Contents

Abstract

The material, polyurethane (PUR), has been used in spray process for years. It is sprayed into a mould or a substrate with mixed long fibers to obtain a composite material. In the research project SFPURC (Spray-simulation Fiber PUR-Composites), a new simulation model for the spray-process technique is developed and applied, in order to estimate and optimize the corresponding distributions of PUR and fibers within a substrate.

In this study, a simulation model was established to estimate the layer structure during the PUR-spray process without the presence of fibers. Thus, the trajectory of PUR-particles together with their inter-collision and their final distribution on the substrate were simulated using different approaches and compared with experimental results.

The spray material is regarded as discrete phase, which is modeled as particles with help of the Discrete Phase Model (DPM) of the CFD environment *ANSYS FLUENT*, because the volume fraction of PUR particles is less than the continuum's volume fraction (air) [1]. PUR droplets are disintegrated in smaller droplets by the atomizer's external air injection. In this work a cell is defined as an atomizer so that it is easy to model a mobile injector. For the external air injection an air-blast atomizer has been chosen according to the experimental requirements. The velocity of the external air was optimized so that the cell-based velocity of air calculated by *ANSYS FLUENT* corresponds to the practical requests.

In order to estimate the PUR-layer structure, a two-phases flow (PUR and air) with help of the *ANSYS FLUENT*'s volume of fluid (VOF) model was adopted. This VOF model is based on a concept of a fractional volume of fluid, as previously proposed by Hirt and Nichols [7], where the interface between the primary and secondary phase flow is tracked. In the system of multiphase flows, air is the continuous phase, also regarded as primary phase, and PUR-particles represent the discrete phase, also named the secondary phase. The key point in modeling the PUR-layer structure is that the mass and momentum generated by the PUR-particles hitting the plate should be assigned through user-defined functions (UDFs) to the mass and momentum sources, generating the second phase.

In addition, the standard wall-film model based on the work of Stanton [3] and O'Rourke [4] was used to estimate the PUR-layer structure. At last, the results of VOF model, wall-film model and practical experiment were compared.

Nomenclature

A	Area (m^2, ft^2)
$\vec{a}$	Acceleration (m/s^2, ft/s^2)
CFL	Courant number (dimensionless)
E	Total energy, activation energy (J, kJ, cal, Btu)
$\vec{F}$	Force vector (N, lbf)
$\vec{g}$	Gravitational acceleration (m/s^2, ft/s^2); standard values = 9.80665 m/s^2, 32.1740 ft/s^2
I	Impulse (kg·m/s)
$\dot{m}$	Mass flow rate (kg/s, lbm/s)
p	Pressure (Pa, atm, mm Hg, lbf /ft^2)
S	Total entropy (J/K, J/kgmol-K, Btu/lbmmol-°F)
T	Temperature (K, °C, °R, °F)
t	Time (s)
u	Velocity (m/s, ft/s)
V	Volume (m^3, ft^3)
$\vec{v}$	Overall velocity vector (m/s, ft/s)
ρ	Density (kg/m^3, lbm/ft^3)
τ	Shear stress (Pa, lbf/ft^2)
$\bar{\bar{\tau}}$	Stress tensor (Pa, lbf/ft^2)
α	Volume fraction (dimensionless)
Δ	Change in variable, final − initial
μ	Dynamic viscosity (cP, Pa-s, lbm/ft-s)

Index for Nomenclature

c	Index for cell
b	Index for boiling
w	Index for wall-film
p	Index for particle
g	Index for gas
s	Index for surface
f	Index for film
α	Index for the current face where particles reside
imp	Index for impingement
$n+1$	Index for current time step
n	Index for previous time step
h	Index for heat

Index of Figures

Index of Tables

1. Introduction

1.1 Overview

The manufacture of single- or multi-layer fiber composite materials based on polyurethane has been used in spray process for years, by injecting (spraying) reactive, liquid polyurethane (PUR) into a mould or on a substrate and simultaneously introducing long fibers for stiffening. The mixture, then, cures either under or without pressure to a composite material.

The fiber reinforcement improves stiffness and strength of the composite properties significantly. Nevertheless, there are limited information about the mechanic properties, because of the unknown distribution and direction of fibers due to the complicate dynamics of the spray process. However, these properties are essential for crash simulations of PUR automobile parts. Therefore, in the research project SFPURC (Spray-simulation of Fiber PUR-Composites) a new simulation model for this process technique should be developed and applied, in order to estimate and optimize the distributions of liquid and fibers on the substrate. The simulation model should indicate the distribution of the flow and particles as well as the interaction of the mixed fibers with the flow field of spray stream and the interaction between the fibers themselves. The key point of this scientific work should focus on the development and integration of a fiber model, for within this model the dynamic and mutual reaction (impulse exchange) of the fibers and their reaction upon the flow field will be described. To ensure the practical application in the industrial development project, the model should be built as far as possible on the commercially available CFD software.

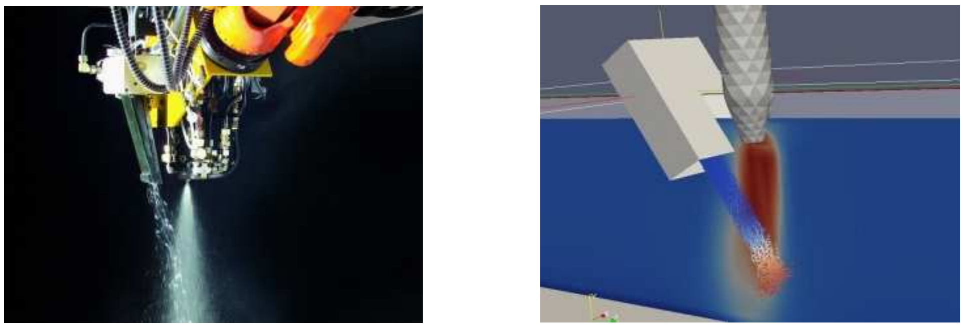


Figure 1.1: PUR-CSM Process (© Hennecke GmbH) (Left) and Simulation of Fiber Spray Process (Right)

1.2 Task

As a part of the SFPURC project, the layer structure resulting of the PUR-spray process is modeled and numerically simulated in this study. The major task includes:

1. optimization of the air-PUR spray model concerning the air velocity at the injector (discussed in Section 4.2.2: Air Source at Spray Injector)
2. modeling the layer structure with the volume of fluid (VOF) model for free surfaces instead of the existing *ANSYS FLUENT*'s wall-film model.
   The integral mass and momentum conservation by the transition of PUR-particles into a wall film are respected (discussed in Section 4.3: VOF Model for Layer Structure)
3. comparison between experimental reports and results of the standard wall-film model or the VOF model (discussed in Chapter 5: Results and Comparisons)

In this study, user-defined functions (UDFs) of *ANSYS FLUENT* are significant. They are used to optimize the velocity of extern air at spray injector, and to process the mass and impulse of particles on the layer.

2. Theory

In this chapter, the significant concepts used for the study are briefly described. The basic theory for fluid flow can be found in Section 2.1. Dispersed phase flow is described in Section 2.2 including an injection model, *ANSYS FLUENT*'s air-blast atomizer model and a boundary condition model, *ANSYS FLUENT*'s wall-film model. Section 2.3 introduces multiphase flows with *ANSYS FLUENT*'s volume of fluid (VOF) model.

2.1 Continuity and Momentum Equation

In *ANSYS FLUENT*, all flows are solved based on the conservation equations for mass and momentum. [1]

The mass conservation equation or continuity equation is formed as follows:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = S_m \tag{2.1}$$

This form is valid for incompressible as well as compressible flows. S_m denotes the mass source added to the continuous phase and can be defined as a user-defined source.

The momentum conservation equation for an inertial (non-accelerating) reference frame is formed:

$$\frac{\partial}{\partial t}(\rho \vec{v}) + \nabla \cdot (\rho \vec{v} \vec{v}) = -\nabla p + \nabla \cdot (\bar{\bar{\tau}}) + \rho \vec{g} + \vec{F} \tag{2.2}$$

where p is the static pressure, $\bar{\bar{\tau}}$ is the stress tensor (its calculation can be found in [1]), $\rho \vec{g}$ is the gravitational body force and $\vec{F}$ is external body forces (e.g., that arise from interaction with the dispersed phase). $\vec{F}$ can also contain other model-dependent sources.

2.2 Discrete Phase

Discrete phase model (DPM) is described as Lagrangian discrete phase model in *ANSYS FLUENT,* which follows the Euler-Lagrangian approach. The equation of motion of each particle can be written as follows:

$$\frac{du_p}{dt} = a_1 + a_2 + a_3 \tag{2.3}$$

where u_p describes the particle velocity, a_1 is the drag force per unit particle mass based on the relative velocity between fluid and particle, a_2 is the density based gravitational acceleration and a_3 gathers additional accelerations arising from thermophoretic, Brownian, Saffman's lift motions for example.

DPM is used to model particles, droplets, or bubbles dispersed in a continuous fluid phase. Momentum, mass and energy transfer between the dispersed phase and the continuous phase can be coupled. In practice, the discrete phase model is usually used, when the volume fraction of the particles in the computational domain is less than 10-12%. The equation for volume fraction of the dispersed phase has the following form:

$$\alpha_q = \frac{V_q}{V_c} \tag{2.4}$$

where V_q is the volume of this dispersed phase in a cell or domain, V_c is the volume of the cell or domain.

2.2.1 Air-Blast Atomizer Model

The realistic PUR-particles are generated from an atomizer. Air-blast atomizer is one of the complex injection types of spray in *ANSYS FLUENT*, which is the closest to the industrial atomizers for PUR. With physical parameters for the given atomizer, the initial conditions of particles such as size, velocity and position are calculated. The process of air-blast atomization requires an additional air stream to generate small drops from liquid sheets. (See Figure 2.1)

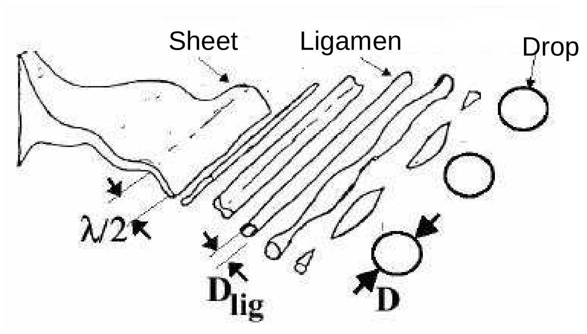


Figure 2.1: Sheet Breakup [16]

The following point properties for air-blast atomizer need to be specified in *ANSYS FLUENT* [2]:

- Position
  The x, y, and z (if 3D) positions of the injected stream in the Cartesian system
- Axis (only for 3D)
  The x, y, and z components of the vector defining the axis of the orifice
- Temperature
  The temperature of the streams
- Mass flow rate
  The mass flow rate of the streams in the atomizer
- Duration of injection
  The starting and ending time for the injection when the particle tracking is unsteady
- Inner diameter
  The inner diameter of the injector
- Outer diameter

The outer diameter of the injector, it determines the thickness of the liquid sheet with the inner diameter

- Spray angle
  The initial trajectory of the film as it leaves the end of the orifice
- Relative velocity
  The maximum relative velocity that is produced by the sheet and air
- Sheet breakup
  The value of the empirical constant that determines the length of the ligaments that are formed after sheet breakup
- Ligament diameter
  For short waves, the proportionality constant that linearly relates the ligament diameter to the wavelength that breaks up the sheet
- Dispersion angle
  It determines the initial velocities for a smooth distribution of the droplets.

The equations for these point parameters are described in [1]. For the PUR-spray model, the atomizer is moving along a Cartesian axis, thus requires a user-defined function. Details are described in Section 4.2.1.

2.2.2 Wall-Film Model Theory

In *ANSYS FLUENT*, wall-film model is a specific boundary condition for simulation of internal combustion engines. A typical application is modeling of the wall-film phenomena inside port fuel injected (PFI) and direct injection (DI) engines [1]. The particles within the discrete phase model (DPM) are used to model the wall-film. Hereby a single component liquid drop can impinge upon a boundary surface and form a thin film. The model is composed of four major subtopics: interaction during the initial impact with a wall boundary, subsequent tracking on surfaces, calculation of film variables, and coupling to the gas phase. Figure 2.2 shows the basic mechanisms for wall-film model.

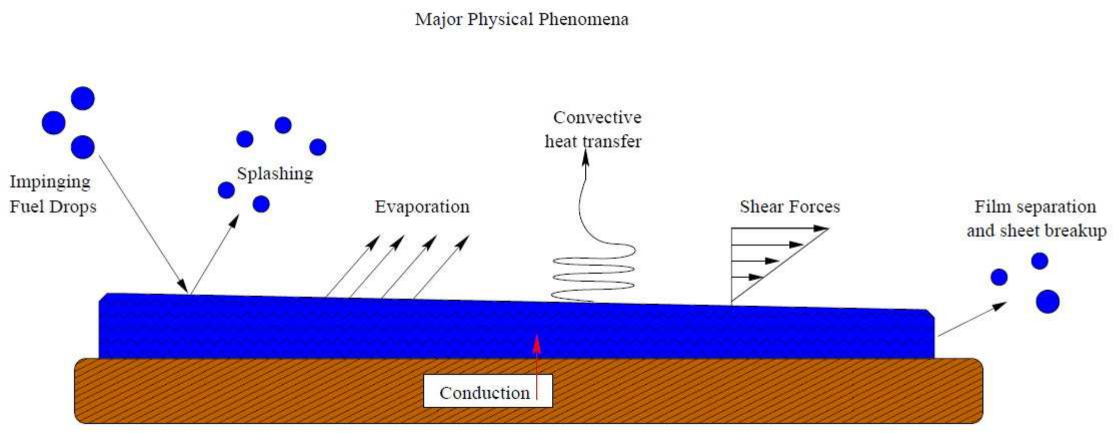


Figure 2.2: Mechanisms of Splashing, Momentum, Heat and Mass Transfer for the Wall-Film [1]

The wall interaction is based on the work of Stanton [3] and O'Rourke [4]. The outcomes of drop-film interactions include four regimes based on the impact energy and wall temperature: stick, rebound, spread and splash. (See Figure 2.3) Sticking droplets hit the wall film and stay on it. Rebounding droplets rebound off the wall with diminished momentum. Entire spreading droplets hit the wall at higher impact energies than sticking droplets. In the splashing regime, the incoming droplets produce secondary droplets. Below the boiling temperature of the liquid, the impinging droplet can stick, spread or splash, while above the boiling temperature, the particle can either rebound or splash.

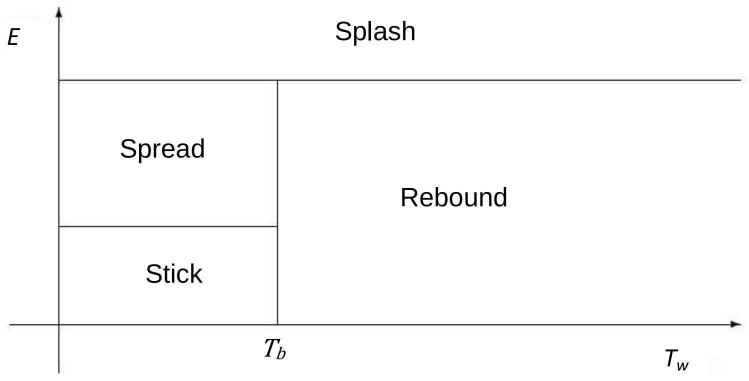


Figure 2.3: Simplified Decision Chart for Wall Interaction Criterion [1]

Conservation equations for momentum and mass of each parcel in the wall-film are described in [1]. This particle-based approach for thin films was first formulated by O'Rourke [5] and most of the derivation for the conservation equations is based on that work.

The equation for the momentum of a parcel on the film is

$$\rho h \frac{d\vec{u}_p}{dt} + h(\nabla_s p_f)_\alpha = \tau_g \vec{t}_g + \tau_w \vec{t}_w + \dot{\vec{P}}_{imp,\alpha} - \dot{M}_{imp,\alpha}\vec{u}_p + \dot{\vec{F}}_{n,\alpha} + \rho h(\vec{g} - \vec{a}_w) \tag{2.5}$$

where

$\frac{d\vec{u}_p}{dt}$	=	particle position over the time
α	=	index of the current face where particles reside
h	=	current film height at the particle location
∇_s	=	gradient operator restricted to the surface
p_f	=	pressure on the surface of the film
τ_g	=	magnitude of the shear stress of the gas flow on the film surface
$\vec{t}_g$	=	unit vector in the direction of the relative motion of the gas and the film surface
τ_w	=	magnitude of the stress that the wall exerts on the film
$\vec{t}_w$	=	unit vector in the direction of the relative motion of the wall and the film surface
$\dot{\vec{P}}_{imp,\alpha}$	=	impingement pressure on the film surface
$\dot{M}_{imp,\alpha}$	=	impingement momentum source
$\dot{\vec{F}}_{n,\alpha}$	=	force to keep the film on the surface
$\rho h(\vec{g} - \vec{a}_w)$	=	body force term

The detailed equations for these variables can be found in [1].

The mass transition is based on the film vaporization law. In this study, the temperature of the wall film does not reach T_b, therefore, the mass transfer is not considered.

In this study, wall-film model was used to model the PUR-liquid film on the sprayed wall for the boundary condition of DPM model. It was then compared with the solution of VOF model. The explanation for VOF model is described in Section 2.3.1.

2.3 Multiphase Flows

A large number of flows in nature and technology are called mixture of phases. Multiphase flow regimes can be grouped into four categories: gas-liquid or liquid-liquid flows, gas-solid flows, liquid-solid flows, and three-phase flows. The fluid system is defined by a primary and multiple secondary phases. Primary phase is considered to be a continuous phase such as air and secondary phase is considered as a dispersed phase within the continuous phase. (See Figure 2.4) PUR-spray model is modeled as Euler-Lagrange-two-phase flows: the primary phase is air, and the PUR-droplets are defined as the secondary phase regarded as particles. Particles deposing on the substrate are transferred via source terms to the Volume Of Fluid (VOF) model.

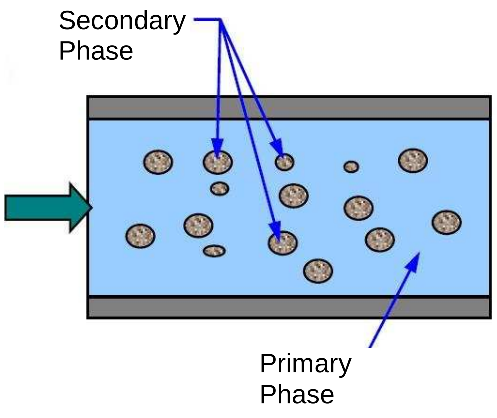


Figure 2.4: Primary Phase and Secondary Phase in a Domain [6]

2.3.1 Volume of Fluid (VOF) Model Theory

The simulation of layer structure of PUR-spray was carried out by using *ANSYS FLUENT*'s volume of fluid (VOF) model, of which the method is based on a concept of a fractional volume of fluid, as previously proposed by Hirt and Nichols [7]. The VOF method, one of the Euler-Euler multiphase models, is used to capture the interface of two or more immiscible fluids. Except tracking liquid-gas interface, VOF model is typically used for the prediction of jet breakup, the motion of large bubbles in a liquid, the motion of liquid after a dam break.

In VOF model, the governing equations are solved by using the volume fraction in each cell. In each control volume, the volume fractions of all phases α_k sum to unity.

$$\sum_{k=1}^{n} \alpha_k = 1 \qquad (2.6)$$

If the volume fraction of the q^{th} phase in the cell is denoted as α_q, then the following three conditions are possible:

- $\alpha_q = 0$: This cell is empty of the q^{th} phase.
- $0 < \alpha_q < 1$: This cell represents the interface region between the q^{th} phase and one or more other phases.
- $\alpha_q = 1$: This cell is full of the q^{th} phase.

The fields for all variables and properties are shared by all the phases and represent volume-averaged values in any given cell depending upon the volume fraction value. For example, the volume-fraction-averaged density of an n-phase system takes on the following form:

$$\rho = \sum \alpha_q \rho_q \tag{2.7}$$

where α_q is the tracked volume fraction and ρ_q is the density of its phase. All other properties (e.g., volume-fraction-averaged viscosity) are computed in the same manner.

The interface(s) between the phases is tracked by the solution of a continuity equation for the volume fraction of one (or more) of phases. For the q^{th} phase, this equation has the following form:

$$\frac{\partial \alpha_q}{\partial t} + \nabla \cdot (\alpha_q \vec{v}_q) = \frac{1}{\rho_q}\left[S_{\alpha_q} + \sum_{p=1}^{n} (\dot{m}_{pq} - \dot{m}_{qp})\right] \tag{2.8}$$

where $\dot{m}_{pq}$ is the mass transfer from the p^{th} phase to the q^{th} phase and $\dot{m}_{qp}$ is the mass transfer from the q^{th} phase to the p^{th} phase. S_{α_q} is the source term which is zero by default.

The volume fraction equation may be solved through either implicit or explicit time discretization. For implicit scheme:

$$\frac{\alpha_q^{n+1}\rho_q^{n+1} - \alpha_q^n \rho_q^n}{\Delta t} V + \sum_f (\rho_q^{n+1} U_f^{n+1} \alpha_{q,f}^{n+1}) = \left[S_{\alpha_q} + \sum_{p=1}^{n} (\dot{m}_{pq} - \dot{m}_{qp})\right] V \tag{2.9}$$

This equation requires the volume fraction value at the current time step. Thus, a standard scalar transport equation is solved iteratively for each of secondary-phase volume fractions at each time step.

In the explicit approach, the standard finite-difference interpolation schemes are applied to the volume fraction values which were computed at the previous time step:

$$\frac{\alpha_q^{n+1}\rho_q^{n+1} - \alpha_q^n \rho_q^n}{\Delta t} V + \sum_f (\rho_q U_f^{n} \alpha_{q,f}^{n}) = \left[S_{\alpha_q} + \sum_{p=1}^{n} (\dot{m}_{pq} - \dot{m}_{qp})\right] V \tag{2.10}$$

where	$n+1$	=	index for current time step
	n	=	index for previous time step

$\alpha_{q,f}$	=	face value of the q^{th} volume fraction
V	=	volume of cell
U_f	=	volume flux through the face, based on normal velocity

One of the techniques to calculate the face fluxes of volume fraction is geometric reconstruction scheme, generalized for unstructured meshes from the work of Youngs [8]. It assumes that the interface between two fluids has a linear slope within each cell, and uses this linear shape for calculation of the advection of fluid through the cell faces. Another is Donor-Acceptor Scheme, used to determine the amount of fluid flowing through all the faces of a cell [7]. (See Figure 2.5)

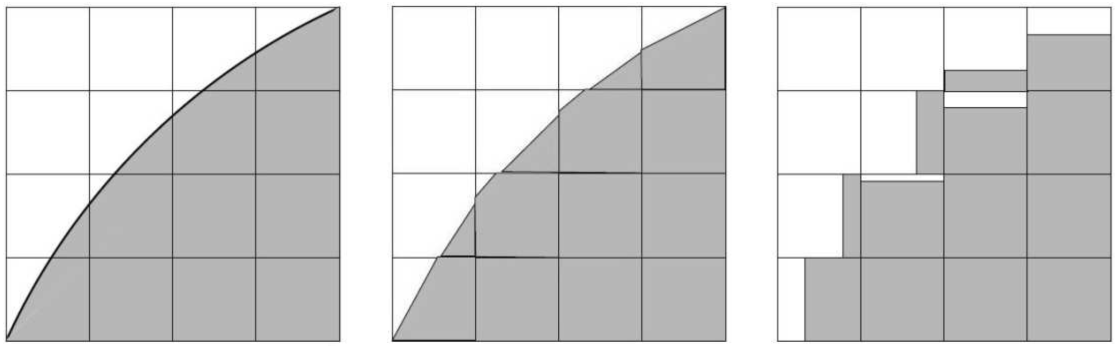

actual interface shape
interface shape represented by the geometric reconstruction scheme
interface shape represented by the donor acceptor scheme

Figure 2.5: Interface Calculations [1]

The high viscosity ratio between air-phase and PUR-phase (ca. 10^7) may lead to convergence difficulties. As a high resolution differencing scheme, the compressive interface capturing scheme for arbitrary meshes (CICSAM), based on Ubbink's work [9] was used for solving the problem of poor convergence.

A single momentum equation is solved throughout the domain, and the obtained velocity field is shared among the phases:

$$\frac{\partial}{\partial t}(\rho\vec{v}) + \nabla\cdot(\rho\vec{v}\vec{v}) = -\nabla p + \nabla\cdot[\mu(\nabla\vec{v} + \nabla\vec{v}^T)] + \rho\vec{g} + \vec{F} \tag{2.11}$$

The momentum equation is dependent on the volume fractions of all phases through ρ and μ, the density and the viscosity.

The energy equation, also shared among the phases, is shown below:

$$\frac{\partial}{\partial t}(\rho E) + \nabla\cdot(\vec{v}(\rho E + p)) = \nabla\cdot(k_{eff}\nabla T) + S_h \tag{2.12}$$

where S_h is the volumetric heat source. The properties ρ and k_{eff} (effective thermal conductivity) are shared by the phases.

The energy E and temperature T are treated as mass-averaged variables by VOF model:

$$E = \sum_{q=1}^{n} \alpha_q \rho_q E_q \,/ \sum_{q=1}^{n} \alpha_q \rho_q \tag{2.13}$$

where E_q is based on the specific heat of the q^{th} phase and the shared temperature. However, heat transfer is neglected in PUR-spray model.

Otherwise, surface tension and wall adhesion effects can be taken into account in VOF model.

3. User-Defined Function (UDF)

In *ANSYS FLUENT*, we can program a user-defined function (UDF) to enhance the standard features of the code. UDFs can be used to define boundary conditions, material properties or source terms of flow regime, and initialize the solution, or enhance post-processing as well. They are also necessary for fitting the particular model needs (e.g. DPM, multiphase models). UDFs are written in C programming language and the source code is stored with X.c extension.

3.1 Mesh Terminology

Most UDFs access data from an *ANSYS FLUENT* solver, of which data is defined in terms of mesh components. Before writing a UDF, some basic mesh terminologies need to be learnt. Figure 3.1 schematically shows the mesh components.

- domain
  grouping of node, face, cell threads in a mesh
- node/face/cell thread
  grouping of nodes/faces/cells
- cell
  A mesh is broken up into control volumes, and this control volume is called as cell.
- cell center
  location where cell data is stored
- face
  boundary of a cell
- edge
  boundary of a face
- node
  mesh point

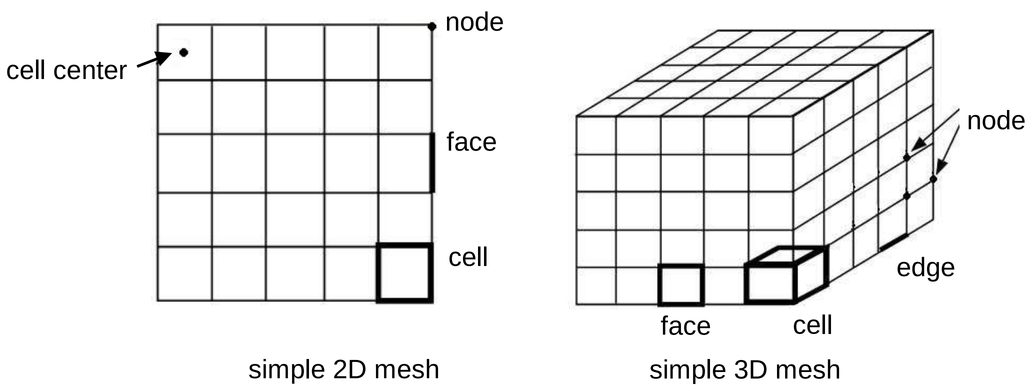


Figure 3.1: Mesh Components [10]

3.2 DEFINE Macros and Additional Macros

DEFINE macros predefined macros provided by *ANSYS FLUENT*, Inc. must be used to define UDFs for specific purposes. Definitions for DEFINE macros are contained within the udf.h file. In this study, some general purpose DEFINE macros and DPM DEFINE macros were used. They are discussed in Chapter 4.

ANSYS FLUENT provides the predefined additional macros to facilitate the programming of UDFs and the use of CFD objects as defined inside it. Data access macros can be used to obtain or specify some information, e.g. face area of a face, volume of a cell, adjacent cell thread, velocity of a particle and so on. A listing and discussion of each macro is presented in [10]. In this study, additional macros for mesh component and particles are frequently utilized. In addition, a set of predefined looping macros are also required in UDFs when performing repeated operations. For example, in order to track the particles which reside on the wall-boundary, each face on this boundary should be visited so that face looping for this boundary is required. The detailed description of these macros with the specific purposes for the study can be found in Chapter 4.

3.3 Parallel Processing

In order to increase the running speed for complex *ANSYS FLUENT*'s models, parallel processing is needed. Multiple Processes are essential when using *ANSYS FLUENT*'s parallel solver and execute on the same computer, or on different computers in a Network. (Figure 3.2)

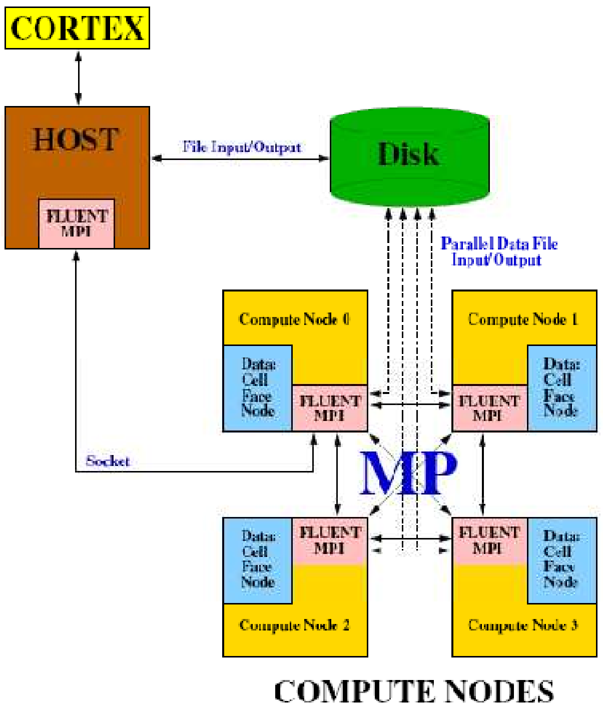


Figure 3.2: Parallel *ANSYS FLUENT* Architecture [2]

The mesh model is separated into a set of parts, and the solution of each part is computed by a parallel process or computer node. A host process does not contain mesh data (cell, face, and node data), but parallel processes do. Instead, it only interprets commands from *ANSYS FLUENT*'s graphics-related interface, cortex. Thus, the mesh data must be defined only for computer nodes in UDFs.

In short, we should pay attention to following points when using parallel processing:

- Suitable mesh partitioning and load balancing are necessary.
- When DPM model works with the multiphase flows models (e.g. VOF model), **Message Passing** option is automatic enabled.
- Mesh data are defined only for computer processes or nodes.
- Looping macro for interior cells is used to avoid repeating cell looping. For the same reason, the specific macro for face looping is also used.
- The definition of global variables for parallel process is more complex than for serial process.
- The transfer and reduction of data between host process and computer processes is important.
- UDFs are specified for both parallel and serial version.

4. Model

This chapter describes the mesh modeling, the air-particle-spray modeling and layer structure modeling (VOF, wall-film) using the preprocessor *Gambit* and the solver *ANSYS FLUENT*.

4.1 Mesh

An appropriate mesh for the model can not only calculate accurately but also save the running time. Thus, an appropriate mesh is modeled within the *Gambit* environment.

4.1.1 Objectives of Mesh

In order to track the position of transient interface between two fluids (air as primary phase and PUR as secondary phase throughout the domain), the structure of PUR-layer is modeled using *ANSYS FLUENT*'s Volume of Fluid (VOF) model by solving a single set of momentum equations and tracking the volume fraction of PUR-particles and air. In the VOF model, a very fine mesh is important for an accurate solution due to the expected, tiny PUR-layer on the substrate. However, only the interface of two fluids on the plate needs to be tracked, so the grids far away from the plate need not to be as fine as the grids nearby. Considering the problem above, two kinds of models were created (See Figure 4.1 and Figure 4.2):

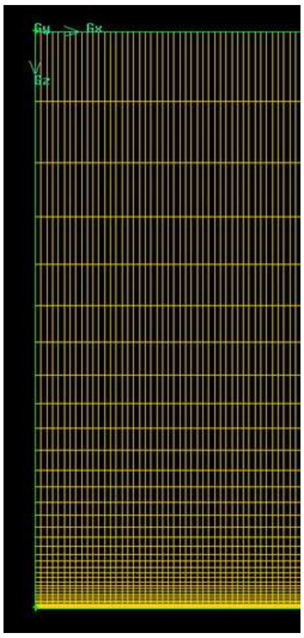


Figure 4.1: From-Top-To-Layer-Finer-Meshed Model

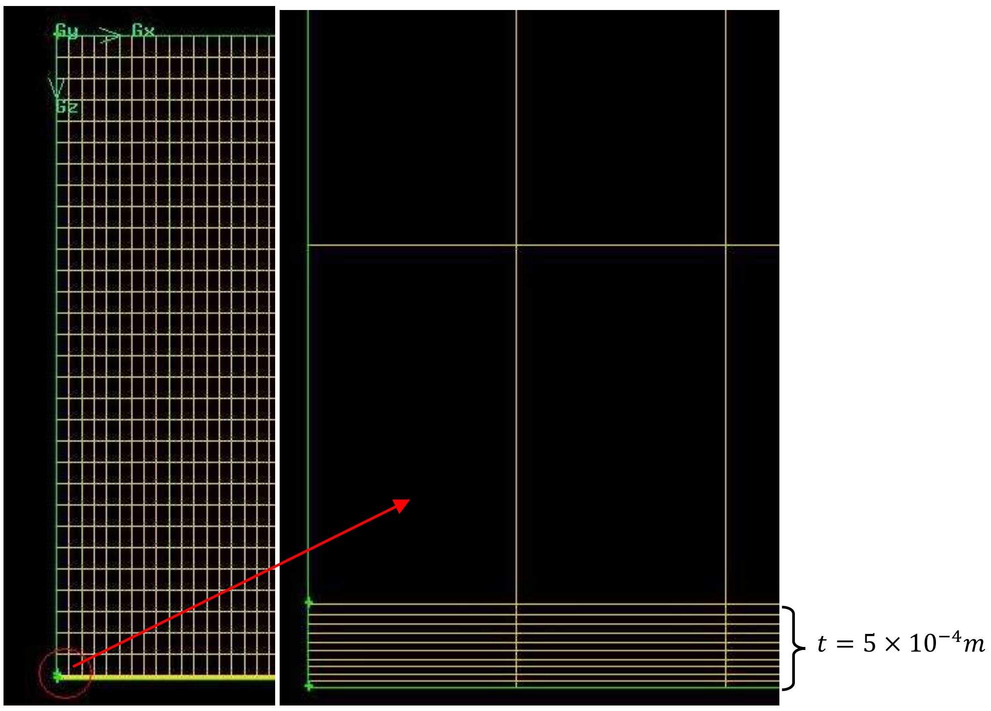


Figure 4.2: Near-Layer-Finer-Meshed Model

Figure 4.1 and Figure 4.2 show the meshed model viewed in the direction of x-z plane. The line located on the bottom of the model is the spray-layer. The models are the rectangular container considered as the surroundings of the PUR-spray process. The mesh showed in Figure 4.1 is refined from top to layer, while in Figure 4.2 only the grid near layer ($t = 5 \times 10^{-4} m$) is refined.

The unique advantages and disadvantages of two grid methods are listed in Table 4.1:

	Advantages	Disadvantages
From-Top-To-Layer-Finer-Meshed Model	1. The simulation is stabilized with the grids gradually finer in spray direction.	1. The volume of each grid is different, which means the sizes of injection cells at different positions are not same. This leads to inconsistent experimental conditions. 2. The prohibitive number of grid cells cost more running time.
Near-Layer-Finer-Meshed Model	1. There are fewer cells in this case because finer grids are only built near the bottom. 2. The sizes of injection cells at different positions (e.g. distance between plate and injection is $0.2m, 0.4m$, or $0.6m$) are same.	1. The ratio of grid changes rapidly, which may lead to the unsteady simulation, e.g. Courant number increases unexpectedly.

Table 4.1: Comparison of Meshes between Two Models

Additionally, the size of grids is important for the post-processing of simulated results. In order to compare the distribution of layer weight obtained from *ANSYS FLUENT*'s models and practical experiments, the size of each sample cut from the layer by experiment should be the integer multiple size of grids. Figure 4.3 shows the cutting tool used in experiment, the size of each part cut from the sprayed layer is $11mm \times 75mm$. It cuts the layer into a group of samples and these samples are weighted to obtain the distribution of weight on the layer.

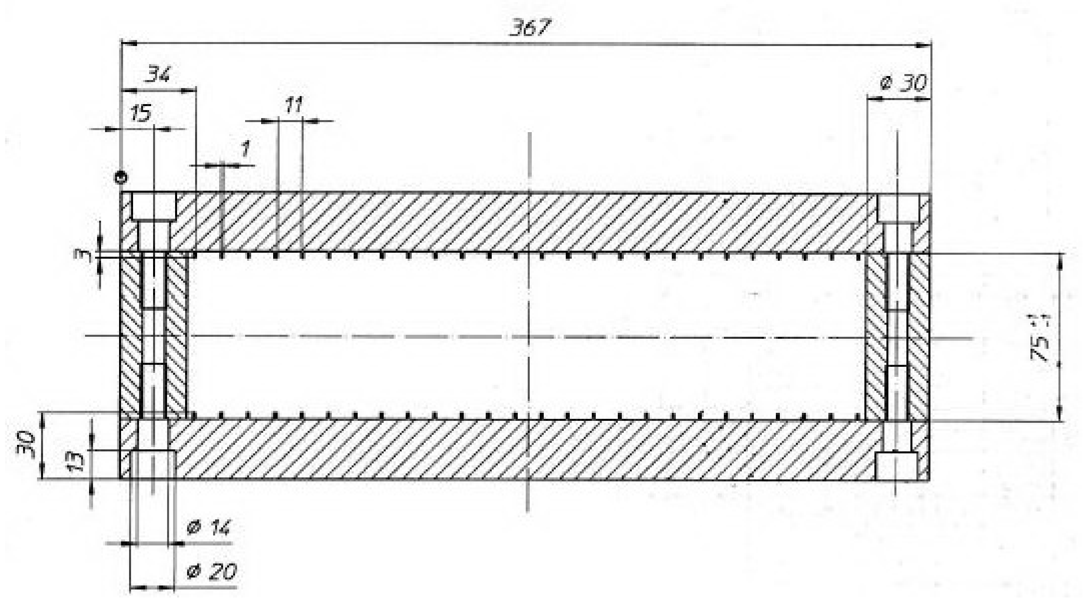


Figure 4.3: Conduction-Pusher for Cutting the Layer [11]

Furthermore, the working environment including the plate of PUR-spray is modeled as a rectangular container. It should be larger than the space that spray particles may pass through, so that except the sprayed plate, the other five boundaries of container will not affect the spray process.

Therefore, a simply container with meshes and boundary conditions created in *Gambit* includes following points:

- The grids are finer from the plane on the top of the container to the spray plate.
- Change of the size of grids should be gentle.
- The size of grids is based on the size of the conduction-pusher's sample.
- Size of the container is larger than the spray space.

4.1.2 Final Mesh Model

Geometry:

The model created in *Gambit* is a rectangular container, of which size is:

$$Length_x \times Width_y \times Height_z = 1.5m \times 0.561m \times 0.65m$$

X-axis direction is considered as the trajectory of the moving PUR-spray injector and the bottom of the container is the sprayed layer (see Figure 4.4 and Figure 4.5).

Coordinate System:

As Figure 4.4 shows, the Cartesian coordinate origin is located in the middle of the short edge of the plane on the top of the container. Positive z direction is defined as the spray direction.

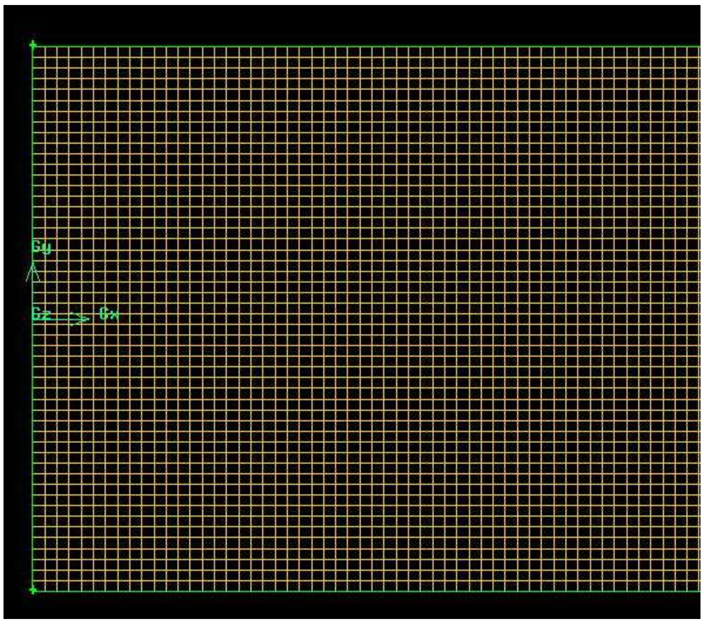


Figure 4.4: The Final Scheme of the Meshed Model (Viewed in Z Direction)

Mesh:

As described above, the model is optimized by combining the advantages of two methods and avoiding the disadvantage. The gentle ratios of grids are necessary for the simulation. Thus, the optimization will be described as following: (see Figure 4.5)

- On the basis of near-layer-finer-meshed model, the height is divided into two parts, one is from $z = 0m$ to $z = 0.5m$, the other is from $z = 0.5m$ to $z = 0.65m$.
- For the upper part ($z = 0 \rightsquigarrow 0.5m$), the successive ratio is 1 and the number of grids is 20, the length of grids is $0.025m$. We obtain equally spaced cells.
- For the lower part ($z = 0.5 \rightsquigarrow 0.65m$), in order to get the gentle changing of ratio, the first length of grids is defined as same as the length of grids for the upper part. And the number of grids is also 20 to ensure the length of grids near the layer is fine enough.

The terminology and performing of mesh can be found in [12]. The details of settings for mesh described above are listed in the following Table 4.2:

Edge / Parameters	Length (m)	Mesh Length (m)	Number of Grids	Ratio
x positive	1.5	0.0125	120	1
y positive	0.561	0.011	51	1
z positive	$0 \rightsquigarrow 0.5$	0.025	20	1
z positive	$0.5 \rightsquigarrow 0.65$	first length: 0.025	20	

Table 4.2: Meshes Specified in *Gambit*

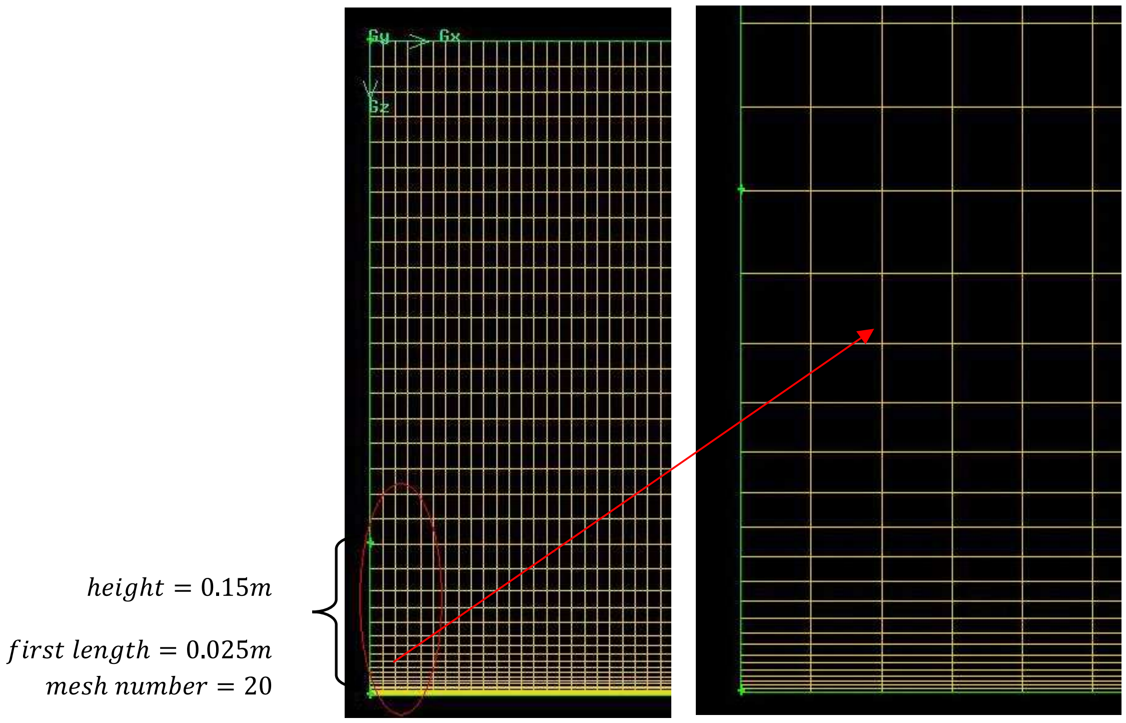


Figure 4.5: Final Scheme of the Mesh Model (Viewed in Y Direction)

Boundary conditions:

The boundary condition of sprayed plate is defined as "**Wall**", while the other five planes of the container are defined as "**Pressure Outlet**".

The key point in this study is that the VOF model for the plate requires fine meshes, and the other area can be consists of a normal and rough mesh. The solving time would be shortened by this method of mesh.

4.2 Model of PUR-Spray

In this study, the air-PUR spray injector was specified as an air-blast atomizer. It moves along a direction with a fixed speed. Under the influence of impulse generated by the extern air into the injector, particle dispersion occurs by using *ANSYS FLUENT*'s Discrete Phase Models (DPM).

Usually when modeling the particles sprayed from air-blast atomizer, a nozzle with specific shape (see Figure 4.6) is created, and then defined as Air-Blast Atomizer in *ANSYS FLUENT* (See tutorial 17 in [13]). But an air-blast atomizer created with specific shape cannot work as a moving spray injector because of the restrictions on the definition in the *ANSYS FLUENT*. Therefore, another method was used to model the PUR-spray process.

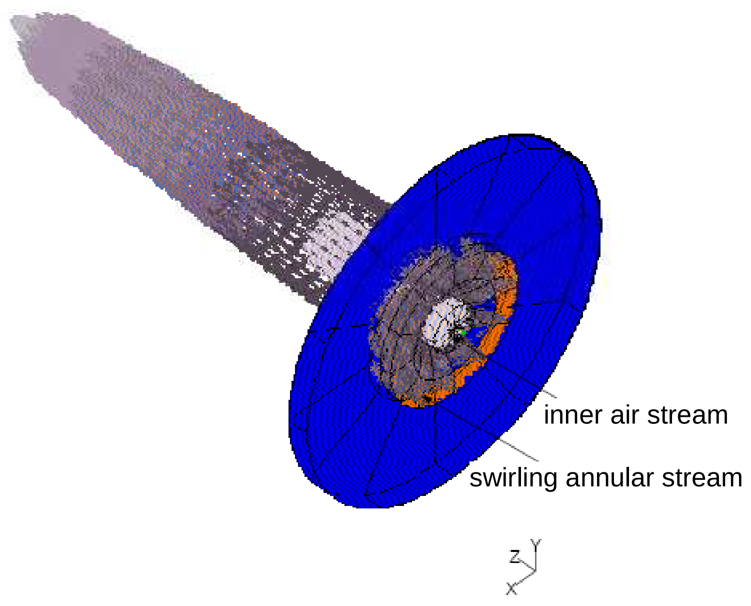


Figure 4.6: Nozzle of Air-Blast Atomizer [13]

Due to small nozzle (diameter: $5mm \rightsquigarrow 7mm$), a cell was regarded as a spray injector. In the following description, it is called as injection cell. Using the defined injection cell, *ANSYS FLUENT* can solve the problem of the moving PUR-spray injector with extern air source. This issue involves three specific aspects:

- How to move PUR-spray injector along a certain direction?
- How to define the extern air source at PUR-spray injector?
- How to control the impulse of this air source?

4.2.1 Moving Spray Injector

As mentioned above, the spray injector was modeled using *ANSYS FLUENT*'s air-blast atomizer model. For an air-blast atomizer, the point properties needed to be specified are: position, axis (if 3D), temperature, mass flow rate, duration of injection (if unsteady), injector inner diameter, injector outer diameter, spray angle, relative velocity, sheet breakup, ligament diameter, azimuthal angles (if relevant) and dispersion angle. However, with these parameters the injector is still unable to move. In this case, user-defined functions (UDFs) are useful for this specific application.

When the injector moves, the particles at injector move at the same speed simultaneously, which means we can define the position of the injector by specifying the position of the particles at injector indirectly. In UDFs, the macro called P_INIT_POS(p)[i], returns the position of the particles at injection. Expressed in mathematical language, a moving injector implies that the particles at injection shift from one position to another position with the distance returned by the product of the speed and the flow time. Or in other words, the current position of the particles at injection changes with a certain rate over solving time. Suppose the starting position locates at the point $(0,0,0)$ in Cartesian coordinate system and moves along x positive direction, the macro called P_POS0(p)[i] can be compiled in an appropriate macro function as:

P_INIT_POS(p)[0] = C * CURRENT_TIME;

where C is the speed of injection, CURRENT_TIME is a time-dependent macro which returns the current flow time (in seconds). The argument type p is the tracked particles, i = 0 means it is in x direction, corresponding 1 and 2 mean in y and z direction.

There comes the next problem with the selection of macro function, which will be used to define the injector with the command of P_INIT_POS(p)[i]. The DEFINE macro for DPM model, DEFINE_DPM_INJECTION_INIT, has a purpose to initialize a particle's injection properties such as location, diameter, and velocity. Table 4.3 below lists the usages of it:

DEFINE_DPM_INJECTION_INIT(name,I)		
Argument Type	Description	Function returns
symbol name	UDF name.	void
Injection *I	Pointer to the Injection structure which is a container for the particles being created. This function is called twice for each Injection before the first DPM iteration, and then called once for each Injection before the particles are injected into the domain at each subsequent DPM iteration.	

Table 4.3: Usage of DEFINE_DPM_INJECTION_INIT

During the first few attempts using DFEINE_DPM_INIT_INJECTION with the command for P_INIT_POS(p)[i], the problem was identified that the injector didn't move actually because the particles stayed at the starting position. The position of particles at current position (see Figure 4.7) should also be defined, although the initial position of particles has been defined. The macro for current position of particle is P_POS(p)[i]. In *ANSYS FLUENT*, when looping over the particles of the injection, P_POS(p)[i] returns the position a little bit below the nozzle.

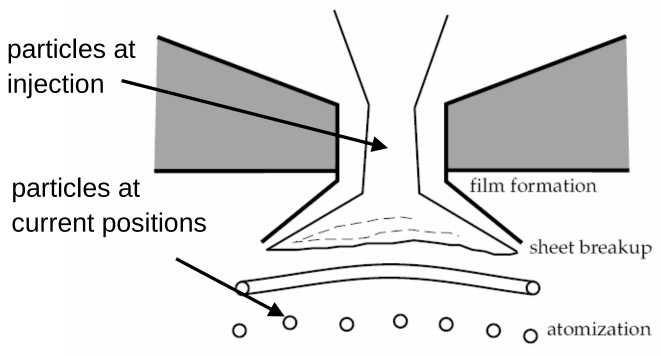
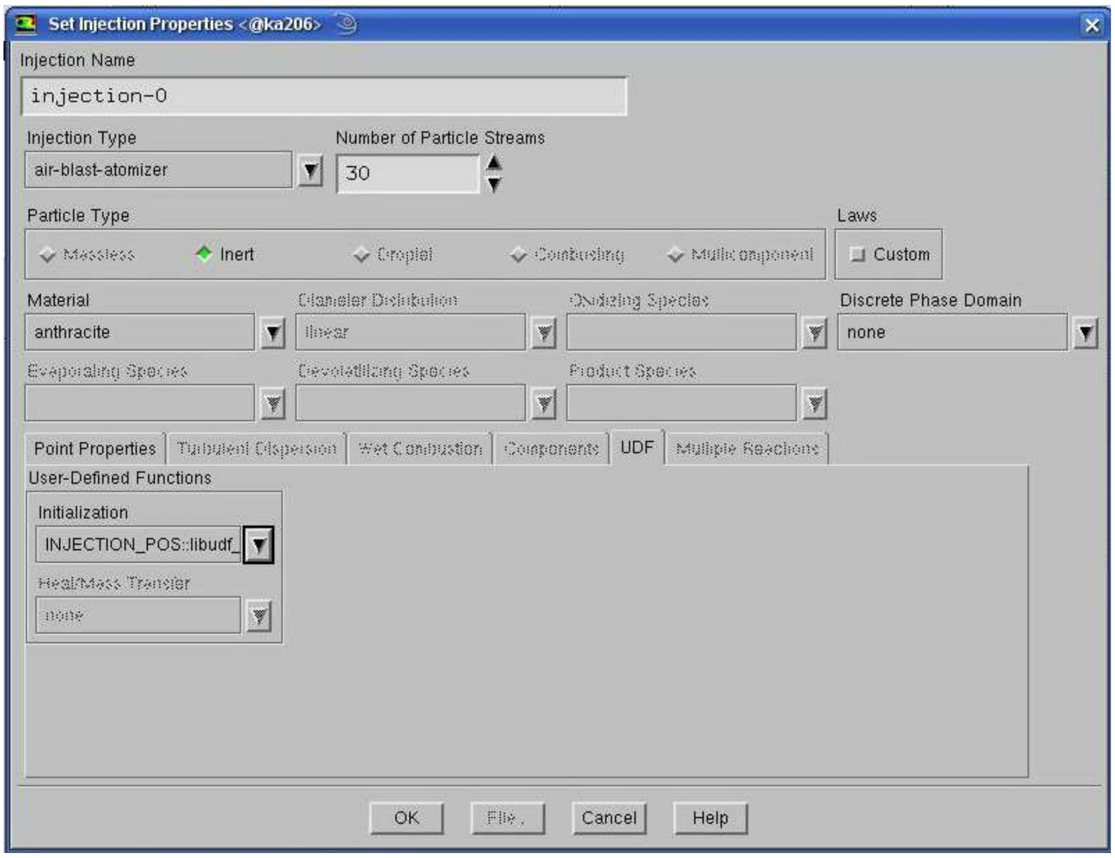


Figure 4.7: Position of the Particles at Different Flow Time

Therefore, for a moving injector we can hook the UDF as the following simplified function into the **Set Injections properties** box showed in Figure 4.8:

```
DEFINE_DPM_INJECTION_INIT(INJECTION_POS,I)
{
   Particle *p;
   loop(p,I->p_init)
   {
      P_INIT_POS(p)[0] = 0.5 * CURRENT_TIME;   /* 0.5 is the velocity of injection along x
                                                   direction, in m/s */
      P_POS(p)[0] = 0.5 * CURRENT_TIME;
   }
}
```

where loop(p,I->p_init) loops over the particles at or around the injector. It is used for transient particles, while loop(p,I->p) is for steady simulation.


Set Injection Properties <@ka206>

Injection Name: injection-0

Injection Type: air-blast-atomizer

Number of Particle Streams: 30

Particle Type: Massless [✓ Inert] Droplet Combusting Multicomponent

Laws: Custom

Material: anthracite

Diameter Distribution: linear

Oxidizing Species:

Discrete Phase Domain: none

Evaporating Species:

Devolatilizing Species:

Product Species:

User-Defined Functions

Initialization: INJECTION_POS::libudf_

Heat/Mass Transfer: none

Figure 4.8: Dialog Box of Set Injection Properties

In addition, the velocity of particles in the discrete direction needs to be defined as initial velocity, and it will also be compiled in DFEINE_DPM_INIT_INJECTION. For 3D model, the magnitude of velocity of particles at current position is:

$$v_p = \sqrt{v_{p,x}^2 + v_{p,y}^2 + v_{p,z}^2} \tag{4.1}$$

where $v_{p,x}$, $v_{p,y}$, $v_{p,z}$ are the velocity of particles (parcels) in x, y, z direction. The required initial magnitude velocity of particles is equal to $13 m/s$ [11] and constant for all initial particles. But the vector of each particle is not the same value because the spray half angle is defined for the injection. No equations, which explain the relationships between the velocity of particles (parcels) and the properties of spray injection, have been found in [1]. Thence, we can at first read the default initial magnitude velocity resulted by the pre-set properties of spray injection, and then divide the maximal velocity value to the required velocity to get our correction factor, and then multiply this factor to each velocity component of each particle (parcel). At the end, the required initial magnitude velocity of particles (parcels) is specified. Through the following simplified UDF, this idea will be clearly declared:

```
DEFINE_DPM_INJECTION_INIT(INJECTION_VEL,I)
{
  Particle *p;
  loop(p,I->p_init)
  {
    real p_vel_mag;
    p_vel_mag = sqrt( SQR(P_VEL(p)[0]) + SQR(P_VEL(p)[1]) + SQR(P_VEL(p)[2]) );
    /* Read default initial magnitude velocity of particles with P_VEL(p)[i] */
    /* P_VEL(p)[i] is the macro for the velocity of particles (parcels) at current position. */
    /* SQR(k) returns the square of the given variable k, or k * k. */

    real rate = 13 / p_vel_mag ;
    /* 13 is the required initial magnitude velocity of particles (parcels) in m/s */

    int i;
    for(i=0;i<3;i++)
      P_VEL(p)[i] = rate * P_VEL(p)[i] ;
    /* Define the velocity component by multiplying the rate to the default velocity. */
  }
}
```

However, DEFINE_DPM_INJECTION_INIT(INJECTION_POS,I) and DEFINE_DPM_INJECTION_INIT(INJECTION_VEL,I) were combined into one function. The complete UDFs about it can be found in A.3.

4.2.2 Air Source at Spray Injector

Application of air source for the spray injector is one of the key points in this study. It is an important definition for particle dispersion. With the impulse of extern air, the diameter of particles decreases from the nozzle to the sprayed layer.

The volume stream of extern air source has been given as $300L/min$ [11]. Through the following equations, we can get the mass stream and velocity of it:

$$\dot{m} = \rho\dot{V} \tag{4.2}$$

$$v = \dot{V}/A \tag{4.3}$$

where $\dot{m}$ is mass steam of the extern air source in kg/s, $\dot{V}$ is volume stream in m^3/s, ρ is the density of air as $1.225kg/m^3$, v is the impulse velocity generated by air source, whose direction is the same as the face area vector of the face whose area is A. The face in this case is the surface of the nozzle.

As Figure 4.9 shows, air source is modeled by setting of volumetric mass source of $\frac{\dot{m}}{V} = \frac{\rho_j A_j v_j}{V}$ in $kg/m^3 s$ and momentum source of $\frac{\dot{m}v}{V} = \frac{\dot{m}v_j}{V}$ in N/m^3 as compiled in function macro DEFINE_SOURCE, where V is the cell volume. These source terms are added to the injection cell, where the moving injector currently locates.

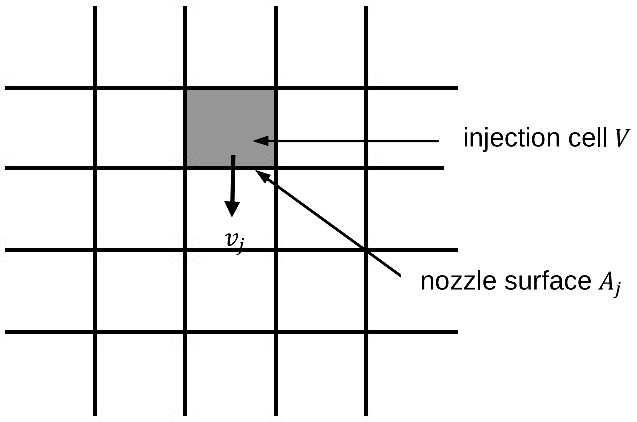


Figure 4.9: Defining a Source for a Tiny Inlet

Note that if only mass source is defined for a cell, the mass will enter the domain with no momentum or thermal heat. The mass of air transpires therefore in all the direction of the cell. Thus, a volumetric momentum source is necessary to be defined.

The theoretical velocity of air source calculated through Equation 4.3 with diameter of nozzle ($5mm$) is about $150m/s$, and mass stream is about $6.125 \times 10^{-3} kg/m^3 s$. Anyway, with both of the parameters, the solution of ANSYS FLUENT shows, that the maximum cell-based velocity is much smaller than the velocity by hand calculation ($150m/s$). It is hard to find out the reason of this deviation by *ANSYS FLUENT*. In order to obtain the precise velocity generated by air source, a UDF used to achieve this purpose should be compiled. The main idea of it is to increase the velocity (v_j) defined in momentum source ($\frac{\dot{m}v}{V} = \frac{\dot{m}v_j}{V}$) every a few iterations until the velocity showed as the cell-based velocity (v_{FLU}) in the next cell of the source cell in the direction of the momentum source reaches the hand calculated value (v_r). This correction is executed under steady simulation, and it needs many iterations (about 1000), even though the whole solution is transient and it needs much fewer iterations (about 20) for every time step. 20 iterations in every narrow time step are enough for getting the precise solution and more iterations cost more solving time. *ANSYS FLUENT* cannot control the changing of iteration-number in different time step. Therefore, the solution of searching the required velocity and the solution of VOF Model must be separated into two parts, one is by steady with many iterations, and the other is by transient with fewer iterations in every time step.

The UDFs for this purpose includes following basic macro functions:

- one DEFINE_SOURCE for air mass source in injection cell,

- one DEFINE_SOURCE for air momentum source in spray direction,
- and one DEFINE_ADJUST for correction of the air velocity.

Both mass source and momentum source of air must be defined for the same cell where the injection point locates, since the different cell volume in the container model previously described leads to different result of velocity. The UDF defined for spray injector must be combined with these UDFs to find the source cell not only for spray injector but also for air source by using the extern variables of injection position. Although the injector moves along a direction, or further to say, it stays at different cell from time to time, it does not affect the correction of the velocity solved by steady and related with the cell volume, owing to the equal cell volume in the no-spray direction.

DEFINE_ADJUST executes at each iteration and is called at the beginning of every iteration before transport equations are solved and it can also be used to modify the flow variables includes velocity. The usage of this macro is listed in Table 4.4. It is still an appropriate function for correction of velocity even if the solution must be separated into two parts. Some points are sequential involved in DEFINE_ADJUST,

- Find out the injection cell,
- Find out the neighbor cell of injection cell in the spray direction,
- Correct the cell-based velocity of this neighbor cell.

DEFINE ADJUST(name,d)		
Argument Type	Description	Function returns
symbol name	UDF name.	void
Domain *d	Pointer to the domain over which the adjust function is to be applied. The domain argument provides access to all cell and face threads in the mesh. For multiphase flows, the pointer that is passed to the function by the solver is the mixture-level domain.	

Table 4.4: Usage of DEFINE_ADJUST

<u>Find out the injection cell:</u>

As previously described, injection cell is the cell where the injector locates. Using command extern can import the injection position from the UDF defined for spray injector to find out the injection cell. After extraction of the injection position, the injection cell can be found using the cxboolean macro SV_is_point_in_cell predefined in dpm.h. Then this injection cell will be marked with user-defined memory (UDM) macro. C_UDMI can be used to store a value like 2.0 (except 0.0) in this injection cell and the value of C_UDMI in the other cells without definition is a default value 0.0. A simple example is showed as following:

```
if(SV_is_point_in_cell(&cx_cell,c,t,inj_init))
/* inj_init is the extern variable of injecton position */
{
   C_UDMI(c,t,0) = 2.0;
   /* Mark the injection cell by storing it as value 2.0 */
}
```

Later, this marked cell is used as a condition to define the air source in the UDF for correction of velocity. In another word, if the visited cell is the marked cell, UDFs add the volumetric source items into it or, if not, nothing happens to this cell.

Find out the neighbor cell of injection cell in the spray direction:

In Section 4.1.2, the spray direction has been described and it sprays along z-positive direction. So this neighbor cell is just the next cell of injection cell in z-positive direction. (See Figure 4.10) In the following description, it will be named as z-neighbor cell.

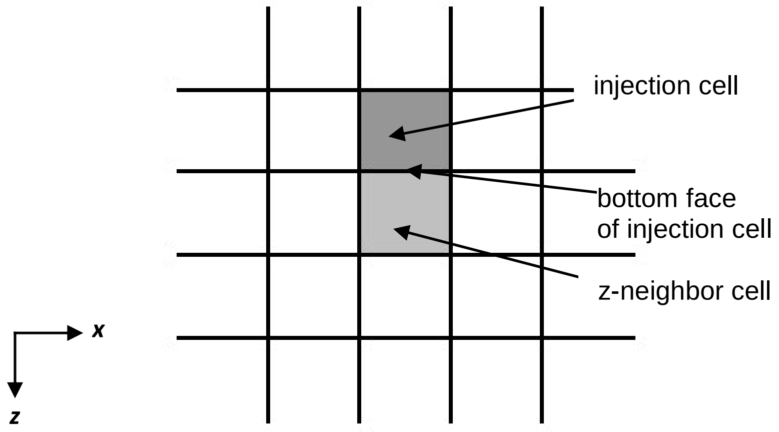


Figure 4.10: Injection Cell and Z-Neighbor Cell

Searching z-neighbor cell is not as direct as searching injection cell because a cubical cell contains six neighbor cells. Z-neighbor cell satisfies two conditions: 1. The absolute value of z component of the face area vector (A) of the injection cell is greater than zero. As Figure 4.11 shows, A is the area normal vector of a face directed from adjacent cell c0 to c1. This condition filters two faces whose face area vector is parallel to z axis from six faces of the injection cell. 2. z component of face centroid of the injection cell is greater than the cell centroid of the injection cell. This condition further filters the bottom face (see Figure 4.10) from those two faces. On either side of the bottom face there are two cells, one is injection cell and the other is the z-neighbor cell. If A of bottom face directs from injection cell to z-neighbor cell, it means that injection cell is c0 and z-neighbor cell is c1. After obtaining the z-neighbor cell, the cell-based velocity of it (v_{FLU}) can be immediately solved.

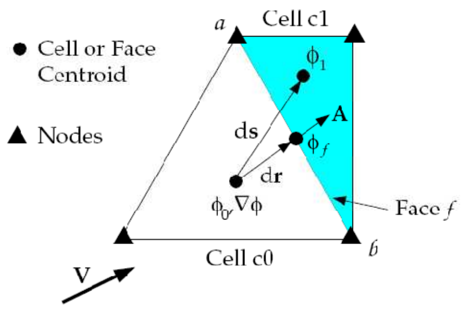


Figure 4.11: Adjacent Cells c0 and c1 with Vector and Gradient Definitions [10]

Correction of cell-based velocity of this neighbor cell:

The velocity (v_j) defined in momentum source ($\frac{\dot{m}v}{V} = \frac{\dot{m}v_j}{V}$) will be increased by a fixed dimension every few iterations (about 10iterations). With increasing of v_j, momentum source changes and the cell-based velocity of z-neighbor cell (v_{FLU}) also changes. The increasing of v_j does not stop until v_{FLU} reaches the hand calculation value (v_r). Through the following flow diagram (Figure 4.12), it will be clearly figured out:

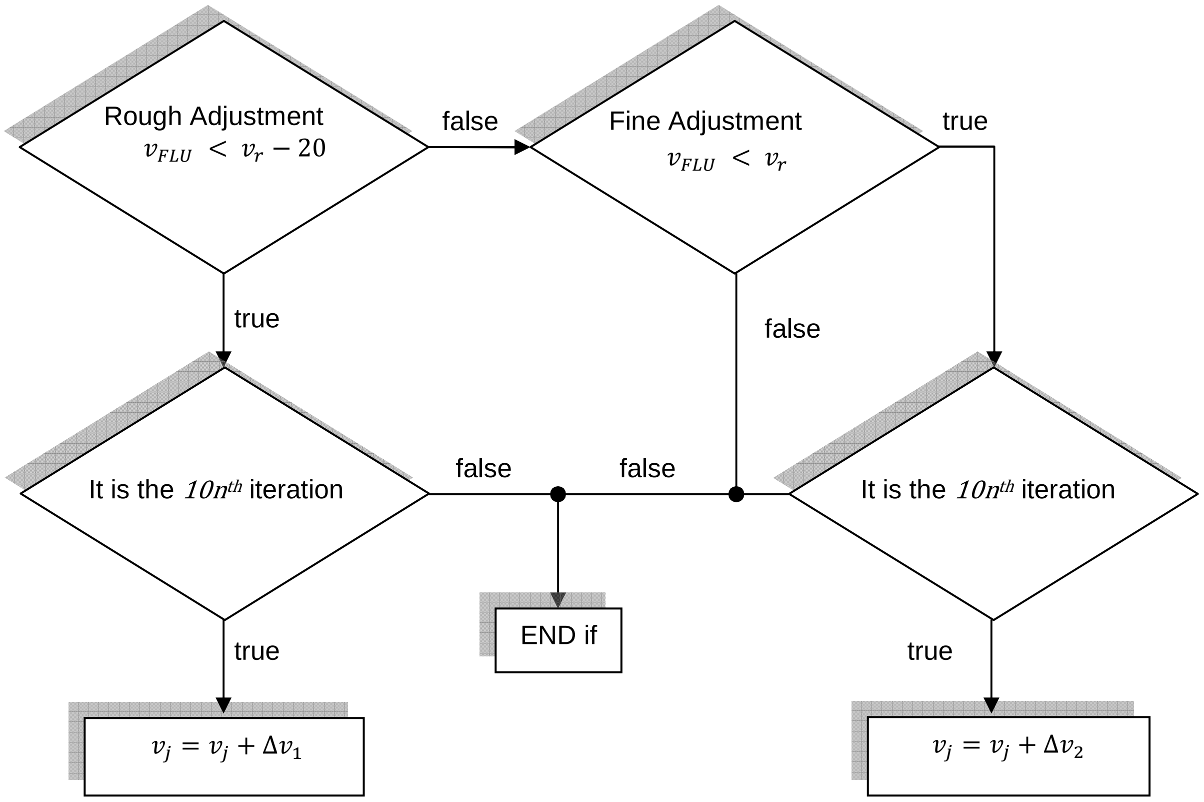


Figure 4.12: Flow Diagram of Velocity Correction

In the flow diagram, n is an integer, which means, the adjustment executes every 10 iterations. Δv_1 for rough adjustment is larger than Δv_2 for fine adjustment.

After obtaining the value of velocity for momentum source which makes the value of the cell-based velocity of z-neighbor cell equal as the hand calculation value, the air source by transient solution was defined with this value.

At this point, the main definition and method of DPM including the moving spray injector and the extern air source are described. In the next section, the model of PUR-layer structure will be discussed in detail.

4.3 VOF Model for Layer Structure

This section is the emphasis also the difficulty of this study. It describes how to model a layer structure of PUR-spray process using *ANSYS FLUENT*'s VOF model. VOF model is for the solution of the multiphase flows. There are two phases in the model, the primary phase (continuous phase) as air and the secondary phase (dispersed phase), which is material PUR hitting the plate. With VOF model the interfaces between the two phases at every time step can be solved. Usually the boundary conditions for secondary phases are defined as inlet or outlet by VOF model. For this PUR-spray model, PUR-particles as secondary phase do not come from any boundaries but from the injection cell and then remain on the plate whose boundary condition is "**wall**". Therefore, it is complex to define a VOF model for PUR-layer structure. The basic idea to solve this problem is to use the information of the PUR particles hitting the plate to define the mass source and momentum source for the *first layer of cells lying on the plate*. With these two sources, the volume fraction of PUR or air can be automatically calculated with the density of PUR predefined for materials.

Otherwise, another model, *ANSYS FLUENT*'s wall-film model was used to study the PUR-layer structure. It will be discussed in Section 4.4.

4.3.1 Collection of Information of PUR on Layer

In order to model the PUR-layer structure to identify the interface between the PUR fluid on the plate and the air as the primary phase at all times, some information of particles hitting the plate was collected to be the source terms of fluid for cell conditions in VOF model. As soon as a particle hits the plate, the mass, velocity, position of it can be obtained from *ANSYS FLUENT*. Mass sources and momentum sources can be calculated with this information. Initially, the basic realization of this objective requires following three steps:

- Track the mass, velocity and position of particles which are currently hitting the plate at every time step.
- Transform the information above into mass and momentum sources for volumetric cell conditions.
- By each end of time step, reset the information above (set back to 0) after the source terms become effective so as to refresh the information for next time step.

Before compiling the UDFs for solving these problems, the UDF calling sequence in the solution process must be learned. Knowing the sequence of function calls within iteration in the *ANSYS FLUENT* solution process can help you determine which data are at current and available at any given time. In this case, we use the pressure-based coupled solver (Figure 4.13), because one of the limitations for *ANSYS FLUENT*'s VOF model is that only the pressure-based solver can be used and VOF model is not available with the density-based solver.

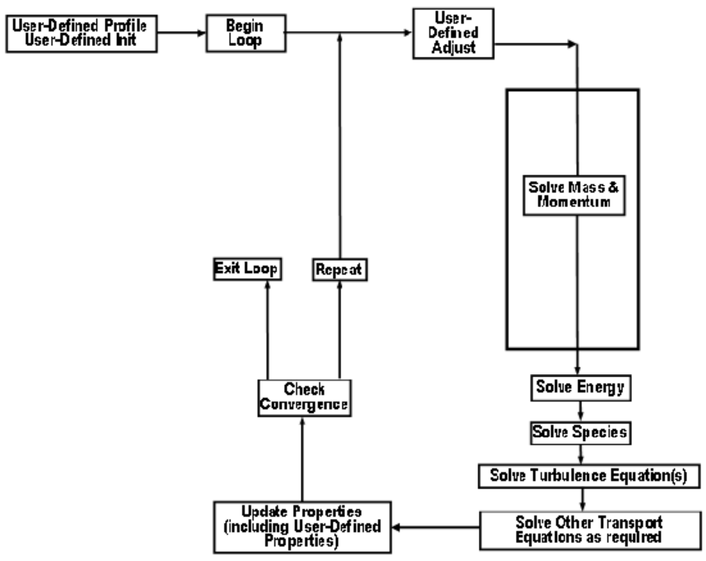


Figure 4.13: Procedure for the Pressure-Based Coupled Solver [10]

Generally the function DEFINE_ADJUST can be used to modify the necessary variables of particles on the plate for VOF model. Looping macros can be used to loop cells, faces, threads and so on. When using DEFINE_ADJUST, we need to find the face thread of the plate at first, and then loop all faces of this face thread to find the face that a particle is hitting. However, it needs a complex program to track the face that a particle is hitting by using this function.

Thus, in order to simplify the program and reduce the calculating time, another DEFINE macro can be used, whose argument has been already defined as the face that a particle is hitting. Because of DPM, this macro may be found out in the section about DPM model in [10]. The macro, DEFINE_DPM_BC just meets these requirements. It includes the argument types: the face that a particle is hitting and the particle on the plate. See Table 4.5 below shows the detail information of this DEFINE macro. Commonly, DEFINE_DPM_BC is used to specify the own boundary conditions for particles, but its arguments can be used for the special purpose as tracking the information of particles on the plate which are necessary for PUR-source terms. This function is executed every time when a particle touches a boundary of the domain.

DEFINE_DPM_BC(name,p,t,f,f_normal,dim)		
Argument Type	Description	Function returns
symbol name	UDF name.	int
Tracked_Particle *p	Pointer to the Tracked_Particle data structure which contains data related to the particle being tracked.	
Thread *t	Pointer to the face thread the particle is currently hitting.	
face_t f	Index of the face that the particle is hitting.	
real f_normal[]	Array that contains the unit vector which is normal to the face.	
int dim	Dimension of the flow problem. The value is 2 in 2D, for 2D-axisymmetric and 2D-axisymmetric-swirling flow, while it is 3 in 3d flows.	

Table 4.5: Usage of DEFINE_DPM_BC

Otherwise, refreshing the information of particles hitting the face at each time step is an emphasis. Without this step, the information calculated for the mass source and momentum source is the sum of mass and impulse of all the particles hitting the face.

4.3.2 Mass Source

When a flow source cannot be represented by an inlet, volumetric sources of mass for the cells on the plate whose boundary condition is "**wall**" is then defined. The mass source is

$$\frac{\dot{m}}{V} = \frac{\rho_j A_j v_j}{V} \tag{4.4}$$

where V is the cell volume, and $\dot{m}$ is mass stream, an argument in the equation for both mass source and momentum source, is

$$\dot{m} = \frac{\Delta m}{\Delta t} \tag{4.5}$$

where Δm is mass difference and Δt is time difference or time step. There are two methods to get Δm by UDFs. The steps of method 1 are described as following:

- Track the mass of new particles hitting on a face in current flow time step using DEFINE_DPM_BC.
- Use this mass as Δm for mass source using DEFINE_SOURCE.
- Initialize all the information of these particles to null using DEFINE_EXECUTE_AT_END, which is executed at the end of a time step in a transient run. (The usage of it is detailed in Table 4.6)

DEFINE_EXECUTE_AT_END(name)		
Argument Type	Description	Function returns
symbol name	UDF name.	void

Table 4.6: Usage of DEFINE_ EXECUTE_AT_END

Therefore, by method 1, the calling sequence of these three DEFINE macros is showed in Figure 4.14. However, method 1 is not feasible because its sequence does not meet the UDF calling sequence in *ANSYS FLUENT* showed in Figure 4.15. Which means, after initializing the information of particles using DEFINE_EXECUTE_AT_END, DEFINE_SOURCE will receive the empty signal which leads to null source term. For this reason, method 2 is designed to avoid this problem.

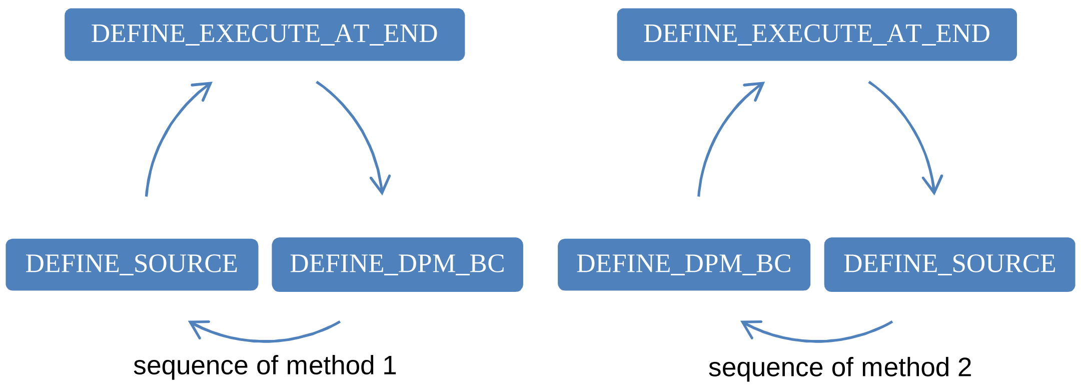


Figure 4.14: UDF Calling Sequence by Two different methods

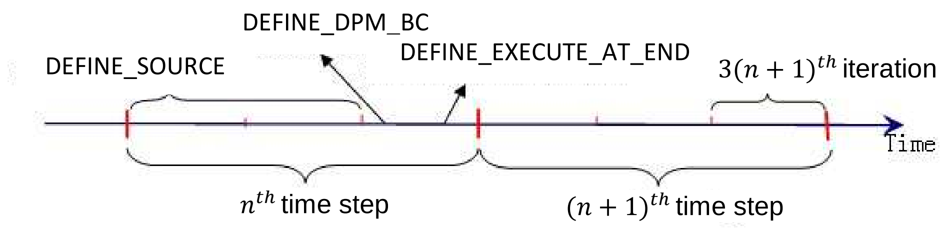


Figure 4.15: UDF Calling Sequence in *ANSYS FLUENT*

The steps of method 2:

- Track the mass of all particles which have been loaded on the faces using DEFINE_DPM_BC, regarding this mass as sum of mass at current flow time(m^{n+1}), (see Figure 4.16),
- Use DEFINE_EXECUTE_AT_END for taking the difference Δm^{n+1} between sum of mass at current flow time (m^{n+1}) and sum of mass at previous flow time (m^{n}), and then store m^{n+1} used in next time step as previous mass. (The first previous m^{0} mass during all the simulation is zero.)
- And then put Δm^{n+1} into Equation 4.5 for the mass source.

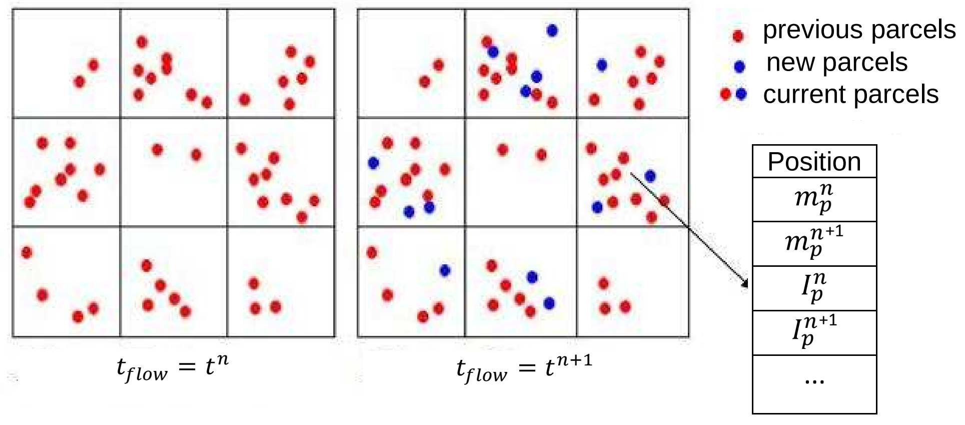


Figure 4.16: An Example of the Situation of Parcels Loaded on the Different Grids of the Plate at Different Flow Time

In Figure 4.16, the red points indicates the total parcels of the injection from the beginning of the simulation t^0 to previous flow time t^n, and the blue points indicates the total new parcels which is hitting the plate during the current time step. Thus, when the injection sprays at current flow time t^{n+1}, the total parcels on the plate are the combination of the red and the blue points. Every grid of the plate is called as face (boundary of a cell) in UDFs of *ANSYS FLUENT*. Each face or cell can store and retrieve up to 500 values of the user-defined memory (UDM) using macros F_UDMI (for a face) or C_UDMI (for a cell) for post-processing. Therefore, the position of the parcels, the current and previous mass of parcels, and the current and previous impulse of parcels and so on can be stored and retrieved as UDM on a face or in a cell.

4.3.3 Momentum Source

If a mass source is defined for a cell zone, a momentum source should be also defined. If only mass source is defined, that mass enters the domain with no momentum or thermal heat. The mass will therefore have to be accelerated and heated by the flow and, consequently, there may be a drop in velocity or temperature. This drop may or may not be perceptible, depending on the size of the source.

The momentum source is

$$\frac{\dot{m}v}{V} = \frac{\dot{m}v_j}{V} \tag{4.6}$$

compared with Equation 4.4, it is multiplied with a velocity of source v_j. The item $\dot{m}v_j$ is the impulse stream,

$$\dot{I} = \dot{m}\vec{v}_j = \frac{\Delta I}{\Delta t} \tag{4.7}$$

where ΔI is momentum difference and Δt is time difference or time step called in *ANSYS FLUENT*. ΔI can be also obtained with method 2 for catching Δm. Momentum I on a face is an accumulation momentum of every particle, which is expressed as:

$$I_i = \sum n_i m_i \vec{v}_i \tag{4.8}$$

where i is the direction index, n_i is the number of particles of each tracked parcel. In UDFs, the mass and velocity of each particle in a parcel is equivalent and represented by m_i and $\vec{v}_i$ are the mass and velocity of this parcel which in UDFs represent the mass and velocity of each particle belongs to this parcel.

The logic in the computation of the momentum source is the same as explained for the mass source. Therefore the 3 components of the momentum vector are stored in 3 different C_UDMIs and refreshed at the end of every time step.

4.3.4 Size of Time Step

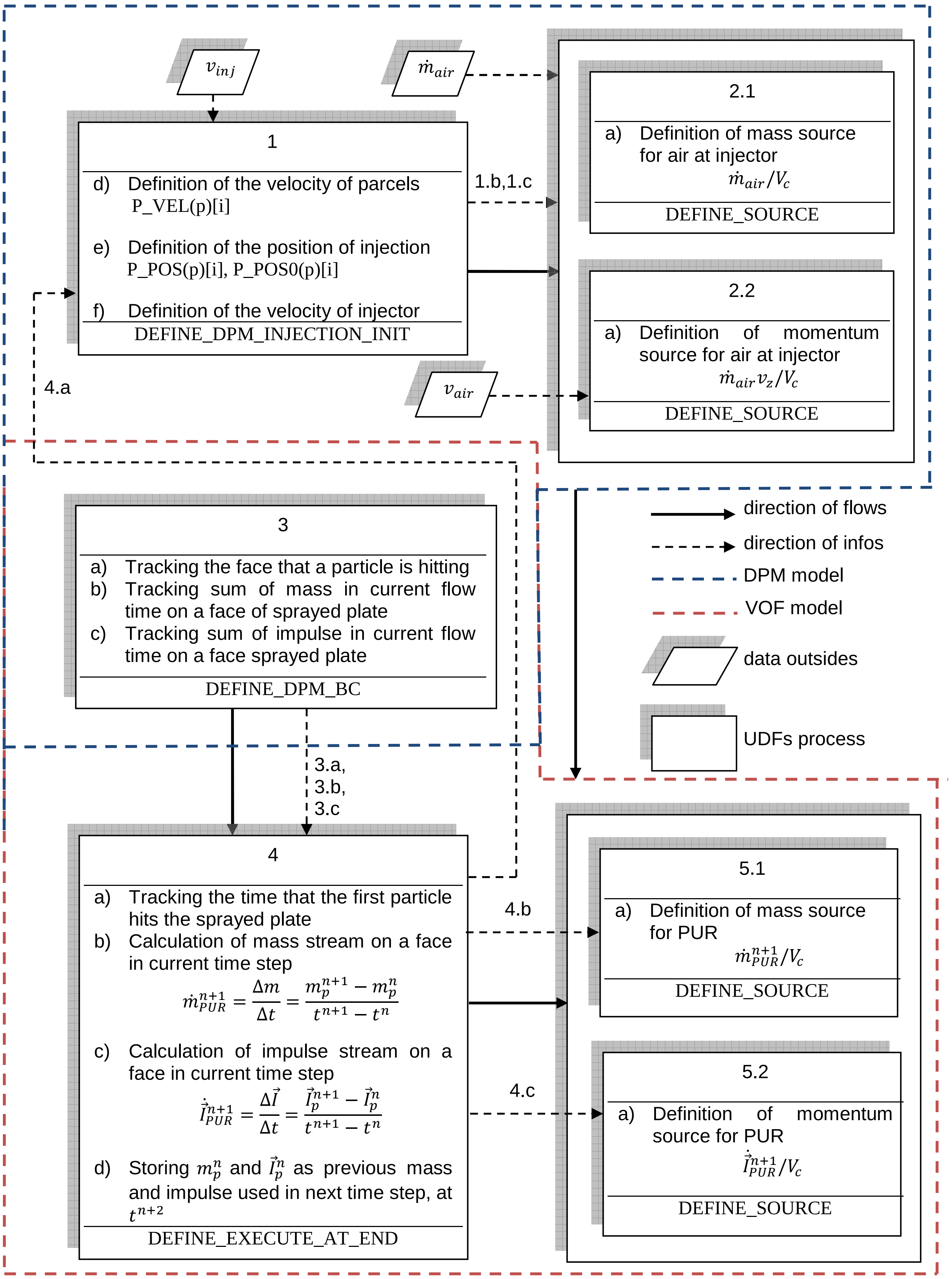
When using VOF Model, non-convergence problems may happen. In order to avoid this problem, dynamic time step for the simulation is necessary. [14] The range for a variable time step needs to be set. Time step can be estimated as [15]:

$$\Delta t = CFL \times \frac{V_{cell,min}^{1/3}}{U} \tag{4.9}$$

where $V_{cell,min}$ is the minimum volume of grid obtained from the **Grid Check** panel. U is the velocity scale of problem, the maximum one along particle velocity, air velocity and gravitational velocity. CFL is Courant number in the range of 0.5 to 1.

4.3.5 Solution Procedure of UDFs for VOF Model

In order to get an intuitional view over the UDFs for this VOF model running in *ANSYS FLUENT*, the following figure summarizes their particular tasks and sequence over each other which have been discussed in Section 4.2 and Section 4.3.



1

d) Definition of the velocity of parcels P_VEL(p)[i]

e) Definition of the position of injection P_POS(p)[i], P_POS0(p)[i]

f) Definition of the velocity of injector

DEFINE_DPM_INJECTION_INIT

2.1

a) Definition of mass source for air at injector $\dot{m}_{air}/V_c$

DEFINE_SOURCE

2.2

a) Definition of momentum source for air at injector $\dot{m}_{air}v_z/V_c$

DEFINE_SOURCE

3

a) Tracking the face that a particle is hitting
b) Tracking sum of mass in current flow time on a face of sprayed plate
c) Tracking sum of impulse in current flow time on a face sprayed plate

DEFINE_DPM_BC

4

a) Tracking the time that the first particle hits the sprayed plate
b) Calculation of mass stream on a face in current time step

$$\dot{m}_{PUR}^{n+1} = \frac{\Delta m}{\Delta t} = \frac{m_p^{n+1} - m_p^n}{t^{n+1} - t^n}$$

c) Calculation of impulse stream on a face in current time step

$$\dot{\vec{I}}_{PUR}^{n+1} = \frac{\Delta \vec{I}}{\Delta t} = \frac{\vec{I}_p^{n+1} - \vec{I}_p^n}{t^{n+1} - t^n}$$

d) Storing m_p^n and $\vec{I}_p^n$ as previous mass and impulse used in next time step, at t^{n+2}

DEFINE_EXECUTE_AT_END

5.1

a) Definition of mass source for PUR $\dot{m}_{PUR}^{n+1}/V_c$

DEFINE_SOURCE

5.2

a) Definition of momentum source for PUR $\dot{\vec{I}}_{PUR}^{n+1}/V_c$

DEFINE_SOURCE

Figure 4.17: Flow Diagram of UDFs for VOF Model

4.4 Wall-Film Model for Layer Structure

In *ANSYS FLUENT*, wall-film model is a specific boundary condition for simulation of internal combustion engines. This model is usually important for the port fuel injected (PFI) engines. DPM particles are used to model the wall-film. The wall-film model in *ANSYS FLUENT* allows a single component liquid drop to impinge upon a boundary surface and form a thin film.

4.4.1 Collection of information using UDFs

In order to obtain a spray-wall of wall-film for this PUR-spray model, "**wall-film**" should be selected in the dialog box of the boundary conditions after the DPM model has been enabled. Compared with the VOF model, no source items stimulated by particles on the surface are defined for cell zone conditions, which means, there is no need to collect the information about the spray particles on the surface. Discussed in Section 4.2.1, the injection begins to move itself straightly at a constant speed as soon as the first particle hits the plate (key time). When calculating a VOF model the time is collected through two DEFINE macros, DEFINE_DPM_BC and DEFINE_EXECUTE_AT_END. However, for wall-film model, the boundary condition is defined as "**wall-film**" instead, while for VOF model it is defined as "**user-defined function**" which is compiled with DEFINE_DPM_BC.

So another DEFINE macro is compiled to get this key time. Considering to simplify the compiling (No looping of faces or cells), it is better to use a DEFINE macro which contains an argument of the face that the particle is hitting just like the argument 'face_t' in DEFINE_DPM_BC. DEFINE_DPM_EROSION is a DEFINE macro, which is usually used to specify the erosion and accretion rates by being calculated as the particle stream strikes a wall surface. The steps obtaining the key time is described as following:

- Mark the face that the particles are hitting with UDM macro in DEFINE_DPM_EROSION.
- In DEFINE_EXECUTE_AT_END, if the marked face is found out, record the flow time as the time the first particle hits the wall, and this step is called only once during all the simulation.
- Use this key time as global variable to define the situation of moving injection in DEFINE_DPM_INIT_INJECTION.

For VOF Model, it is the same way to find out this key time except the face here is marked by using DEFINE_DPM_BC instead.

Besides using face_t to obtain the key time, noting will be defined in DEFINE_DPM_EROSION, so it will not specify any erosion or accretion for the reflecting surface to influence the wall-film model. This function will be visible and selectable in the Discrete Phase Model dialog box in *ANSYS FLUENT* after enabling the Interaction with Continuous Phase option under Interaction in the Discrete Phase Model dialog box.

4.4.2 Solution Procedure of UDFs for Wall-Film Model

Compared with VOF Model, UDFs for wall-film model are simpler and its procedure is showed as following:

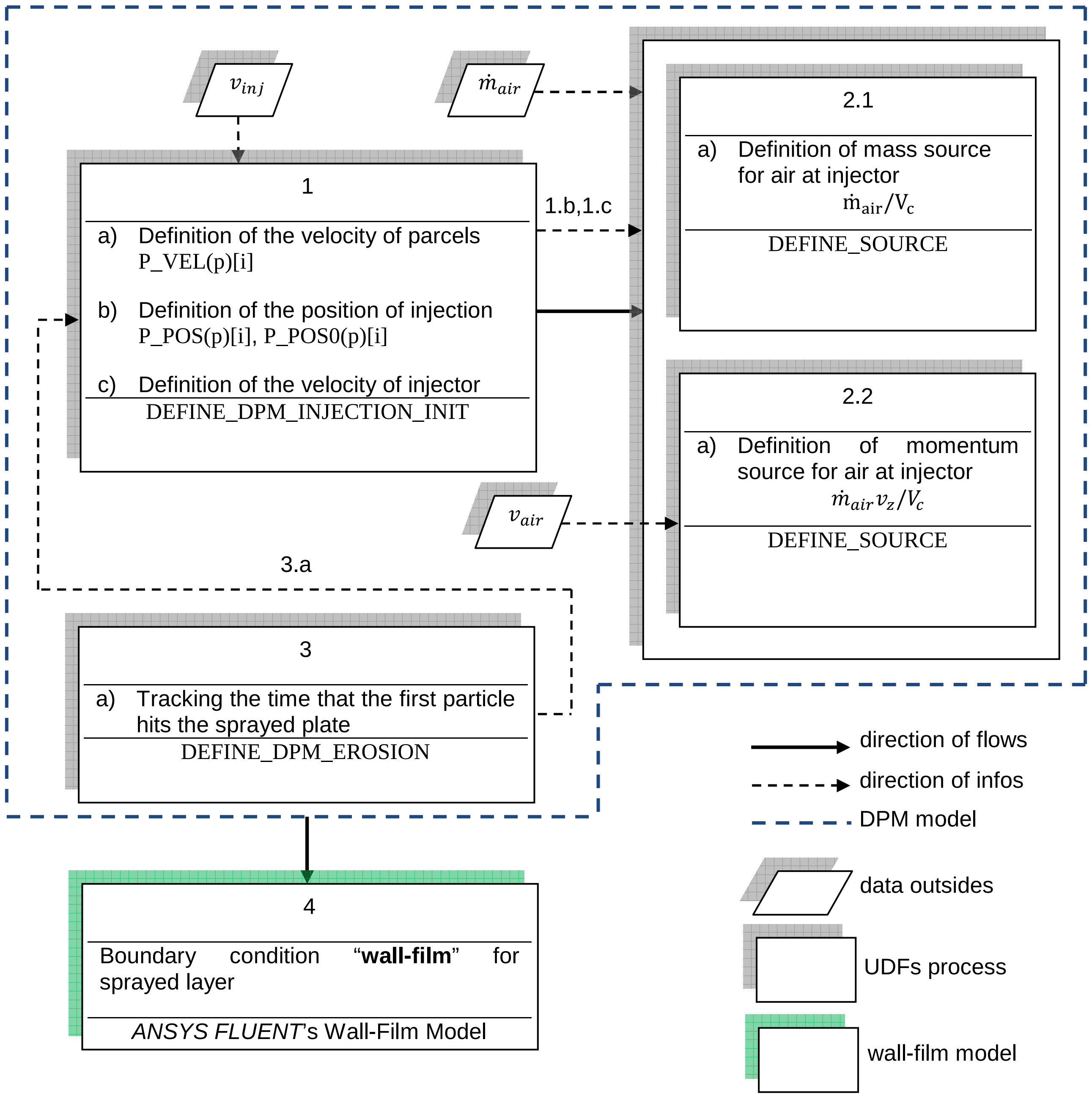

v_{inj}
$\dot{m}_{air}$
1
a) Definition of the velocity of parcels P_VEL(p)[i]
b) Definition of the position of injection P_POS(p)[i], P_POS0(p)[i]
c) Definition of the velocity of injector
DEFINE_DPM_INJECTION_INIT
1.b,1.c
2.1
a) Definition of mass source for air at injector $\dot{m}_{air}/V_c$
DEFINE_SOURCE
2.2
a) Definition of momentum source for air at injector $\dot{m}_{air}v_z/V_c$
DEFINE_SOURCE
v_{air}
3.a
3
a) Tracking the time that the first particle hits the sprayed plate
DEFINE_DPM_EROSION
4
Boundary condition "wall-film" for sprayed layer
ANSYS FLUENT's Wall-Film Model
direction of flows
direction of infos
DPM model
data outsides
UDFs process
wall-film model

Figure 4.18: Flow Diagram of UDFs for Wall-Film Model

5. Results and Comparisons

In this chapter, the results of experiment and simulation, wall-film model and VOF model, are discussed and compared. Section 5.1 explains concisely the method of the practical experiment. Section 5.2 shows the results of the layer structure calculated and simulated with wall-film model. And Section 5.3 shows the results of the layer structure calculated and simulated with VOF model.

5.1 Practical Experiment

5.1.1 Method of Practical Experiment

The method of the practical experiment is optimized on the base of Hennecke method [11]. A PUR-air atomizer with fixed parameters (e.g. outer and inner diameter) executed at different height at the same moving speed along a direction. (See Table 5.1) The plate was a special paper with a constant weight-area ratio. After spraying, the layer structure was punched with the conduction-pusher tool (Figure 4.3) accurately into a group of samples with the same surface area. Each sample was weighted, and the weight of the PUR-film was deducted from the weight of the sample. PUR used in the experiment were Baytec, Baypreg, Bayflex and Multitec. Here, the results of Baytec are discussed and compared.

Spray height (mm)	600	400
Flow rate (g/s)	21	21
Moving speed (mm/s)	500	500
Nozzle	$5mm$ conical	$5mm$ conical

Table 5.1: Parameter for Practical Experiment

5.1.2 Results of Practical Experiment

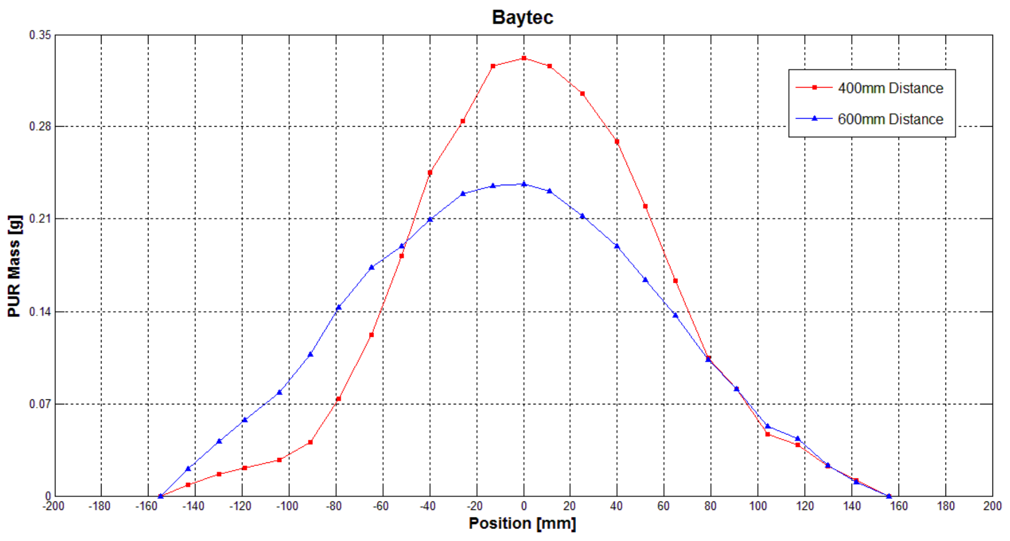
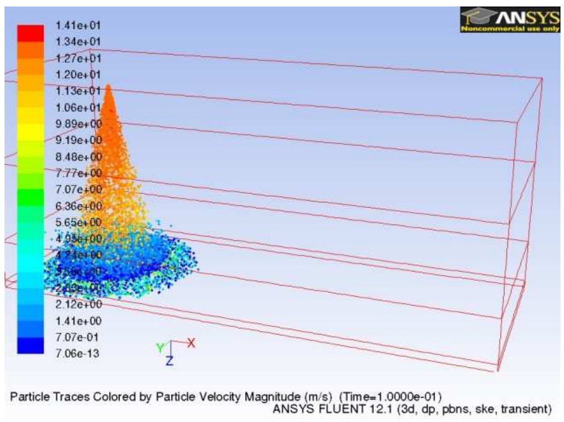


Figure 5.1: Comparisons of Baytec $400mm$, $600mm$ Distance at $21g/s$ Flow Rate [11]

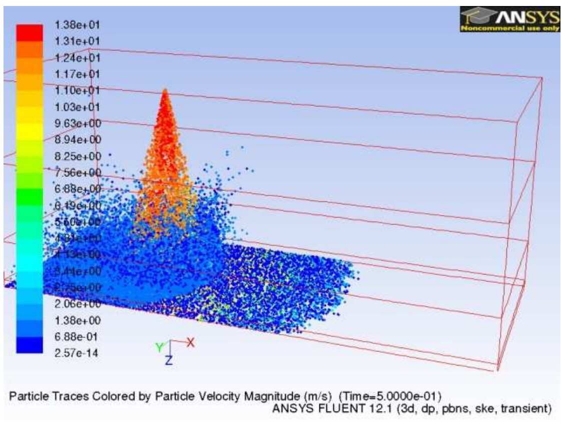
Figure 5.1 shows the distribution of the PUR mass in width. More PUR particles deposited in the middle of the layer than on two side of the layer. The larger the distance between the atomizer and the sprayed layer is, the wider the distribution is. Because of the limited source of information, the spray time and spray trajectory are unknown. However, the atomizer trajectory by CFD simulation in *ANSYS FLUENT* is a simple straight trajectory along the middle line of the plate. It stops just when the atomizer exceeds the simulated area.

5.2 Results of Wall-Film Model

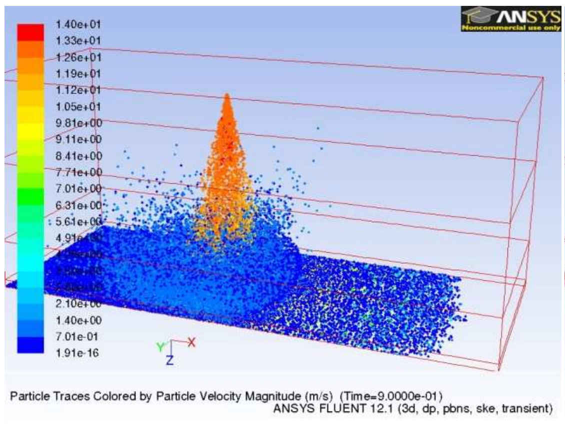
By standard film model, wall-film model, two-way coupling was enabled because the effect of the discrete phase is included on continues phase. Owing to the high energy when particles hit the plate, splash was also enabled in wall-film model. To model a spray atomizer which generates small droplets, breakup and collision of droplet were considered. The layer was sprayed only once during the CFD simulation.


Particle Traces Colored by Particle Velocity Magnitude (m/s) (Time=1.0000e-01)
ANSYS FLUENT 12.1 (3d, dp, pbns, ske, transient)

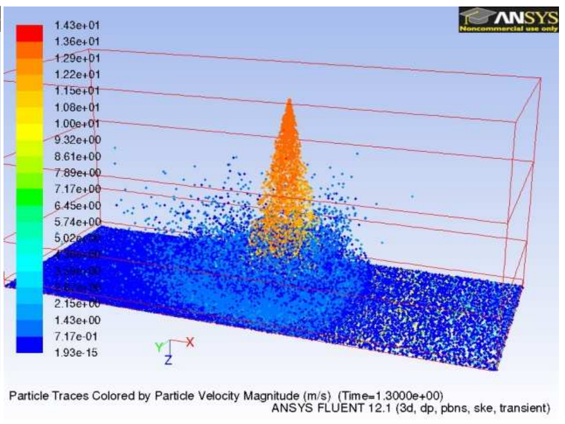
a) $t_{flow} = 0.10\ s$


Particle Traces Colored by Particle Velocity Magnitude (m/s) (Time=5.0000e-01)
ANSYS FLUENT 12.1 (3d, dp, pbns, ske, transient)

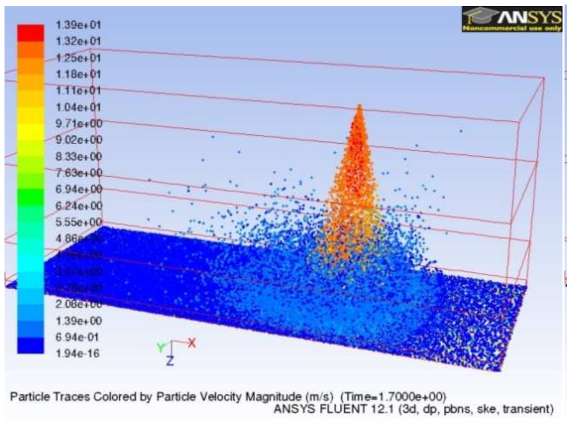
b) $t_{flow} = 0.50\ s$


Particle Traces Colored by Particle Velocity Magnitude (m/s) (Time=9.0000e-01)
ANSYS FLUENT 12.1 (3d, dp, pbns, ske, transient)

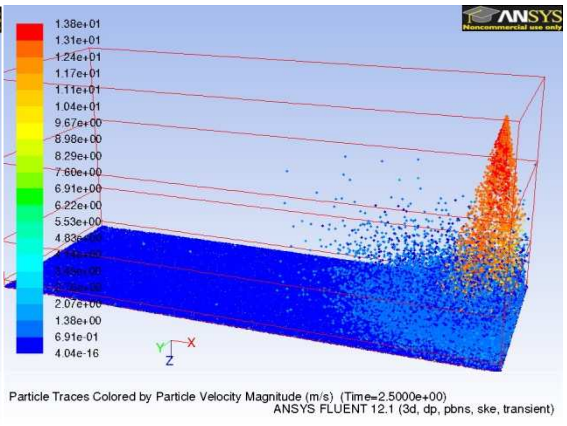
c) $t_{flow} = 0.90\ s$


Particle Traces Colored by Particle Velocity Magnitude (m/s) (Time=1.3000e+00)
ANSYS FLUENT 12.1 (3d, dp, pbns, ske, transient)

d) $t_{flow} = 1.30\ s$

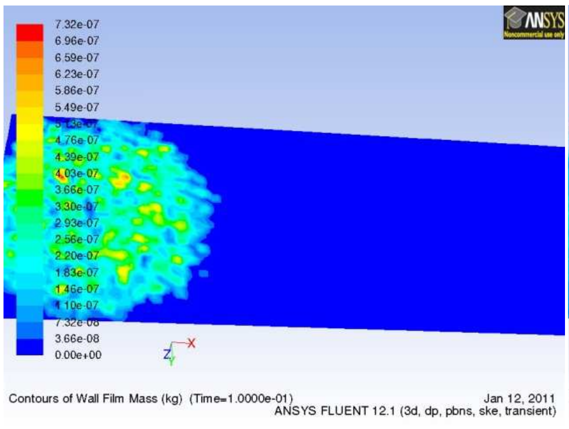

Particle Traces Colored by Particle Velocity Magnitude (m/s) (Time=1.7000e+00)
ANSYS FLUENT 12.1 (3d, dp, pbns, ske, transient)

e) $t_{flow} = 1.70\ s$

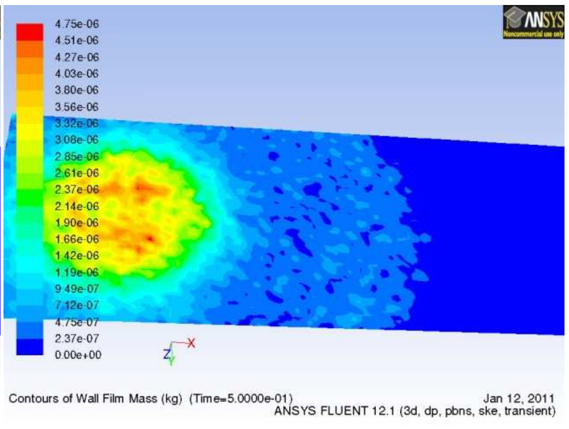

Particle Traces Colored by Particle Velocity Magnitude (m/s) (Time=2.5000e+00)
ANSYS FLUENT 12.1 (3d, dp, pbns, ske, transient)

f) $t_{flow} = 2.50\ s$

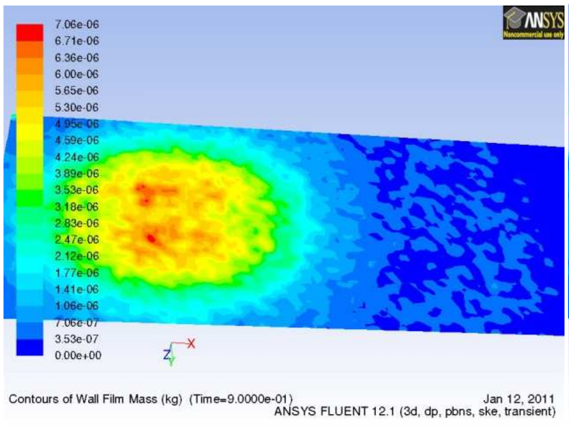
Figure 5.2: Particle Traces Colored by Particle Velocity Magnitude, 600mm Distance, wall-film model



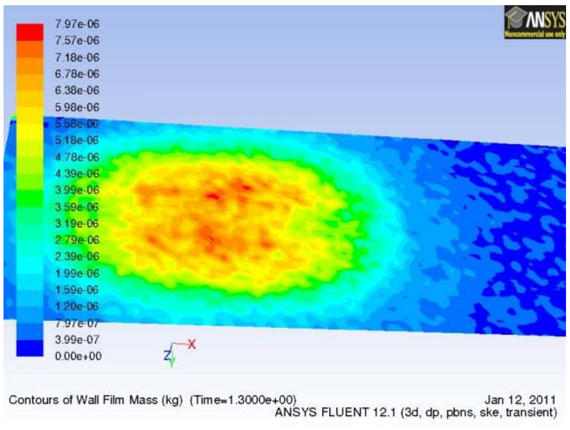
a) $t_{flow} = 0.10\ s$



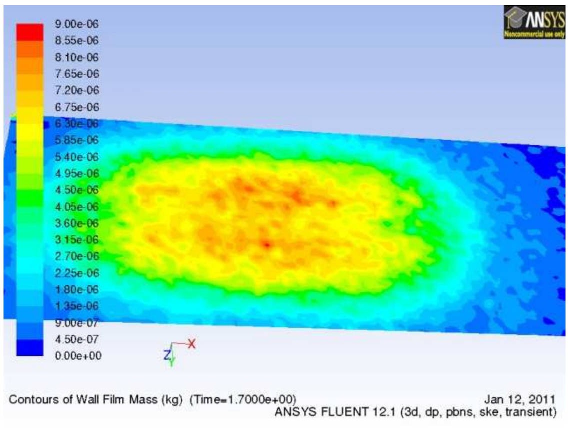
b) $t_{flow} = 0.50\ s$



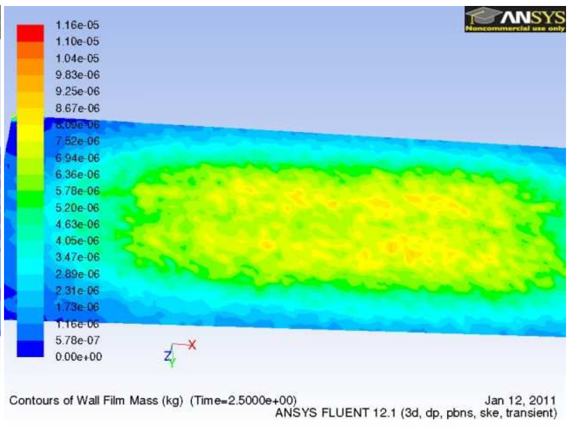
c) $t_{flow} = 0.90\ s$



d) $t_{flow} = 1.30\ s$



e) $t_{flow} = 1.70\ s$



f) $t_{flow} = 2.50\ s$

Figure 5.3: Contours of Wall Film Mass, 600mm Distance, wall-film model

Figure 5.2 shows the particle traces colored by magnitude velocity of particle. The injector moves straightly from the left side of x axis. When particles reside on the plate, the velocity of almost all the particles changed to zero. The whole area of the plate was full of the particles although the area of the spray area which related with the half spay angle is more narrow than the width of the plate. We can see that many droplets rebound back into the air instead of residing on the plate. However, the results are different as predicted in practice. The distributions of film mass are showed in Figure 5.3.

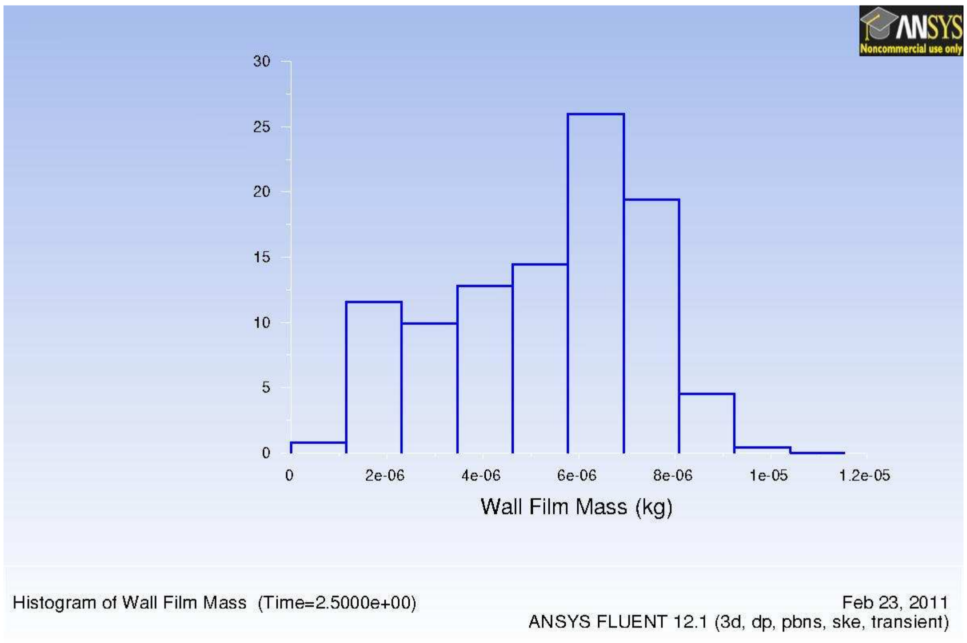


Figure 5.4: Histogram of Wall Film Mass, $600mm$ Distance, wall-film model

Figure 5.4 shows this mass distribution related with the quantity of faces. For example, in this diagram on each face from ca. 27% (6th bar) of the whole plate was loaded 6×10^{-6} to 8×10^{-6} kg of the mass.

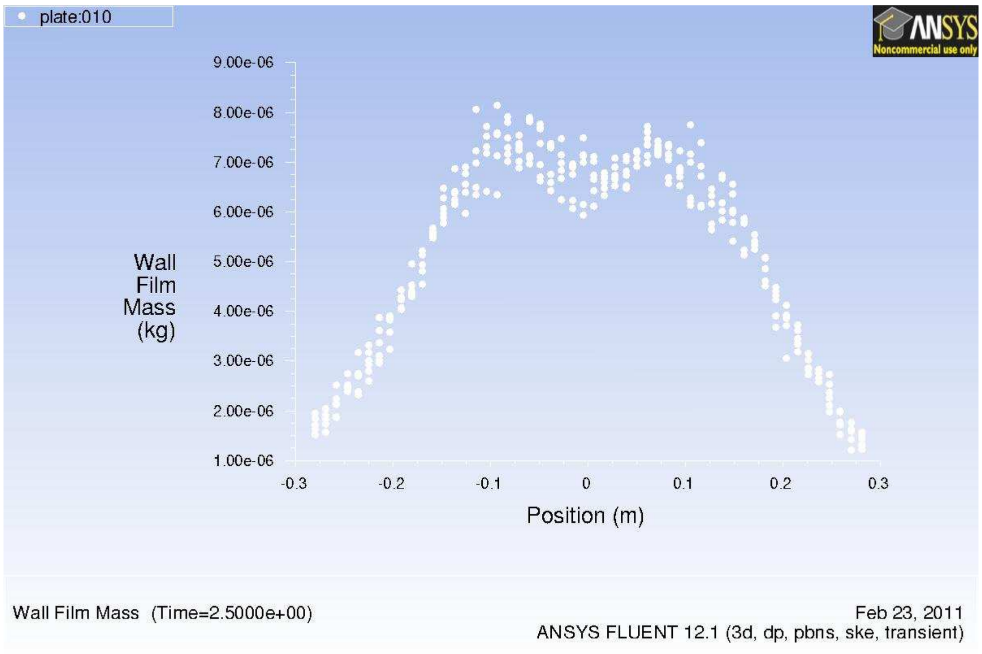


Figure 5.5: Cell-Based Distribution of Wall Film Mass in Width, $600mm$ Distance

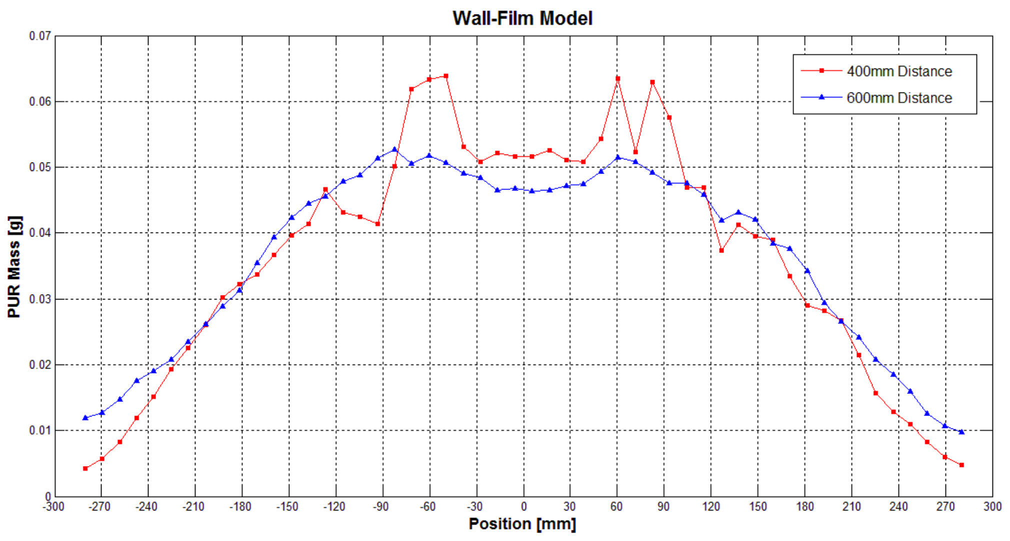
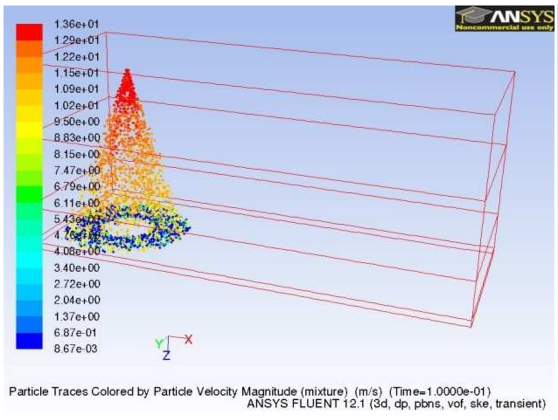


Figure 5.6: Distribution of Wall Film Mass in Width, $400mm$, $600mm$ Distance at $21g/s$ Flow Rate

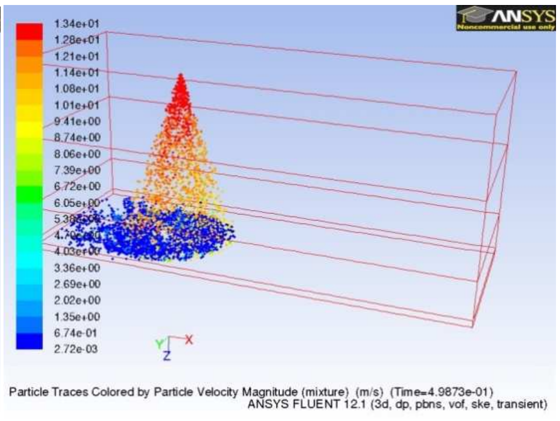
According to the experimental sample mentioned in Section 5.1, a region with same area was marked from the sprayed layer in order to capture simulation data. Figure 5.5 shows the distribution of wall film mass in width. Different with the experimental data (see Figure 5.1), the curve bends downward a little bit at the top middle because the nozzle is a ring form (see Figure 5.7a). Each white point denotes a control volume. The distribution of total mass in length direction is showed in Figure 5.6. As the figure shows, the mass distribution is relatively stable in the condition when the injector works at a high position. By $400mm$ and $600mm$ distance, the average masses of the samples are both ca. 0.0364g, and the maximum masses of the samples are respectively 0.0638g and 0.0528g.

5.3 Results of VOF Model

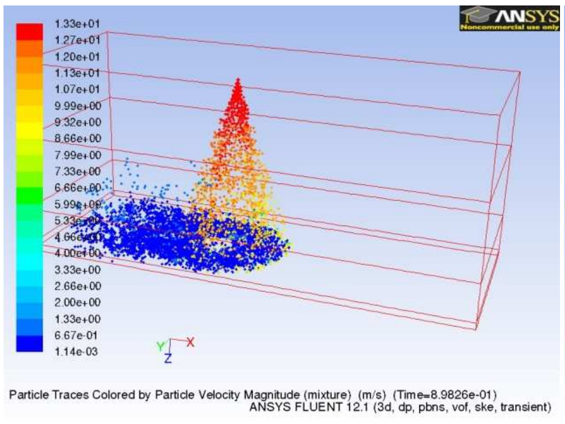
By VOF model, two-way coupling was also enabled with the same reason as wall-film model. To model a spray atomizer which generates small droplets, breakup and collision of droplet were considered. The layer was sprayed only once during the CFD simulation. It runs with a dynamic time step.



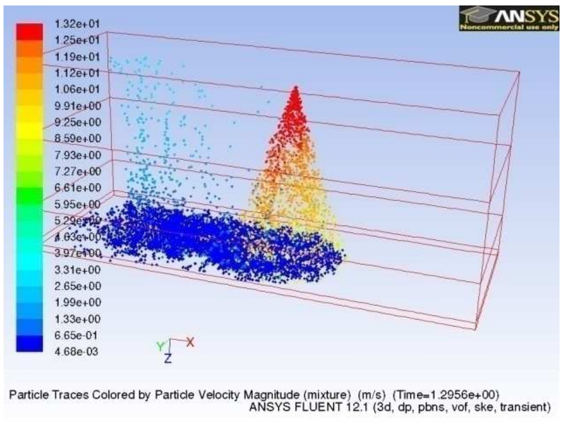
a) $t_{flow} = 0.10\ s$



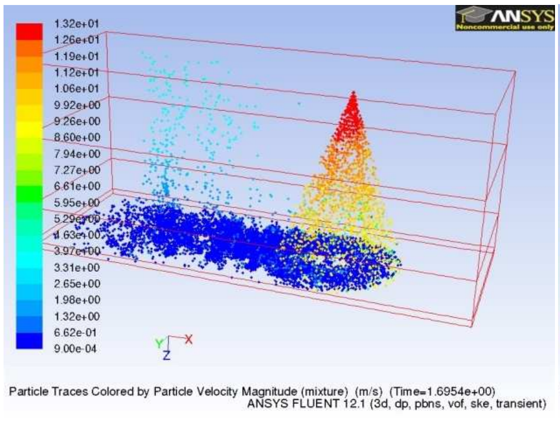
b) $t_{flow} = 0.49873\ s$



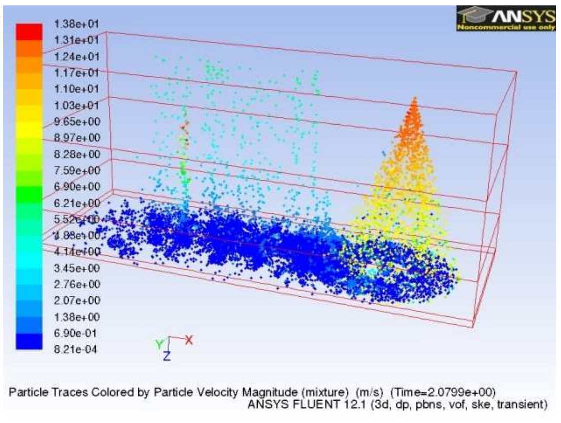
c) $t_{flow} = 0.89826\ s$



d) $t_{flow} = 1.2956\ s$

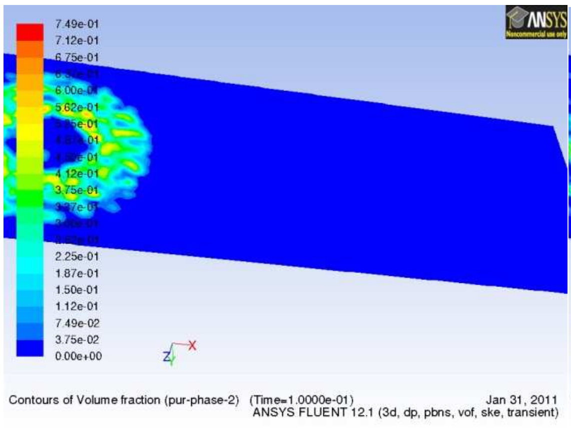


e) $t_{flow} = 1.6954\ s$



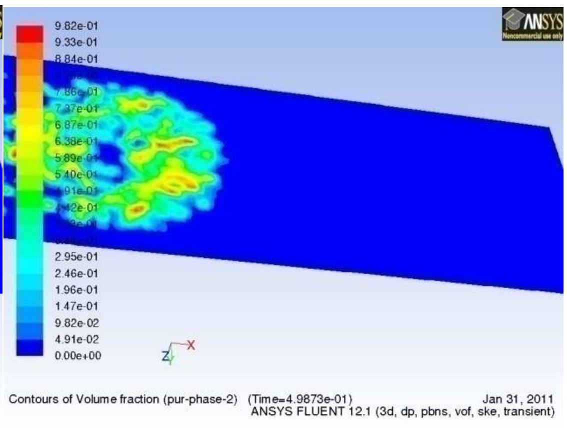
f) $t_{flow} = 2.0799\ s$

Figure 5.7: Particle Traces Colored by Particle Velocity Magnitude, 600mm Distance, VOF Model



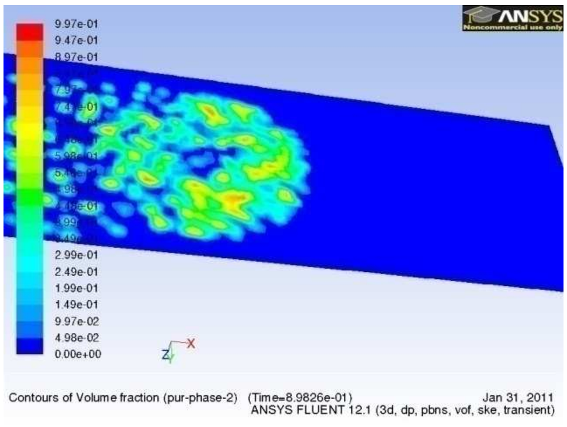
Contours of Volume fraction (pur-phase-2) (Time=1.0000e-01) Jan 31, 2011
ANSYS FLUENT 12.1 (3d, dp, pbns, vof, ske, transient)

a) $t_{flow} = 0.10\ s$



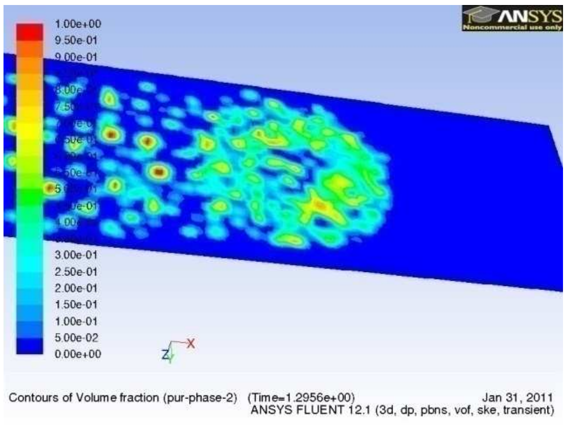
Contours of Volume fraction (pur-phase-2) (Time=4.9873e-01) Jan 31, 2011
ANSYS FLUENT 12.1 (3d, dp, pbns, vof, ske, transient)

b) $t_{flow} = 0.49873s$



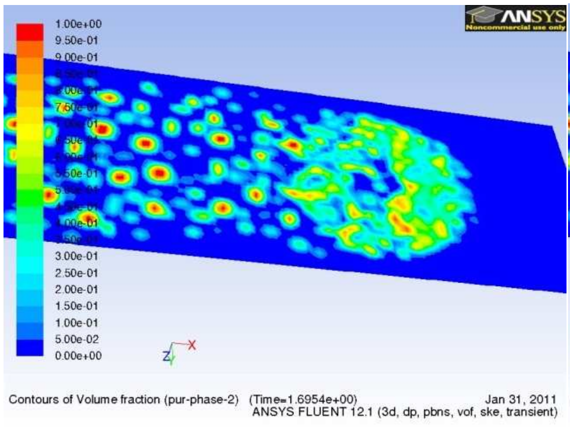
Contours of Volume fraction (pur-phase-2) (Time=8.9826e-01) Jan 31, 2011
ANSYS FLUENT 12.1 (3d, dp, pbns, vof, ske, transient)

c) $t_{flow} = 0.89826\ s$



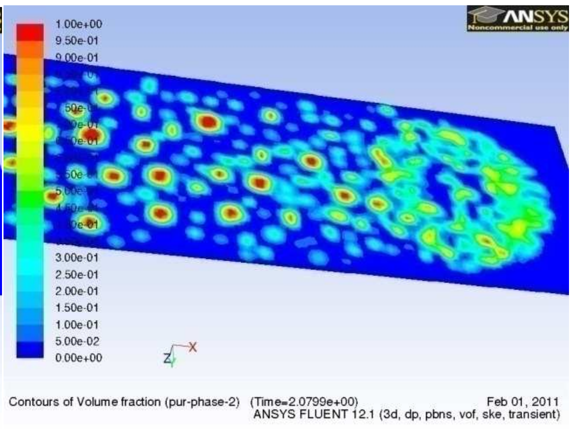
Contours of Volume fraction (pur-phase-2) (Time=1.2956e+00) Jan 31, 2011
ANSYS FLUENT 12.1 (3d, dp, pbns, vof, ske, transient)

d) $t_{flow} = 1.2956\ s$



Contours of Volume fraction (pur-phase-2) (Time=1.6954e+00) Jan 31, 2011
ANSYS FLUENT 12.1 (3d, dp, pbns, vof, ske, transient)

e) $t_{flow} = 1.6954\ s$



Contours of Volume fraction (pur-phase-2) (Time=2.0799e+00) Feb 01, 2011
ANSYS FLUENT 12.1 (3d, dp, pbns, vof, ske, transient)

f) $t_{flow} = 2.0799\ s$

Figure 5.8: Contours of Volume Fraction (PUR-Particles) on the Plate by VOF Model, 600mm Distance

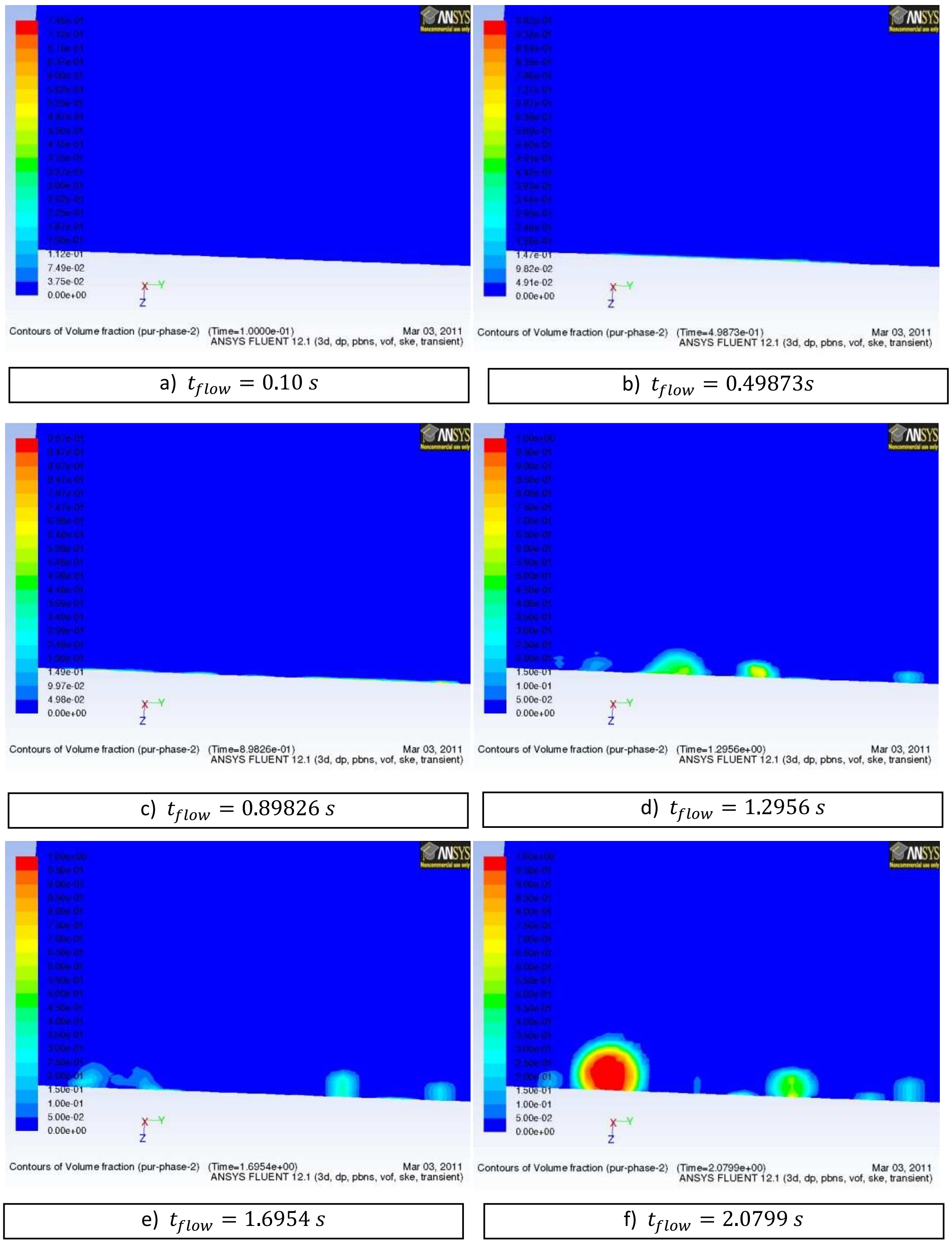


Figure 5.9: Contours of Volume Fraction (PUR-Particles) on the cross section of the plate by VOF Model, $600mm$ Distance

Figure 5.7 shows the tracking of PUR-particles at different time. The boundary condition for particles after they hit the plate was not defined in VOF model. With regard to these particles which are hitting the plate, only mass and momentum of them were collected. Thus, as figures showed, it is unexplained, that the particles rebounded back into the air. Compared with wall-film model, less rebounded particles were included, and the direction of them was single.

Figure 5.8 shows the tracking of volume fraction of PUR on the plate at different time. By VOF model, not every volume fraction (PUR-phase) in control volume increased over the flow time. It can be conjectured, that the PUR particles did not resided on the faces that they hit originally. The condition of the particles hit the faces was unknown. As Figure 5.8f shows, PUR particles distributed inhomogeneous by the end of the simulation. Thus, *ANSYS FLUENT*'s VOF model is not suitable for modeling the layer structure.

Figure 5.9 shows the volume of PUR on the cross section of the plate in the view of x direction. Only the cells of the plate were occupied by PUR particles in the beginning (see Figure 5.9a, b, c). And then PUR-layer turned thicker in some regions. As Figure 5.9f shows, the red area was full of PUR particles, the volume fraction of PUR was 1.

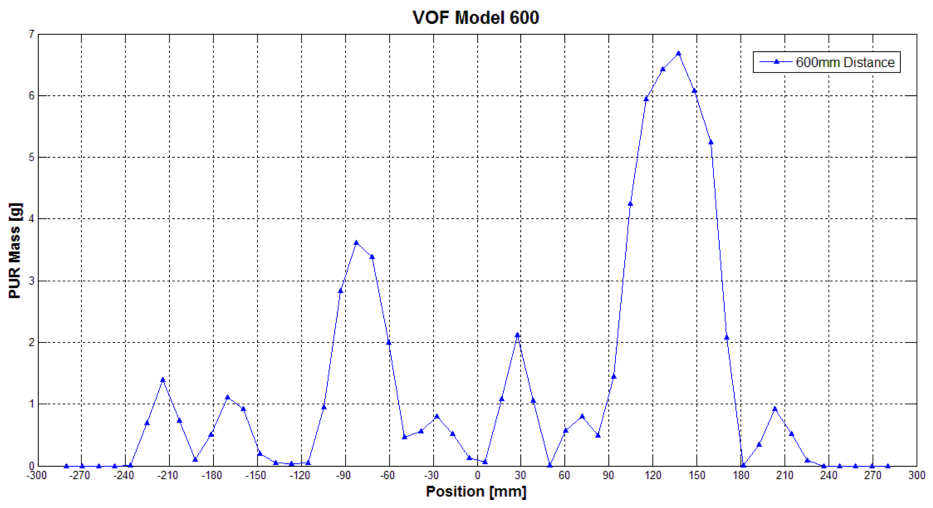


Figure 5.10: Distribution of PUR Mass in Width by VOF Model

PUR mass on the sprayed plate simulated by VOF model cannot be directly captured from *ANSYS FLUENT*. Through Equation 5.1, mass was obtained.

$$m_q = \rho_q V_c \alpha_q \tag{5.1}$$

Figure 5.10 shows the distribution of PUR mass in width at $600mm$ by VOF model. Compared with experimental data and wall-film data, the curve of VOF model does not show up like a parabola. The average and the maximum mass of the samples are respectively 1.2939g and 6.6825g. With the same flow rate and nearly same flow time, the average mass by VOF model is about 36 times as large as it by wall-film model.

During the simulation for VOF model, "**Floating Point Error**" often occurred to stop the calculation. (See Figure 5.11) In order to optimize the model, the following possibilities which lead to the error were considered as following:

a) There were particles whose velocities were unrealistically and unexplainably large, and the positions of these particles were arbitrary (maybe on the plate, maybe near the plate or in the air).

To avoid this problem, these particles were slowed down by using UDFs.

b) There were control volumes whose mixture velocities were unrealistically and unexplainably large. The mixture velocity of a control volume is calculated with the velocities of all the phases. The large velocity of continuum (air) may be the reason of the large mixture velocity as well as the large velocity of discrete phase (PUR-particle).
   Although the problem a) was avoided, “**Floating Point Error**” remained. Assume that the large velocities of continuum (air) leaded to the large velocities of PUR-particles, the mixture velocities of these control volumes were also slowed down.

c) An insufficiently long time step size may not satisfy a large velocity.
   Dynamic time step was used to solve the problem.

d) The under-relaxation factors for particles/droplets were reduced. The interphase exchange of momentum, heat, and mass is under-relaxed during the calculation. Reduced under-relaxation factors improve the stability of coupled calculations

However, after solving the problems described above, it ran still unstably by VOF model.

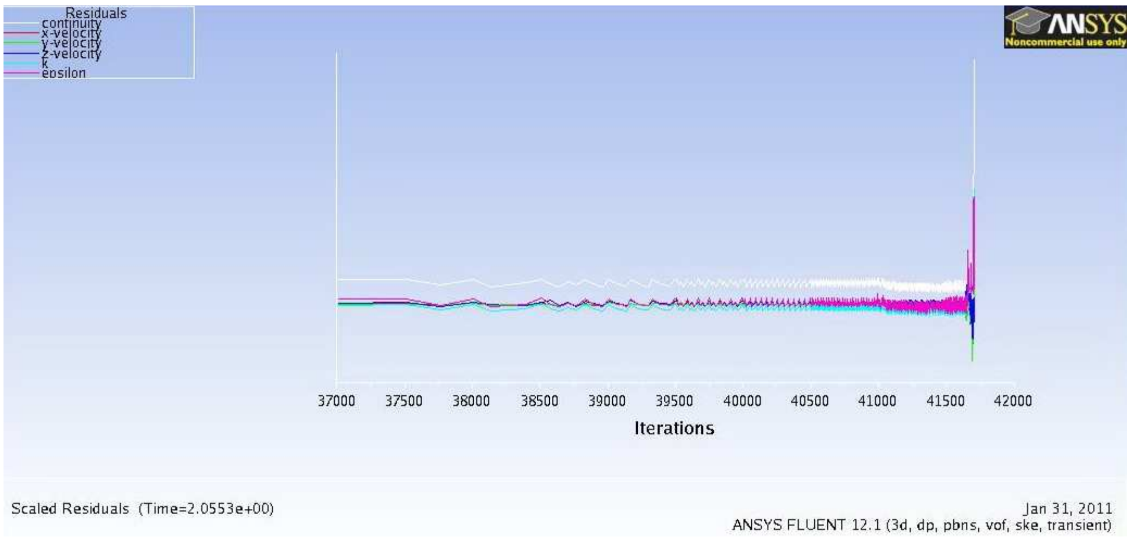


Figure 5.11: Scaled Residuals, $600mm$ Distance, VOF Model

5.4 Alternative simulation (Eulerian Model)

As Figure 5.7 shows, a large amount of particles rebounded from the plate. In order to study this unexplainable phenomenon, another multiphase flows model, Eulerian model, was tried.

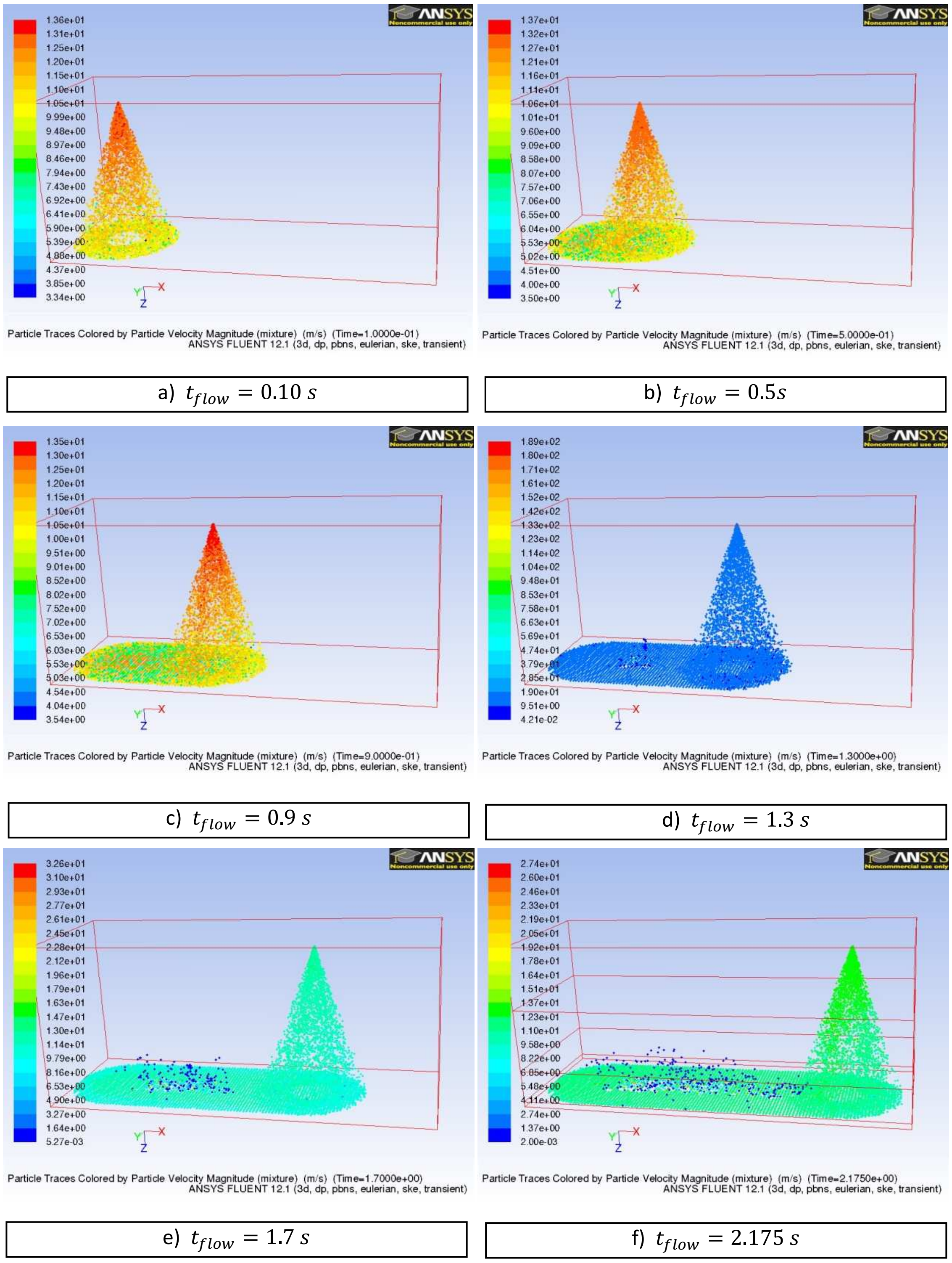
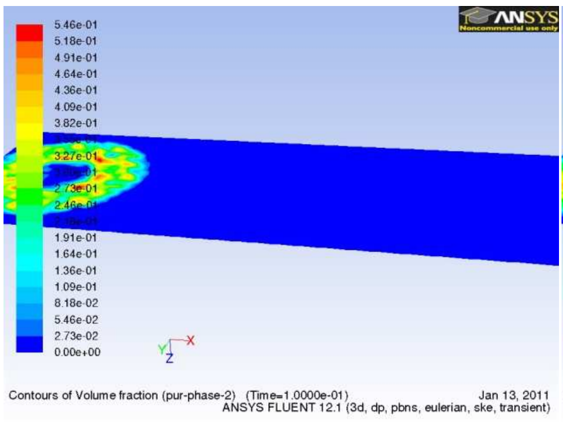
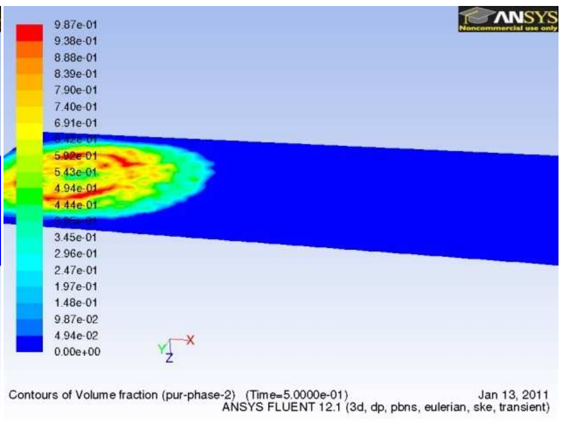


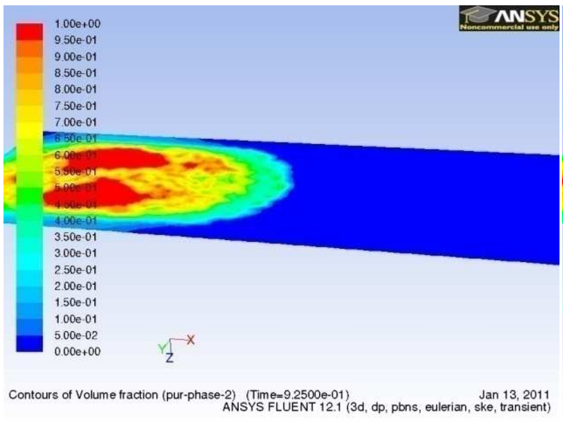
Figure 5.12: Particle Traces Colored by Particle Velocity Magnitude, 600mm Distance, Eulerian Model



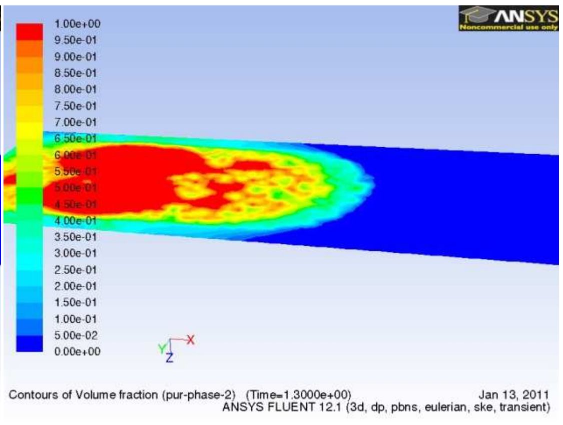
a) $t_{flow} = 0.10\ s$



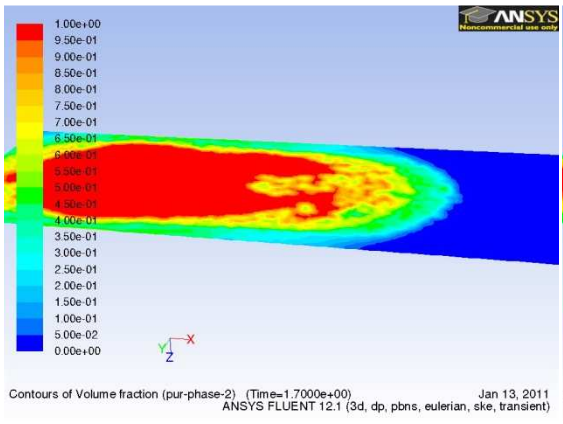
b) $t_{flow} = 0.5s$



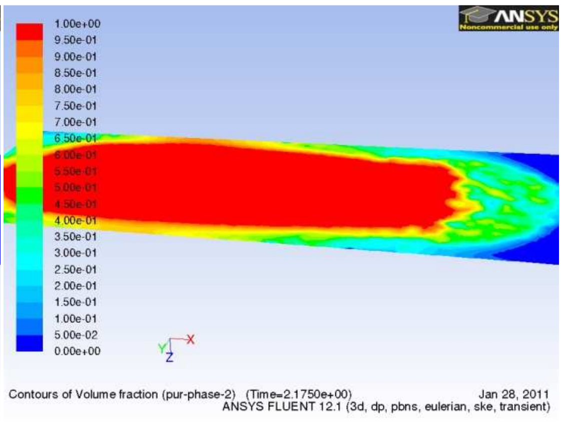
c) $t_{flow} = 0.9\ s$



d) $t_{flow} = 1.3\ s$



e) $t_{flow} = 1.7\ s$



f) $t_{flow} = 2.175\ s$

Figure 5.13: Contours of Volume Fraction (PUR-Particles) on the Plate by Eulerian Model, 600mm Distance

Figure 5.12 and Figure 5.13 show the solution of Eulerian model. Compared with VOF model, the number of rebounded particles was much less. And the plate was occupied almost everywhere by the particles.

6. Conclusions and Outlooks

Neither the VOF-simulation nor the Wall-Film simulation leads to accurate results, when qualitatively compared with the experimental data. Neither the shape of Figure 5.1 nor the position of the maximal value has been achieved by the different models.

One of the major purposes of this study is to optimize the injector model. Because of the limitation of definition in CFD software, the mobile nozzle can be only setup with a control cell. Using this injection cell, the trajectory of the spray can be well controlled. But, the momentum component of extern air cannot be modeled in a cone form according to the physics. This leads to the phenomenon, that the particles in the middle of the injection cell are obviously accelerated, while the particles at the border of the spray cone keep their initial velocity. For the further optimization of spray model, this problem should be solved.

The other major purpose of this study is to setup a VOF model to track the PUR-film on the plate. A “**wall**” boundary condition cannot be simultaneously an “inlet” boundary for the production of the VOF-film. Using UDFs, the mass source and momentum source generated by PUR-particles on “**wall**” can be collected, and with this work, the volume of fluid can be tracked. However, traces of particles by VOF model and by standard wall-film model are different. The scattered distribution of particles on the plate in the VOF simulation can be explained by an insufficient resolution of the film, due to its tinny thickness. Compared with wall-film model, the average mass of particles residing on the plate is relatively larger (about 36 times); the difference between maximum mass and minimum mass is also relatively greater (about 155 times) by VOF model. The reason for these differences might lie in the high amount of particles flying around during the simulation with the wall-film model compared to the VOF simulation. For the optimization of the whole simulation, a routine should written, which avoids the detachment of a landed particle from the substrate.

In addition, VOF model runs relatively unstable. An extra short time step size may prevent this error, but it will cost a relatively greater running time. Thus, parallel processing with more processors (computer nodes) is necessary for VOF model, especially together with DPM model.

Besides the Eulerian multiphase model should be considered as an alternative to the VOF model to study the PUR layer structure.

Appendix

A.1 Experimental and Simulation Data

position in width (mm)	mass (g)	position in width (mm)	mass (g)
-155	0	11	0.231
-143	0.020625	25	0.212025
-130	0.04125	40	0.18975
-119	0.05775	52	0.164175
-104	0.078375	65	0.13695
-91	0.10725	79	0.103125
-79	0.142725	91	0.08085
-65	0.17325	104	0.0528
-52	0.18975	117	0.043725
-40	0.20955	130	0.0231
-26	0.22935	142	0.010725
-13	0.235125	156	0
0	0.236775		

Table A.1: Experimental Data of Baytec $600mm$ Distance at $21g/s$ Flow Rate

position in width (mm)	Mass (g)	position in width (mm)	Mass (g)
-155	0	11	0.325875
-143	0.00825	25	0.30525
-130	0.0165	40	0.26895
-119	0.02145	52	0.21945
-104	0.027225	65	0.16335
-91	0.040425	79	0.104775
-79	0.073425	91	0.08085
-65	0.1221	104	0.047025
-52	0.182325	117	0.038775
-40	0.245025	130	0.022275
-26	0.284625	142	0.01155
-13	0.325875	156	0
0	0.332475		

Table A.2: Experimental Data of Baytec $400mm$, Distance at $21g/s$ Flow Rate

position in width (mm)	mass (g)	position in width (mm)	mass (g)
280.5	9.78E-03	-5.5	4.68E-02
269.5	1.07E-02	-16.5	4.65E-02
258.5	1.26E-02	-27.5	4.83E-02
247.5	1.59E-02	-38.5	4.90E-02
236.5	1.85E-02	-49.5	5.07E-02
225.5	2.08E-02	-60.5	5.17E-02
214.5	2.42E-02	-71.5	5.06E-02
203.5	2.66E-02	-82.5	5.28E-02
192.5	2.95E-02	-93.5	5.14E-02
181.5	3.42E-02	-104.5	4.88E-02
170.5	3.76E-02	-115.5	4.79E-02
159.5	3.84E-02	-126.5	4.55E-02
148.5	4.20E-02	-137.5	4.45E-02
137.5	4.32E-02	-148.5	4.23E-02
126.5	4.20E-02	-159.5	3.93E-02
115.5	4.59E-02	-170.5	3.55E-02
104.5	4.76E-02	-181.5	3.13E-02
93.5	4.76E-02	-192.5	2.89E-02
82.5	4.92E-02	-203.5	2.62E-02
71.5	5.08E-02	-214.5	2.34E-02
60.5	5.15E-02	-225.5	2.08E-02
49.5	4.94E-02	-236.5	1.90E-02
38.5	4.75E-02	-247.5	1.76E-02
27.5	4.72E-02	-258.5	1.48E-02
16.5	4.65E-02	-269.5	1.27E-02
5.5	4.64E-02	-280.5	1.20E-02

Table A.3: Wall Film Mass in Width, $600mm$ Distance at $21g/s$ Flow Rate

position in width (mm)	mass (g)	position in width (mm)	mass (g)
-280.5	0.004214	5.5	0.05167
-269.5	0.005697	16.5	0.05256
-258.5	0.00832	27.5	0.05107
-247.5	0.01185	38.5	0.05085
-236.5	0.01521	49.5	0.05437
-225.5	0.01927	60.5	0.06351
-214.5	0.02261	71.5	0.05232
-203.5	0.02603	82.5	0.06298
-192.5	0.03025	93.5	0.05759
-181.5	0.03223	104.5	0.04694
-170.5	0.03366	115.5	0.04696
-159.5	0.03663	126.5	0.03738
-148.5	0.03965	137.5	0.04129
-137.5	0.04133	148.5	0.03956
-126.5	0.04667	159.5	0.03902
-115.5	0.04311	170.5	0.03347
-104.5	0.04249	181.5	0.02905
-93.5	0.04133	192.5	0.02826
-82.5	0.0502	203.5	0.02673
-71.5	0.06192	214.5	0.02143
-60.5	0.06337	225.5	0.0157
-49.5	0.06384	236.5	0.01289
-38.5	0.05308	247.5	0.01102
-27.5	0.05083	258.5	0.00832
-16.5	0.05218	269.5	0.005987
-5.5	0.05156	280.5	0.004776

Table A.4: Wall Film Mass in Width, $400mm$, Distance at $21g/s$ Flow Rate

position in width (mm)	mass (g)	position in width (mm)	mass (g)
-280.5	0	5.5	0.059804095
-269.5	0	16.5	1.082440024
-258.5	0	27.5	2.115592342
-247.5	0	38.5	1.063116332
-236.5	1.49387E-06	49.5	0.012863889
-225.5	0.69729874	60.5	0.576115425
-214.5	1.389985824	71.5	0.807425408
-203.5	0.736879757	82.5	0.490990814
-192.5	0.099204229	93.5	1.441004592
-181.5	0.505526872	104.5	4.242591131
-170.5	1.109383604	115.5	5.944356642
-159.5	0.926383364	126.5	6.427882596
-148.5	0.197316654	137.5	6.682469704
-137.5	0.045363175	148.5	6.081573144
-126.5	0.035705812	159.5	5.246382494
-115.5	0.047622607	170.5	2.079594265
-104.5	0.945314042	181.5	0.000571034
-93.5	2.830048137	192.5	0.340412231
-82.5	3.612188333	203.5	0.928422104
-71.5	3.392262544	214.5	0.520587705
-60.5	2.005690969	225.5	0.088397501
-49.5	0.468449383	236.5	0
-38.5	0.565331929	247.5	0
-27.5	0.796051093	258.5	0
-16.5	0.519134869	269.5	0
-5.5	0.126833517	280.5	0

Table A.5: PUR Mass in Width by VOF Model, $600mm$ Distance at $21g/s$ Flow Rate

A.2 UDFs for Adjustment of Air Momentum

```
/*******************************************************************
Name: GU_Velo_correction.c

Target:
Set the air source in injection cell to get the velocity.
The air source consists of mass source and momentum source.
The calculation of the velocity during the simulation in fluent is
cell based.
Due to the combination of two sources, it is hard to control the
velocity required in the end.
In order to reach the velocity required, this UDF includes an
automatic searching.

   UDFs terms:           Definition at:
1. velo_correction       Define -> Function Hooks -> Adjust
2. corr_mass_source      Cell Zone Conditions-> Mass
3. corr_momentum_source  Cell Zone Conditions-> Z Momentum

Last edit: 21.Febr.2011
*******************************************************************/
#include "udf.h"
#include "surf.h"
#include "dpm.h"


/**********************************
Get the phase ID
**********************************/
extern int mixture_ID;

/**********************************
UDMI DATA
**********************************/
real Corr_cell = 1.0;

/**********************************
Global Variables
**********************************/
extern real geom[ND_ND];
extern real inj_init[ND_ND];
extern real air_mass_stream;

real e[ND_ND] = {1,1,1};
real mass_stream;           /* Mass steam used for mass source and
                               momentum source */
real velo_momentum;         /* Initial velocity used for momentum
                               source in DEFINE_SOURCE */
real velo_req = 25.0;       /* The velocity required for the air source
                               from the cell in middle */
real velo_delta_1 = 10.0;   /* Rough adjustment of velocity with
                               velo_delta_1 */
real velo_delta_2 = 1.0;    /* Fine adjustment of velocity with
                               velo_delta_2 */
real last_iter = 5.0;       /* After 5 iterations, it begins to search
                               the velo_req */
int iter_delta = 5.0;       /* After each 5 iterations, the velocity
                               adds itself with velo_delta*/
real c_volume;              /* Cell volume */
```

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

1. Get the injection cell.
2. Define the air source for the injection cell.
3. Look for the velocity required.
   Increase the velocity for momentum (velo_momentum)
   with dv(velo_delta) in every iter_delta iterations
   until the velocity of the cell under the marked cell
   in z positive direction (v_z) reaches to the velocity required.

*******************************************************************/
DEFINE_ADJUST(velo_correction,d)
{
  real v_z;    /* Cell based velocity of the neighbor of the injection
                  cell in z-positive direction */

  #if !RP_HOST
  real f_x,f_y,f_z;    /* Surface area in x,y,z direction*/
  real v_x,v_y;        /* Cell based velocity of the neighbor of the
                          injection cell in x,y direction */
  real c_u,c_v,c_w;    /* Three components of the cell based velocity
                          in each cell */

  real c_centroid[ND_ND];    /* Vector of cell centroid */
  real f_centroid[ND_ND];    /* Vector of face centroid */
  real NV_VEC(A);            /* Vector of face area */
  real cell_area = 0;        /* Initialing the surface areas of the
                                cell marked */

  int i = 0;   /* Number of the cells marked */
  int n;   /* Number of surfaces of the cell in the middle of model */

  Thread *t;
  cell_t c;
  Thread *tf;
  face_t f;

  CX_Cell_Id cx_cell;    /* Define the current cell */
  cell_t c_mark = 0;     /* Define and initial the marked cell */
  Thread *t_mark = NULL; /* Define and initial the marked thread */

  thread_loop_c(t,d)     /* Loop all the cell threads in the mixture
                            domain */
  {
    begin_c_loop(c,t)    /* Loop all cells in all cell thread */
    {
      if(SV_is_point_in_cell(&cx_cell,c,t,inj_init))
      /* If the point x[ND_ND] is in the cell, mark it */
      {
        i++;

        c_mark = RP_CELL(&cx_cell);      /* Get the marked cell */
        t_mark = RP_THREAD(&cx_cell);    /* Get the cell thread of
                                            this marked cell */
        C_CENTROID(c_centroid,c_mark,t_mark);  /* Get the cell
                                                  centroid of the marked
                                                  cell*/
        C_UDMI(c_mark,t_mark,0) = Corr_cell;   /* Store the marked
                                                  cell*/
        c_face_loop(c_mark,t_mark,n)     /* Loop all faces on this
                                           marked cell */
        {
          f = C_FACE(c_mark,t_mark,n);         /* Get the faces on the
                                                 marked cell */
          tf = C_FACE_THREAD(c_mark,t_mark,n); /* Get the face threads
                                                 of the marked cell */

          F_AREA(A,f,tf);                  /* Get the vector of face
                                              area on the marked cell */
          cell_area += NV_MAG(A);          /* Calculate the surface
                                              areas of the marked cell */
          F_CENTROID(f_centroid,f,tf);     /* Get all the face centroids
                                              of the marked cell */

          /* Get the cell based velocity components of the neighboring
             cells of the marked cell */
          /* If the face area directs outside of the marked cell,
             marked cell is C0.*/
         if (NV_DOT(A,e) > 0) ND_SET(c_u,c_v,c_w,
                                     C_U(F_C1(f,tf),THREAD_T1(tf)),
                                     C_V(F_C1(f,tf),THREAD_T1(tf)),
                                     C_W(F_C1(f,tf),THREAD_T1(tf)));
          /* If the face area directs inside of the marked cell,
             marked cell is C1.*/
         else if (NV_DOT(A,e) < 0)
           ND_SET(c_u,c_v,c_w,
                  C_U(F_C0(f,tf),THREAD_T0(tf)),
                  C_V(F_C0(f,tf),THREAD_T0(tf)),
                  C_W(F_C0(f,tf),THREAD_T0(tf)));

         /* Get the face of the marked cell in positive x direction*/
         if(fabs(A[0]) > 0 && f_centroid[0] > c_centroid[0])
         {
           f_x = NV_MAG(A);     /* Get the face area of this face */
           v_x = sqrt( SQR(c_u) + SQR(c_v) + SQR(c_w) );
          /* Velocity of positive-x-neighbor cell */
         }
         /* Equal in positive y and z directions */
         else if(fabs(A[1]) > 0 && f_centroid[1] > c_centroid[1])
         {
           f_y = NV_MAG(A);
           v_y = sqrt( SQR(c_u) + SQR(c_v) + SQR(c_w) );
         }
         else if(fabs(A[2]) > 0 && f_centroid[2] > c_centroid[2])
         {
           f_z = NV_MAG(A);
           v_z = sqrt( SQR(c_u) + SQR(c_v) + SQR(c_w) );
         }
        }

        printf("\n**************************************\n");
        printf(" air source at :[%g, %g, %g]\n",
               inj_init[0],inj_init[1],inj_init[2]);
        printf(" Velocity_momentum: %g, Z-Velocity:%g\n",
               velo_momentum,v_z);
        printf("\n**************************************\n");
      }
    }
    end_c_loop(c,t)
  }
  #if RP_NODE  /* Perform node synchronized actions here Does nothing
                  in Serial */
   velo_momentum = PRF_GRHIGH1(velo_momentum);
   /* send pur_on_wall_time to node-0 */
   v_z = PRF_GRHIGH1(v_z);
  #endif  /* RP_NODE */

  #endif /* !RP_HOST */

  /* Pass velo_momentum and v_z from node-0 to the Host */
  node_to_host_real_2(velo_momentum,v_z);   /* Does nothing in
                                                  SERIAL*/

  /* Rough Adjustment of velocity */
  if (v_z < (velo_req - 20.0))
  {
    if (N_ITER == last_iter)
    {
      velo_momentum += velo_delta_1;
      last_iter = last_iter +iter_delta;
    }
  }
  /* Fine Adjustment of velocity */
  else if (v_z >=(velo_req - 20.0) && v_z < velo_req)
  {
    if (N_ITER == last_iter)
    {
      velo_momentum += velo_delta_2;
      last_iter = last_iter +iter_delta;
    }
  }

  /* Pass velo_momentum and v_z from host to all the nodes */
  host_to_node_real_2(velo_momentum,v_z); /* Does nothing in SERIAL */
}
```

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

Define the mass source of air exists in the marked cell.

**********************************************************/
DEFINE_SOURCE(corr_mass_source,c,t,dS,eqn)
{
  #if !RP_HOST
  real mass_source;

  mass_stream = air_mass_stream;

   if (C_UDMI(c,t,0) == Corr_cell) /* The mass source exists in this
                                       marked cell */
  {
    c_volume = C_VOLUME(c,t);      /* Get the volume of the cell */
    mass_source = mass_stream / c_volume;   /* Mass source */
    dS[eqn] = 0;/* No dependent variables of the transport equation */
  }

  else mass_source = dS[eqn] = 0;

  return mass_source;
  #endif
}



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

Define the momentum source of air exists in the marked cell.

***********************************************************/
DEFINE_SOURCE(corr_momentum_source,c,t,dS,eqn)
{
  #if !RP_HOST
  real momentum_source;

  mass_stream = air_mass_stream;

  if (C_UDMI(c,t,0) == Corr_cell)
  {
    c_volume = C_VOLUME(c,t);
    momentum_source = mass_stream * velo_momentum / c_volume;
    dS[eqn] = 0;
  }

  else momentum_source = dS[eqn] = 0;

  return momentum_source;
  #endif
}
```

A.3 UDFs for Mobile PUR-Air Spray Injection

```
/****************************************************
Name: GU_move_injection.c

Target:
1. Define the trajectory of the moving injection
2. Define velocity of particles in the spray injection.
3. Read the position point of moving injection
   and use it for the source file "GU_air_source.c"

   UDFs terms:            Definition at:
1. INJECTION_POS          Injections -> Initialization

Last edit: 21.Ferb.2011
****************************************************/
#include "udf.h"
#include "dpm.h"
#include "surf.h"


/**********************************
Get the phase ID
**********************************/
extern int mixture_ID;

/**********************************
Global Variables
**********************************/
extern real pur_on_wall_time; /* Time when the first particle hits the
                                 wall*/
extern real flow_time;        /* Current flow time */

/* Geometry of the Model */
extern real geom[ND_ND];
extern real mesh[ND_ND];

/* Global initial position of injection */
extern real inj_init[ND_ND];

/* Read the position point of moving injection
   and use it for the source file "GU_air_source.c" */
real inj_pos[ND_ND];

real wish_vel = 13; /*Required maximum velocity of the parcel in m/s*/
int alpha = 0;      /* variable used to insure a unique (only once)
                       computation of "K" */
real K = 0;         /* linear factor between wish_vel and maximum
                       ist_vel_betrag */
```

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

Define the moving injection through changing the initial position of
particles.

*********************************************************************/
DEFINE_DPM_INJECTION_INIT(INJECTION_POS,I)
{
  #if !RP_HOST                 /* Compile this section for computing
                                  processes only (serial and node) */

  flow_time = CURRENT_TIME;    /* Get current flow time */
  int i;                       /* Item for dimension */

  real v_inj = 0.5;            /* Moving speed of injection in m/s */
  real t_tot = geom[0] / v_inj;
  real p_pos_x = 200;

  real inj_vel[ND_ND];

  Particle *p;
  cell_t c;
  Thread *t;
  Domain *d;

  d = Get_Domain(mixture_ID);  /* Get the mixture domain */


  /* Adjustment for the velocity of particles at injector*/
  /* Define the magnitude velocity of particles at injector as
     wish_vel, and retain the vector of the velocity as default */
  if (alpha ==0)
  {
    real ist_vel[ND_ND]={0.0}; /* Define and initialize the actual
                                  velocity of particles */
    real ist_vel_betrag = 0.0; /* Define and initialize the actual
                                  magnitude velocity of particles */

    loop(p,I->p_init) /* Loop all the particles at injection */
    {
      for(i=0;i<3;i++)
      ist_vel[i] = P_VEL(p)[i]; /* Get the actual velocity of
                                   particles */

      if (NV_MAG(ist_vel)> ist_vel_betrag) /* Search the max. velocity
                                              of particles */
      {
        ist_vel_betrag = NV_MAG(ist_vel);  /* Get the actual magnitude
                                              velocity of particles */
        K = wish_vel / ist_vel_betrag;     /* Get the linear factor */
      }
    }
    alpha =1;
  }
```

```
loop(p,I->p_init)
{
  c = P_CELL(p);         /* Cell index of the cell that the particle
                            is currently in */
  t = P_CELL_THREAD(p);  /* Pointer to the thread of the cell that
                            the particle is currently in */

  /* Define the moving trajectory of the injection */
  /* x-y coordinate is in the middle of the contain */
  /* It moves along the x-axis at v_inj m/s from the point
     inj_pos[ND_ND]. The injection stays at this point until a
     particle hits the wall. */
  if(pur_on_wall_time <= flow_time) /* After the first particle hits
                                       the wall */
  {
    /* Define the position of the particles under the injection */
    P_POS(p)[0] = v_inj * (flow_time - pur_on_wall_time)  +
                  inj_init[0];
    /* Define the position of the particles in the injection */
    P_INIT_POS(p)[0] = v_inj*(flow_time - pur_on_wall_time) +
                       inj_init[0];
    P_INIT_POS(p)[1] = inj_init[1];
    P_INIT_POS(p)[2] = inj_init[2];

    /* Get the position of particles in the injection */
    for(i=0;i<3;i++)
    inj_pos[i] = P_INIT_POS(p)[i];

    /* Define the velocity of the particles under the injection */
    for(i=0;i<3;i++)
    inj_vel[i] = P_VEL(p)[i];
    for(i=0;i<3;i++)
    P_VEL(p)[i] = K * inj_vel[i];

    if (p_pos_x != P_INIT_POS(p)[0])
    {
      Message("\n*********DEFINE_DPM_INJECTION_INIT**********\n");
      Message("\n At time: %g\n
                  Injector moves to: [%g, %g, %g]\n",
                  flow_time,inj_pos[0],inj_pos[1],inj_pos[2]);
       Message("\n*********DEFINE_DPM_INJECTION_INIT**********\n");
      p_pos_x = P_INIT_POS(p)[0];
    }
    Message("\n velocity of particle: [%g, %g, %g]\n",
                inj_vel[0],inj_vel[1],inj_vel[2]);
    Message("\n velocity of particle after increasing:
               [%g, %g, %g]\n",P_VEL(p)[0],P_VEL(p)[1],P_VEL(p)[2]);
  }

  else  /* Before the first particle hits the wall */
  {
   P_POS(p)[0] = inj_init[0];

    P_INIT_POS(p)[0] = inj_init[0];
    P_INIT_POS(p)[1] = inj_init[1];
    P_INIT_POS(p)[2] = inj_init[2];


    for(i=0;i<3;i++)
    inj_pos[i] = P_INIT_POS(p)[i];
```

```
      /* Define the velocity of the particles under the injection */
      for(i=0;i<3;i++)
      inj_vel[i] = P_VEL(p)[i];
      for(i=0;i<3;i++)
      P_VEL(p)[i] = K * inj_vel[i];

      if (p_pos_x != P_INIT_POS(p)[0])
      {
        for(i=0;i<3;i++)
        inj_vel[i] = P_VEL(p)[i];
        Message("\n******DEFINE_DPM_INJECTION_INIT********n");
        Message("\n At time: %g\n
                    Injector stays at:  [%g, %g, %g]\n",
                    flow_time,inj_pos[0],inj_pos[1],inj_pos[2]);
        Message("\n*******DEFINE_DPM_INJECTION_INIT******\n");
        p_pos_x = P_INIT_POS(p)[0];
      }
      Message("\n velocity of particle: [%g, %g, %g]\n",
                  inj_vel[0],inj_vel[1],inj_vel[2]);
      Message("\n velocity of particle after increasing:
                 [%g, %g, %g]\n",P_VEL(p)[0],P_VEL(p)[1],P_VEL(p)[2]);
    }
  }

  #endif /* !RP_HOST */
}
```

A.4 UDFs for DPM Model

```
/******************************************************************
Name: GU_DPM_VOF.c

Model:
real geom[ND_ND] = {1.5, 0.561, 0.65};
real mesh[ND_ND] = {120, 51, 40};
Target:
1. Set names of user-defined memories (UDMI)
   using udf "on_loading".
2. Get the information of mass and velocity of particles
   hitting the wall using udf
   "Particle_Infos" and "plot_and_store".
3. With the information from Target 2,
   define the mass and momentum sources of PUR-particles
   for the VOF Model using UDFs written by DEFINE_SOURCE.

   UDFs terms:            Definition at:
1. on_loading             automatically loading when compiling UDFs
2. Particle_Infos         Boundary Conditions-> wall-> Mixture -> DPM
3. p_mass_source          Cell Zone Conditions-> Phase-2-> Mass
4. p_x_momentum_source    Cell Zone Conditions-> Mixture-> X Momentum
5. p_y_momentum_source    Cell Zone Conditions-> Mixture-> Y Momentum
6. p_z_momentum_source    Cell Zone Conditions-> Mixture-> Z Momentum
7. plot_and_store         Define -> Function Hooks -> Execute at End

Last edit: 21.Febr.2011
******************************************************************/
#include "udf.h"
#include "dpm.h"
#include "sg.h"


/**********************************
Model from Gambit
**********************************/
real geom[ND_ND] = {1.5, 0.561, 0.65};
real mesh[ND_ND] = {120, 51, 40};

/**********************************
Get ID from ANSYS FLUENT
**********************************/
int mixture_ID = 1;    /* ID for mixture phase */
int secondary_ID = 3;  /* ID for secondary phase */
int wall_ID = 5;       /* ID for wall boundary */

/**********************************
Global Variables
**********************************/
/* Global initial position of injection */
real inj_init[ND_ND] = {0.2111, 0, 0.0511};  /* The initial position
                                                 of injection */

/* Global variables used by serial, host, node versions */
real pur_c_volume;  /* Volume of the cell where particles reside */
real pur_f_centroid[ND_ND];  /* Centroid of the face where particles
                                reside */
real pur_c_centroid[ND_ND];  /* Centroid of the cell where particles
                                reside */
real curr_ts_size;           /* Current time step */
real flow_time;                /* Current flow time */
real  pur_on_wall_time  =  100.0;  /*  Assume  the  time  particle  hits  the
                                      wall  is  100.  This  time  is  updated
                                      if a particle hits the wall. */
real  PUR_cell  =  5.0;      /*  UDMI  number  of  the  cell  where  particles
                                reside */
real p_vel_max = 25.0;      /*  The  velocity of particles on plate cannot
                                be greater than p_vel_max m/s */
real p_vel_soll = 0.1;

extern real velo_req;




/**********************
Setting names of UDMI
**********************/
DEFINE_EXECUTE_ON_LOADING(on_loading, libname)
{
    Set_User_Memory_Name(0,"Source_cell");
    Set_User_Memory_Name(1,"Num_Particle(t)");
    Set_User_Memory_Name(2,"Mass_Particle(t)");
    Set_User_Memory_Name(3,"Deltat_mass_Particle");
    Set_User_Memory_Name(4,"Mass_Particle(t-dt)");
    Set_User_Memory_Name(5,"Mass_Stream");
    Set_User_Memory_Name(6,"Impluse_x(t)");
    Set_User_Memory_Name(7,"Impluse_y(t)");
    Set_User_Memory_Name(8,"Impluse_z(t)");
    Set_User_Memory_Name(9,"Deltat_Num_Parcel");
    Set_User_Memory_Name(10,"Num_Parcel(t-dt)");
    Set_User_Memory_Name(11,"ave_Impluse_x(t)");
    Set_User_Memory_Name(12,"Impluse_x(t-dt)");
    Set_User_Memory_Name(13,"ave_Impluse_y(t)");
    Set_User_Memory_Name(14,"Impluse_y(t-dt)");
    Set_User_Memory_Name(15,"ave_Impluse_zx(t)");
    Set_User_Memory_Name(16,"Impluse_z(t-dt)");
    Set_User_Memory_Name(17,"Num_Parcel(t)");
}
```

```
/*********************************************************************
Store the information of the particles on the wall for all the time.
*********************************************************************/
DEFINE_DPM_BC(Particle_Infos,p,t,f,f_normal,dim)
{
  #if !RP_HOST
  Thread  *p_cell_thread;     /*  Threads  of  the  cells  where  particles
                                 reside */

  cell_t p_cell;              /* Cells where particles reside */
  real p_n;                   /* Number of particles in a parcel */
  real p_mass;                /* Mass of each particle */
  real pur_vel[ND_ND];        /* Velocity of each particle */
  int i;                      /* vector number*/
  flow_time = CURRENT_TIME;   /* Get current flow time */

  /* return cell index in which face "f" is */
  p_cell = F_C0(f,t);
  /* return cell thread index in which face thread "t" is */
  p_cell_thread = THREAD_T0(t);
  /* Get the face centroid from ANSYS FLUENT */
  F_CENTROID(pur_f_centroid,f,t);
  /* Get the number of the particles in a parcel from FLUENT */
  p_n = P_N(p);
  /* Get the mass of the particle from ANSYS FLUENT */
  p_mass = P_MASS(p);
  /* Get the velocity of the particle from ANSYS FLUENT */
  for(i=0;i<dim;i++)
    pur_vel[i] = P_VEL(p)[i];

  /* When the particles hit the wall face, mark the cell,
     which consists this face.*/
  C_UDMI(p_cell,p_cell_thread,0) = PUR_cell;

  /* Store the number of particles hitting this face all the time. */
  C_UDMI(p_cell,p_cell_thread,1) += p_n;

  /************************************************
  Information of particles used for mass source
  ************************************************/
  /* mass of particles hitting the face */
  C_UDMI(p_cell,p_cell_thread,2) += p_n * p_mass;

  /*************************************************
  Information of particles used for momentum source
  *************************************************/
  /* Impulse = mass * velocity */
  /* Total impulse of all parcels in x direction in a cell */
  C_UDMI(p_cell,p_cell_thread,6) += p_n * p_mass * pur_vel[0];
  /* Equal in y and z direction */
  C_UDMI(p_cell,p_cell_thread,7) += p_n * p_mass * pur_vel[1];
  C_UDMI(p_cell,p_cell_thread,8) += p_n * p_mass * pur_vel[2];

  /*****************************************************************
   Store the number of parcels hitting on this face all the time.
   *****************************************************************/
  C_UDMI(p_cell,p_cell_thread,17) += 1;

  return PATH_ABORT;
  #endif
}
```

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

This function is used only for wall film model, to search the time
when the first
particle is hitting on the wall face.

By VOF model, this time is obtained through function "Particle_Infos"
and
"plot_and_store", which is hooked into DPM_BC as user defined
function.

However, when the wall film model is running, DPM_BC is defined as
"wall film",
so another function which can get the particle_on_wall_time can be
hooked into
another place should be compiled.

DEFINE_DPM_EROSION is hooked for Erosion/Accretion from
Discrete Phase Model dialog box.

*********************************************************************/
DEFINE_DPM_EROSION(on_wall_time,p,t,f,normal,alpha,Vmag,mdot)
{
  #if !RP_HOST
  Thread *p_cell_thread;
  cell_t p_cell;

  /* return cell index in which face "f" is */
  p_cell = F_C0(f,t);
  /* return cell thread index in which face thread "t" is */
  p_cell_thread = THREAD_T0(t);

   /******************************************************************
    Mark the cell, when there are particles hit the wall face, which
   consists this face.
 ********************************************************************/
  C_UDMI(p_cell,p_cell_thread,0) = PUR_cell;

  #endif
}




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

Define the mass source of the particles hitting on the wall

***********************************************************/
DEFINE_SOURCE(p_mass_source,c,t,dS,eqn)
{
  #if !RP_HOST /* Compile this section for computing processes only
                (serial and node) */
  real pur_mass_source;

  /* Get the volume of the cell from FLUENT*/
  pur_c_volume = C_VOLUME(c,t);

  /* When the particles hit on the face, define the mass source*/
  if(C_UDMI(c,t,0) == PUR_cell)
  {
    /* Mass source of the particle in a face */
    pur_mass_source = C_UDMI(c,t,5) / pur_c_volume;
    /* No dependent variables of the transport equation */
    dS[eqn] = 0;
  }
  else
    pur_mass_source = dS[eqn] = 0;

  return pur_mass_source;

  #endif  /* !RP_HOST */
}


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

Define the momentum source in x direction aroused
by the particles hitting on the wall

*********************************************************************/
DEFINE_SOURCE(p_x_momentum_source,c,t,dS,eqn)
{
  #if !RP_HOST /* Compile this section for computing processes only
                  (serial and node) */
  real pur_momentum_source_x;

  pur_c_volume = C_VOLUME(c,t);

  if(C_UDMI(c,t,0) == PUR_cell)
  {
    /* Momentum source of the particle in a face */
    pur_momentum_source_x = C_UDMI(c,t,11)/ pur_c_volume;
    dS[eqn] = 0;
  }
  else
    pur_momentum_source_x = dS[eqn] = 0;

  return pur_momentum_source_x;
  #endif  /* !RP_HOST */
}
```

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

Define the momentum source in y direction aroused
by the particles hitting on the wall

**********************************************************************/
DEFINE_SOURCE(p_y_momentum_source,c,t,dS,eqn)
{
  #if !RP_HOST /* Compile this section for computing processes only
                  (serial and node)  */
  real pur_momentum_source_y;

  pur_c_volume = C_VOLUME(c,t);

  if(C_UDMI(c,t,0) == PUR_cell)
  {
    pur_momentum_source_y = C_UDMI(c,t,13)/ pur_c_volume;
    dS[eqn] = 0;
  }
  else
    pur_momentum_source_y = dS[eqn] = 0;

  return pur_momentum_source_y;
  #endif  /* !RP_HOST */
}


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

Define the momentum source in z direction aroused
by the particles hitting on the wall

**********************************************************************/
DEFINE_SOURCE(p_z_momentum_source,c,t,dS,eqn)
{
  #if !RP_HOST /* Compile this section for computing processes only
                  (serial and node)  */
  real pur_momentum_source_z;

  pur_c_volume = C_VOLUME(c,t);

  if(C_UDMI(c,t,0) == PUR_cell)
  {
    pur_momentum_source_z = C_UDMI(c,t,15)/ pur_c_volume;
    dS[eqn] = 0;
  }
  else
    pur_momentum_source_z = dS[eqn] = 0;

  return pur_momentum_source_z;
  #endif  /* !RP_HOST */
}
```

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

DEFINE_EXECUTE_AT_END runs after DEFINE_DPM_BC at each end of a time
step. In this function, current mass, mass source and momentum source
of particles are calculated just after the running of DEFINE_DPM_BC.
And it store the total mass of particles and total impulse of parcels
in current time step, for using in next time step as previous mass and
previous impulse.

**********************************************************************/
int counter = 0;
int p_large = 0;
int air_large = 0;

DEFINE_EXECUTE_AT_END(plot_and_store)
{
  #if !RP_HOST /* Compile this section for computing processes only
                  (serial and node)  */
  Domain *d;
  Thread *t;
  cell_t c;
  Injection *I;
  Injection *dpm_injections = Get_dpm_injections();
  Particle *p;

  d = Get_Domain(mixture_ID); /* Get mixture domain */

  int dim = 3;
  int i;
  int only_one = 0;
  real wall_vel_max = 0.0;
  real wall_vel = 0.0;
  real mixture_vel = 0.0;
  curr_ts_size = CURRENT_TIMESTEP;  /* Get current time step */
  flow_time = CURRENT_TIME;  /* Get current flow time */

  thread_loop_c(t,d)
  {
    begin_c_loop_int(c,t)
    {
      if(C_UDMI(c,t,0) == PUR_cell)
      {
        if (counter == 0)
        {
          pur_on_wall_time = flow_time;
          counter = 1;
        }
        if (only_one == 0)
        {
          Message("\n IN RP_NODE-%d, from time: %g,
                         Particles are hitting the wall.\n",
                         myid,pur_on_wall_time);
          only_one = 1;
        }

        /***************************************
         Begin running for mass source
        ***************************************/
        /* mass_deltat = mass(t) -mass(t-t-deltat) */
        C_UDMI(c,t,3) = C_UDMI(c,t,2) - C_UDMI(c,t,4);
        /* mass stream = mass_deltat / t_deltat */
        C_UDMI(c,t,5) = C_UDMI(c,t,3) / curr_ts_size;

        /* Store the total mass of particles in current time step,
           for using in next time step as previous mass. */
        C_UDMI(c,t,4) = C_UDMI(c,t,2);


        /***************************************
         Begin running for momentum source
        ***************************************/
        /* Begin running for x momentum source*/
        /* impulse stream (dI) in x direction of a cell =
           current total impulse of all particles in current time step
           / current number of particles */
        C_UDMI(c,t,11) = ( C_UDMI(c,t,6) - C_UDMI(c,t,12) )
                         / curr_ts_size;
        /* Store the total impulse of parcels in current time step,
           for using in next time step as previous impulse.*/
        C_UDMI(c,t,12) = C_UDMI(c,t,6);

        /* Equal in y and z direction */
        /* Begin running for y momentum source*/
        C_UDMI(c,t,13) = ( C_UDMI(c,t,7) - C_UDMI(c,t,14) )
                         / curr_ts_size;
        C_UDMI(c,t,14) = C_UDMI(c,t,7);

        /* Begin running for z momentum source*/
        C_UDMI(c,t,15) = ( C_UDMI(c,t,8) - C_UDMI(c,t,16) )
                         / curr_ts_size;
        C_UDMI(c,t,16) = C_UDMI(c,t,8);
      }

      mixture_vel = sqrt( SQR(C_U(c,t))
                          + SQR(C_V(c,t)) + SQR(C_W(c,t)) );

      /******************************************************
      adjust the cells whose velocity is too large to
      lead to the convergence difficulties.
      ******************************************************/
      if (mixture_vel > velo_req)
      {
        C_CENTROID(pur_c_centroid,c,t);
        air_large++;
        C_U(c,t) = 0;
        C_V(c,t) = 0;
        C_W(c,t) = 1;
        Message("\n***Adjust the air with large velocity***\n");
        Message("\n The mixture_vel: %g.
                    After changing: %g, position:[%g, %g, %g]",
                    mixture_vel,
                    sqrt(SQR(C_U(c,t))+SQR(C_V(c,t))+SQR(C_W(c,t))),
                    pur_c_centroid[0],
                    pur_c_centroid[1], pur_c_centroid[2]);
        Message("\n***Adjust the air with large velocity***\n");
      }
    }
    end_c_loop_int(c,t)
  }
  /*****************************************************
  Adjust the particles whose velocity is too large to
  lead to the convergence difficulties.
  *****************************************************/
  loop(I,dpm_injections)/* Loop all particles in the mixture domain */
  {
    loop(p,I->p)
    {
      real pur_vel_mag;
      pur_vel_mag = NV_MAG(P_VEL(p));
      c = P_CELL(p);
      t = P_CELL_THREAD(p);
      if (pur_vel_mag > p_vel_max)
      {
        p_large++;
        C_CENTROID(pur_c_centroid,c,t);
        for(i=0;i<dim;i++)
        P_VEL(p)[i] = p_vel_soll;
        Message("\n***Adjust the particle with large velocity***\n");
        Message("\n The pur_vel_mag: %g.
                    After changing: %g, position:[%g, %g, %g]",
                    pur_vel_mag, (NV_MAG(P_VEL(p))),
                    pur_c_centroid[0],pur_c_centroid[1],
                    pur_c_centroid[2]);
        Message("\n***Adjust the particle with large velocity***\n");
      }
    }
  }

  #if RP_NODE  /* Perform node synchronized actions here Does nothing
                  in Serial */
  /* send pur_on_wall_time to node-0 */
  pur_on_wall_time = PRF_GRLOW1(pur_on_wall_time);

  air_large = PRF_GRSUM1(air_large);
  p_large = PRF_GRSUM1(p_large);

  Message("\n The mixture velocity of %d cells,
              the velocity of %d particles are too large.",
              air_large,p_large);

  #endif  /* RP_NODE */

  #endif  /* !RP_HOST */

  /* Pass the pur_on_wall_time from node-0 to the Host */
  node_to_host_real_1(pur_on_wall_time);
  /* Pass the pur_on_wall_time from host to all the nodes */
  host_to_node_real_1(pur_on_wall_time);
}
```

A.5 UDFs for Air Source

```
/**************************************************
Name: GU_air_source.c

Target:
This Source file is used to define the air source for spray injection.
It includes mass source and momentum source of air.

    UDFs terms:           Definition at:
1. air_mass_source       Cell Zone Conditions-> Phase-1-> Mass
2. air_momentum_source   Cell Zone Conditions-> Mixture-> Z Momentum

Last edit: 21.Feb.2011
****************************************************************/
#include "udf.h"
#include "dpm.h"



/********************************
Global Variables
*******************************/
real air_c_centroid[ND_ND];  /* Centroid of the injection cell */
real air_velo_momentum = 0.0;
real air_mass_stream = 6.125e-3; /* Mass steam used for mass source
                                    and momentum source, in kg/s */
real air_c_volume;           /* Volume of the injection cell */
extern real inj_pos[ND_ND];
extern real velo_momentum;
```

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

Define the mass source of air exists in the marked cell.

**********************************************************/
DEFINE_SOURCE(air_mass_source,c,t,dS,eqn)
{
  #if !RP_HOST /* Compile this section for computing processes only
                (serial and node)  */
  real air_mass_source;

  CX_Cell_Id cx_cell;    /* Define the current cell */
  cell_t c_mark = 0;     /* Define and initial the marked cell */
  Thread *t_mark = NULL; /* Define and initial the marked thread */

  if (inj_pos[0] != 0.)  /* Excludes this variables as 0.0
                            in the other computer node or host */
  {
    if(SV_is_point_in_cell(&cx_cell,c,t,inj_pos))
    /* If the injection point in this cell */
    {
      c_mark = RP_CELL(&cx_cell);      /* Get the marked cell */
      t_mark = RP_THREAD(&cx_cell);    /* Get the thread of this
                                          marked cell */
      C_CENTROID(air_c_centroid,c_mark,t_mark); /* Get the cell
                                                   centroid of the
                                                   marked cell*/
      air_c_volume = C_VOLUME(c_mark,t_mark);   /* Get the volume of
                                                   the cell from ANSYS
                                                   FLUENT*/

      air_mass_source = air_mass_stream / air_c_volume;  /* Mass
                                                            source */
      dS[eqn] = 0;                     /* No dependent variables of
                                          the transport equation */

      Message("\n***************air source****************\n");
      Message("\n air inj_pos: [%g, %g, %g]\n",
                inj_pos[0],inj_pos[1],inj_pos[2]);
      Message("\n air source centroid: [%g, %g, %g]\n",
              air_c_centroid[0],air_c_centroid[1],air_c_centroid[2]);
      Message("\n***************air source****************\n");
    }
  }
  else air_mass_source = dS[eqn] = 0;

  return air_mass_source;

  #endif  /* !RP_HOST */
}
```

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

Define the momentum source of air exists in the marked cell.

***********************************************************/
DEFINE_SOURCE(air_momentum_source,c,t,dS,eqn)
{
  #if !RP_HOST
  real air_momentum_source;

  air_velo_momentum = RP_Get_Real("air/velo_momentum");
  /* initial velocity used for momentum source in DEFINE_SOURCE */
  /* To set up this user Scheme variable in cortex type */
  /* (rp-var-define ' air/velo_momentum 1 ' real #f) */
  /* After set up you can change it to another variable: */
  /* (rpsetvar ' air/velo_momentum 34) */

  CX_Cell_Id cx_cell;
  cell_t c_mark = 0;
  Thread *t_mark = NULL;

  if (inj_pos[0] != 0.)
  {
    if(SV_is_point_in_cell(&cx_cell,c,t,inj_pos))
    {
      c_mark = RP_CELL(&cx_cell);
      t_mark = RP_THREAD(&cx_cell);
      air_c_volume = C_VOLUME(c_mark,t_mark);

      air_momentum_source = air_mass_stream * air_velo_momentum
                            / air_c_volume;
      dS[eqn] = 0;
    }
  }
  else air_momentum_source = dS[eqn] = 0;

  return air_momentum_source;

  #endif /* !RP_HOST */
}
```

Bibliography

[1] ANSYS FLUENT 12.0 Theory Guide, ANSYS, Inc., April, 2009.

[2] ANSYS FLUENT 12.0 User’s Guide, ANSYS, Inc., April, 2009.

[3] D. W. Stanton and C. J. Rutland. Modeling Fuel Film Formation and Wall Interaction in Diesel Engines. SAE Paper 960628, 1996.

[4] P. J. O’Rourke and A. A. Amsden. A Spray/Wall Interaction Submodel for the KIVA-3 Wall Film Model. SAE Paper 2000-01-0271, 2000.

[5] P. J. O’Rourke and A. A. Amsden. A Particle Numerical Model for Wall Film Dynamics in Port-Fuel Injected Engines. SAE Paper 961961, 1996.

[6] Modeling Multiphase Flows, Introductory FLUENT Training, ANSYS Inc, 2008.

[7] C. W. Hirt and B. D. Nichols. Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries. J. Comput. Phys., 39:201–225, 1981.

[8] D. L. Youngs. Time-Dependent Multi-Material Flow with Large Fluid Distortion. In K. W. Morton and M. J. Baines, editors, Numerical Methods for Fluid Dynamics. Academic Press, 1982.

[9] O. Ubbink. Numerical Prediction of Two Fluid Systems with Sharp Interfaces. PhD thesis, Imperial College of Science, Technology and Medicine, London, England, 1997.

[10] ANSYS FLUENT 12.0 UDF Manual, ANSYS, Inc., April, 2009.

[11] FLORIAN HELPA, Dipolom Thsis, HENNECKE GMBH, 2006.

[12] Gambit 2.4, Modeling Guide, Volume 2, Fluent, Inc., May, 2007.

[13] ANSYS FLUENT 12.0 Tutorial Guide, ANSYS, Inc., April, 2009.

[14] Practices for the VOF Model, On-line FLUENT Training, ANSYS, Inc., 2006.

[15] Volume Of Fluid Model, Advanced Multiphase Modeling Course, ANSYS, Inc., 2007.

[16] Genong Li, Modeling Spray Behavior: Tools and Applications, Prepared for Fluent UGM2003.