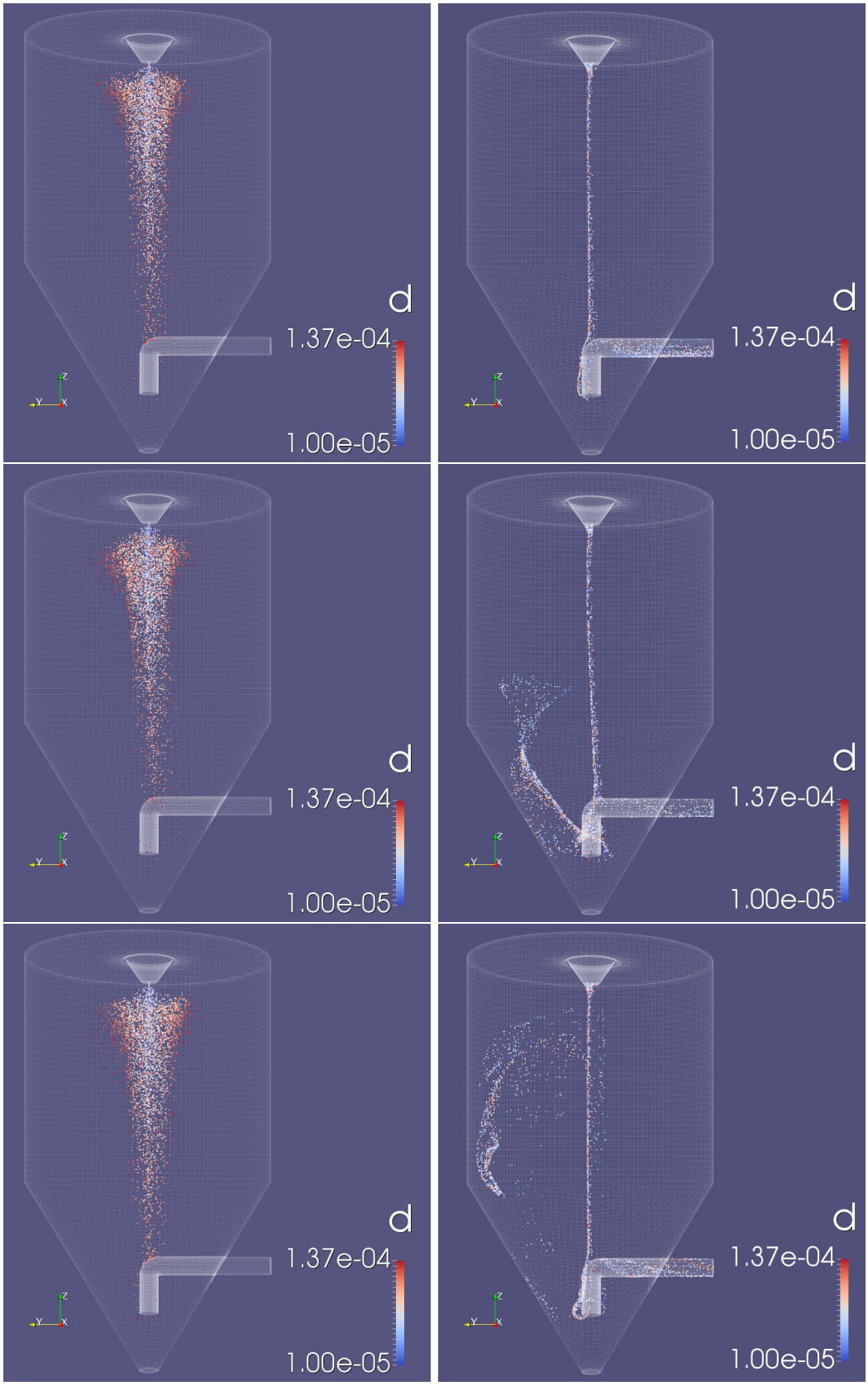


Figure 9.5.15: Comparison between liquid and solid phase. Images in left columns represent liquid phase, while those in right columns represent solid phase. Figures on the top represent instant t=1.5s, those in the middle t=4.5s and those on the bottom t=6.5s.

10 Conclusions

10.1 Final remarks

Despite spray dryers are used in a wide range of industries like food manufactures, chemical and pharmaceutical industry and other product processes, they are still principally designed by virtue of pilot experiments. CFD models developed through the last years are based on expensive software. That is the reason why the aim of this thesis is to develop a model of spray drying using CFD with an open-source software, freely available and without license costs.

To achieve this, different step have been gradually followed. First, an original geometry has been created starting from the data literature; different geometries of the atomizer and of the length of the *PipeDown* have been studied. The latter parameter resulted more important, influencing the results in a relevant way.

Moreover, mesh and grid generation and also the turbulence model have been tested in order to have the best setup able to accurately approach the velocity and the temperature profiles founded in the literature. Once this goal has been achieved, particles have been implemented. Two different dispersed phases have been tested: liquid and solid varying principally the number of the injected particles per second. As concerns the liquid case, it has been noticed that the increasing of the injected particles does not affect the evaporation phase. Regarding the solid case, a steady behavior of the collected particles has been reached for a time simulation t=6 sec. Finally, a brief comparison between the liquid and solid case has been presented, showing the relevant influence of the particle density on its trend position in the spray dryer volume.

10.2 Future developments

The course of the work has been made in such way that the initial parts regarding the creation of the geometry, flow solution and temperature solution are in accordance with data found in the literature. For this reason these parts can be regarded developed and do not need much work for the continuation of the project.

The modeling part regarding particles, liquid/solid, do need further development. At the present stage particles are treated as liquid or solid, collisions are not accounted and rotational motion is neglected. The most crucial part to incorporate is the liquid/solid two-stage evaporation process. This essentially means that a drop is treated as liquid initially and, at full evaporation, turns in to the solid phase. Models for this do exist but needs further implentation and verification. The other parts mentioned above are also of interest but considered of less priority for the next step in developing this numerical spray drying tool.

References

[1] D.E. Oakley, R.E. Bahu, Computational modelling of spray dryers, Computers and Chemical Engineering 17 (Suppl. 1) (1993) 493–498.

[2] F.G. Kieviet, P.J.A.M. Kerkhof, Using computational fluid dynamics to model product quality in spray drying: air flow, temperature and humidity, in: A.S. Mujumdar (Ed.), Drying'96-Proceedings of the 10th International Drying Symposium (IDS'96), vol. A, Lodz Technical University, Lodz, 1996, pp. 259–266.

[3] F.G. Kieviet, P.J.A.M. Kerkhof, Air flow, temperature and humidity patterns in a co-current spray dryer: modelling and measurements, Drying Technology 15 (1997) 1763–1773.

[4] F.G. Kieviet, Modelling Quality in Spray Drying. Ph.D. Thesis, Eindhoven University of Technology, Netherlands, 1997.

[5] L. Huang, K. Kumar, A.S. Mujumdar, A parametric study of the gas flow patterns and drying performance of co-current spray dryer: results of a computational fluid dynamics study, Drying Technology 21 (2003) 957–978.

[6] L. Huang, K. Kumar, A.S. Mujumdar, Use of computational fluid dynamics to evaluate alternative spray dryer chamber configurations, Drying Technology 21 (2003) 385–412.

[7] L. Huang, K. Kumar, A.S. Mujumdar, A comparative study of a spray dryer with rotary disc atomizer and pressure nozzle using computational fluid dynamic simulations, Chemical Engineering and Processing 45 (2006) 461–470.

[8] F. Ducept, M. Sionneau, J. Vasseur, Superheated steam dryer: simulations and experiments on product drying, Chemical Engineering Journal 86 (2002) 75–83.

[9] T.A.G. Langrish, D.F. Fletcher, Spray drying of food ingredients and applications of CFD in spray drying, Chemical Engineering and Processing 40 (2001) 345–354.

[10] T.A.G. Langrish, T.K. Kockel, The assessment of a characteristic drying curve for milk powder for use in computational fluid dynamics modeling, Chemical Engineering Journal 84 (2001) 69–74.

[11] D.B. Southwell, T.A.G. Langrish, D.F. Fletcher, Use of computational fluid dynamics techniques to assess design alternatives for the plenum chamber of a small spray dryer, Drying Technology 19 (2001) 257–268.

[12] D.F. Fletcher, B. Guo, D.J.E. Harvie, T.A.G. Langrish, J.J. Nijdam, J.Williams, What is important in the simulation of spray dryer performance and how do current CFD models perform? Applied Mathematical Modelling 30 (2006) 1281–1292.

[13] I. Zbicinski, Development and experimental verification of momentum, heat and mass transfer model in spray drying, Chemical Engineering Journal 58 (2) (1995) 123–133.

[14] I. Zbicinski, X.Y. Li, Conditions for accurate CFD modelling of spray-drying process, Drying Technology 24 (2006) 1109–1114.

[15] M. Mezhericher, A. Levy, I. Borde, Droplet–droplet interactions in spray drying using 2D computational fluid dynamics, Drying Technology 26 (2008) 265–282.

[16] M. Mezhericher, Drying of Slurries in Spray Dryers. Ph.D. Thesis, Ben Gurion University of the Negev, Israel, 2008.

[17] M. Mezhericher, A. Levy, I. Borde, Modelling of droplet drying in spray chambre using 2D and 3D computational fluid dynamics, Drying Technology 27 (3) (2009) 359–370.

[18] M.W. Woo, W.R.W. Daud, A.S. Mujumdar, Z.H. Wu, M.Z.M. Talib, S.M. Tasirin, CFD evaluation of droplet drying models in a spray dryer fitted with a rotary atomizer, Drying Technology 26 (2008) 1180– 1198.

[19] Anderson, J. D. (1984). Computational fluid dynamics e The basics with applications. New York: McGraw-Hill Inc.

[20] Masters, K. (1991). Spray drying. Essex: Longman Scientific Technical / John Wiley Sons Inc.

[21] Mujumdar, A. S. (1987). Handbook of industrial drying. New York: Marcel Dekker.

[22] Fellows, P. J. (1998). Food processing technology-principles and practice. Cambridge: Woodhead Publishing Limited.

[23] Langrish, T. A. G., Fletcher, D. F. (2001). Spray drying of food ingredients and applications of CFD in spray drying. Chemical Engineering and Processing, 40, 345e354.

[24] C. Anandharamaskrishnan (2008). Experimental and Computational Fluid Dynamics Studies on SprayFreeze-Drying and Spray-Drying of Proteins.

[25] Mostafa, A. A., Mongia, H. C. (1987). On the modeling of turbulent evaporating sprays: Eulerian versus Lagrangian approach. International Journal of Heat and Mass Transfer, 30(12), 2583e2593.

[26] Jakobsen, H. A., Sannaes, B. H., Grevskott, S., Svendsen, H. F. (1997). Modelling of vertical bubbledriven flows. Industrial Engineering Chemistry Research, 36, 4052e4074.

[27] Nijdam, J. J., Guo, B., Fletcher, D. F., Langrish, T. A. G. (2006). Lagrangian and Eulerian models for simulating turbulent dispersion and coalescence of droplets within a spray. Applied Mathematical Modelling, 30, 1196e1211.

[28] Bakker, A. (2002). Computational fluid mixing. Lebanon, NH, USA: Fluent Inc.

[29] Launder, B. E., Spalding, D. B. (1972). Lectures in mathematical models of turbulence. London, UK: Academic Press.

[30] Launder, B. E., Spalding, D. B. (1974). The numerical computation of turbulent flows. Computer Methods in Applied Mechanics and Engineering, 3, 269e289.

[31] Yakhot, V., Orszag, S. A. (1986). Renormalization group analysis of turbulence: i. basic theory. Journal of Scientific Computing, 1, 1e51.

[32] Shih, T. H., Liou,W.W., Shabbir, A., Zhu, J. (1995). A new ke3 eddyviscosity model for high Reynolds number turbulent flows e model development and validation. Computers Fluids, 24(3), 227e238.

[33] Launder, B. E., Reece, G. J., Rodi, W. (1975). Progress in the development of a reynoldsstress turbulence closure. Journal of Fluid Mechanics, 68, 537e566.

[34] Fletcher, A. J. (2000). Computational techniques for fluid dynamics, (2nd ed.). New York: SpringerVerlag.

[35] Bakker, A. (2002). Computational fluid mixing. Lebanon, NH, USA: Fluent Inc.

[36] J. Sloth (2010). Method for Improving Spray Drying Equipment and Product Properties.

[37] H. Versteeg and W. Malalasekera, An introduction to computational fluid dynamics. Harlow, England: Pearson Education Ltd., 2007.

[38] J. Jaramillo, C. Pérez-Segarra, A. Oliva and K. Claramunt, 'Analysis of different RANS models applied to turbulent forced convection', International Journal of Heat and Mass Transfer, vol. 50, no. 19-20, pp. 3749-3766, 2007.

[39] X. Liu, A. Godbole, C. Lu, G. Michal and P. Venton, 'Source strength and dispersion of CO2 releases from high-pressure pipelines: CFD model using real gas equation of state', Applied Energy, vol. 126, pp. 56-68, 2014.

[40] B. Xu, J. Zhang, J. Wen, S. Dembele and J. Karwatzki, Numerical Study of a Highly Underexpanded Hydrogen Jet, 1st ed. London: School of Engineering, Kingston University, 2005.

[41] Wikipedia, 'Equation of state', 2015. [Online].

[42] Thermalfluidscentral.org, 'Thermal-FluidsPedia | Heat and mass transfer | ThermalFluids Central', 2015.

[43] D. Hahn and M. Ozisik, Heat conduction. Hoboken, N.J.: John Wiley Sons, 2012.

[44] Thermalfluidscentral.org, 'Thermal-FluidsPedia | Basics of heat conduction | ThermalFluids Central', 2015.

[45] Y. Cengel and A. Ghajar, Heat and mass transfer. New York: McGraw-Hill, 2011.

[46] B. Guo, T.A.G. Langrish, D.F. Fletcher, Simulation of turbulent swirl flow in an axisymmetric sudden expansion, AIAA J. 39 (2001)96–102

[47] Gosman, A.D., Ioannides, E., 1983, Aspects of computer simulation of liquid-fuelled combustors, J. Energy, 7(6): 482-490.

[48] http://www.openfoam.com

[49] CFD Direct - OpenFOAM User Guide: 4.1 File structure of OpenFOAM cases - http://cfd.direct/openfoam/user-guide/case-file-structure/

[50] http://www.cfd-online.com

[51] https://computing.llnl.gov

[52] R. Kuriakose, C. Anandharamakrishnan, Computational fluid dynamics (CFD) applications in spray drying of food products, 2010

[53] CFD Direct - OpenFOAM User Guide: 5.2 Boundaries - http://cfd.direct/openfoam/user-guide/boundaries/

[54] http://www.wolfdynamics.com/

[55] CFD Direct - OpenFOAM User Guide: 7.1 Thermophysical models - http://cfd.direct/openfoam/user-guide/thermophysical/

[56] CFD Direct - OpenFOAM User Guide: 5.4 Mesh generation with the snappyHexMesh utility - http://cfd.direct/openfoam/user-guide/snappyhexmesh/

57 F. Greifzu, C. Kratzsch, T. Forgber, F. Lindner R. Schwarze - Assessment of particle-tracking models for dispersed particle-laden flows implemented in OpenFOAM and ANSYS FLUENT (2016)

[[58] Ranz, W.E., Marshall, W.R., Evaporation from Drops, Chem.Eng.Prog,48 (1952),22, pp.141-146]

Nomenclature

Acronyms

CFD Computational Fluid Dynamics

CPU Central Processing Unit

DKA Drying Kinetics Analyzer

FDM Finite Difference Method

FEM Finite Element Method

FVM Finite Volume Method

GNU GNU is Not Unix

IC Integrate Circuit

MPI Message Passing Interface)

NP Number of Physical cores

OOP Object-Oriented Programming

OpenFOAM Open source Field Operation And Manipulation

PDEs Partial Differential Equations

PISO Pressure Implicit with Split Operator

RANS Reynolds-averaged Navier–Stokes equations

RNG Re-Normalisation Group

RSM Reynolds Stress Model

SIMPLE Semi-Implicit Method for Pressure Linked Equations

SST Shear Stress Transport

STL STereoLithography

URANS Unsteady Reynolds-averaged Navier–Stokes equations

Greek Symbols

α	turbulent diffusity	$\mathrm{kg}/(\mathrm{m}\cdot\mathrm{s})$
ϵ	turbulent dissipation	$\mathrm{m}^2/\mathrm{s}^3$
κ	turbulent kinetic energy	$\mathrm{m}^2/\mathrm{s}^2$
μ	dynamic viscosity	$\mathrm{kg}/(\mathrm{m}\cdot\mathrm{s})$

ν	kinematic viscosity	m^2/s
ω	turbulent specific dissipation	$1/s$
ϕ	hypothetical field	1

Roman Symbols

p	pressure	Pa
T	temperature	K
t	time	s
U	velocity magnitude	m/s
Z	vorticity magnitude	$1/s$

Subscripts

a	ambient
p	particle
T	Turbulent
w	wall

Appendices

A Case with the only air flow - fvSchemes

```
ddtSchemes
{
    default           Euler;
}

gradSchemes
{
    default           Gauss linear;
    grad(U)           Gauss linear;
}

divSchemes
{
    default           none;

    div(phi,U)        Gauss linearUpwind grad(U);
    div(phi,k)        Gauss upwind;
    div(phi,omega)    Gauss upwind;
    div(phi,R)        Gauss limitedLinear 0.777;
    div(R)            Gauss linear;
    div(phi,nuTilda)  Gauss limitedLinear 0.777;
    div((nuEff*dev(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default           Gauss linear corrected;
}

interpolationSchemes
{
    default           linear;
}

snGradSchemes
{
    default           corrected;
}
fluxRequired
{
    default           no;
    p;
}
```

B Case with the only air flow - fvSolution

```
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.01;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration on;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels     1;
    }
    pFinal
    {
        $p;
        tolerance       1e-06;
        relTol          0;
    }
    "(U|k|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol          0.001;
    }

    "(U|k|omega)Final"
    {
        $U;
        tolerance       1e-08;
        relTol          0;
    }
}
PIMPLE
{
    nNonOrthogonalCorrectors 2;
    nCorrectors         2;
}
potentialFlow
{
    nNonOrthogonalCorrectors 10;
}
cache
{
    grad(U);
}
```

C Implementation of the temperature - fvSchemes

```
ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default cellLimited leastSquares 1.0;
    grad(U) cellLimited leastSquares 1.0;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind grad(U);
    div(phi,K)      Gauss upwind;
    div(phi,h)      Gauss upwind;
    div(phi,T)      Gauss upwind;
    div(phi,k)      Gauss upwind;
    div(phi,omega)  Gauss upwind;
    div((muEff*dev2(T(grad(U))))) Gauss linear;
}


laplacianSchemes
{
    default        Gauss linear limited 0.5;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default limited 0.5;
}

fluxRequired
{
    default         no;
    p_rgh;
}
```

D Implementation of the temperature - fvSolutions

```
solvers
{
    "rho.*"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       0;
        relTol          0;
    }
    p
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.001;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration on;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels     1;
    }

    pFinal
    {
        $p;
        tolerance       1e-6;
        relTol          0;
    }

    p_rgh
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.001;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration on;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels     1;
    }
```

```
    p_rghFinal
    {
        $p_rgh;
        relTol          0;
    }

    "(U|h|k|T|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0.001;
    }
    "(U|h|k|T|omega)Final"
    {
        $U;
        tolerance       1e-8;
        relTol          0;
    }
}


SIMPLE
{
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}


PIMPLE
{
    nNonOrthogonalCorrectors 3;
    nCorrectors         2;
    pRefPoint        (0 0 0);
    pRefValue       0;
    residualControl
    {
    "(U|h|k|T|omega|p|p_rgh)"
        {
                tolerance  1e-14;
                relTol  0;
        }
     }

}
```

```
potentialFlow
{
    nNonOrthogonalCorrectors 10;
}
relaxationFactors
{
    fields
    {
        rho             1;
                p_rgh           0.3;
    }
    equations
    {
        U               0.7;
        h               0.7;
        "(k|epsilon|omega)" 0.7;
    }
}
cache
{
    grad(U);
}
```

E Implementation of the particles - ThermoIncompressiblePoly

```
H20
{
    specie
    {
        nMoles          1;
        molWeight       18.0153;
    }
    equationOfState
    {
        rhoCoeffs<8>    ( 2.5039 -0.010587 2.0643e-05 -1.8761e-08 6.4237e-12 0 0 0 );
    }
    thermodynamics
    {
        Hf              -13423000;
        Sf              10482;
        CpCoeffs<8>     ( 1563.1 1.604 -0.0029334 3.2168e-06 -1.1571e-09 0 0 0 );
    }
    transport
    {
        muCoeffs<8>     ( 1.5068e-06 6.1598e-08 -1.8188e-11 0 0 0 0 0 );
        kappaCoeffs<8>  ( 0.0037972 0.00015336 -1.1859e-08 0 0 0 0 0 );
    }
}
air
{
    specie
    {
        nMoles          1;
        molWeight       28.85;
    }
    equationOfState
    {
        rhoCoeffs<8>    ( 4.0097 -0.016954 3.3057e-05 -3.0042e-08 1.0286e-11 0 0 0 );
    }
    thermodynamics
    {
        Hf              0;
        Sf              0;
        CpCoeffs<8>     ( 948.76 0.39171 -0.00095999 1.393e-06 -6.2029e-10 0 0 0 );
    }
    transport
    {
        muCoeffs<8>     ( 1.5061e-06 6.16e-08 -1.819e-11 0 0 0 0 0 );
        kappaCoeffs<8>  ( 0.0025219 8.506e-05 -1.312e-08 0 0 0 0 0 );
    }
}
```

F Implementation of the particles - fvSchemes

```
ddtSchemes
{
    default         Euler;
}
gradSchemes
{
    default         cellMDLimited Gauss linear 1.0;
    grad(U)         cellMDLimited Gauss linear 1.0;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind grad(U);

    div(phi,K)      Gauss upwind;
    div(phi,h)      Gauss upwind;
    div(phi,T)      Gauss upwind;
    div(phi,k)      Gauss upwind;
    div(phi,omega)  Gauss upwind;
    div(phi,Yi_h)   Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear limited 0.5;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default limited 0.5;
}
fluxRequired
{
    default         no;
        p;
        Phi;
}
wallDist
{
method meshWave;
}
```

G Implementation of the particles - fvSolutions

```
solvers
{
    "rho"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       0;
        relTol          0;
    }
    rhoFinal
    {
        $rho;
        tolerance       0;
        relTol          0;
    }
    p
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.01;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration on;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels     1;
    }

    pFinal
    {
        $p;
        tolerance       1e-6;
        relTol          0;
        minIter 1;
    }
    "(U|h|k|T|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0.001;
    }
```

```
    "(U|h|k|T|omega)Final"
    {
        $U;
        tolerance       1e-8;
        relTol          0;
        minIter 1;
    }
 Phi
    {
        $p;
    }

    "(Yi|air|O2|H2O)"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-6;
        relTol          0;
    }
    h
    {
        $Yi;
        relTol          0;
    }
    hFinal
    {
        $Yi;
        minIter 1;
    }

}

PIMPLE
{
    momentumPredictor yes;
    nNonOrthogonalCorrectors 2;
    nCorrectors         5;
    pRefPoint        (0 0 0);
    pRefValue       0;
    residualControl
    {
    "(U|h|k|T|omega|p|p_rgh)"
        {
                tolerance  1e-14;
                relTol  0;
        }
     }
}
```

```
potentialFlow
{
    nNonOrthogonalCorrectors 10;
}


relaxationFactors
{
    fields
    {
        rho             1;
        p               0.3;
    }
    equations
    {
        U               0.8;
        h               0.8;
        "(k|epsilon|omega)" 0.8;
    }
}

cache
{
    grad(U);
}
```