VON KARMAN INSTITUTE FOR FLUID DYNAMICS
TURBOMACHINERY & PROPULSION DEPARTMENT

UNIVERSITÉ DE LA ROCHELLE - UFR SCIENCE ET TECHNOLOGIE
ECOLE DOCTORALE SCIENCES ET INGÉNIERIE EN MATÉRIAU, MÉCANIQUE, ÉNERGÉTIQUE ET AÉRONAUTIQUE
LABORATOIRE DES SCIENCES DE L'INGÉNIEUR POUR L'ENVIRONNEMENT

ROYAL MILITARY ACADEMY
DEPARTMENT OF MECHANICAL ENGINEERING

Numerical modeling and experimental investigation of fine particle coagulation and dispersion in dilute flows

Cover art: Particle-laden turbulent channel flow, with vertical cuts and streamlines colored by streamwise particle velocity and the bottom wall colored by particle concentration.

Thesis presented by Bart Janssens in order to obtain the degree of "Doctor of Philosophy in Applied Sciences - Mechanics", Université de la Rochelle, France and Royal Military Academy, Belgium, 10th of July 2014.

Promoter: Prof. Tony Arts (von Karman Institute for Fluid Dynamics, Belgium)
Supervisor: Prof. Walter Bosschaerts (Royal Military Academy, Belgium)
Co-Supervisor: Karim Limam (Maître de conférences, Université de La Rochelle, France)

Doctoral Committee:

Prof. Gérard Degrez (Université Libre de Bruxelles, Belgium)
Asst Prof. Benoît Marinus (Royal Military Academy, Belgium)
Prof. Hassane Naji (Université d'Artois, France)
Asst Prof. Elmar Recker (Royal Military Academy, Belgium)
Prof. Dirk Saelens (KU Leuven, Belgium)

A selection of doctoral theses published by the von Karman Institute:

Benefits of flow control on compressor blade aerodynamics
(F.C. Şahin, Université Catholique de Louvain, Belgium, June 2014)

Spectroscopic measurements of sub- and supersonic plasma flows for the investigation of atmospheric re-entry shock layer radiation
(D. Le Quang Huy, Université Blaise Pascal, France, June 2014)

Development of a high temperature cooled fast response probe for gas turbine applications
(M. Mersinligil , Université Catholique de Louvain, Belgium, May 2014)

Experimental investigation of induced supersonic boundary layer transition
(H. Bottini, Universidad Politecnica de Catalunya (UPC), Spain, March 2014)

Multi-scale models and computational methods for aerothermodynamics
(A. Munafo, École Centrale de Paris, France, January 2014)

Experimental aerothermal performance of turbofan bypass flow heat exchangers
(L. Villafañe, Universidad Politécnica de Valencia, Spain, December 2013)

A full catalogue of publications is available from the library.

Numerical Modeling and Experimental Investigation Fine Particle Coagulation and Dispersion in Dilute Flows

Keywords: finite element method, disperse multiphase flow, Eulerian method, particle coagulation, experimental validation

D/2014/0238/649, T. Magin, Editor-in-Chief
Published by the von Karman Institute for Fluid Dynamics with permission.

ISBN 978-2-87516-081-2

For Francis, in memoriam.

Contents

Abstract

The present work deals with the development of a framework for the modeling of dispersed flows, including the effect of coagulation on the particle size distribution. We also explore some techniques for experimental validation. Models are developed for incompressible, isothermal flow containing particles that have a small relaxation time compared to the fluid time scale.

For the dispersed phase, an equilibrium Eulerian approach is used, extrapolating the particle velocity from the fluid velocity. The size distribution is modeled using the Direct Quadrature Method of Moments. In practice, this results in solving transport equations for the weights and abscissa of a Dirac delta approximation of the size distribution. To model the effect of coagulation, a collision kernel that makes use of the resolved instantaneous velocity is developed.

All transport equations are solved using the Finite Element Method. For the fluid, the Streamline Upwind and Pressure Stabilized Petrov-Galerkin method are used, with additional grad-div stabilization. To decrease the solution time for DNS, a segregated formulation with an explicit advection term is proposed. The particle transport equations require cross-wind diffusion in addition to the streamline upwind stabilization when large gradients occur.

All work is available in the open source Coolfluid 3 framework, using an Embedded Domain Specific Language we developed for the implementation of finite element models. The resulting code closely resembles the variational form of the equations and is generic in terms of element type and the number of spatial dimensions.

A first validation uses literature results as reference. Correctness and accuracy of the methods are verified using the Taylor-Green vortex flow. For the fluid and particle concentration, direct numerical simulation of a turbulent channel flow is performed. The particle coagulation kernel is tested using particles of different sizes falling through a Burgers vortex.

Finally, some experimental validation techniques are used on a small test chamber. Particle image velocimetry is used for the fluid motion, while the size distributions are measured using Phase Doppler Anemometry and Multiple Wavelength Light Extinction. The light extinction technique was found to produce size distributions that could provide valuable reference data for our particle model.

Acknowledgments

The successful completion of this work would not have been possible without the help of many people, so I would like to express my gratitude here.

First of all I thank my promotors, Prof. Arts, Prof. Bosschaerts and Prof. Limam for offering me the opportunity to do this work and for their valuable support throughout the research. I am grateful for the great amount of freedom they allowed, without which it would have been impossible to delve into the C++ programming aspects of the work. I also thank the reporters and other members of the jury for their questions and comments on the manuscript.

I am also most grateful to the Coolfluid 3 team. Tiago, thanks for introducing me to the team and for involving me in the Coolfluid 3 transition. Tamás, thank you for your "How to implement a Navier-Stokes solver in 10 easy steps" tutorial, and the countless intersting discussions and debugging sessions (yes there were bugs but they are all gone now). Willem, thank you for working out lots of the Coolfluid 3 basics, especially the wonderful mesh structure that is essential for all other work.

Of course my colleagues at the RMA were always ready to lend a hand, and I thank all of them. Elmar, thank you for the many hours you helped me with the experiments and all the useful comments on my work. Benoît, many thanks for all the interesting discussions we had as well as the detailed comments made on the manuscript.

The most interesting results in the experimental work are without a doubt the light extinction measurements. Many thanks, Imre, for making this possible and for all the help with the experiments and discussions about the results.

Finally, I thank my family and especially my lovely wife. Lynn, thank you for your love and support during all these years, I know it was difficult at times. I know your father would have loved to see me finish this work, so I dedicate it to him. He is sorely missed.

Nomenclature

Roman Symbols

$\boldsymbol{a}$	Acceleration	[m/s^2]
A_e	Element stiffness matrix	
C_α	Cunningham correction for phase α	
d_p	Particle diameter	[m]
d_α	Diameter associated with particle phase α	[m]
D_b	Diffusion coefficient due to Brownian motion	[m^2/s]
f	Particle number distribution function	[m^{-6}]
f_α	Particle number distribution function value at v_α	[m^{-6}]
$\boldsymbol{g}$	Acceleration associated with the body force	[m/s^2]
g_{ij}	Element metric tensor	[m^{-2}]
h	Element length scale	[m]
k_B	Boltzmann constant, $1.3806488 \cdot 10^{-23}$	[J/K]
Kn	Knudsen number	
L	Linear operator	
n	Particle number concentration	[m^{-3}]
n_t	Total particle number concentration for all sizes	[m^{-3}]
N	Number of elements or number of phases	
N_u	Element shape function vector for the variable u	
p	Kinematic pressure	[m^2/s^2]
r_p	Particle radius	[m]
$R_{\alpha\gamma}$	Particle collision radius (i.e. the sum of radii)	[m]
t	Time	[s]
Δt	Time step	[s]
T_e	Element mass matrix	
$\boldsymbol{u}$	Fluid velocity vector	[m/s]
$\tilde{u}$	Value of unknown u interpolated by its associated shape functions	
$\boldsymbol{u}_e$	Element vector of unknown coefficients associated with variable u	
U_a	Horizontal Taylor-Green background velocity	[m/s]
U	Mean x-velocity	[m/s]
U_c	Mean centerline x-velocity in the experiment	[m/s]
U_m	Mean x-velocity across the inlet height	[m/s]
$\boldsymbol{v}$	Particle velocity vector	[m/s]
v_p	Particle volume	[m^3]
v_α	Particle volume associated with particle phase α	[m^3]

V_a	Vertical Taylor-Green background velocity	[m/s]
V_s	Initial maximal Taylor-Green swirl velocity	[m/s]
V	Mean y-velocity	[m/s]
V	Volume	[m^3]
$\boldsymbol{w}$	Relative velocity between particles	[m/s]
w_r	Radial component of the relative velocity between particles	[m/s]
W	Mean z-velocity	[m/s]
$\boldsymbol{x}$	Position vector	[m]
$\boldsymbol{x}_e$	Vector of unknowns for an element	
$\boldsymbol{x}_{u_i}$	Block of the element vector for component i of $\boldsymbol{u}$	
$\tilde{x}$	Finite element approximation of x	

Greek Symbols

α_p	Particle volume fraction	
β	Density ratio parameter	
η_k	Kolmogorov length scale	[m]
μ	Dynamic viscosity of the fluid	[kg/ms]
μ_g	Geometric mean of the particle diameter	[m]
ν	Kinematic viscosity of the fluid	[m^2/s]
ω	Vorticity	[s^{-1}]
Ω_e	Volume of a single element	[m^3]
ρ_f	Density of the fluid	[kg/m^3]
ρ_p	Density of the particle	[kg/m^3]
σ_{ab}	Collision cross section for two particles a and b	[m^2]
σ_g	Geometric standard deviation of d_p	
θ	Parameter for the theta-scheme	
τ	Generalized particle relaxation time	[s]
τ_c	Average time between particle collisions	[s]
τ_k	Kolmogorov time scale	[s]
τ_p	Particle relaxation time	[s]
τ_{PS}	PSPG stabilization coefficient	[s]
τ_{SU}	SUPG stabilization coefficient	[s]
τ_{BU}	Grad-div (bulk) stabilization coefficient	[m^2/s]
ζ_α	Weighted particle diameter $c_\alpha d_\alpha$ for phase α	[m]

Chapter 1

Introduction

1.1 Background

The behavior of fine particles in a carrier fluid is of importance in many fields. One area of concern is the effect of particle pollution in the air we breathe: small particles enter the body through the respiratory system or the skin and may constitute a health risk [1]. This pollution consists of both solid and liquid particles that are much heavier than the air that carries them. A wide range of sizes can be observed, with the smallest particles starting at a few nanometers. Particles larger than 10 µm do not penetrate into the respiratory system and are not considered in most studies. Particles in the size range below 100 nm are thought to be more hazardous [2], though the chemical composition of the particles strongly affects their toxicity [3]. These nanoparticles only make up a small portion of the total mass of the particles, but they are usually present in high numbers. In the case of high number concentration and polydisperse particle distributions, the effect of particle coagulation is enhanced.

Proper understanding of the behavior of these particles can also be applied to the development of air handling units that are able to filter out the pollutants.

A related area is the study of soot particles in exhaust plumes [4]. Particle number concentrations near the point of emission are high enough to promote coagulation, so we must take this effect into account to obtain a correct particle size distribution. In this context, a detailed study of the evolution of soot particles through a turbine would also be possible.

Volcanic aerosols in the high altitude atmosphere also fall into the size range that is considered here. This kind of aerosol is transported over long distances during long periods of time (weeks or months). Numerical modeling can aid in assessing the risk that these aerosols pose to air traffic [5].

Finally, in the context of defense and security, it is useful to be able to predict the effects of an attack using chemical or biological aerosols, such as anthrax spores [6]. A practical model could be used to evaluate possible release scenarios in a building, allowing for the construction of a database of emergency response plans.

1.2 Scope and objectives

The main objective of the current work is to develop a numerical model for the behavior of discrete particles in a continuous carrier phase. The particles are assumed to be heavy with respect to the carrier phase, with a size range from 10 nm to 10 µm when we use air at standard conditions. The continuous phase is restricted to an isothermal, incompressible flow.

We assume that the motion of the particles is governed by the influence of the flow rather than inter-particle collisions — i.e. a "dilute" regime. Furthermore, the influence of the particles on the continuous phase is neglected (one-way coupling). We do consider the effect of particle coagulation, i.e. particles may collide and form larger particles. The term coagulation is used synonymously with aggregation here, and applies to both droplets and solid particles.

The numerical work is part of Coolfluid 3, a cutting-edge C++ framework [7]. The actual numerical models are implemented using an embedded domain specific language, allowing for a concise formulation that stays close to the original mathematical formulation. This abstraction greatly improves code readability and simplifies experimentation with the many different models that can be found in the literature. This kind of simplification usually implies a performance penalty, but this effect is negligible in our code.

Finally, the model needs to be validated. We use available data from literature as well as our own experiment. The objective of the experiment is to investigate the feasibility of producing validation data sets that are detailed enough to validate the model. It is beyond the scope of this work, however, to produce an actual quantitative validation from our own experiments.

The original contribution of the work lies in the development of a finite element model for a dispersed flow that takes into account the effect of coagulation on the size distribution. Many models are described in literature, but the effect of coagulation is rarely taken into account. We develop a coagulation kernel that relies on DNS data of the fluid model, intended as a basis for future development of coarser models. The implementation using a domain specific language is also new in this field, and helps to ensure the continued performance, maintainability and extensibility of the code. Finally, the lack of experimental data on particle coagulation in the literature led to the development of our own experimental setup to explore some new validation techniques.

1.3 Thesis outline

After this introduction, the work is divided into three parts: modeling, validation and conclusions. Chapter 2 starts with an overview of some fundamental concepts for disperse flows. Next, we provide an overview of the

available methods for modeling dispersed two-phase flow, and we explain some of the choices that we make for the rest of the work. Chapter 3 deals with the modeling of the continuous and the dispersed phase. The domain specific language that we use to implement the models is explained in chapter 4, together with the application to the current fluid model. These chapters make up part I of the work. In part II we validate the model, using data from literature in chapter 5 and our own experimental setup as described in chapter 6. The conclusions of the work are summarized in part III.

Part I

Modeling

Chapter 2

Dispersed flow modeling: a review

2.1 Fundamental concepts

In this section, we present some fundametal concepts related to dispersed flow, using [8] and [9] as reference.

2.1.1 Particle concentration

In this work, we will mostly work with particle number concentrations, defined as the ratio of the number of particles δN in a volume δV large enough to get a stationary average:

$$n \equiv \lim_{\delta V \to \delta V_0} \frac{\delta N}{\delta V} \tag{2.1}$$

When dealing with measurements, we will also need the particle volume fraction, writing δV_p for the volume occupied by the particles in the reference volume:

$$\alpha_p \equiv \lim_{\delta V \to \delta V_0} \frac{\delta V_p}{\delta V} \tag{2.2}$$

2.1.2 Particle response time

The particle response time follows directly from the equation of motion of a small sphere in Stokes flow [8]:

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{18\mu}{\rho_p d_p^2}(\boldsymbol{u} - \boldsymbol{v}) \tag{2.3}$$

The factor on the right hand side has dimensions [s^{-1}], so the particle response time is:

$$\tau_p \equiv \frac{\rho_p d_p^2}{18\mu} \tag{2.4}$$

The particle response time can be compared to a fluid time scale τ_f using the Stokes number:

$$\mathrm{St} \equiv \frac{\tau_p}{\tau_f} \tag{2.5}$$

The Stokes number plays a fundamental role in the particle behavior. For $\mathrm{St} \ll 1$, the particle velocity is closely related to the fluid velocity, while for $\mathrm{St} \gg 1$ the particle velocity undergoes little influence from the fluid. Consequently, modeling assumptions for both kinds of particles are fundamentally different and we must have an idea of the Stokes numbers in order to make an appropriate choice.

Numerical example

We consider a plane channel flow (see also figure 5.6) with air at standard conditions, i.e. $\mu = 1.8 \times 10^{-5}$ kg/(ms) and $\rho_f = 1.2$ kg/m^3. The channel half-height h is 10 cm. This situation could represent a simple model for a ventilation duct, as might be studied in an air quality problem. A fluid time scale can be defined as $\tau_f = \nu/u_\tau^2$. As shown in [10], this time scale is smaller than the Kolmogorov time scale at all locations in the channel. The friction velocity u_τ can be computed from Dean's correlation [11] for the skin friction coefficient $c_f = 2u_\tau^2/U^2 = 0.073\mathrm{Re}_m^{-0.25}$, where U is the channel bulk velocity and Re_m is the Reynolds number with respect to U and the channel height $2h$.

The particles have a density of $\rho_p = 1000$ kg/m^3. Using this data, we can compute the particle response times for different particle sizes as well as the bulk velocity that results in $\mathrm{St} = 1$. Table 2.1 presents the results

d_p (µm)	τ_p (μs)	U (m/s)	Re_m
10	309	4.57	61000
2	12.3	28.8	384000
0.1	0.0309	883	1.18×10^7

Table 2.1: Particle response times and flow properties to obtain $\mathrm{St} = 1$ for a plane channel flow.

for different sizes. Even at the largest particle size targeted in this work, the mean velocity is already higher than what is used in many indoor air quality applications. As the particle size decreases, the flow velocity must increase sharply to match the very small particle response times, reaching values far outside our application domain.

From this result, we conclude that models that assume $\mathrm{St} \lesssim 1$ are suitable for our work. Both the fluid and particle time scale depend on the exact application, so this hypothesis should be checked for each specific case.

2.1.3 Dilute flow

A dispersed flow is considered dilute when the particle motion is dominated by fluid forces, rather than inter-particle collisions. This condition is satisfied if the average time between collisions τ_c is large when compared to the

particle response time, i.e. a flow is dilute if

$$\frac{\tau_p}{\tau_c} < 1 \tag{2.6}$$

The average time between collisions is the inverse of the particle collision frequency, which can be derived from the kinetic theory presented in section 3.2.2:

$$\tau_c = \left(n\pi d_p^2 v_r\right)^{-1} \tag{2.7}$$

Here, v_r is the relative velocity between particles, and it depends on the flow conditions. The above formula is valid for a monodisperse flow. In the case of a polydisperse flow where the instantaneous fluid velocity is known (i.e. DNS) the derivations from sections 3.2.2 and 3.2.2 can be used to obtain a more accurate value.

2.1.4 Rarefied flow effects

As the particle size decreases it may approach the mean free path of the molecules of the carrier phase. This effect is quantified by the Knudsen number:

$$\mathrm{Kn} \equiv \frac{\lambda_f}{r_p} \tag{2.8}$$

Equation (2.4) is obtained assuming Stokes flow, thus implying a continuous carrier flow and $\mathrm{Kn} \ll 1$. Various corrections are possible as Kn increases, the simplest being the Cunningham correction factor of the form:

$$C_p \equiv 1 + A\mathrm{Kn} \tag{2.9}$$

Equation (2.4) is then corrected by multiplying with C_p. According to [12] and using $A = 1.591$, this formula should be valid up to $\mathrm{Kn} \approx 5$.

Many more advanced corrections are available, see e.g. [13]. Recent measurements, valid for a wider range of Knudsen numbers, can be found in e.g. [14, 15].

2.2 Literature review

2.2.1 Particle transport

The techniques for dealing with particles fall into two categories: Lagrangian methods, where individual particles are tracked; and Eulerian methods, where properties related to the particles are treated as a continuous field. A recent overview of advances related to both methods is provided in [16]. In [17], the methods are classified as a function of Stokes number. In the limit $\mathrm{St} \to 0$, particles track the fluid exactly and their properties are transported using the fluid velocity. For $\mathrm{St} \lesssim 0.2$, an equilibrium Eulerian approach allows extrapolating the fluid velocity from the carrier flow velocity without

solving additional transport equations [17, 18]. The applicability of the Eulerian approach can be further extended up to St ≈ 1 by including equations for the particle momentum. This method was successfully applied to sprays in [19] and is sometimes called the "two-fluid approach".

As the Stokes number increases further, the use of a Lagrangian method is necessary. Since these methods require the least amount of modeling, they are often used to generate direct numerical simulation data for disperse flow, as in e.g. [20, 10]. In air quality applications, the Lagrangian method has been used in e.g. [21].

Another concept in the modeling of the dispersed flow is the coupling between the phases. One-way coupling means that only the influence of the fluid on the particles is considered. In two-way coupling, the influence of the particles on the flow is considered, as in e.g. [22]. The particles may modulate the turbulence spectrum, provided that the particle diameter and volume fraction are significant. Finally, four-way coupling adds the effect of particle collisions [23] on the motion of the particles. Both these effects can be neglected for the dilute flows considered here.

Given the small Stokes numbers targeted in this work, the Eulerian methods appear to be the most appropriate. These methods can be derived from a probability density function that statistically describes the particle properties (e.g. mass, velocity, temperature, ...). Writing a transport equation for this function and then integrating over the appropriate set of unknowns results in transport equations for particle concentration and momentum [24]. For small particle relaxation times, the momentum equations can be simplified to directly extrapolate the particle velocity from the fluid velocity, resulting in the "diffusion-inertia model" as used in [25] and [26]. This model makes assumptions regarding turbulence modeling to include effects of velocity fluctuations on the particles.

An easier method to obtain essentially the same equations is to extrapolate the particle velocity from the fluid velocity using the general equation of motion of a particle (presented in [27]) and use the resulting velocity in a transport equation for the particle concentration. This approach is followed in [17], where no assumptions regarding turbulence modeling are made.

2.2.2 Particle coagulation

Even though the flows under consideration are characterized by a low volume fraction for the dispersed phase, the number concentration typically increases as the particle size decreases [28]. For high number concentrations, coagulation may have an impact on the evolution of the particle size distribution [29]. Accounting for this effect requires a solution of the population balance equation [30]. A direct simulation is possible using Monte-Carlo methods [31, 32, 33], but this is expensive in the context of fluid simulations. In [34], a stochastic particle method is applied, in which particles are traced through phase-space much like they are traced through the physical space.

Another approach is to divide the particle size distribution into size bins. This method requires a transport equation for each bin, and source terms describe the effects of coagulation. It is applied in [25] and also related to sectional methods [35]. The drawback here is that the number of bins can be rather large and is fixed for all positions and times.

Recently, finite difference schemes were applied to the population balance equation, obtaining an accurate direct solution for the size distributions [36]. The authors concede that this method is computationally expensive, however.

A promising method that fits well with Eulerian methods is the Quadrature Method of Moments [37, 38]. It relies on the transport of lower order moments of the size distribution, which are approximated using a quadrature method. It was later extended to obtain the Direct Quadrature Method of Moments [39], where the size distribution is approximated using Dirac delta functions and the weights and abscissa are transported, rather than the moments. At a first glance, selecting delta functions appears similar to choosing size bins, but the difference is that both the weights (e.g. concentration) and the abscissa (e.g. volume) are transported. This means the initial choice of the delta approximation does not need to account for possible growth of particles, since the required abscissa will be recomputed for each time step and location. Consequently, even a few delta functions can suffice to obtain accurate moments for the size distribution. Another advantage of this method is that it can be applied to problems where the phase-space is multi-dimensional, as in e.g. [19], where the particle velocity is included in the phase space along with the size. The diffusion-inertia model was also extended using this technique to study bubble coagulation [40].

The disadvantage of the moment methods is that only a finite set of moments of the distribution are recovered. Techniques to reconstruct a distribution from these moments exist [41, 42], but in general this problem is ill conditioned. A recent overview of the moment methods and related techniques is available in [43].

2.2.3 Stabilized finite element methods

The finite element method is attractive because of its solid mathematical foundation and natural ability to deal with higher order approximations. Complex geometries can be studied using unstructured grids. Its application to the Navier-Stokes equations has been hindered by several sources of instability. First, the Galerkin finite element method corresponds to a central differencing method, which is unstable for advection-dominated problems. Second, only certain combinations of element types for the pressure and velocity are stable, due to the Babuška-Brezzi condition [44, 45].

Using the Streamline Upwind Petrov-Galerkin (SUPG) method, the instability due to the central difference type approximation was removed in [46]. Later, a pressure-stabilizing Petrov-Galerkin term was added [47]. At

higher Reynolds numbers, an additional grad-div stabilization is needed [48]. In [49], Crank-Nicolson time stepping was combined with an extrapolation of the velocity to linearize the equations, resulting in a method that only requires the solution of a single linear system each time step.

The SUPG method was applied successfully to perform direct numerical simulation of a turbulent channel flow in [50]. The stabilization parameters influence the solution of a turbulent flow, and in [51], the interaction between numerical dissipation for SUPG and the LES subgrid model is studied. Computations of the stabilization parameters can be found in [52, 53].

A theoretical foundation for these methods is established in [54], giving rise to the variational multiscale method. This also provides an alternative way to derive Large Eddy Simulation methods, as derived in [55, 56]. The method was applied using NURBS elements in [57]. We also note that in the case of linear shape functions as used in this work, the SUPG method is equivalent to the Galerkin/Least Squares (GLS) method (see e.g. [54]).

2.2.4 Experimental methods

The LaSIE laboratory in La Rochelle has conducted a series of experiments related to particle behavior in air quality applications [58, 59, 60, 61, 62]. Particles are counted using an optical particle counter (see e.g. [9] for the working principle) and the technique allows quick measurements of the size distribution even for very dilute flows. A disadvantage is that it relies on sampling the air, making it difficult to use this technique to validate a CFD model where particle concentration may be heterogeneous.

Phase Doppler Anemometry overcomes this limitation and allows non-intrusive measurements at the intersection of two laser beams. The disadvantage is that it is limited to (in our context) large particles: in [63], 1 µm is the smallest particle radius considered. Typically, the technique is used on even larger particles, as in e.g. [64] where particles of 60 µm or larger were used.

The Multiple Wavelength Light Extinction technique [65, 66] uses the attenuation of a beam of light to measure the size distribution of particles. The measurement volume is the intersection between the particle cloud and the beam of light, resulting in a much larger measurement volume than PDA. Nevertheless, it is a non-intrusive technique that can provide local information about the particle size distribution and concentration in a flow. When a combined UV-halogen light source is used, particle diameters ranging from 10 nm to 2 µm can be measured.

2.3 Modeling choices

Since the Stokes numbers encountered in our applications are expected to be small, an equilibrium Eulerian approach seems most appropriate to

model the particle phase. The coagulation will be modeled using the Direct Quadrature Method of Moments, since this appears to be a good compromise between efficiency an accuracy. All transport equations, including the fluid, will be implemented in a finite element framework.

Chapter 3

Numerical method

3.1 Fluid model

We present an overview of the SUPG method as detailed in [49]. This implicit method using a coupled system of equations is then reformulated using a semi-implicit, predictor-multicorrector scheme in an attempt to improve scalability for large, unsteady 3D problems requiring small time steps.

3.1.1 Coupled formulation

The conservation equations for incompressible flow are:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{3.1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\,\boldsymbol{u} + \frac{\boldsymbol{u}(\nabla \cdot \boldsymbol{u})}{2} + \nabla p - \nu \nabla^2 \boldsymbol{u} = \boldsymbol{0} \tag{3.2}$$

We use the skew symmetric formulation for the advection term in the momentum equation for improved conservation of energy [67]. To obtain the finite element formulation, we multiply the equations with a set of weighting functions, interpolate the unknowns between discrete nodes using shape functions and integrate over the domain. The weighting and shape functions are chosen to be identical, yielding a Galerkin formulation. The time derivative is approximated using the θ-method. This procedure yields a discrete system, with an unknown pressure and velocity at time level $n+1$ to be computed at each node in the mesh. The global shape functions are non-zero only in a node and its surrounding elements. This means that the integrals can be evaluated as a sum of integrals over these elements. This global system can thus be written as the sum of N element contributions:

$$\sum_{e=1}^{N} \left(\frac{1}{\Delta t} T_e + \theta A_e \right) \left(\boldsymbol{x}_e^{n+1} - \boldsymbol{x}_e^n \right) = -A_e \boldsymbol{x}_e^n \tag{3.3}$$

The parameter θ controls the time stepping and should be set to 1 for a forward Euler method or 0.5 for the Crank-Nicolson scheme. The vector of unknowns at the element level is laid out by grouping the nodal values per unknown, i.e. for a 3D element with $m+1$ nodes:

$$\boldsymbol{x}_e^n = \left[p_0^n \cdots p_m^n \, (u_0^n)_0 \cdots (u_0^n)_m \cdots (u_2^n)_m \right] \tag{3.4}$$

This results in the following block structure for the matrices A_e and T_e:

$$A_e = \begin{bmatrix} A_{pp} & A_{pu} \\ A_{up} & A_{uu} \end{bmatrix} = \begin{bmatrix} A_{pp} & A_{pu_0} & A_{pu_1} & A_{pu_2} \\ A_{u_0p} & A_{u_0u_0} & A_{u_0u_1} & A_{u_0u_2} \\ A_{u_1p} & A_{u_1u_0} & A_{u_1u_1} & A_{u_1u_2} \\ A_{u_2p} & A_{u_2u_0} & A_{u_2u_1} & A_{u_2u_2} \end{bmatrix} \tag{3.5}$$

We now apply the stabilized finite element method to equations (3.1) and (3.2) to obtain the following expressions for each block:

$$A_{pp} = \int_{\Omega_e} \tau_{\mathrm{PS}} \nabla N_p^{\mathrm{T}} \nabla N_p \mathrm{d}\Omega_e \tag{3.6}$$

$$\begin{aligned} A_{pu_i} = \int_{\Omega_e} \Bigg(& \left(N_p + \frac{\tau_{\mathrm{PS}} \tilde{\boldsymbol{u}}_{\mathrm{adv}} \cdot \nabla N_p}{2}\right)^{\mathrm{T}} (\nabla N_u)_i \\ & + \tau_{\mathrm{PS}} \left(\nabla N_p\right)_i^{\mathrm{T}} \tilde{\boldsymbol{u}}_{\mathrm{adv}} \nabla N_u \Bigg) \mathrm{d}\Omega_e \end{aligned} \tag{3.7}$$

$$A_{u_iu_j} = \int_{\Omega_e} \Bigg(\tau_{\mathrm{BU}} \left(\nabla N_u\right)_i \tag{3.8}$$

$$+ \frac{1}{2} \left(\tilde{\boldsymbol{u}}_{\mathrm{adv}}\right)_i \left(N_u + \tau_{\mathrm{SU}} \boldsymbol{u} \nabla N_u\right) \Bigg)^{\mathrm{T}} \left(\nabla N_u\right)_j \mathrm{d}\Omega_e \tag{3.9}$$

$$\begin{aligned} A_{u_iu_i} = \int_{\Omega_e} \Bigg(& \nu \nabla N_u^{\mathrm{T}} \nabla N_u \\ & + \left(N_u + \tau_{\mathrm{SU}} \tilde{\boldsymbol{u}}_{\mathrm{adv}} \nabla N_u\right)^{\mathrm{T}} \tilde{\boldsymbol{u}}_{\mathrm{adv}} \nabla N_u \Bigg) \mathrm{d}\Omega_e + A_{u_iu_j} \end{aligned} \tag{3.10}$$

$$A_{u_ip} = \int_{\Omega_e} \left(N_u + \tau_{\mathrm{SU}} \tilde{\boldsymbol{u}}_{\mathrm{adv}} \nabla N_u\right)^{\mathrm{T}} \left(\nabla N_p\right)_i \mathrm{d}\Omega_e \tag{3.11}$$

$$T_{pu_i} = \int_{\Omega_e} \tau_{\mathrm{PS}} \left(\nabla N_p\right)_i^{\mathrm{T}} N_u \mathrm{d}\Omega_e \tag{3.12}$$

$$T_{u_iu_i} = \int_{\Omega_e} \left(N_u + \tau_{\mathrm{SU}} \tilde{\boldsymbol{u}}_{\mathrm{adv}} \nabla N_u\right)^{\mathrm{T}} N_u \mathrm{d}\Omega_e \tag{3.13}$$

Here, N_u and N_p are the shape functions for the velocity and pressure, respectively. They are row vectors of size m+1 with coefficients depending on the spatial coordinates. We use mapped coordinates so the integrals can easily be evaluated numerically using Gaussian quadrature. The indices i and j iterate over the number of dimensions of the problem, indicating a single component of a vector variable or a row of a gradient matrix ∇N. A recent overview of the stabilization terms is provided in [52]. The stabilization terms are multiplied with their respective stabilization coefficients τ_{PS} for the PSPG stabilization, τ_{SU} for the SUPG stabilization and τ_{BU} for the bulk viscosity term. The PSPG term allows the use of equal-order interpolation for the velocity and the pressure. It introduces a non-zero A_{pp} block, consisting of the Laplacian of the pressure. The SUPG stabilization

corresponds to upwinding in the streamwise direction, i.e. the weight of upstream nodes is increased. The bulk viscosity term is necessary for flows that are strongly dominated by advection, and is sometimes called "grad-div" stabilization [52] or the "least squares on incompressibility constraint" [53]. The values for the stabilization parameters must be chosen carefully: they should be large enough to obtain the stabilizing effect, but if they are too large the scheme becomes too dissipative and accuracy suffers. The definitions given in [53] are:

$$\tau_{\mathrm{SU1}} = \frac{h}{2\left\|\tilde{\boldsymbol{u}}_{\mathrm{adv}}\right\|} \tag{3.14}$$

$$\tau_{\mathrm{SU2}} = \frac{\Delta t}{2} \tag{3.15}$$

$$\tau_{\mathrm{SU3}} = \frac{h^2}{4\nu} \tag{3.16}$$

$$\tau_{\mathrm{SU}} = \left(\frac{1}{\tau_{\mathrm{SU1}}} + \frac{1}{\tau_{\mathrm{SU2}}} + \frac{1}{\tau_{\mathrm{SU3}}}\right)^{-1} \tag{3.17}$$

$$\tau_{\mathrm{PS}} = \tau_{\mathrm{SU}} \tag{3.18}$$

$$\tau_{\mathrm{BU}} = \tau_{\mathrm{SU}} \left\|\tilde{\boldsymbol{u}}_{\mathrm{adv}}\right\|^2 \tag{3.19}$$

Here, h is a characteristic element length. In [68], a systematic study comparing different definitions of h was conducted, concluding that a length scale based on the minimal edge length of an element gives the best result in the case of high aspect ratio elements. For aspect ratios closer to one, results were comparable to other possible definitions (maximum edge length and edge length in the streamwise direction). This leads us to choose the minimum element edge length as our definition for h.

In [50], definitions based on the element metric tensor g_{ij} (i.e. the element shape function inverse Jacobian matrix multiplied with its transpose) are used (using index notation):

$$\tau_{\mathrm{SU1}}^2 = \frac{1}{u_i g_{ij} u_j} \tag{3.20}$$

$$\tau_{\mathrm{SU2}} = \frac{\Delta t}{2} \tag{3.21}$$

$$\tau_{\mathrm{SU3}}^2 = \frac{1}{\nu^2 g_{ij} g_{ij}} \tag{3.22}$$

$$\tau_{\mathrm{SU}} = \left(\frac{1}{\tau_{\mathrm{SU1}}^2} + \frac{c_1^2}{\tau_{\mathrm{SU2}}^2} + \frac{c_2}{\tau_{\mathrm{SU3}}^2}\right)^{-\frac{1}{2}} \tag{3.23}$$

$$\tau_{\mathrm{PS}} = \tau_{\mathrm{SU}} \tag{3.24}$$

$$\tau_{\mathrm{BU}} = \frac{1}{\tau_{\mathrm{SU}} g_{ii}} \tag{3.25}$$

The parameters c_1 and c_2 are introduced in [50] and allow further control

of the stabilization. Values $c_1 = 1$ and $c_2 = 16$ correspond to the definitions in [53], while in [50] the authors choose $4 \leq c_1 \leq 16$ and $c_2 = 36$, adjusting values to obtain better results for a direct simulation of channel flow. When c_1 is increased, the time component of the stabilization gains in importance, resulting in an overall lower PSPG/SUPG stabilization. Due to equation (3.25), however, the grad-div term increases. It seems necessary to only apply the adjustment using c_1 and c_2 to τ_{SU} and τ_{PS} and ignore it for τ_{BU}, to avoid a too high dissipation.

The advection velocity $\boldsymbol{u}_{\text{adv}}$ is calculated using a Taylor series expansion:

$$\boldsymbol{u}_{\text{adv}} = 2.1875\boldsymbol{u}^n - 2.1875\boldsymbol{u}^{n-1} + 1.3125\boldsymbol{u}^{n-2} - 0.3125\boldsymbol{u}^{n-3} \tag{3.26}$$

This technique allows us to linearize the equations without resorting to an iterative technique, thus solving only one linear system per time step, at the (very modest) cost of storing the velocity for the previous 4 time steps for every node.

In [49], the global linear system 3.3 is solved by directly applying the GMRES method. It is preconditioned either with algebraic multigrid or ILU factorization. There is no stability condition for the time step if we set $0.5 \leq \theta \leq 1$, and the scheme is second order accurate in time when setting $\theta = 0.5$. Storing the coupled system is expensive, and depending on the mesh and the flow configuration the iterative method may converge slowly.

3.1.2 Segregated formulation

An alternative to solving the complete system is to split it into separate linear systems for the velocity and the pressure. We follow the method proposed in [46]. Introducing the velocity- and pressure differences between two time levels $\Delta\boldsymbol{u}$ and Δp, we can rewrite the momentum equation as a function of the acceleration $\boldsymbol{a} = \Delta\boldsymbol{u}/\Delta t$:

$$\left(T_{uu} + \theta\Delta t A_{uu}\right)\boldsymbol{a} + \theta A_{up}\Delta p = -A_{uu}\boldsymbol{u}^n - A_{up}p^n \tag{3.27}$$

Defining $\boldsymbol{a}^* = \boldsymbol{a} + (T_{uu} + \theta\Delta t A_{uu})^{-1}\theta A_{up}\Delta p$ we obtain a linear system that can be solved for $\boldsymbol{a}^*$:

$$\left(T_{uu} + \theta\Delta t A_{uu}\right)\boldsymbol{a}^* = -A_{uu}\boldsymbol{u}^n - A_{up}p^n \tag{3.28}$$

The continuity equation is:

$$\left(T_{pu} + \Delta t A_{pu}\right)\boldsymbol{a} + A_{pp}\Delta p = -A_{pu}\boldsymbol{u}^n - A_{pp}p^n \tag{3.29}$$

Using the definition of a^*, we can rewrite this into a linear system for the pressure difference Δp between two time steps:

$$\begin{aligned}\Big(\left(T_{pu} + \Delta t A_{pu}\right)\left(T_{uu} + \theta\Delta t A_{uu}\right)^{-1}\theta A_{up} - A_{pp}\Big)\Delta p \\ = T_{pu}\boldsymbol{a}^* + A_{pu}\left(\boldsymbol{u}^n + \Delta t\boldsymbol{a}^*\right) + A_{pp}p^n\end{aligned} \tag{3.30}$$

Note that the system matrix for the pressure is the Schur complement of the velocity block in the original system. We can now first solve the linear system for $\boldsymbol{a}^*$, then solve the Δp system and finally get the acceleration from

$$\boldsymbol{a} = \boldsymbol{a}^* - (T_{uu} + \theta \Delta t A_{uu})^{-1} \theta A_{up} \Delta p \tag{3.31}$$

So far, we have only solved the coupled system in a different way, algebraically equivalent to a direct solution. In addition to solving a separate linear system for the velocity and the pressure, we also need the inverse $(T_{uu} + \theta \Delta t A_{uu})^{-1}$ to construct the matrix for the pressure system. Doing this directly is not possible on a large mesh, so simplification is needed to obtain an efficient method. When we simplify steps in the algorithm, the result will no longer be identical to the solution of the coupled system, so we introduce an iterative algorithm that can be executed M times each time step. The linear systems will be solved for the difference between two inner iterations m and $m+1$, i.e. $\Delta \boldsymbol{a} = \boldsymbol{a}^{m+1} - \boldsymbol{a}^m$ and $\Delta p^{m+1} = p^{m+1} - p^m$. From this, we also have $\boldsymbol{u}^m = \boldsymbol{u}^n + \Delta t \boldsymbol{a}^m$ and $p^m = p^n + \sum_{i=0}^{m} \Delta p^i$. The modified $\boldsymbol{a}^*$ is then:

$$\boldsymbol{a}^{*m} = \boldsymbol{a}^m + (T_{uu} + \theta \Delta t A_{uu})^{-1} \theta A_{up} \left(\sum_{i=0}^{m} \Delta p^i \right) \tag{3.32}$$

Filling this into the original system (3.28) and using the definition of $\boldsymbol{a}^*$ yields:

$$\begin{aligned} (T_{uu} + \theta \Delta t A_{uu})\, \Delta \boldsymbol{a}^* = &-A_{uu} \boldsymbol{u}^m - A_{up} p^m \\ &- (T_{uu} + \theta \Delta t A_{uu})\, \boldsymbol{a}^m + A_{uu} \Delta t \boldsymbol{a}^m \\ &+ (1-\theta)\, A_{up} \left(\sum_{i=0}^{m} \Delta p^i \right) \end{aligned} \tag{3.33}$$

For the continuity equation we start from:

$$\begin{aligned} (T_{pu} + \Delta t A_{pu}) \Delta \boldsymbol{a} + A_{pp} (\sum_{i=0}^{m+1} \Delta p^i) = \\ - A_{pu} \boldsymbol{u}^n - A_{pp} p^n - (T_{pu} + \Delta t A_{pu}) \boldsymbol{a}^m \end{aligned} \tag{3.34}$$

With $\Delta \boldsymbol{a} = \Delta \boldsymbol{a}^* - (T_{uu} + \theta \Delta t A_{uu})^{-1} \theta A_{up} \Delta p^{m+1}$ this becomes:

$$\begin{aligned} \Big((T_{pu} + \Delta t A_{pu})\, (T_{uu} + \theta \Delta t A_{uu})^{-1}\, \theta A_{up} - A_{pp} \Big) \cdot \\ \Delta p^{m+1} = T_{pu} \Delta \boldsymbol{a}^* + A_{pu}\, (\boldsymbol{u}^m + \Delta t \Delta \boldsymbol{a}^*) \\ + A_{pp} p^m + T_{pu} \boldsymbol{a} \end{aligned} \tag{3.35}$$

With the problem formulated this way, we can now apply a predictor-multicorrector iterative scheme:

$\boldsymbol{u}^0 = \boldsymbol{u}^n$
$p^0 = p^n$
$\boldsymbol{a}^0 = \boldsymbol{0}$
for $m = 0$ to $M - 1$ **do**
  Solve linear system (3.33) for $\Delta \boldsymbol{a}^*$
  Solve linear system (3.35) for Δp^{m+1}
  Compute:
  $\Delta \boldsymbol{a} = \Delta \boldsymbol{a}^* - (T_{uu} + \theta \Delta t A_{uu})^{-1} \theta A_{up} \Delta p^{m+1}$
  Update $\boldsymbol{u}^{m+1} = \boldsymbol{u}^m + \Delta t \Delta \boldsymbol{a}$
  Update $p^{m+1} = p^m + \Delta p^{m+1}$
end for

Without simplifications to the systems, executing the iteration once will immediately provide the correct velocity- and pressure updates. The solution of the velocity system is difficult due to the advective terms. These terms have an important impact on the convergence rate of the iterative solvers and introduce a direct dependency of the matrix coefficients on the velocity. An easy fix is to drop the advection terms from the velocity system matrix, treating them explicitly (i.e. setting θ=0 for those terms). We assemble the simplified velocity matrices $\widetilde{A_{uu}}$ and $\widetilde{T_{uu}}$ using the following expressions:

$$\widetilde{A_{u_i u_i}} = \int_{\Omega_e} (\nu + \tau_{\mathrm{BU}}) \nabla N_u^{\mathrm{T}} \nabla N_u \, \mathrm{d}\Omega_e \tag{3.36}$$

$$\widetilde{A_{u_i u_j}} = \int_{\Omega_e} \tau_{\mathrm{BU}} (\nabla N_u)_i^{\mathrm{T}} (\nabla N_u)_j \, \mathrm{d}\Omega_e \qquad (i \neq j) \tag{3.37}$$

$$\widetilde{T_{u_i u_i}} = \int_{\Omega_e} N_u^{\mathrm{T}} N_u \, \mathrm{d}\Omega_e \tag{3.38}$$

The system matrix of this simplified velocity system is now symmetric and much better conditioned, due to the removal of the advective terms. The coefficients only depend on the viscosity and τ_{BU}, so they do not vary much in time, especially if the time step is small. This allows us to reuse the same velocity matrix over a range of time steps, reducing the time required for assembly and preconditioner setup. For the pressure system, we first need an approximation for $(T_{uu} + \theta \Delta t A_{uu})^{-1}$. As suggested in [46], the inverse of the lumped velocity mass matrix M_L is a good candidate, i.e. we sum all the elements of a row and then put that value on the diagonal, making the inverse trivial to compute. The same approximation is used in [53]. After this change, the pressure system matrix becomes:

$$\left((T_{pu} + \Delta t A_{pu}) M_L^{-1} \theta A_{up} - A_{pp} \right) \tag{3.39}$$

If we ignore the stabilization terms and apply partial integration to the pressure gradient term in the momentum equation, we have $A_{up} = -A_{pu}^{\mathrm{T}}$, hinting that the structure of the pressure matrix and the Poisson problem

are similar. This leads us to simplify the pressure matrix as follows:

$$((T_{pu} + \Delta t A_{pu}) M_L^{-1} \theta A_{up} - A_{pp}) \approx \\ - \theta(\tau_{\text{PS}} + \Delta t) \int_{\Omega_e} \nabla N_p^{\text{T}} \nabla N_p \, \mathrm{d}\Omega_e \quad (3.40)$$

This approximation is remarkably similar to the approximation of the Schur complement of the velocity block for flows where the time term is dominant, as described in [69]. In our work the time term is indeed large due to the small time steps under consideration. The validity of our approximation is further confirmed by the Taylor-Green test case. The resulting matrix is symmetric and only depends on the solution through the value of τ_{PS}. Numerical experiments show that the adjustment of τ_{PS} in the matrix has no effect on the accuracy, so we can reuse the same pressure matrix during the complete calculation. This opens up the possibility of using a direct solution method or reusing the preconditioner. While the original matrix required a sparse matrix product, the simplification can be assembled on a per-element basis. This greatly simplifies the code and speeds up the assembly. This can be important in the case of deforming meshes or variable time steps, where the pressure matrix does change with each time step.

3.1.3 Performance aspects

In this section we assess the performance of the segregated method. We use the direct numerical simulation of plane channel flow as a basis for the different tests, limiting the computations to 100 time steps to reduce the computational overhead. This test fits the objectives of the method perfectly, since a DNS requires small time steps due to the physics of the flow. The meshes used in the tests are based on those from [12] and [15], using a hexahedral mesh that is refined towards the walls. We note the mesh size in the $N_x \times N_y \times N_z$ format, where each N_i represents the number of nodes in the corresponding direction i (not counting periodic nodes twice). The streamwise direction corresponds to x, the wall-normal direction is y and z is the spanwise direction. Specifications for the machines used in the tests are listed in table 3.1. The computationally expensive steps in

	RMA Cluster	**VKI Cluster**
Processor type	Xeon E5520	Opteron 6376
Cores per node	8	64
RAM per core	3 GB	4 GB
Interconnect	1 Gb ethernet	InfiniBand
Nb. nodes	30	28

Table 3.1: Machines used in the tests

the algorithm are the solution of the linear systems and the computation of

the coefficients for the system matrices and right hand side vectors, i.e. the evaluation of the element integrals. For the fully coupled method, the matrix coefficients need to be recomputed each time step, since they depend on the advection velocity and only one linear system needs to be solved. For the segregated method, the matrix for the pressure system is constant for the whole simulation. The velocity matrix depends on the solution only through the τ_{BU} stabilization parameter. This means that the matrix coefficients are also approximately constant. Surprisingly, this results in a linear increase of the solution time per time step, as shown by the dashed line in figure 3.1. We can eliminate this effect by recomputing the coefficients every 100 time steps, resulting in a constant solution time (solid line in figure 3.1).

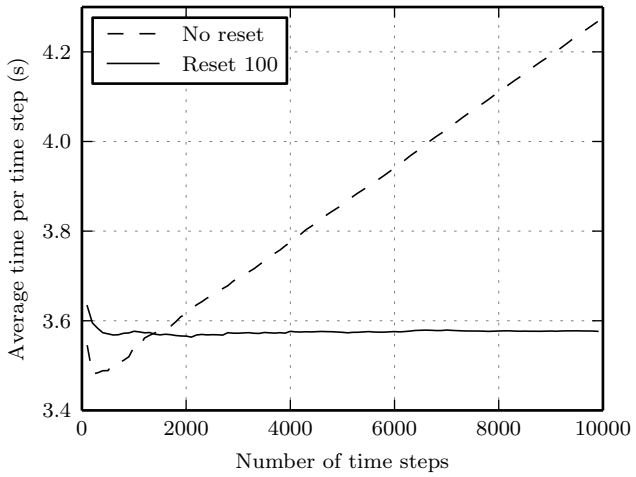


Figure 3.1: Average wall clock time per time step for a $64 \times 129 \times 64$ mesh on 64 cores on the RMA cluster. "No reset" means one single velocity matrix assembly. "Reset 100" means one velocity matrix assembly every 100 time steps.

Since we mostly eliminated the matrix assembly from the computation, the cost for the segregated method is dominated by the solution of both linear systems and the computation of the right hand side coefficients. This work must be done every iteration, i.e. twice every time step in practice. Figure 3.2 presents the scaling of the segregated method in the strong sense, i.e. keeping the problem size constant while increasing the number of processors. The assembly operations follow the ideal scaling (i.e. half the time each time the number of cores is doubled) closely. This is to be expected, since this step does not depend on communication and only requires additional ghost elements as the number of mesh partitions is increased. The timing marked as "other" corresponds to some aspects of the computation

that take a negligible amount of time, such as the update of the solution and the extrapolation of the velocity for the linearization. The matrix assemblies are also included here and confirmed to be negligible in time, since they are executed at most once every 100 steps. Most of the time is spent solving the linear systems. We solve the velocity system using the Conjugate Gradient (CG) method, preconditioned using ILU factorization. The scaling is not ideal, but better than the scaling for the pressure system, which we solve using CG preconditioned with algebraic multigrid (AMG). This no longer scales when moving from 256 to 512 cores, making the solution of the pressure system the dominant factor at 512 cores.

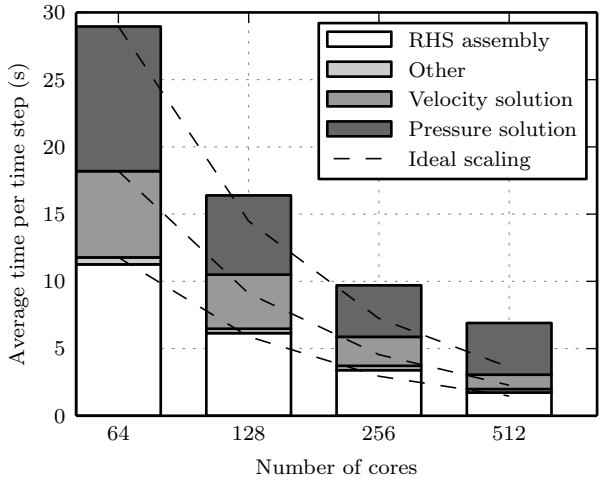


Figure 3.2: Strong scaling of the average wall clock time per time step for a $128 \times 257 \times 128$ mesh on the VKI cluster. The RHS assembly timing comprises the sum of all coefficient computations for the right hand side vectors.

For smaller problems, it is feasible to solve the pressure system using a direct method. Figure 3.3 illustrates the effect on the timings for two different mesh sizes. The AMG timings are obtained using the same settings as before, while the MUMPS timings use the MUMPS parallel sparse direct solver [16] for the pressure system. Since the pressure matrix is constant, we only need to perform the expensive factorization once and can then apply this in all subsequent time steps. The small ($32 \times 65 \times 32$) and large ($64 \times 129 \times 64$) problems used 8 and 64 cores, respectively, thus keeping the workload per core constant. According to the ideal scaling law, the timings for the large and small problems should be identical, but especially on the RMA cluster the communication overhead becomes prohibitive for the large mesh. This is an effect of the 1Gb Ethernet interconnect. The assembly

operations —which do not require intensive communication— do follow the ideal scaling almost perfectly.

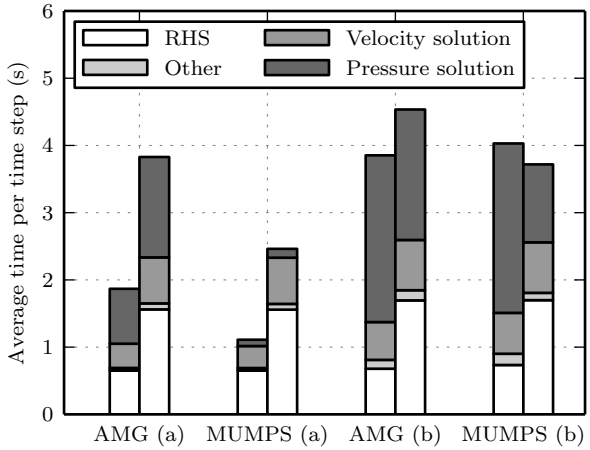


Figure 3.3: Comparison between Algebraic Multigrid (AMG) and a direct solver (MUMPS) for the pressure system, using mesh sizes (a) $32 \times 65 \times 32$ and (b) $64 \times 129 \times 64$. Left bars are for the RMA cluster, right bars for the VKI cluster. Results are averaged over 1000 time steps.

The switch to MUMPS is very effective on the small problem: the solution time for the pressure system becomes negligible compared to the total timing, while it is the dominant factor when using AMG. The total solution time is nearly halved as a result. For the large problem, we see that the scaling for MUMPS is much worse than AMG, resulting in very little benefit. On the RMA cluster, the initial factorization took 842 s. Since we averaged the timing over 2000 iterations, there is still a contribution of 0.42 s from the initial factorization in the average timing. This part will diminish as the number of iterations increases. On the VKI cluster the initial factorization only took 95 s, illustrating the importance of communication in this step. As is typical for a direct method, the cost of the factorization increases non-linearly: on the small problem the timings were 5.3 s (RMA) and 6.6 s (VKI). For very large problems, the cost of the initial factorization becomes prohibitive, and the factorization itself can no longer be stored because it is much denser than the original matrix. The poor scaling of the application of the factorization is surprising, and might be due to our use of the Trilinos interface to access MUMPS. This interface also forbids the use of the symmetric solver, so better results should be possible by interfacing with MUMPS directly.

Memory usage is dominated by the storage of the linear systems. For the

coupled method, the memory required to store the sparse matrix can be computed as follows, using a structured hexahedral grid where 27 nodes are adjacent to each other:

$$(27\,\text{nodes per row} \cdot 4\,\text{variables} \cdot 12\,\text{bytes} + 4\,\text{bytes}) \cdot (4\,\text{equations} \cdot \text{number of nodes})$$

We assume double precision, i.e. 8 bytes per coefficient and 32 bits for integers, i.e. 4 bytes per integer to store the coefficient index and row size. For a mesh with 10 million nodes, this yields a storage cost of 52 GB. In the case of the segregated solver, two matrices need to be stored, but because each matrix is smaller the total size is less: 3.25 GB for the pressure system and 29.25 GB for the velocity system. The savings are modest, and when using a direct solver for the pressure the segregated solver will even use more memory than the coupled method. On modern hardware, such as the clusters used in this work, problems are typically distributed over a large number of CPUs with a sufficient amount of RAM to allow either method to be chosen.

	Segregated		*Coupled*		
Mesh	t_s (s)	Scaling	t_c (s)	Scaling	t_c/t_s
$32 \times 65 \times 32$	1.44	-	7.64	-	5.31
$32 \times 129 \times 32$	4.78	3.32	35.46	4.64	7.42
$32 \times 257 \times 32$	12.26	2.57	277.96	7.84	22.67
$32 \times 65 \times 32$ random	1.70	-	15.26	-	8.98
$32 \times 129 \times 32$ random	3.60	2.12	66.00	4.33	18.35
$32 \times 257 \times 32$ random	11.99	3.33	575.30	8.72	47.98

Table 3.2: Comparison of the average time per time step for the segregated and the coupled method. Scaling is the time on the current mesh divided by the time on the previous mesh. All simulations are carried out on the RMA cluster on 8 cores.

In a final test, we compare the timings per time step for the segregated and the coupled method on three different meshes, gradually finer in the wall-normal direction. The segregated solver uses the AMG method for the pressure system as described before. For the coupled method, we also use algebraic multigrid preconditioning, using the defaults optimized for advection-diffusion problems. The iterative solver is GMRES from the Belos package. Two sets of initial conditions were used: the laminar solution and a random disturbance of the laminar solution. The latter is typically used to initialize a DNS. Time steps were chosen to obtain a Courant number of around 0.15.

Table 3.2 summarizes the results. All tests are carried out on 8 cores on the RMA cluster, so we expect the solution time to double each time the

number of mesh nodes doubles. The "mesh scaling" factor in the table lists the ratio between the current time and the time on the previous mesh, and it is always above the ideal value of 2, with significantly higher values for the coupled method. For the segregated method, the initial condition has little impact on the solution time, but for the coupled method the added randomness appears to double the solution time. As the mesh is refined, the coupled method can take up to 48 times as long as the segregated method, so for short time steps there is a clear benefit of using the segregated approach.

3.2 Discrete phase model

We use the Eulerian approach for the modeling of the fluid phase. In what follows, we first present the transport equation for the concentration of particles of a single, fixed size. We then extend this method to account for polydisperse particle distributions with coagulation effects.

3.2.1 Monodisperse flows

For monodisperse flows, it is possible to write a single transport equation for the particle concentration. We follow the approach of Ferry et al. [17], which is a fairly intuitive description of how the transport equation can be obtained. Similar models are also described by Simonin [24] and Zaichik et al. [26], where the model is obtained by integrating a probability density function in phase space. While more rigorous, this approach is much harder to follow and the resulting transport equation is the same.

The basic assumption is that we can write a transport equation for the particle number concentration n as follows:

$$\frac{\partial n}{\partial t} + \nabla \cdot (n\boldsymbol{v}) = D_b \nabla^2 n \tag{3.41}$$

The velocity $\boldsymbol{v}$ is the particle velocity, and the above equation is valid only if we suppose that the particle velocity can be represented as an Eulerian field. This is a strong hypothesis, since the particle velocity not only depends on the fluid velocity but also on the particle initial conditions. Two particles with different initial velocities might still have different velocities at the same time t and position $\boldsymbol{x}$. If the particle relaxation time τ_p is sufficiently small with respect to the fluid velocity time scale, it can be shown (see [17]) that the effect of particle initial conditions decreases exponentially fast, and an Eulerian velocity field may be used.

Generally speaking, the particle velocity field $\boldsymbol{v}$ is obtained from a transport equation, as computed from the probability density function in i.e. [26]. For the small particle relaxation times considered here, however, it is possible to omit this extra set of transport equations and to write the particle velocity as an expansion of the fluid velocity, resulting in the equilibrium

Eulerian method. The expansion can be obtained from the particle equation of motion, to be discussed in detail in the following section.

Particle equation of motion

The Maxey and Riley [27] equation describes the motion of a small spherical particle moving in a constant density flow and reads:

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \underbrace{\frac{\rho_p - \rho_f}{\rho_p}\boldsymbol{g}}_{\text{Buoyancy}} + \underbrace{\frac{\rho_f}{\rho_p}\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t}}_{\text{Fluid acceleration}} - \underbrace{\frac{1}{\tau_p}(\boldsymbol{v}-\boldsymbol{u})}_{\text{Stokes drag}} + \underbrace{\frac{\rho_f}{2\rho_p}\left(\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} - \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\right)}_{\text{Added mass}} - \underbrace{\frac{9\sqrt{\mu\rho_f}}{d_p\rho_p\sqrt{\pi}}\int_0^t \frac{1}{\sqrt{1-s}}\frac{\mathrm{d}}{\mathrm{d}s}(\boldsymbol{v}-\boldsymbol{u})\,\mathrm{d}s}_{\text{Basset history}} \quad (3.42)$$

Here, $\mathrm{d}/\mathrm{d}t$ denotes the total derivative following the particle, while $\mathrm{D}/\mathrm{D}t$ is the total derivative following the fluid. The first term is the buoyancy or Archimedes force felt by the particle. The second term, labeled "fluid acceleration" here, represents the force that the particle would feel if its density were equal to the fluid density. Together with the $\rho_f\boldsymbol{g}$ term in the buoyancy, this is an effect that can be computed from the undisturbed flow. The remaining terms take into account the effect of the disturbed flow, and are obtained from the solution of an unsteady Stokes problem around a sphere. This implies that the Reynolds number associated with the relative velocity between fluid and particle is assumed to be small. The first of these terms is the drag force, inversely proportional to the particle relaxation time. The added mass term can be interpreted as an inertia term that is also derived from the disturbed flow computation. Finally, the Basset history term models the unsteady development of the flow around the particle.

Before we exploit this equation, we will examine possible simplifications, based on the expected importance of the different terms. The fluid acceleration, added mass and Basset history forces are present only when unsteady flow or particle motion is considered. They are usually important if $\rho_p \lesssim O(\rho_f)$. As found in [70], with $\rho_p \gg \rho_f$ their importance must be evaluated using the ratio d_p/η_k, where η_k is the Kolmogorov length scale, i.e. the smallest fluid length scale in the present context of incompressible flow. If $d_p < \eta_k$, the unsteady terms can be neglected. In [18], numerical tests are performed using the equilibrium Eulerian approach, indicating that the model remains valid when the particle response time stays below the order of the Kolmogorov time scale, i.e. $\tau_p \lesssim \tau_k$. For particles with $\rho_p \gg \rho_f$, this

results in $d_p < \eta_k/2$. This means that for the entire validity range of the equilibrium Eulerian approach, the unsteady terms can be neglected, since we are in the case $\rho_p \gg \rho_f$ and $d_p < \eta_k$. The latter condition can be verified from table 2.1 by estimating the fluid time scale as $\sqrt{\nu\tau_f}$, indicating that the particle diameter is at least 6 times smaller than the Kolmogorov length scale for the largest particles.

Nevertheless, we choose to retain the fluid acceleration and added mass terms, since they bring no significant complexity to the model and including them allows a comparison with the bubble simulations of [71]. The Basset history force greatly complicates the implementation of the model, so we will always neglect it. This is perfectly valid for the targeted applications, but care should be taken if the flow strays too much from the criteria $\rho_p \gg \rho_f$ and $d_p < \eta_k$. Armenio et al. [72] show that for larger particles, the Basset force is important for all density ratios. The importance of memory effects is also demonstrated in [73] and [74], again showing stronger effects for larger and lighter particles. Efficient methods to evaluate the Basset term can be found in i.e. [75, 76].

In equation (3.42) we also omitted the so-called Faxén correction terms, which are proportional to $d_p^2\nabla^2\boldsymbol{u}$. These terms become important only when $d_p > \eta_k$ [77, 78], so they can be neglected on the same grounds as the unsteady terms.

Rewriting equation (3.42) by regrouping terms and removing the Basset force yields:

$$\left(1 + \frac{\rho_f}{2\rho_p}\right)\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\rho_p - \rho_f}{\rho_p}\boldsymbol{g} - \frac{1}{\tau_p}(\boldsymbol{v} - \boldsymbol{u}) + \frac{3}{2}\frac{\rho_f}{\rho_p}\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} \tag{3.43}$$

We now look for an approximate solution for the particle velocity from the above equation. First we have:

$$\boldsymbol{v} = \boldsymbol{u} - \tau_p\left[\left(1 + \frac{\rho_f}{2\rho_p}\right)\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} - \frac{\rho_p - \rho_f}{\rho_p}\boldsymbol{g} - \frac{3}{2}\frac{\rho_f}{\rho_p}\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t}\right] \tag{3.44}$$

If we consider τ_p as a small parameter, we can already see that this equation yields the trivial zeroth order approximation $\boldsymbol{v} = \boldsymbol{u}$, valid only for perfect tracer particles. Recursively replacing $\boldsymbol{v}$ on the right hand side using equation (3.44) itself yields the first order approximation:

$$\boldsymbol{v} = \boldsymbol{u} - \tau_p\left[\left(1 + \frac{\rho_f}{2\rho_p}\right)\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} - \frac{\rho_p - \rho_f}{\rho_p}\boldsymbol{g} - \frac{3}{2}\frac{\rho_f}{\rho_p}\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t}\right] + O(\tau_p^2) \tag{3.45}$$

Higher order expansions can be obtained by writing out the $O(\tau^2)$ part and recursively substituting again the expression for $\boldsymbol{v}$. However, as concluded in [17] the second order approximation does not always yield better results, so it is not worth the effort in practice and we will use equation (3.45) without the $O(\tau_p^2)$ terms to obtain the particle velocity. Note that this is

equivalent to the approximation $\mathrm{d}\boldsymbol{v}/\mathrm{d}t \approx \mathrm{d}\boldsymbol{u}/\mathrm{d}t$ and results in the locally implicit method as formulated in [18].

The equation is still implicit in $\boldsymbol{v}$ due to the definition of the total derivatives:

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{u} = \frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} + (\boldsymbol{v} - \boldsymbol{u}) \cdot \nabla \boldsymbol{u} \tag{3.46}$$

In [17], this is avoided by approximating $\mathrm{d}\boldsymbol{v}/\mathrm{d}t \approx \mathrm{D}\boldsymbol{u}/\mathrm{D}t$. This is less accurate because now $(\boldsymbol{v} - \boldsymbol{u}) \cdot \nabla \boldsymbol{u}$ is neglected. As indicated in [18], this term is important in the prediction of wall-normal transport of streamwise velocity, since the wall-normal component of $\boldsymbol{v}$ may differ significantly from that of $\boldsymbol{u}$. Filling in equation (3.46) into equation (3.45) results in:

$$\boldsymbol{v} = \boldsymbol{u} - \tau_p \left(1 - \frac{\rho_f}{\rho_p}\right) \left(\boldsymbol{I} + \tau_p \left(1 + \frac{\rho_f}{2\rho_p}\right) \nabla \boldsymbol{u}\right)^{-1} \left(\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} - \boldsymbol{g}\right) \tag{3.47}$$

Here, $\boldsymbol{I}$ is the identity matrix. The matrix that needs to be inverted is only size 3x3 for three dimensional problems, so this equation is still very cheap to solve compared to a full momentum transport equation for the particle phase.

Equation (3.47) is difficult to apply to the limit case of bubbles, where $\rho_f/\rho_p \to \infty$ and $\tau_p \to 0$. This can be avoided by introducing the density ratio parameter β and the generalized relaxation time τ as defined in [17]:

$$\beta \equiv \frac{3}{2\rho_p/\rho_f + 1} \tag{3.48}$$

$$\tau \equiv \tau_p \left(1 + \frac{\rho_f}{2\rho_p}\right) \tag{3.49}$$

Substituting into equation (3.47) yields:

$$\boldsymbol{v} = \boldsymbol{u} - \tau \left(1 - \beta\right) \left(\boldsymbol{I} + \tau \nabla \boldsymbol{u}\right)^{-1} \left(\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} - \boldsymbol{g}\right) \tag{3.50}$$

Comparing with the equations found in literature, equation (3.50) is identical to the result found in [18] except for the factor $(1 - \beta)$. This stems from the fact that [18] only accounts for gravity and Stokes drag forces. This also corresponds to $\beta = 0$, which is the limit for heavy particles with $\rho_p/\rho_f \to \infty$ and $\tau \to \tau_p$. It is another way to illustrate that the added mass and fluid acceleration forces become negligible as the density ratio increases. If we neglect the locally implicit contribution $(\boldsymbol{I} + \tau \nabla \boldsymbol{u})^{-1}$ we recover the general formulation of [17]. Finally, in the limit case of bubbles we have $\beta = 3$ as $\rho_p/\rho_f \to 0$ and we obtain the equation given in [71]. Note that for bubbles we have, using equations (3.49) and (2.4) with $\tau_p \to 0$:

$$\tau = \tau_p \frac{\rho_f}{2\rho_p} = \frac{d_p^2}{36\nu} \tag{3.51}$$

This is exactly the bubble relaxation time as given in [71].

In conclusion, with equation (3.50) we have generalized the locally-implicit approach of [18], accounting for the added mass and fluid acceleration forces, thus making the equation applicable to both bubbles and heavy solid particles.

Finite element formulation

Equation (3.41) is basically a scalar advection-diffusion equation, where the velocity field is not divergence free. We apply the same SUPG stabilization as for the flow field, resulting in the following expressions for the element matrices:

$$A_n = \int_{\Omega_e} \left(N_n + \tau_{\mathrm{SU}} \tilde{\boldsymbol{v}} \nabla N_n\right)^{\mathrm{T}} \left(\tilde{\boldsymbol{v}} \nabla N_n + \operatorname{div} \tilde{\boldsymbol{v}} N_n\right) + D_b \nabla N_n^{\mathrm{T}} \nabla N_n \,\mathrm{d}\Omega_e \tag{3.52}$$

$$T_n = \int_{\Omega_e} \left(N_n + \tau_{\mathrm{SU}} \tilde{\boldsymbol{v}} \nabla N_n\right)^{\mathrm{T}} N_n \,\mathrm{d}\Omega_e \tag{3.53}$$

As for the fluid model, we assemble a linear system, using θ time integration (we note m for the time step to avoid confusion with the number concentration n):

$$\sum_N \left(\frac{1}{\Delta t} T_n + \theta A_n\right) \left(\boldsymbol{n}_e^{m+1} - \boldsymbol{n}_e^m\right) = -A_n \boldsymbol{n}_e^m \tag{3.54}$$

Before the previous equation can be solved, we need to supply it with the particle velocity. The finite element formulation for equation (3.50) reads, when omitting the local-implicit term for clarity and only writing the time step at $(n-1)$:

$$\left(\int_{\Omega_e} N_v^{\mathrm{T}} N_v \,\mathrm{d}\Omega_e\right) \left(\boldsymbol{v}_e\right)_i = \int_{\Omega_e} N_u^{\mathrm{T}} \left[\tilde{\boldsymbol{u}} - \tau \left(1 - \beta\right) \left(\frac{\tilde{\boldsymbol{u}} - \tilde{\boldsymbol{u}}^{n-1}}{\Delta t} + \tilde{\boldsymbol{u}} \cdot \operatorname{grad} \tilde{\boldsymbol{u}}\right)\right]_i \mathrm{d}\Omega_e \tag{3.55}$$

On the left hand side, $\left(\boldsymbol{v}_e^{n+1}\right)$ appears next to a mass matrix, so we need to solve a linear system of equations over the domain in order to find the particle velocity. Even though a linear system involving only the mass matrix is easy to solve, the cost is still much greater than a direct application of the locally implicit formulation of equation (3.50). This becomes prohibitive especially when polydisperse particle distributions need to be considered. One easy fix would be to lump the mass matrix, resulting in a diagonal system that is trivial to solve. As another alternative, we can also approximate

the gradient matrix $\nabla \boldsymbol{u}$ as the average of the gradient at the center of each element surrounding a node. The resulting gradient field can then be used to evaluate the particle velocity at each node k using equation (3.50) with a first-order difference in time:

$$\boldsymbol{v}_k = \boldsymbol{u}_k - \tau\,(1-\beta)\,(\boldsymbol{I} + \tau\,(\nabla \boldsymbol{u})_k)^{-1} \left(\frac{\boldsymbol{u}_k - \boldsymbol{u}_k^{n-1}}{\Delta t} + \boldsymbol{u}_k\,(\nabla \boldsymbol{u})_k - \boldsymbol{g}_k \right) \tag{3.56}$$

Numerical experiments in section 5.2.1 show that this approach maintains second order accuracy and is more accurate than lumping the mass matrix of the full finite element formulation.

Stabilization

Equations (3.52) and (3.53) already contain the SUPG stabilization as used in the flow equations. In the presence of steep concentration gradients, this is not enough to prevent spurious oscillations and additional stabilization is needed [79]. Different methods are compared in [79, 80], but none of them provide a complete solution to the problem. We add an artificial dissipation term of the form

$$\int_{\Omega_e} a_0 \tau_{\mathrm{AD}} \nabla N_n^{\mathrm{T}} \nabla N_n \,\mathrm{d}\Omega_e \tag{3.57}$$

Here, the stabilization coefficient τ_{AD} depends on the local element size and concentration gradient and is computed following the crosswind diffusion method presented in [79]. The parameter a_0 allows ad-hoc control over the amount of diffusion that is added. There is no general rule to determine the value of this parameter. It must be chosen so that the dissipation is enough to avoid the oscillations, but small enough so the accuracy of the solution is not affected too much. In order to guarantee that the concentration remains positive in the entire domain we also resorted to cutting negative values. Note that the authors of [81] also had to apply this technique when solving a chemically reacting flow using SUPG. They also conclude that the elimination of spurious oscillations in the presence of steep concentration gradients is currently an unsolved problem.

3.2.2 Polydisperse flows

When we take into account the effect of coagulation, particles of different sizes can appear in the flow. We deal with this problem using the direct quadrature method of moments (DQMOM) of [39].

In a polydisperse flow, particles with many different sizes may coexist, so we need a way to describe what the different particle sizes are. To this end, we define a particle distribution function $f(v_p, \boldsymbol{x}, t)$. It depends on the particle volume v_p as well as position and time. The particle volume is the property we wish to track, while position and time are external variables.

In this sense, we can think of $f(v_p, \boldsymbol{x}, t)$ as representing a particle size distribution function in terms of particle volume, defined at every point in the domain and at every time. Integrating this function at a given position and time over a particle volume range $[v_a, v_b]$ will give the number of particles per unit of volume in the physical space at that point and at that time that have a particle volume comprised in the interval $[v_a, v_b]$. This also implies that f has units $[\mathrm{m}^{-6}]$. The choice of particle volume as internal coordinate allows some simplification of the coagulation source term equations (3.70) and (3.71).

If we now want to generalize the monodisperse particle model given by equation (3.41), we need to write the transport of the distribution function, instead of just the number concentration for a single size (using $f \equiv f(v_p, \boldsymbol{x}, t)$ for clarity):

$$\frac{\partial f}{\partial t} + \nabla \cdot (f\boldsymbol{v}) = D_b \nabla^2 f + S_f \tag{3.58}$$

This equation has the same form as (3.41), except for the additional source term S_f, which we will use to model the particle coagulation. More generally, this term would also include effects such as fragmentation, evaporation, nucleation and growth. Since we have no direct way to translate a transport of a distribution function into a numerical model, some kind of approximation is required. In [39], the function f is approximated by a weighted sum of Dirac delta functions:

$$f(v_p, \boldsymbol{x}, t) \approx \sum_{\alpha=1}^{N} f_\alpha(\boldsymbol{x}, t)\delta\left(v_p - v_\alpha(\boldsymbol{x}, t)\right) \tag{3.59}$$

The set of weights f_α and abscissas v_α associated with each Dirac function form different particle "phases", each with a number density function value f_α and particle volume v_α. They depend on time and position, though we will omit this from the notation from now on to avoid the clutter. The idea of the DQMOM method is to write transport equations for f_α and the weighted particle volume $\zeta_\alpha = f_\alpha v_\alpha$:

$$\frac{\partial f_\alpha}{\partial t} + \nabla \cdot (f_\alpha \boldsymbol{v}_\alpha) = D_\alpha \nabla^2 f_\alpha + a_\alpha \tag{3.60}$$

$$\frac{\partial \zeta_\alpha}{\partial t} + \nabla \cdot (\zeta_\alpha \boldsymbol{v}_\alpha) = D_\alpha \nabla^2 \zeta_\alpha + b_\alpha \tag{3.61}$$

These equations are very similar to the original equation for the concentration transport (3.41), but the difference in particle size means that each phase will also have a distinct particle velocity $\boldsymbol{v}_\alpha$. This is a consequence of the dependency of the particle relaxation time τ_p on the particle volume. For the same reason, we also have a different diffusion coefficient D_α for each phase. The terms a_α and b_α are source terms that are linked to the source term S_f in equation (3.58). We remind that these terms also depend

on the position and time. By formally substituting the distribution function (3.59) into equation (3.58) and using equations (3.60) and (3.61) and then applying a moment transform (details in [39]), it is possible to obtain the following equation for each moment k:

$$(1-k)\sum_{\alpha=1}^{N} v_\alpha^k a_\alpha + k\sum_{\alpha=1}^{N} v_\alpha^{k-1} b_\alpha = \bar{S}_k + \bar{C}_k \tag{3.62}$$

The moment source term $\bar{S}_k$ is given by the moment transform:

$$\bar{S}_k = \int_0^\infty v_p^k S_f(v_p)\,\mathrm{d}v_p \tag{3.63}$$

The term $\bar{C}_k$ arises from the diffusion and is defined as follows:

$$\bar{C}_k = k(k-1)\sum_{\alpha=1}^{N} v_\alpha^{k-2} f_\alpha D_\alpha \frac{\partial v_\alpha}{\partial x_i}\frac{\partial v_\alpha}{\partial x_i} \tag{3.64}$$

Since we have N unknown a_α and N unknown b_α, we need to write equation (3.62) for the first $2N$ moments (i.e. $k = 0, ..., 2N-1$) to obtain a linear system. Since the coefficients of the system matrix always contain 1 in the first two rows and powers of the particle volume in the other rows, it is necessary to normalize the particle volumes to obtain volumes near unity, or the matrix determinant will be too close to zero to obtain an accurate solution in double precision arithmetic. This system must then be solved at every node to obtain the values of a_α and b_α. Once these source terms are known in the whole domain, the transport equations (3.60) and (3.61) can be solved. Before we can do that, however, we need to define the source term S_f, which is the subject of the next section.

Particle coagulation: kinetic theory

Equation (3.58) describes the time rate of change of the particle distribution function. One possible source of change is the coagulation of particles. In this work, we consider only the possibility that two particles may collide to form a new one, i.e. we neglegt the interaction between 3 or more particles at the same time. We also deal only with spherical particles.

The expression for the source term can be derived using the kinetic theory of gases. To build up the source term, we start with the probability that a particle of volume v_a undergoes a collision as it moves a distance $\mathrm{d}s$ through a cloud of particles with volumes v_b. The situation is illustrated in figure 3.4, viewing the particle a (dark gray) and the cloud of particles with volume v_b along the direction that a is moving. Note that this configuration is identical to that used for the calculation of the mean free path length in a gas in [82]. The probability of collision is determined by the ratio of the area covered

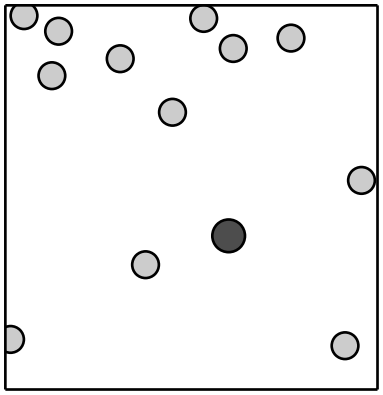


Figure 3.4: Dark-gray particle a moving over a distance $\mathrm{d}s$ through an area $\mathrm{d}A$ perpendicular to the direction of motion, encountering a cloud of light-gray particles b.

by particles and the empty area. For the area of the particles, we have to consider both particles. If they are closer together than the sum of their radii $r_a + r_b$, they will make contact, so the effective collision cross section is $\sigma_{ab} \equiv \pi(r_a + r_b)^2$, or in terms of the volume:

$$\sigma_{ab} = \pi \left(\left(\frac{3}{4\pi} v_a \right)^{\frac{1}{3}} + \left(\frac{3}{4\pi} v_b \right)^{\frac{1}{3}} \right)^2 \tag{3.65}$$

To get the total area over which a collision can happen, we just have to multiply σ_{ab} with the total number of particles b that can be encountered, assuming there is no overlap between particles as in figure 3.4. The number of particles b in the volume $\mathrm{d}s\,\mathrm{d}A$ is given by $f(v_b)\,\mathrm{d}v_b\,\mathrm{d}s\,\mathrm{d}A$. Multiplying with the collision cross section and dividing by the total area yields the probability that a single particle a will collide with any of the particles b:

$$\sigma_{ab} f(v_b)\,\mathrm{d}v_b\,\mathrm{d}s \tag{3.66}$$

Of course we do not have a single particle a, but their number is also given by the distribution function, so we must multiply (3.66) with $f(v_a)$. The result has the dimension of f and describes the number of collisions between a and b per unit volume and per elementary volume variation around v_a. To obtain a source term for equation (3.58) we still need to account for the time variation, which we do by dividing by an elementary time change $\mathrm{d}t$. The factor $\frac{\mathrm{d}s}{\mathrm{d}t}$ appears, which is equal to the relative velocity between particles a and b. This can be seen by supposing that group b is not moving

in figure 3.4 and particle a is then moving at a velocity equal to the relative velocity between both groups. Finally, we recognize that equation (3.58) is a function of v_p, so we need to consider all possible combinations of v_a and v_b that yield a particle of v_p. Writing $v_a = v_p - v_b$, we can now integrate over all possible v_b to obtain:

$$S_f^+ = \frac{1}{2}\int_0^{v_p} \beta\left(v_p - v_b, v_b\right) f\left(v_p - v_b\right) f\left(v_b\right)\, \mathrm{d}v_b \tag{3.67}$$

Note that we only integrate up to v_p since both colliding particles must be smaller than v_p to create a particle of volume v_p. We also divide by 2, since otherwise we would count each pair of particles twice. β is the collision kernel, and it encompasses the collision cross section and the relative velocity between any two particles α and γ:

$$\beta\left(v_\alpha, v_\gamma\right) = \left|\boldsymbol{v}_\alpha - \boldsymbol{v}_\gamma\right| \pi \left(\left(\frac{3}{4\pi} v_\alpha\right)^{\frac{1}{3}} + \left(\frac{3}{4\pi} v_\gamma\right)^{\frac{1}{3}}\right)^2 \tag{3.68}$$

So far, we have only considered the creation of new particles with volume v_p, which is why we wrote S_f^+ instead of just S_f in equation (3.67). When a particle p collides with any other particle b, however, it will form a new particle of a different volume, so this constitutes a removal from the number of particles with volume v_p. Following the same reasoning as above, we can write a sink term for this:

$$S_f^- = f(v_p) \int_0^{\infty} \beta\left(v_p, v_b\right) f\left(v_b\right)\, \mathrm{d}v_b \tag{3.69}$$

The final source term is then $S_f = S_f^+ - S_f^-$, where S_f^+ is typically called the birth rate due to coagulation and S_f^- the death rate. The expressions we obtained here are very similar to [83], except that there the particle velocity is included in the particle phase space.

To close equation (3.62), we have to compute the integral (3.63). Starting with S_f^+, we first approximate f using (3.59), yielding, after distributing the sum:

$$S_f^+ \approx \frac{1}{2}\sum_{\alpha=1}^{N}\sum_{\gamma=1}^{N}\int_0^{v_p} \beta\left(v_p - v_b, v_b\right) f_\alpha f_\gamma \delta\left(v_p - v_b - v_\alpha\right) \delta\left(v_b - v_\gamma\right)\, \mathrm{d}v_b$$

Using the property of Dirac delta functions where $\int_{-\infty}^{\infty} g(x)\delta(x-a)\,\mathrm{d}x = g(a)$ for any function g and knowing that $v_b \leq v_p$, we can eliminate the integral:

$$S_f^+ \approx \frac{1}{2}\sum_{\alpha=1}^{N}\sum_{\gamma=1}^{N} \beta\left(v_p - v_\gamma, v_\gamma\right) f_\alpha f_\gamma \delta\left(v_p - v_\gamma - v_\alpha\right)$$

To evaluate the moment transform (3.63) for the k-th moment, we need to multiply with v_p^k and integrate with v_p ranging to infinity. This allows us to remove the remaining Dirac function and yields:

$$\bar{S}_k^+ = \frac{1}{2}\sum_{\alpha=1}^{N}\sum_{\gamma=1}^{N}\left(v_\alpha + v_\gamma\right)^k \beta\left(v_\alpha, v_\gamma\right) f_\alpha f_\gamma \tag{3.70}$$

Applying the same transformation to S_f^- as for S_f^+ results in:

$$\bar{S}_k^- = \sum_{\alpha=1}^{N}\sum_{\gamma=1}^{N} v_\alpha^k \beta\left(v_\alpha, v_\gamma\right) f_\alpha f_\gamma \tag{3.71}$$

Coagulation in non-uniform flow

In the previous section, we derived a collision model based on a relatively straightforward kinetic theory. Unfortunately, this theory does not hold in the case of a non-uniform flow, even if we suppose the velocities are completely known as in i.e. DNS. The problem lies in the definition of the collision kernel β. When we examine equation (3.68), it becomes clear that the relative velocity $(\boldsymbol{v}_\gamma - \boldsymbol{v}_\alpha)$ in the above formulation is non-zero only for particles of different size. Saffman and Turner already pointed out in their famous paper [84] that turbulence also causes particles of the same size to collide, and they derive a formula for the collision kernel in isotropic turbulence. Further study of the literature reveals that the theory of section 3.2.2 is commonly known as the cylindrical formulation [85]. Using this description, the collision kernel can also be interpreted as a volumetric flow rate through a disk with radius $R_{\alpha\gamma} = r_\alpha + r_\gamma$. The underlying assumption is therefore that the velocity remains constant on a length scale of the order of the collision radius. In turbulent flow, this assumption is invalid, even on length scales as small as the collision radius. On this small scale, the relative velocity can be computed from the velocity gradient tensor, so it depends on the relative direction between the two particles. For this reason, the cylindrical approximation is often replaced by the spherical formulation, which can correctly model this anisotropy by considering all possible orientations. The difference between both approaches is discussed in great detail in [85]. The collision kernel is now written as the incoming volumetric flow rate through the collision sphere (figure 3.5). Following [86], we can split up the radial component of the relative velocity w_r as:

$$\begin{aligned} w_r^- &= w_r & w_r^+ &= 0 & (w_r < 0)\\ w_r^- &= 0 & w_r^+ &= w_r & (w_r > 0) \end{aligned}$$

Now, the integral of the incoming flux over the collision sphere is the collision kernel:

$$\beta_{\alpha\gamma} = R_{\alpha\gamma}^2 \int_0^{2\pi} \mathrm{d}\phi \int_0^{\pi} -w_r^- \sin\theta \,\mathrm{d}\theta \tag{3.72}$$

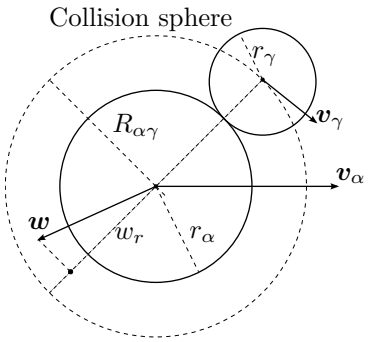


Figure 3.5: The collision sphere model for two colliding particles α and γ.

Let us now determine an expression for w_r, applied to the situation where we have DNS data for the flow field, from which we derive particle velocities using equation (3.50). Suppose we have a particle α moving with a velocity $\boldsymbol{v}_\alpha$ and a second particle γ at a distance $R_{\alpha\gamma} = r_\alpha + r_\gamma$. Its velocity at that location is then, using index notation for the gradient and vector between the particles:

$$\boldsymbol{v_\gamma}|_{\boldsymbol{x}=\boldsymbol{x}_\gamma} = \boldsymbol{v_\gamma} + (R_{\alpha\gamma})_j \frac{\partial (v_\gamma)_i}{\partial x_j} \tag{3.73}$$

The particle location is only indicated when $\boldsymbol{x} \neq \boldsymbol{x}_\alpha$. The relative velocity is then:

$$\boldsymbol{w} = \boldsymbol{v_\gamma}|_{\boldsymbol{x}=\boldsymbol{x}_\gamma} - \boldsymbol{v}_\alpha = \boldsymbol{v_\gamma} - \boldsymbol{v_\alpha} + (R_{\alpha\gamma})_j \frac{\partial (v_\gamma)_i}{\partial x_j} \tag{3.74}$$

The difference $\boldsymbol{v_\gamma} - \boldsymbol{v_\alpha}$ is as before only due to the different particle size, while the gradient term encapsulates the effect of non-uniform flow. In [85], it is shown that the cylindrical and spherical approaches are equivalent when the flow is uniform, so we can keep our previously found kernel (3.68) and apply the spherical approach only to the velocity difference due to the gradient. Both kernels can then be added to find the global collision kernel.

The radial component of the relative velocity is:

$$w_r = \frac{1}{|R_{\alpha\gamma}|} (R_{\alpha\gamma})_j \underbrace{\frac{\partial (v_\gamma)_i}{\partial x_j}}_{g_{ij}} (R_{\alpha\gamma})_i \tag{3.75}$$

If we assume a direct simulation of the flow, the elements of the gradient tensor g_{ij} are constant, known coefficients. The challenge now lies in the computation of the integral (3.72) as an easy to evaluate function of these coefficients. To do this, we need to either determine where w_r is negative, or combine integrals of w_r and its absolute value over the entire sphere. Since integrating the absolute value also requires knowledge of the location of sign changes, both methods present the same difficulty. Because of the symmetry

of the multiplication with the radius vector, we can replace g_{ij} with the rate of strain tensor $s_{ij} = (g_{ij} + g_{ji})/2$, reducing the number of coefficients to 6. If in equation (3.75) we let the components of the separation vector be (x, y, z), then the equation $w_r = 0$ represents a quadric surface. Its principal axes are given by the eigenvectors of the strain rate matrix s_{ij}. Since we are integrating over a sphere, the orientation of the quadric surface in space is of no importance, and the problem becomes much simpler if we can work in a coordinate system that is aligned with the principal axes. Because s_{ij} is real and symmetric, we can always find an orthonormal set of eigenvectors and replace s_{ij} by the diagonal matrix of its eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$. This proces yields the following equation for the locus where the radial relative velocity is zero:

$$\lambda_1 x^2 + \lambda_2 y^2 + \lambda_3 z^2 = 0 \tag{3.76}$$

The shape of this surface can be classified according to the signs of the eigenvalues. To simplify further, we assume $\lambda_1 \geq \lambda_2 > 0$ and $\lambda_3 < 0$. Later we will show how other cases can be reduced to this one, or solved more easily. Dividing by $-\lambda_3$ and setting $k_1 = -\lambda_1/\lambda_3$ and $k_2 = -\lambda_2/\lambda_3$ yields:

$$k_1 x^2 + k_2 y^2 = z^2 \tag{3.77}$$

This is the equation for an elliptic cone, with negative values of w_r found only inside its volume. The problem of evaluating integral (3.72) can now be stated geometrically: integrate $-w_r$ over the surface of the parts of a sphere that lie inside an elliptic cone. This is illustrated in figure 3.6. To

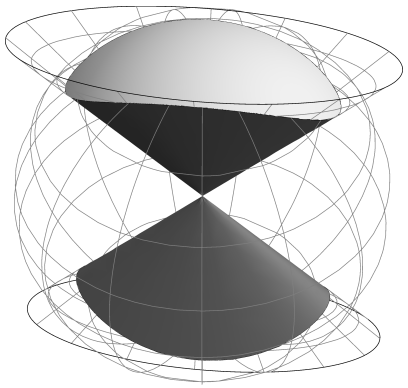


Figure 3.6: Intersection of an elliptic cone (aligned with the Z-axis) and a sphere. Negative radial relative velocities are found only inside the cone and must be integrated over the light-gray part of the sphere.

simplify the integration procedure, we will use spherical coordinates. As $R_{\alpha\gamma}$ only appears as a constant factor, we will ignore it in the derivations

and multiply the final result with $R^3_{\alpha\gamma}$. Using $x = \cos\phi\sin\theta$, $y = \sin\phi\sin\theta$ and $z = \cos\theta$, equation (3.77) becomes:

$$\sin^2\theta\left(k_1\cos^2\phi + k2\sin^2\phi\right) - \cos^2\theta = 0 \tag{3.78}$$

Figure 3.6 shows that the boundary of the spherical cap ranges over all values of the azimuthal angle, i.e. $0 \leq \phi \leq 2\pi$, while the polar angle θ ranges from 0 to some value determined by the intersection. To find this value, we solve equation (3.78) for θ. Since the problem is symmetric, only values in the range $0 \leq \theta \leq \pi/2$ are of interest, and the solution is:

$$\theta_{\max} = \cos^{-1}\left(\sqrt{\frac{k_1\cos^2\phi + k_2\sin^2\phi}{k_1\cos^2\phi + k_2\sin^2\phi + 1}}\right) \tag{3.79}$$

The integrand of equation (3.72) is minus the left hand side of equation (3.78) multiplied with $\sin\theta$. The primitive function is:

$$\frac{1}{3}\cos\theta\left(3\left(k_1\cos^2\phi + k_2\sin^2\phi\right) - \cos^2\theta\left(k_1\cos^2\phi + k_2\sin^2\phi + 1\right)\right)$$

Substituting the integration limits yields:

$$\frac{1}{R_{\alpha\gamma}}\int_0^{\theta_{\max}}\frac{-w_r^-}{-\lambda_3}\sin\theta\,\mathrm{d}\theta = \frac{2\left(k_1\cos^2\phi + k_2\sin^2\phi\right)^{\frac{3}{2}}}{3\sqrt{k_1\cos^2\phi + k_2\sin^2\phi + 1}} - \frac{1}{3}\left(2k_1\cos^2\phi + 2k_2\sin^2\phi - 1\right)$$

This expression must be integrated over ϕ. Because of the symmetry, we also restrict this to $0 \leq \phi \leq \pi/2$, remembering that the result is a factor 8 too small. The second term, corresponding to $\theta = 0$, is easy to integrate. The first term corresponds to the limit imposed on θ by the intersection of the cone and the sphere. It is impossible to find a closed form for this integral. We can prove this using figure 3.6 and the divergence theorem: the integral we seek is also the volume integral of $\nabla \cdot \boldsymbol{w}$ in the solid volume minus the surface integral of $\boldsymbol{w}$ over the dark gray surface. The integral over the dark surface is closely related to the calculation of the circumference of an ellipse, so it has no closed form. The final expression for the collision kernel is then:

$$\beta_{\alpha\gamma} = 8(-\lambda_3)R^3_{\alpha\gamma}\left(\int_0^{\frac{\pi}{2}}\frac{2\left(k_1\cos^2\phi + k_2\sin^2\phi\right)^{\frac{3}{2}}}{3\sqrt{k_1\cos^2\phi + k_2\sin^2\phi + 1}}\,\mathrm{d}\phi - \frac{\pi}{6}\left(k_1 + k_2 - 1\right)\right) \tag{3.80}$$

The second term here is proportional to the divergence of the particle velocity field, while the integral in the first term requires numerical evaluation.

The function is smooth over the integration interval and we find that a six point Gaussian quadrature yields satisfactory results. To complete the discussion, we must still account for all possible combinations of eigenvalue signs.

First, consider the case of only one positive eigenvalue, i.e. $\lambda_1 \leq \lambda_2 < 0$ and $\lambda_3 > 0$. With respect to the above procedure, the sign of the cone expression changes, so we are calculating the integral of $-w_r^+$. Following [86], this can be used to retrieve the integral of w_r^- as follows:

$$\int_{\Omega_R} w_r^- \, \mathrm{d}\Omega_R = \int_{\Omega_R} w_r \, \mathrm{d}\Omega_R - \int_{\Omega_R} w_r^+ \, \mathrm{d}\Omega_R \tag{3.81}$$

The integral of w_r is easy to find, using directly the full expression of w_r:

$$R_{\alpha\gamma}^2 \int_0^{2\pi} \mathrm{d}\phi \int_0^{\pi} \underbrace{R_{\alpha\gamma} \left(\sin^2\theta \left(\lambda_1 \cos^2\phi + \lambda_2 \sin^2\phi \right) + \lambda_3 \cos^2\theta \right)}_{w_r} \sin\theta \, \mathrm{d}\theta = \frac{4\pi R_{\alpha\gamma}^3}{3} \left(\lambda_1 + \lambda_2 + \lambda_3 \right) \tag{3.82}$$

Combining equations (3.81) and (3.82), we can use the result of (3.80) (i.e. the integral of $-w_r^+$ now) and add the result of equation (3.82). The collision kernel is then the opposite of this value. Working this out yields:

$$\beta_{\alpha\gamma} = 8\lambda_3 R_{\alpha\gamma}^3 \left(\int_0^{\frac{\pi}{2}} \frac{2 \left(k_1 \cos^2\phi + k_2 \sin^2\phi \right)^{\frac{3}{2}}}{3\sqrt{k_1 \cos^2\phi + k_2 \sin^2\phi + 1}} \, \mathrm{d}\phi \right) \tag{3.83}$$

Note that the result (3.82) is proportional to the divergence of the particle velocity field. This shows the connection of the current theory to i.e. [84, 87, 85], where zero-inertia particles are studied in incompressible flow. In this case the particle velocity field is divergence-free everywhere and the absolute values of w_r^+ and w_r^- are equal. The collision kernel then reduces to the integral of $|w_r|$ over half the sphere. In our formulation, equations (3.83) and (3.80) would yield the same value except for the sign.

A second possibility is that of having one of the eigenvalues equal to 0. Setting $\lambda_1 > \lambda_2 = 0$ and $\lambda_3 < 0$, the cone degenerates into a wedge, but expression (3.80) remains valid. Due to the simplification, however, a closed form can be found:

$$\beta_{\alpha\gamma} = 8(-\lambda_3) R_{\alpha\gamma}^3 \left(\frac{1}{3} \left(\sqrt{k_1} + (k_1 - 1) \tan^{-1} \left(\sqrt{k_1} \right) \right) - \frac{\pi}{6} (k_1 - 1) \right) \tag{3.84}$$

In practice, this can be used when one of the eigenvalues gets too close to 0, to avoid using a very small λ_3 that would produce very high values of

k_1 or k_2, which would degrade the accuracy of the numerical integration. It is also the most general case in 2D simulations, so there the numerical integration is avoided.

The final possibility is that of all eigenvalues having the same sign, possibly including one or more zeros. If they are all positive, w_r is positive for any direction and the collision kernel is 0. If they are all negative, the problem reduces to the opposite of equation (3.82), i.e.:

$$\beta_{\alpha\gamma} = -\frac{4\pi R_{\alpha\gamma}^3}{3}\left(\lambda_1 + \lambda_2 + \lambda_3\right) \tag{3.85}$$

Table 3.3 presents a summary of all the possibilities, recalling that the eigenvalues may always be reordered to fit exactly one of these cases. We can

λ_1	λ_2	λ_3	$\beta_{\alpha\gamma}$
> 0	> 0	< 0	Equation (3.80)
< 0	< 0	> 0	Equation (3.83)
> 0	0	< 0	Equation (3.84)
≤ 0	≤ 0	≤ 0	Equation (3.85)
> 0	> 0	> 0	0

Table 3.3: The possible combinations of eigenvalues of the particle rate of strain tensor.

immediately verify that the method recovers the correct result in the case of pure shear flow. Setting $\partial v_x/\partial z = a$, the eigenvalues of the rate of strain tensor are: $\lambda_1 = a/2$, $\lambda_2 = 0$ and $\lambda_3 = -a/2$. This case is covered by equation (3.84) with $k_1 = 1$, yielding:

$$\beta_{\alpha\gamma} = \frac{4}{3} a R_{\alpha\gamma}^3 \tag{3.86}$$

This corresponds to the correct value as given in i.e. [85] and first computed by von Smoluchowski in 1917. After deriving the above equations, we discovered —unsurprisingly— that this has been done before. In [88], the equations are developed assuming an incompressible particle velocity field (i.e. the limit of zero inertia), while [89] generalizes this for a compressible field as we have done. We do however make a small new contribution with the geometrical interpretation of the problem (figure 3.6) and the closed form for a single zero eigenvalue in equation (3.84).

Brownian coagulation

So far, we only considered the velocity gradient and particle relaxation time differences as possible sources of relative particle velocities. In reality, for very small particles, the Brownian motion also causes relative motion between particles. In our experiments, we always have $\mathrm{Kn} < 1$, so we can use

the Brownian coagulation kernel for the continuum regime as given in e.g. [9]:

$$\beta^{\mathrm{B}}_{\alpha\gamma} = \frac{2k_B T}{3\mu}\left(\frac{C_\alpha}{d_\alpha} + \frac{C_\gamma}{d_\gamma}\right)(d_\alpha + d_\gamma) \tag{3.87}$$

Here, we use the Cunningham slip correction factor as in [12]: $C_\alpha = 1 + 1.591\,\mathrm{Kn}_\alpha$. In [90], a more general formulation is given that is also valid much smaller particles, i.e. in the free molecular regime where $\mathrm{Kn} > 1$. As shown in e.g. [91], the Brownian collision kernel may not be just added to our previously found kernels, although this is still a common approximation. In [92], using a quadratic interpolation between the collision kernel due to fluid motion and the Brownian collision kernel was found to be a better approach.

The log-normal distribution

The log-normal distribution is often used as an approximation for real distributions, especially when considering Brownian coagulation [93, 94]. Deviations may be significant when sedimentation of particles becomes important [95]. Nevertheless, the log-normal approximation is widely used and provides a useful test case for the collision model when considering only Brownian coagulation. In this section, we will present the equations for the log-normal distribution, as well as the relations between the various possible formulations using different dependent and independent variables. We will then use the moments for the distribution to calculate the Dirac delta function approximation and finally compare the results of the DQMOM method with [93].

A log-normal distribution for the particle number concentration as a function of the particle diameter can be written as:

$$f(d_p) = \frac{n_t}{\sqrt{2\pi}\ln\sigma_g d_p}\exp\left(\frac{-\left(\ln\frac{d_p}{\mu_g}\right)^2}{2\ln^2\sigma_g}\right) \tag{3.88}$$

Here, μ_g and σ_g are geometric mean and standard deviation of d_p, i.e when sampling n particles:

$$\mu_g = \sqrt[n]{\prod_{i=1}^{n} d_i} \tag{3.89}$$

$$\sigma_g = \exp\left(\sqrt{\frac{\sum_{i=1}^{n}\left(\ln\frac{d_i}{\mu_g}\right)^2}{n}}\right) \tag{3.90}$$

From these definitions, it is seen that μ_g has length units while σ_g is dimensionless. In the following expressions, μ_g always appears in a ratio with

d_p in the exponent, thus ensuring dimensional correctness. Note that $\ln \mu_g$ and $\ln \sigma_g$ are the mean and standard deviation of the normal distribution of $\ln d_p$. The factor n_t is the total number density of the particles, i.e. the result of integrating the distribution over all particle sizes.

Since we use the particle volume v_p as independent variable it is useful to rewrite equation (3.88) as a function of $v_p = (\pi/6)d_p^3$. The change of variables results in:

$$f(v_p) = \frac{n_t}{\sqrt{2\pi} 3 \ln \sigma_g v_p} \exp\left(\frac{-\left(\ln \frac{6v_p}{\pi \mu_g^3}\right)^2}{18 \ln^2 \sigma_g} \right) \tag{3.91}$$

Note that this is equivalent to using the geometric mean and standard deviation of the particle volume.

Instead of the number concentration, some measurements use the volume fraction as dependent variable. For this, equation (3.88) needs to be multiplied with the volume of the particle v_p, yielding:

$$f_x(d_p) = \frac{\pi n_t d_p^2}{6\sqrt{2\pi} \ln \sigma_g} \exp\left(\frac{-\left(\ln \frac{d_p}{\mu_g}\right)^2}{2 \ln^2 \sigma_g} \right) \tag{3.92}$$

The factor d_p^3 can be brought into the exponent as $\exp(3 \ln d_p)$, clearly demonstrating that the resulting distribution is still log-normal, although the expression is more complicated:

$$f_x(d_p) = \frac{n_t}{\sqrt{2\pi}\sigma_d d_p} \frac{\pi}{6} \mu_g^3 \exp\left(\frac{9}{2} \ln^2 \sigma_g \right) \exp\left(\frac{-\left(\ln d_p - \mu_x\right)^2}{2 \ln^2 \sigma_g^2} \right) \tag{3.93}$$

Here, the mean is $\mu_x = \ln \mu_g + 3 \ln^2 \sigma_g$.

Once the distribution is known, we can derive a Dirac delta function approximation as input for boundary or initial conditions for the simulations. To get an approximation using N phases, we can compute the first $2N$ moments of the distribution and solve the polynomial system for f_α and v_α with $0 \leq k \leq 2N - 1$:

$$\sum_{\alpha=1}^{N} f_\alpha v_\alpha^k = \int_0^\infty v_p^k f(v_p) \, \mathrm{d}v_p \tag{3.94}$$

Application

Here, we consider a simple application using initially log-normally distributed particles undergoing Brownian coagulation, suitable for comparison with the results from [93]. Since the particle diameters are on the order of microns or

even less, we normalize all diameters using the geometric mean, so all computations can be done using a dimensionless $\mu_{gn} = 1$. The solution of the polynomial system for a monovariate distribution as is the case here could be done efficiently using the Product-Difference algorithm [38], but since we only need to solve it once to obtain initial conditions we used Mathematica[1] to solve the system. Table 3.4 lists the parameters for the Dirac delta ap-

σ_g	v_1	v_2	f_1	f_2
1.5	0.92677	25.049	0.99293	7.0661×10^{-3}
2.0	4.4908	2.6269×10^{4}	1.0000	2.2352×10^{-6}
3.0	1.1961×10^{2}	3.2339×10^{11}	1.0000	7.1054×10^{-15}

Table 3.4: Values of the abscissa v_α and weights f_α, with $N = 2$ and for three different geometric standard deviations.

proximation of equation (3.59), using two phases. The geometric standard deviation has a dramatic effect on the values of the parameters, resulting in a steep increase for the abscissa and a strong decrease of the weight of the second Dirac function. If we choose more than 2 Dirac functions, the third volume and weight take even more extreme values. These results are due to the nature of the log-normal distribution, with the moment values increasing sharply as the order and σ_g increase.

In [93], analytical formulas are derived for the evolution of the parameters of a log-normal size distribution for particles undergoing Brownian coagulation in the continuum regime, i.e. using kernel (3.87) without Cunningham correction. Figures 3.7 to 3.9 compare the results of our DQMOM implementation with the analytical results. The quality of the results depends greatly on the value of σ_g. For $\sigma_g = 1.5$, all quantities are reproduced correctly. For the decay in number concentration, the DQMOM appears to be insensitive to the changes in standard deviation, with the error reaching a value of up to 20% for $\sigma_g = 3$. Likewise, there is a tendency to underpredict the geometric mean diameter, as seen in figure 3.8. The standard deviation, on the other hand, is reproduced quite well even for a the initial σ_g of 3 (figure 3.9).

If we reconstruct log-normal distributions using the mean and standard deviation obtained from the DQMOM, the results from figures 3.10 and 3.11 are obtained. Again, the deviation is greatest for the highest σ_g.

We believe the above difficulties stem from the properties of the log-normal distribution, which lends itself poorly to approximation by Dirac delta functions. This is evident from table 3.4, where the values for $\sigma_g = 3$ are very close to a monodisperse distribution, with the weight f_2 almost zero. Increasing the number of phases also does not help: the next Dirac function will have a lower weight and higher volume still, and in fact numerical

[1] `http://www.wolfram.com`

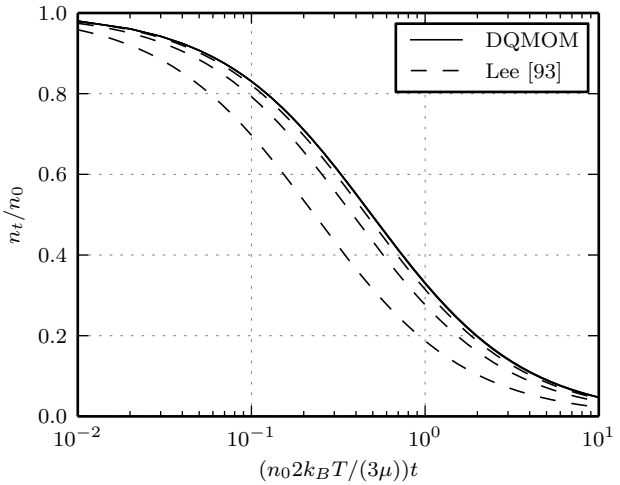


Figure 3.7: Evolution of the number concentration n_t with respect to the initial concentration n_0. Initial geometric standard deviations, from top to bottom: 1.5, 2.0, 3.0.

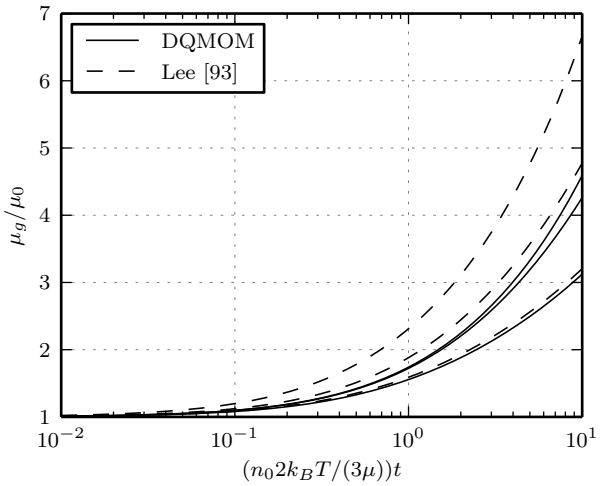


Figure 3.8: Evolution of the normalized mean particle diameter. Initial geometric standard deviations, from top to bottom: 1.5, 2.0, 3.0.

instabilities arise making a solution impossible. In [93], it was found that σ_g evolves to an asymptotic value of about 1.32, which is below the value of 1.5 for which the DQMOM still produces accurate results. Therefore, it

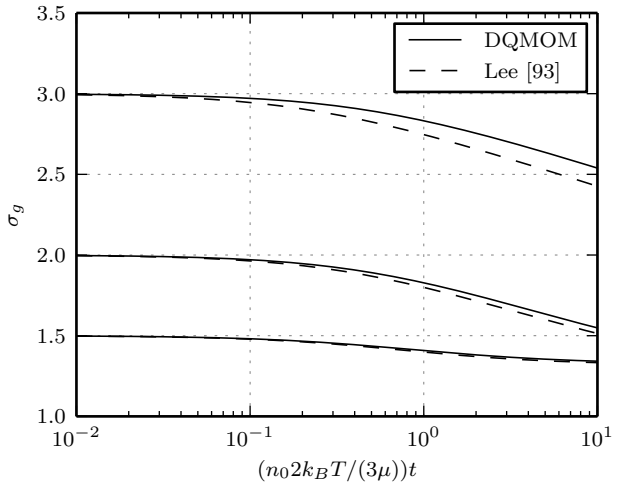


Figure 3.9: Evolution of the geometric standard deviation. Initial geometric standard deviations, from top to bottom: 1.5, 2.0, 3.0.

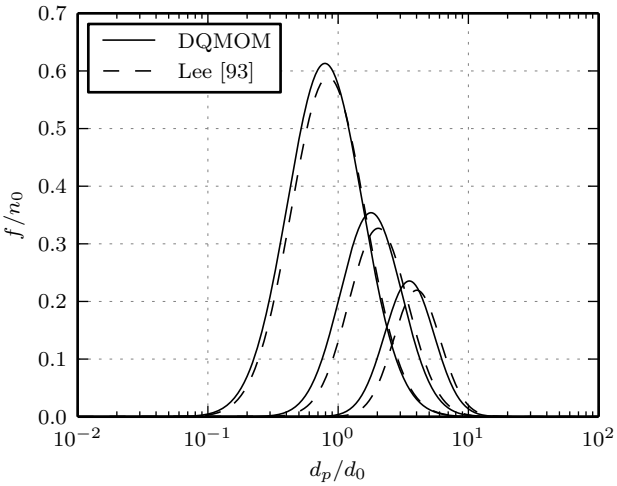


Figure 3.10: The reconstructed log-normal distributions at dimensionless times $(n_0 2k_B T/(3\mu))t$ 0.25, 2.5 and 10 for initial $\sigma_g = 2$

is expected that for distributions resulting from coagulation, no great issues regarding accuracy should arise.

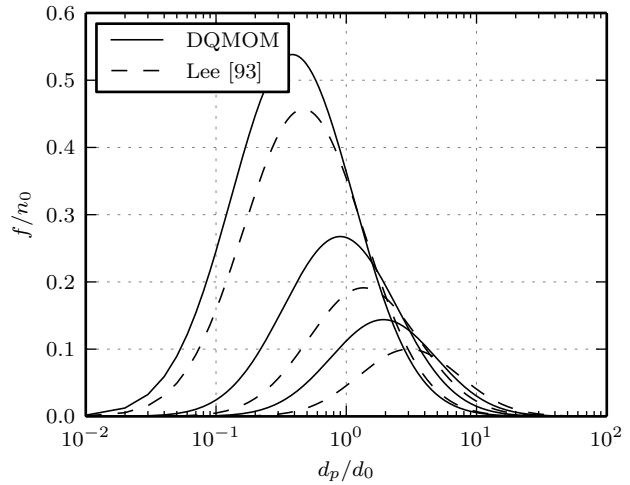


Figure 3.11: The reconstructed log-normal distributions at dimensionless times $(n_0 2k_B T/(3\mu))t$ 0.25, 2.5 and 10 for initial $\sigma_g = 3$

Colission efficiency

We close this discussion of particle coagulation with an important remark regarding the assumptions of our model. The above theory equates collision with coagulation. Indeed, if we apply equations (3.70) and (3.71) directly to close the system (3.62), then we have assumed that every collision results in coagulation. As explained in i.e. [9] and [83], a collision efficiency needs to be taken into account. For the sake of simplicity, we will ignore this issue in the current work and continue to assume that every collision leads to coagulation, with exact mass conservation.

Algorithm implementation

The algorithm for polydisperse flow is assembled from the transport equations for monodisperse flow, so it can be seen as a layer on top of a single transport equation in the form of (3.58), but solving this equation multiple times and calculating the source terms as needed. No additional finite element discretizations are needed. The algorithm works as follows:

1: Choose a set of N particle volumes v_α and distribution function values f_α
2: Compute the fluid flow field
3: **for all** α in 1..N **do**
4: Compute the velocity field $\boldsymbol{v}_\alpha$ using equation (3.56)

5: Compute the particle velocity field gradients at the nodes, using the average of surrounding elements
6: **end for**
7: **for all** nodes in the mesh **do**
8: Compute the moment source terms using equations (3.70) and (3.71). The coagulation kernel is the sum of equation (3.68) and the result of table 3.3.
9: Compute the diffusion effect using (3.64)
10: Solve the linear system (3.62) (with $2N$ equations)
11: **end for**
12: **for all** α in 1..N **do**
13: Solve equations (3.60) and (3.61)
14: **end for**
15: Start the next time step, i.e. go to step 2.

Chapter 4

Domain specific language

The application of the finite element method (FEM) requires the discretization of the integral form of the partial differential equations that govern the physics, thus transforming the problem into a set of algebraic equations that can be solved numerically. In essence, the resulting numerical model only depends on the governing equations and the set of basis and test functions. From the point of view of a model developer, it would therefore be ideal to only need to specify these parameters in a concise form that closely resembles the mathematical formulation of the problem, without sacrificing computational efficiency.

Our implementation is part of Coolfluid 3 [7], a C++ framework intended primarily for the solution of fluid dynamics problems and available as open source under the LGPL v3 license. The system is designed around a "Component" class that can provide a dynamic interface through Python scripting and a GUI. Model developers will typically write a set of Components to build a solver for a given set of equations, using functionality provided by our framework and external libraries. Problem-dependent settings such as the mesh, boundary conditions and model parameters can then be controlled through Python or a GUI, though these aspects are beyond the scope of the current work. In this context, we provide an Embedded Domain Specific Language (EDSL) that can be used to implement finite element solver components. It uses a notation similar to the mathematical formulation of the weak form of a problem (see e.g. equations (3.6) - (3.13) and (3.52) - (3.53)), allowing the programmer to focus on the physics rather than coding details. We assume a typical finite element workflow, where a global linear system is assembled from element matrix contributions. Our language can also describe boundary conditions and field arithmetic operations.

All language elements consist of standard C++ code, so they easily embed into the components of our framework. An extension mechanism is available, allowing any developer to implement his own language elements without touching the library code. The performance overhead will be shown to be quantified and compared to the global solution time for our applications. The implementation uses the Boost.Proto [96] framework, which builds on template meta programming techniques for generating highly efficient code.

The automatic generation of code based on an intuitive specification of the physical problem is of course a feature that is highly desirable for any numerical simulation framework. One example of previous work providing

such functionality is OpenFOAM [97], a C++ library primarily intended for fluid mechanics. It allows for the easy expression of differential equations by providing an embedded tensor manipulation language, specific to finite volume discretizations. The FEniCS project [98] provides an easy and efficient method for developing finite element discretizations. The approach differs from ours in that the language is embedded into Python and compiled using a just-in-time compiler. It also offers the option of automated analytical evaluation of integrals [99]. Another example is the FEEL++ project [100], which also provides a domain specific language embedded into C++ using expression templates.

The existence of all these excellent projects notwithstanding, we decided to implement our own framework for tight integration with the Coolfluid data structures and to support our workflow of building user-configurable components. The implementation relies on expression templates, just like FEEL++, but we use Boost.Proto to handle the expression template generation and to define a grammar for our language, resulting in simpler and easier to maintain code. Even though our language is not as feature-complete as the more established FEEL++ and FEniCS projects, we do believe that the easy integration of user-defined extensions into the language is a unique feature. The purpose of the current chapter is to present how Proto can be used to construct a language for finite element modeling —based on a generic example— and to show the capabilities and performance of the language we developed.

This chapter is structured as follows. In section 4.1 the mathematical formulation of the integral form will be laid out in order to clearly define the class of problems that is supported. Next, in section 4.2 the mechanics for constructing and interpreting the EDSL are explained in detail, based on a simple application. Section 4.3 contains some application examples —including the SUPG method from section 3.1.1— and section 4.4 a performance analysis. Finally, in section 4.5 we present our conclusions and suggest future work.

4.1 Finite element discretization

This section presents a brief overview of the finite element method, based on the weighted residual Galerkin formulation as reviewed in [101]. The objective is to introduce the notation with which we align our language. Consider the following differential equation with a linear operator L, to be solved over a domain Ω:

$$L(u) = 0 \tag{4.1}$$

The above equation (together with the boundary conditions) represents the mathematical model for the phenomenon we want to study, and the solution u is a function of the independent variables, typically the spatial dimensions and time. We now multiply each equation j from equation (4.1) with a

weighting function v and then integrate over the domain:

$$\int_{\Omega} vL\left(u\right) \,\mathrm{d}\Omega = 0 \tag{4.2}$$

If a solution satisfies this integral formulation for all possible weighting functions, it also solves equation (4.1). The Finite Element Method approximates the unknown function u using a linear combination of n shape functions $N_i\left(\boldsymbol{x}\right)$ with the unknown coefficients u_i:

$$u \approx \sum_{i=1}^{n} N_i\left(\boldsymbol{x}\right) u_i \equiv \tilde{u} \tag{4.3}$$

The shape functions depend only on the spatial coordinates $\boldsymbol{x}$ and to simplify the notation we will from here on just write N_i instead of $N_i\left(\boldsymbol{x}\right)$. We can also write the linear combination for one unknown in vector form as $\tilde{u} = N_u \boldsymbol{u}_e$. Here, N is a row vector with the shape functions associated with unknown u and $\boldsymbol{u}_e$ is the column vector of the unknowns coefficients.

Due to the linearity of the operator we have $L\left(N\boldsymbol{u}_e\right) = L\left(N\right)\boldsymbol{u}_e$. This hypothesis is acceptable for the applications targeted by our framework. In the Galerkin method, equation (4.2) is written by substituting v with the shape function associated with each coefficient. This results in a discrete linear system with unknowns $\boldsymbol{u}_e$:

$$\int_{\Omega} N^{\mathrm{T}} L\left(N\right)\boldsymbol{u}_e \,\mathrm{d}\Omega = \boldsymbol{0} \tag{4.4}$$

If we choose piecewise shape functions, defined over n_Ω elements Ω_e that subdivide the problem domain, the global integral can be replaced by a sum of integrals:

$$\sum_{e=1}^{n_\Omega} \int_{\Omega_e} N_u^{\mathrm{T}} L\left(N_u\right)\boldsymbol{u}_e \,\mathrm{d}\Omega_e = \boldsymbol{0} \tag{4.5}$$

For each element, the contributions to the system matrix are $\int N_u^{\mathrm{T}} L\left(N_u\right) \,\mathrm{d}\Omega_e$, which we call the element matrix A. It is a square matrix with a size equal to the number of coefficient values (nodes) in the element. Each entry of the element matrix contributes to a value in the global matrix, and only elements that share a given node need to participate in the sum for its global value. We also assume that the calculation of the shape function values in each element happens in a local coordinate system, replacing the dependency on the global coordinates $\boldsymbol{x}$ mentioned earlier by a dependency on mapped coordinates $\boldsymbol{\xi}$, again omitted to keep the notation clear.

In the case of a system of linear differential equations, we have a vector of unknowns $\boldsymbol{u}$ and there is an equation for each unknown. In this case, the above procedure is applied to each equation and variable, where we are free to choose a different shape function for each unknown. The matrix

notation of equation (4.5) can be preserved in this case, but $\boldsymbol{u_e}$ is now a column vector with all of the unknowns for an element, forming a segment of n_i unknowns for each unknown u_i, where n_i is the number of coefficients that is used in its associated shape function. The shape function vector is arranged in the same way, so the final element matrix can be divided into blocks associated with each combination of unknowns, as in equation (3.5). To distinguish the shape function associated with each unknown, we add an index to the shape function vector, so shape functions associated with u are written as N_u

The matrix assembly procedure is the same for all problems, only the values of the element matrices depend on the initial equation (4.1). This observation drives the current work: we will present a convenient language to write out the element equations and automate the steps that are common for all finite element discretizations. Conceptually, we support any approximation that fits into the above formulation, but for now only shape functions of the Lagrange family are implemented.

As an example, we consider the steady heat conduction equation $k\nabla^2 T + \dot{q} = 0$, where T is the unknown temperature, k is the thermal conductivity and $\dot{q}$ is a volumetric heat source term. We choose interpolating shape functions so that $\tilde{T} = N_T \boldsymbol{T}_e$ and $\tilde{\dot{q}} = N_{\dot{q}} \boldsymbol{\dot{q}}_e$, allowing for different shape functions for the heat and temperature. Introducing these into equation (4.5) and applying integration by parts to the Laplacian yields the following discrete linear system:

$$\sum_{e=1}^{n_\Omega} \int_{\Omega_e} k \left(\nabla N_T\right)^{\mathrm{T}} \nabla N_T \boldsymbol{T}_e \,\mathrm{d}\Omega_e - \int_{\Gamma_e} N^{\mathrm{T}} \left(\nabla N_T \boldsymbol{T}_e\right) \cdot \mathbf{d}\boldsymbol{n} - \int_{\Omega_e} N_T^{\mathrm{T}} \tilde{\dot{q}} \,\mathrm{d}\Omega_e = \mathbf{0} \tag{4.6}$$

The first term contains the element matrix: $A = \int k \left(\nabla N_T\right)^{\mathrm{T}} \nabla N_T \,\mathrm{d}\Omega_e$. The second term is the integral over the boundary of the normal derivative of the temperature, i.e. the surface heat flux. It can be used to set a Neumann boundary condition or must vanish when we impose a Dirichlet boundary condition. The third term depends on the imposed nodal values of the heat source term, so it contributes to the right hand side of the linear system. From this simple example, the basic requirements for our language can be identified: provide access to the shape functions and their derivatives, as well as nodal values of the variables; control the evaluation of integrals; and allow the application of boundary conditions. Finally, we must be able to express the assembly into global matrices and vectors.

4.2 Construction of the language

Our language consists of three levels of implementation, as shown in figure 4.1. The top level contains the language itself, which consists of keywords that the user combines into an expression. Because the language is embed-

ded, only valid C++ expressions are allowed, but we can define the interpretation freely via operator overloading. This is the job of the algorithm implementation layer. Here, the language is parsed and appropriate actions —called semantic actions in language grammar terms— are linked to each part of the input. Some of the actions may result in calls to a shape function library or matrix operations, so the final layer contains all external libraries needed to execute these. The remainder of this section will focus on each layer in more detail, using a simple stand-alone example. This allows us to easily demonstrate the techniques used in Coolfluid 3 while avoiding the need to explain the details of our mesh structure and linear algebra interface, which are beyond the scope of the present work. The example will result in a program that can use a shape function to interpolate the coordinates on each element and print out the result, using the following syntax:

```
1 // Create a mesh
2 mesh_data some_mesh;
3 // ... mesh initialization skipped here
4 // Loop over the elements and print out the centroid coordinates
5 for_each_element(some_mesh, cout_ << N(centroid)*element_coords << "\n");
```

The second argument in the call at line 5 is a Proto expression, where `N` is the shape function, `centroid` represents the mapped coordinates of element center and `element_coords` represents the coordinates of all element nodes. In what follows, we will show how we can evaluate this expression.

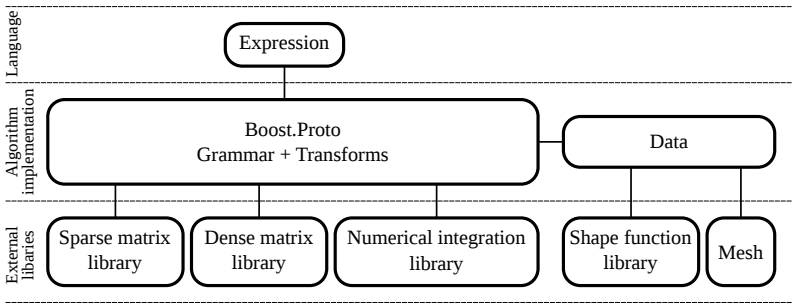


Figure 4.1: The structure of the program, showing the three different levels of the implementation

4.2.1 The language layer

Using the Boost.Proto [96] library, expressions are constructed from so-called “terminals”. These are C++ objects that are instances of the Proto terminal class, which overloads all of the C++ operators. Combining different terminals using operators builds an expression that can be evaluated later. This expression is analogous to the expression templates used in e.g. FEEL++ [100]. Building such an expression is extremely easy: listing 4.1 is

a complete C++ program, where lines 20 and 21 contain expressions, built from the terminals defined at lines 12 to 15. The terminals can be combined using any of the overloadable C++ operators. Because this includes the function call operator, they can also be used as functions, as on line 21. The result of each expression is an expression tree, retaining the concrete type of each element in the expression. This is the same kind of data structure that we encounter in any expression template library and it is generated for us by Proto here. Figure 4.2 shows the expression tree for line 21.

In Coolfluid 3, the language elements are also defined using terminals, but we have of course many more keywords to perform the required operations to build a complete finite element model. Many of these elements will be demonstrated in section 4.3.

So far, the expressions do nothing. The goal of the current example is to correctly interpret the expression at line 21 and make it print the centroid of all the elements in a mesh. This is the subject of the next section, where we will show how to parse the expression and take appropriate action.

```
1  #include <iostream>
2  #include <boost/proto/proto.hpp>
3
4  using namespace boost;
5
6  // Different types to distinguish terminals at compile time
7  struct element_coords_tag {};
8  struct centroid_tag {};
9  struct shape_func_tag {};
10
11 // Some statically created terminals
12 proto::terminal< centroid_tag >::type const centroid = {};
13 proto::terminal< element_coords_tag >::type const element_coords = {};
14 proto::terminal< shape_func_tag >::type const N = {};
15 proto::terminal< std::ostream & >::type cout_ = { std::cout };
16
17 int main(void)
18 {
19   // The terminals can be combined into any valid C++ expression
20   (centroid + element_coords * cout_) / N;
21   cout_ << N(centroid)*element_coords << "\n";
22 }
```

Listing 4.1: Generating expressions from Boost.Proto terminals.

4.2.2 The algorithm implementation layer

The algorithm implementation layer in figure 4.1 takes care of the interpretation of expressions and the execution of the associated actions. In this section, we will take a top-down approach to evaluating our example expression. To this end, we will first define a class —grammar in Proto terminology— capable of parsing the expression. The class defined in listing 4.2 describes the language that we use in our example expression. The grammar is defined entirely in a templated type from which `fem_grammar`

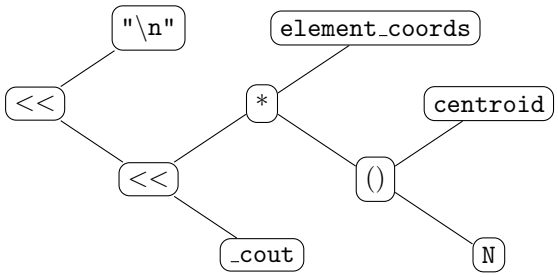


Figure 4.2: The expression tree for line 21 in listing 4.1

derives. This notation is actually the embedded domain specific language provided by Proto itself for defining a grammar. The `or_` construct allows us to list a series of rules (each a Proto grammar in itself) that are checked in sequence. The use of the `when` statement allows us to first describe what kind of syntax we expect, followed by a second argument —the semantic action— that describes what to do when a given expression is encountered. In the case of terminals, we can use the type (classes ending in `_tag` here) to distinguish them at compile time. The grammar can be used to check the validity of an expression at compile time, using the `proto::matches` metafunction. In this case, only the first argument of each `when` clause is considered and no action is taken. We can also use `fem_grammar` as a C++ functor on an expression, in which case our grammar acts as a Proto "transform", evaluating the expression by executing the semantic actions embedded into the grammar. The classes that perform these actions are not defined yet, so that will be our next step.

To evaluate the code in our example, we need to retrieve the element coordinates, get the shape function parameters associated with the centroid and evaluate the shape functions. These are executed by the semantic actions in our grammar and they are implemented as C++ functors that are named "primitive transforms" in Proto. Listing 4.3 shows the primitive transform that evaluates shape functions. The shape function values are computed by the function call operator overload at line 16, taking two arguments: the mapped coordinates `coord` and a `data` argument that provides access to shape function data. Their types are specified through template parameters that embed the size of the involved vectors: this is necessary because we use the Eigen [102] library for matrix computations, which allows setting matrix sizes at compile time. Operations on small matrices are more efficient if their size is fixed like this. The concrete type `CoordT` is a vector with a size equal to the number of spatial dimensions, while the type `DataT` depends on the element type in use, since the size of the element shape function vector is equal to the number of element nodes as described in section 4.1. On line 20, we use the element shape function to compute the value and store it back in the data. This technique allows us to return a reference on line 22,

```
1  // Grammar to evaluate the expression:
2  // cout_ << N(centroid)*element_coords << "\n"
3  struct fem_grammar :
4    // Match the following rules in order:
5    proto::or_
6    <
7      // Evaluate element_coords using the eval_element_coord transform:
8      proto::when<proto::terminal<element_coords_tag>, eval_element_coord(proto::_data)>,
9      // Evaluate centroid using the eval_centroid transform
10     proto::when<proto::terminal<centroid_tag>, eval_centroid(proto::_data)>,
11     // Evaluate shape functions using the eval_shape_func transform
12     proto::when
13     <
14       proto::function<proto::terminal<shape_func_tag>, proto::_>,
15       eval_shape_func(fem_grammar(proto::_child1), proto::_data)
16     >,
17     // On any other expression: perform the default C++ action
18     proto::_default<fem_grammar>
19   >
20 {
21 };
```

Listing 4.2: Grammar capable of parsing a simple expression for interpolating coordinates using a shape function.

avoiding a potentially expensive copy of the data. The implementation of `shape_function()` is provided by a simple element type, that in the conceptual overview of figure 4.1 is part of the external libraries layer (see section 4.2.3 listing 4.6). The rest of the code (lines 3 to 13) is only used at compile time and computes the return type of the functor, needed by Proto so it can pass along the result. Starting with C++11 the result type of a functor can be obtained from the compiler and this code is no longer needed.

We still need to complete the definition of the data used by the functor. It holds the knowledge about the concrete element type and holds the finite element mesh that we use, as can be seen from listing 4.4. We are completely free to define this type as we see fit, since Proto passes it along as a template parameter. In our example, we provide convenience typedefs to access the shape function types and storage for results. The data changes for each element, so we have a `set_element` function that is called by the element looping algorithm (still to be discussed). The entire class is templated on the element type.

So far, we have shown how to implement leaf nodes like the shape function `N` in figure 4.2. What is still missing is the implementation of the binary operators and the output. The top-level grammar in listing 4.2 takes care of this on line 18: we ask Proto to perform the default C++ action when none of the earlier rules are matched, recursively calling `fem_grammar` to interpret any subtrees. This way, the terminal `_cout` is expanded to its value —the standard output stream— and the shift operator has its usual meaning for stream output. When the product is encountered, the left and right side get evaluated using `fem_grammar`, and the results are multiplied using the

```
1  struct eval_shape_func : proto::callable
2  {
3    // C++ result_of declaration
4    template<typename Signature>
5    struct result;
6
7    // C++ result_of implementation
8    template<class ThisT, typename CoordT, typename DataT>
9    struct result<ThisT(CoordT, DataT)>
10   {
11     typedef const typename
12       boost::remove_reference<DataT>::type::element_t::shape_func_t& type;
13   };
14
15   template<typename CoordT, typename DataT>
16   const typename DataT::element_t::shape_func_t& operator()(const CoordT& coord,
17                                                             DataT& data) const
18   {
19     // Evaluate the shape function at the given mapped coordinates
20     data.shape_func = DataT::element_t::shape_function(coord);
21     // Return a reference to the result, stored in data
22     return data.shape_func;
23   }
24 };
```

Listing 4.3: Primitive transforms to evaluate the shape functions

default C++ multiplication, which is in this case a matrix product from the Eigen library.

Using predefined transforms from the Proto framework and user-defined function objects allows for the creation of arbitrarily complex grammars, which can call themselves as transforms recursively. This concise embedded language describing grammars is a key feature of Proto: it makes it possible to indicate how to evaluate an expression without resorting to complicated metaprogramming techniques, resulting in code that is easier to read and maintain.

To obtain a working code, we still need to implement the loop over the elements and generate the data that is used in expression evaluation. The data is templated on the concrete element type, but as can be seen from listing 4.6 the mesh data at line 20 lacks this information at compile-time. This makes sense, since in a real application the user loads the mesh and so the software cannot know what concrete type is used at compile time. To select the element at run time, we need to generate code for all the element types that are supported by the solver. In this example, we will support both 1D line elements and 2D triangle elements. The difference in the mesh data lies in the number of columns in the coordinates and connectivity tables, so we can use that to determine which mesh we are using. By checking for each supported element type if the mesh matches, we can execute the correct code at runtime. We use the MPL functor defined in listing 4.5 to match the correct element type to the mesh. The functor is executed for each item in a list of allowed elements (line 37), resulting in cleaner code

```
1  template<typename ElementT>
2  struct dsl_data
3  {
4    // Required by Eigen to store fixed-size data
5    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
6    // The concrete element type
7    typedef ElementT element_t;
8    // The type of the coordinates for all element nodes
9    typedef Eigen::Matrix<double, element_t::nb_nodes, element_t::dimension> coord_mat_t;
10
11   // Construct using the mesh
12   dsl_data(const fem::mesh_data& d) : mesh_data(d)
13   {
14   }
15
16   // Set the current element
17   void set_element(const int e)
18   {
19     for(int i = 0; i != element_t::nb_nodes; ++i)
20       for(int j = 0; j != element_t::dimension; ++j)
21         coord_mat(i,j) = mesh_data.coordinates[mesh_data.connectivity[e][i]][j];
22   }
23
24   // Reference to the mesh
25   const fem::mesh_data& mesh_data;
26   // Storage for the coordinates of the current element nodes
27   coord_mat_t coord_mat;
28   // Value of the last shape function computation
29   typename element_t::shape_func_t shape_func;
30 };
```

Listing 4.4: Data passed to the primitive transforms

than a long list of if-else statements. Once the element type check at line 13 passes, the data can be constructed and we can start the loop, updating the data for each element at line 23. The actual expression is executed using our grammar at line 24, passing the expression and data as the first and third argument, respectively. The `0` is the Proto state, which is not used here. The expression itself remains a template parameter, allowing us to write any kind of Proto expression in a call to `for_each_element` and keeping all compile-time information. Both of these properties are key advantages over a similar system that could be set up using virtual functions. By generating code for a list of predefined element types, the technique can be seen as a special case of generative programming [103], where the code to generate is defined through the MPL vector of element types.

In Coolfluid 3, the process is more complicated, since we can have different shape functions for the geometry and each variable that appears in an expression. This means we must now generate code for every possible combination of shape functions. We chose to organize the data so that each unknown that appears in an expression has its own data structure, aggregated into a global data structure for the expression. This approach makes it possible to have a unified data structure supporting expressions with an arbitrary number of variables. Another complication is the possibility to

mix different element types in one mesh, e.g. triangles and quadrilaterals. We solve this by organizing the mesh into sets that contain only one element type, and then deal with each region in turn. The complete algorithm to create the data and loop over a set of elements of the same type is as follows:

Require: Mesh with field data for n variables
Require: A compile-time list of m shape functions N_i that may be used
Require: An expression E
Ensure: Construction of context data of the correct type
  for all shape functions N_i **do**
    if N_i matches the geometry shape function **then**
      set geometry shape function: $N_g = N_i$
      for all variables V **do**
        for all N_j compatible with N_g **do**
          if N_j matches the variable shape function **then**
            set $N_V = N_j$
          end if
        end for
      end for
      create context data d using known N_g and N_V $(\forall V)$
      for all elements e **do**
        execute grammar(E, d)
      end for
    end if
  end for

At this point we might wonder what happens if we try to evaluate an expression that makes no sense, such as line 20 of listing 4.1. Unfortunately, this often results in a very long list of compiler messages, exposing Proto implementation details and long template types. In the current example, it would be easy to fix using compile-time error messages, but for a more complex EDSL this is difficult and we have not yet implemented a satisfactory error handling system. We do intend to handle some common cases, and already errors from Eigen indicating incompatible matrix expressions come through, but they are often obscured by a great number of other errors.

4.2.3 External libraries

The example code uses the simplest possible data structures: the mesh simply contains arrays for the coordinates and the element nodes. The shape functions are simplified to only provide the functionality needed for coordinate interpolation. Listing 4.6 shows the code for a first order 1D line element of the Lagrange family, together with the data for the mesh.

In Coolfluid 3, we provide our own mesh data structure, shape function library and numerical integration framework. For operations at the element level, the Eigen [102] library is used, since it provides highly optimized routines for small, dense matrices. Finally, sparse matrix operations —

```
/// MPL functor to loop over elements
template<typename ExprT>
struct element_looper
{
  element_looper(const fem::mesh_data& m, const ExprT& e) : mesh(m), expr(e)
  {
  }

  template < typename ElemT >
  void operator()(ElemT) const
  {
    // Bail out if the shape function doesn't match
    if(ElemT::dimension != mesh.coordinates.shape()[1] || ElemT::nb_nodes != mesh.
         connectivity.shape()[1])
      return;

    // Construct helper data
    dsl_data<ElemT> data(mesh);

    // Evaluate for all elements
    const int nb_elems = mesh.connectivity.size();
    for(int i = 0; i != nb_elems; ++i)
    {
      data.set_element(i);
      fem_grammar()(expr, 0, data);
    }
  }

  const fem::mesh_data& mesh;
  const ExprT& expr;
};

/// Execute the given expression for every element in the mesh
template<typename ExprT>
void for_each_element(const fem::mesh_data& mesh, const ExprT& expr)
{
  // Allowed element types
  typedef mpl::vector2<fem::line1d, fem::triag2d> element_types;
  mpl::for_each<element_types>(element_looper<ExprT>(mesh, expr));
}
```

Listing 4.5: The element looping algorithm.

required for the solution of the linear system arising from the element-wise assembly— are performed using the Trilinos [104] library.

4.2.4 User defined terminals

If we want to extend the language with new functionality, this currently requires modification to the grammar (listing 4.2). Since this is part of the programming framework, it's not something that's supposed to be modified by the user of the language. To provide more flexibility, we allow users to define new terminals that can be used in expressions without modifying the grammar. We will show here how this works, by providing an overview of how this could be added to our example code. Let's return to the definition of terminals, tagged using empty classes as in listing 4.7. We created a terminal with the template class `user_op` as type. Its use as a function (line

```
// 1D line element
struct line1d
{
  static const int nb_nodes = 2; // Number of nodes
  static const int dimension = 1;

  // Type of the mapped coordinates
  typedef Eigen::Matrix<double, 1, dimension> coord_t;
  // Type of the shape function vector
  typedef Eigen::Matrix<double, 1, nb_nodes> shape_func_t;

  // Compute the shape function vector at mapped coordinate c
  static shape_func_t shape_function(const coord_t& c);

  // Return the mapped coordinate of the centroid
  static const coord_t& centroid();
};

// Unstructured mesh data
struct mesh_data
{
  // The coordinates for each node
  boost::multi_array<double, 2> coordinates;
  // The nodes (index into coordinates above) for each element
  boost::multi_array<int, 2> connectivity;
};
```

Listing 4.6: Element type for first order line elements of the Lagrange family and mesh data structure

```
template<typename T> struct user_op {};
struct my_callable {};
// A terminal typed using the above structs
proto::terminal< user_op<my_callable> >::type const my_op = {};
// Can be used as a function
my_op(1,2,3);
```

Listing 4.7: Definition of a terminal allowing user extension

5) can be matched using the grammar:

```
proto::function< proto::terminal< user_op<proto::_> >, proto::vararg<proto::_> >
```

Here, `proto::_` is a wildcard that matches any type and `vararg` allows the function to have any number of arguments. This kind of construct is exactly what we need to make `my_op` a user defined terminal, though we still have to provide a way to evaluate the expression. Adding a when clause yields the code that we need to add to `fem_grammar` (listing 4.2):

```
proto::when
<
  proto::function< proto::terminal< user_op<proto::_> >, proto::vararg<proto::_> >,
  evaluate_user_op(proto::function< proto::_, proto::vararg<fem_grammar> >)
>
```

This uses `fem_grammar` to evaluate all function arguments and then calls the `evaluate_user_op` primitive transform to evaluate the call (not listed

here for the sake of brevity). In `evaluate_user_op`, we use the template argument to `user_op` (`my_callable` here) as a functor to evaluate the operator. This allows the user to define how the terminal should be evaluated without ever needing to touch the core grammar, by simply implementing the correct function call operator in the type that is passed to `user_op`.

In Coolfluid 3, this technique is used to implement specialized code for certain solvers, directly setting entries of the element matrix when analytical expressions are known. Some core functions, such as the gradient and divergence operations, are also defined this way. Listing 4.8 shows the implementation for the divergence operation. On line 9, we have the function call implementation, where `var` is expanded into the data associated with the unknown for which we want to compute the divergence. This data is then used to access the gradient matrix (line 12), much like we computed the shape function value in our previous example (i.e. line 20 in listing 4.3). The terminal is statically created on line 30, using the `MakeSFOp` metafunction to avoid a long Proto type name. We can easily overload the function signature by additional function call operators. This is used on line 24 to provide an alternative divergence call that does not take a mapped coordinate as an argument, but evaluates at the current quadrature point instead.

```
1  // Operator definition
2  struct DivOp
3  {
4    // The result is a scalar
5    typedef Real result_type;
6
7    // Return the divergence of unknown var, computed at mapped_coords
8    template<typename VarT>
9    Real operator()(const VarT& var, const typename VarT::MappedCoordsT& mapped_coords)
10   {
11     // Get the gradient matrix
12     const typename VarT::GradientT& nabla = var.nabla(mapped_coords);
13     Real result = 0.;
14     // Apply each component and return the result
15     for(int i = 0; i != VarT::EtypeT::dimensionality; ++i)
16     {
17       result += nabla.row(i) * var.value().col(i);
18     }
19     return result;
20   }
21
22   // Divergence at the current quadrature point
23   template<typename VarT>
24   Real operator()(const VarT& var);
25 };
26
27 // The terminal to use in the language
28 // divergence(v, xi): compute the divergence at any mapped coordinate xi
29 // divergence(v): compute the divergence at current quadrature point
30 static MakeSFOp<DivOp>::type const divergence = {};
```

Listing 4.8: Definition of the Coolfluid 3 gradient operation, using a user-defined terminal

4.2.5 Integration into a framework

So far, we have focused on building a small language, with the element looping function as entry point. In Coolfluid 3, our EDSL is used to implement reusable solver components, so before showing the concrete examples in section 4.3 we have to present a few concepts about our framework and how it interfaces with our EDSL.

Each class in Coolfluid 3 is derived from the `Component` base class, which offers infrastructure to set options at run time and holds an arbitrary number of child components, thus creating a tree. A particular subclass `Action` implements the Command pattern [105], and we derive a class from that for working with expressions, called `ProtoAction`. Expressions can then be added to an object tree by creating a `ProtoAction` and setting its expression using the `set_expression` function. The framework executes it transparently just like any other `Action`. Expressions in Coolfluid 3 can loop over either elements or nodes, using different grammars. The node expressions are primarily used for setting boundary and initial conditions, and to update the solution.

We also need a way of accessing the mesh and the unknowns, which we do by making use of `FieldVariable` terminals. A basic action for looping over nodes could be added to a parent component as in listing 4.9, in this case setting the temperature to 288 K. Here, the temperature is defined on line

```
1 // Terminal that refers to the temperature
2 FieldVariable<0, ScalarField> T("T", "temperature_field");
3 // Create a new action, as a child component of parent
4 Handle<ProtoAction> action = parent.create_component<ProtoAction>("Action");
5 // Set an expression to loop over nodes
6 action->set_expression(nodes_expression(T = 288.));
7 // Execute the action
8 action->execute();
```

Listing 4.9: Loop over nodes, setting a temperature field

2, indicating that it is stored as variable `T` in the field `temperature_field`. This seems redundant here, but fields can store more than one variable. Furthermore, we need to be able to distinguish each variable at compile time, which is why we number each variable using the first template argument. Each distinct variable that is used in the same expression must have a different number. The second template argument specifies whether we are dealing with a scalar or a vector. We then create the expression component and set its expression on line 6, choosing an expression that loops over nodes and simply assigns the value 288 to the temperature field here. Boundary and initial conditions are provided in a similar fashion, but there the component is parametrized on the field and variable names, so it can be reused in any solver.

4.2.6 Compatibility with matrix expression templates

At the element level, matrix calculations are delegated to the Eigen library, which uses its own expression templates. As it turns out, such libraries cause problems when embedded into a Proto expression tree. A library like Eigen will, in the case of complex operations, create temporary matrices on the fly. When a temporary object is passed on through the transforms that evaluate our expressions we return references to these temporary objects, resulting in memory errors. The problem is common to all modern matrix expression template libraries, and Iglberger et al. [106] review some cases where these temporaries might appear. To handle this issue, we preprocess the expressions using a special grammar: whenever a matrix product is found, the multiplication expression is modified to store a temporary matrix —allocated only once— that can hold the result of the multiplication. Any matrix multiplication result is then stored in the Proto expression itself and a reference to it is returned during matrix product evaluation. The preprocessing happens by calling an additional transform in the element looping function, so the whole process is transparent to the user of the expressions and allows writing matrix arithmetic of arbitrary complexity.

4.3 Application examples

In this section we list a few example problems and evaluate the overhead from using a high level abstraction.

4.3.1 Poisson problem

The first test case we present is the Poisson problem for a scalar function u and source term f, defined over the domain Ω and with boundary Γ:

$$\nabla^2 u + f = 0 \qquad \text{over } \Omega \tag{4.7}$$

$$u = u_0 \qquad \text{over } \Gamma \tag{4.8}$$

The problem is completely equivalent to the heat conduction example in equation 4.6, so the equation for each element becomes:

$$\int_{\Omega_e} \nabla \boldsymbol{N}_{\boldsymbol{u}}^{\mathbf{T}} \nabla N_u \boldsymbol{u}_e \, \mathrm{d}\Omega_e = \int_{\Omega_e} \boldsymbol{N}_{\boldsymbol{u}}^{\mathbf{T}} \tilde{f} \, \mathrm{d}\Omega_e \tag{4.9}$$

Listing 4.10 shows how to implement the assembly procedure for the system using our EDSL. The variables for the problem are defined as scalars on lines 2 and 4. The expression itself spans lines 12 to 22, and is valid in this case for all first order elements of the Lagrange family that are implemented in Coolfluid 3. The first statement is `group`, and it is used to join together the following expressions into a single expression, i.e. the grammar is executed for each argument to the group call. Placing all statements inside a single

```
1  // The unknown function. The first template argument is a constant to distinguish each
       variable at compile time
2  FieldVariable<0, ScalarField> u("u", solution_tag());
3  // The source term, to be set at runtime using an initial condition
4  FieldVariable<1, ScalarField> f("f", "source_term");
5
6  // Action handling the assembly
7  Handle<ProtoAction> assembly = create_component<ProtoAction>("Assembly");
8  // Set the expression
9  assembly->set_expression(elements_expression
10 (
11   mesh::LagrangeP1::CellTypes(), // All first order Lagrange elements
12   group
13   (
14     _A = _0, _a = _0,// The element matrix and RHS vector, initialized to 0
15     element_quadrature // Integration over the element
16     (
17       _A(u) += transpose(nabla(u)) * nabla(u),
18       _a[u] += transpose(N(u))*f
19     ),
20     system_matrix +=  _A, // Assemble into the global linear system
21     system_rhs += _a
22   )
23 ));
```

Listing 4.10: Implementation of the assembly procedure for the Poisson problem.

expression ensures we only need to loop over the elements once to perform all actions.

To assemble the system matrix, we use an element matrix A, which is first set to zero on line 14. The element matrix terminal can be used as a regular matrix in computations, but there is also a transform to evaluate the function call operator. If an expression of the form `_A(u)` is encountered, it returns only the rows of the element matrix that refer to the equation for u (i.e. all rows in this case). If we pass a second variable as argument, the returned matrix only contains the columns that refer to it, so `_A(u, u)` would return a block that only contains the u contributions from the u equation (again, all rows and columns in this case). This notation is convenient in the presence of multiple variables to select only the relevant entries in an element matrix. The size of the element matrix is also determined by looking for blocks like this, so here we only use `_A(u)` to indicate that the element matrix contains only a single equation for u. The same applies to the element right hand side vector `_a` using square brackets.

We call `element_quadrature` to perform a numerical integration using Gauss quadrature on all of its arguments. Each expression must be of the form `LHS += RHS`, where the `LHS` is a part of an element matrix or vector and the `RHS` is the expression to integrate. There is a primitive transform that picks these expressions apart, evaluates the right hand side at each Gauss point, multiplies with the weights and Jacobian determinant and then sums up the contributions into the `LHS`. The required quadrature order is deter-

mined based on the orders of the shape functions of the passed arguments. Line 17 represents the product $\nabla \boldsymbol{N}_{\boldsymbol{u}}^{\mathbf{T}} \nabla N_u$, where we simplify ∇N_u into `nabla(u)` to get a shorter notation in our language. Likewise, the computation of the right hand side integral on line 18 uses `N(u)` to get the shape function vector N_u and `f` to get the interpolated value $\tilde{f}$. Since all of these terminals appear inside the quadrature, they are evaluated at the quadrature points and we can omit the mapped coordinates argument. This results in expressions that closely mimic the mathematical notation of equation (4.9). Finally, the element contributions are assembled into the global system at lines 20 and 21. The `system_matrix` and `system_rhs` terminals keep track of wrapper objects for a Trilinos matrix and vector, using a generic linear solver API that is part of Coolfluid. They are initialized in a base class that provides the functionality for working with expressions and linear systems.

The code in listing 4.10 builds a component that assembles the linear system. At run time, it can be combined with other components to add initial and boundary conditions and to solve the linear system, all of which can be controlled from a Python script.

4.3.2 Navier-Stokes equations using Chorin's method

The Navier-Stokes equations for incompressible flow with velocity vector $\boldsymbol{u}$, pressure p and kinematic viscosity ν are:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{4.10}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} + \frac{\nabla p}{\rho} - \nu \nabla^2 \boldsymbol{u} = \mathbf{0} \tag{4.11}$$

We will first solve this system of equations using Chorin's method [107], since this will allow us to easily compare performance later on with the same example from the FEniCS project. Chorin proposed an iterative time stepping scheme, computing first an auxiliary velocity $\boldsymbol{u}^{\text{aux}}$, followed by the pressure and finally the corrected velocity at the new time step. The weak form (written again for a single element) for the $\boldsymbol{u}^{\text{aux}}$ equation is:

$$\left(\frac{1}{\Delta t} \underbrace{\int_{\Omega_e} \boldsymbol{N}_{\boldsymbol{u}}^{\mathbf{T}} N_u \, \mathrm{d}\Omega_e}_{T_{u_i u_i}} + \nu \underbrace{\int_{\Omega_e} \nabla \boldsymbol{N}_{\boldsymbol{u}}^{\mathbf{T}} \nabla N_u \, \mathrm{d}\Omega_e}_{A_{u_i u_i}} \right) (\boldsymbol{u}_e^{\text{aux}})_i = \frac{1}{\Delta t} \int_{\Omega_e} \boldsymbol{N}_{\boldsymbol{u}}^{\mathbf{T}} \tilde{u}_i^n \, \mathrm{d}\Omega_e - \int_{\Omega_e} \boldsymbol{N}_{\boldsymbol{u}}^{\mathbf{T}} \tilde{\boldsymbol{u}}^n \boldsymbol{\nabla} \boldsymbol{N}_{\boldsymbol{u}} \, (\boldsymbol{u}_e^n)_i \, \mathrm{d}\Omega_e \equiv \boldsymbol{a}_{u_i} \tag{4.12}$$

The superscript n indicates the known solution at time step n. The index i represents the i-th component of the velocity here, which is a scalar in the case of the interpolated value $\tilde{u}_i^n$ and a vector of values for each node of the element for the unknowns $(\boldsymbol{u}_e^{\text{aux}})_i$. This means that for each value of i, we insert a square block with dimension equal to the number of nodes into the

element matrix, and a corresponding segment into the element right hand side vector. We split up the element matrix as indicated by the braces, where $T_{u_i u_i}$ indicates the block corresponding to the rows and columns of component i of the velocity. The code to build this linear system is presented in listing 4.11. To get a stable solution, we will interpolate the velocity

```
1  // Allow a mix of first and second order shape functions
2  typedef boost::mpl::vector2<mesh::LagrangeP1::Triag2D, mesh::LagrangeP2::Triag2D>
       LagrangeP1P2;
3  // Kinematic viscosity, as a user-configurable constant:
4  PhysicsConstant nu("kinematic_viscosity");
5  // The velocity
6  FieldVariable<0, VectorField> u("u", "navier_stokes_u_velocity", "cf3.mesh.LagrangeP2");
7  // LSSActionUnsteady links with the linear algebra backend and the time tracking
8  Handle<LSSActionUnsteady> auxiliary_lss =
9    create_component<LSSActionUnsteady>("AuxiliaryLSS");
10 // The matrix assembly
11 Handle<ProtoAction> auxiliary_mat_assembly =
12   auxiliary_lss->create_component<ProtoAction>("MatrixAssembly");
13 auxiliary_mat_assembly->set_expression(elements_expression(LagrangeP1P2(),
14 group
15 (
16   _A(u) = _0, _T(u) = _0,
17   element_quadrature
18   (
19     _A(u[_i], u[_i]) += transpose(nabla(u))*nabla(u),
20     _T(u[_i], u[_i]) += transpose(N(u))*N(u)
21   ),
22   auxiliary_lss->system_matrix += auxiliary_lss->invdt()*_T + nu*_A
23 )));
24
25 // RHS assembly
26 Handle<ProtoAction> auxiliary_rhs_assembly =
27   auxiliary_lss->create_component<ProtoAction>("RHSAssembly");
28 auxiliary_rhs_assembly->set_expression(elements_expression(LagrangeP1P2(),
29 group
30 (
31   _a[u] = _0,
32   element_quadrature
33   (
34     _a[u[_i]] += auxiliary_lss->invdt() * transpose(N(u))*u[_i] -
35                  transpose(N(u))*(u*nabla(u))*_col(nodal_values(u), _i)
36   ),
37   auxiliary_lss->system_rhs += _a
38 )));
```

Listing 4.11: Code to build the linear system for $\boldsymbol{u}^{\text{aux}}$ in Chorin's method

and pressure using second and first order shape functions, respectively (the Taylor-Hood element). If we restrict the solver to triangles, we can obtain this using the typedef on line 2. The actual shape function is chosen at run time, but we enforce the use of second order for the velocity here through the third constructor argument for the velocity variable on line 6. We use separate actions for the system matrix and right hand side assembly, since in this case the matrix coefficients don't change with time, so we can run the assembly procedure only once. The definition of the element matrices A and T is provided on lines 19 and 20. The Laplacian is written exactly

the same as in the Poisson problem (section 4.3.1), but a new aspect is the introduction of the index `_i` when indexing into the element matrix. It is automatically expanded into a loop over the number of physical dimensions, addressing the diagonal blocks of the element matrices as defined in equation (4.12). The same applies for the mass matrix on line 20, and both matrices are combined and assembled into the global system on line 22. The `auxiliary_lss` component provides access to the terminals related to the linear system and the time. Here, `invdt` returns a reference to the inverse of the time step, which is controlled by the user running the simulation.

In the right hand side expression on line 34, the index `_i` is used again, using `u[_i]` to get each component of the interpolated velocity vector $\tilde{\boldsymbol{u}}$. The last term represents the advection, and it requires access to a single component of the nodal values vector $\boldsymbol{u}_e$. We store the nodal values for a vector variable as a matrix, with each column corresponding to one physical component of the vector. This matrix is obtained using the `nodal_values` function while individual columns can be addressed using `_col`. The notation is a bit verbose, but this could be fixed by introducing a user defined terminal for the advection operation.

The next step in Chorin's algorithm calculates the pressure, through the following Poisson problem:

$$\int_{\Omega_e} \nabla \boldsymbol{N}_{\boldsymbol{u}}^{\mathbf{T}} \nabla N_u \,\mathrm{d}\Omega_e \boldsymbol{p}_e^{n+1} = -\frac{1}{\Delta t} \int_{\Omega_e} N_p^{\mathrm{T}} \left(\boldsymbol{\nabla} \boldsymbol{N}_p\right)_i \,\mathrm{d}\Omega_e \left(\boldsymbol{u}_e^{\mathrm{aux}}\right)_i \tag{4.13}$$

Listing 4.12 shows the code for this system, again using two assembly actions. For the right hand side assembly on line 22, we used the `divergence` function that was defined in listing 4.8. We could also write this in terms of the nodal values matrix like this:

```
element_quadrature(_a[p] += transpose(N(p))*nabla(u)[_i]*_col(nodal_values(u), _i))
```

On line 23, we call the `lit` function on `invdt`. This is actually a Proto function that constructs a terminal in-place, and it is needed to delay the evaluation of the minus sign, which would otherwise be evaluated right away by C++, resulting in the storage of a copy of the negative inverse timestep at expression creation, rather than a reference to the current value.

The final step of the algorithm updates the velocity, using the gradient of the newly calculated pressure:

$$\int_{\Omega_e} N_u^{\mathrm{T}} N_u \,\mathrm{d}\Omega_e \left(\boldsymbol{u}_e^{n+1}\right)_i = \int_{\Omega_e} N_u^{\mathrm{T}} \tilde{u}_i^{\mathrm{aux}} \,\mathrm{d}\Omega_e - \Delta t \int_{\Omega_e} N_u^{\mathrm{T}} \left(\boldsymbol{\nabla} \boldsymbol{N}_p\right)_i \boldsymbol{p}_e^{n+1} \,\mathrm{d}\Omega_e \tag{4.14}$$

The system matrix is the mass matrix, so as seen in listing 4.13 we assemble it in its own action. The gradient function on line 18 is defined using the user defined function mechanism, and just as is the case with the divergence it can be written using nodal values as well:

```
transpose(N(u))*(u[_i] - lit(correction_lss->dt()) * (nabla(p)[_i]*nodal_values(p))[0])
```

```
// The pressure field, using the default first order shape function
FieldVariable<1, ScalarField> p("Pressure", pressure_lss->solution_tag());
// The linear system manager
Handle<LSSActionUnsteady> pressure_lss = create_component<LSSActionUnsteady>("
     PressureLSS");
// The assembly action
Handle<ProtoAction> pressure_mat_assembly =
pressure_lss->create_component<ProtoAction>("MatrixAssembly");
pressure_mat_assembly->set_expression(elements_expression(LagrangeP1(),
group
(
  _A(p) = _0,
  element_quadrature(_A(p) += transpose(nabla(p))*nabla(p)),
  pressure_lss->system_matrix += _A
)));

Handle<ProtoAction> pressure_rhs_assembly =
  pressure_lss->create_component<ProtoAction>("RHSAssembly");
pressure_rhs_assembly->set_expression(elements_expression(LagrangeP1P2(),
group
(
  _a[p] = _0,
  element_quadrature(_a[p] += transpose(N(p))*divergence(u)),
  pressure_lss->system_rhs += -lit(pressure_lss->invdt())*_a
)));
```

Listing 4.12: Pressure Poisson problem for Chorin's method

```
Handle<LSSActionUnsteady> correction_lss = create_component<LSSActionUnsteady>("
     CorrectionLSS");

Handle<ProtoAction> correction_matrix_assembly = correction_lss->create_component<
     ProtoAction>("MatrixAssembly");
correction_matrix_assembly->set_expression(elements_expression(LagrangeP1P2(),
group
(
  _A(u) = _0,
  element_quadrature(_A(u[_i], u[_i]) += transpose(N(u))*N(u)),
  correction_lss->system_matrix += _A
)));

Handle<ProtoAction> correction_rhs_assembly = correction_lss->create_component<
     ProtoAction>("RHSAssembly");
correction_rhs_assembly->set_expression(elements_expression(LagrangeP1P2(),
group
(
  _a[u] = _0,
  element_quadrature(_a[u[_i]] +=
    transpose(N(u))*(u[_i] - lit(correction_lss->dt()) * gradient(p)[_i])),
  correction_lss->system_rhs += _a
)));
```

Listing 4.13: The code for the correction step in Chorin's method

The implementation of Chorin's method shows how different systems can be combined to solve a problem with multiple unknowns, each interpolated using a different shape function. The coding of the assembly procedure closely follows the mathematical formulation.

4.3.3 PSPG/SUPG stabilized incompressible Navier-Stokes

The easy implementation of the method presented in section 3.1.1 was one of the main resaons for developing our language. Equations (3.6) to (3.13) are assembled into a single linear system using the elements expression defined in listing 4.14 (showing only the part relevant to the assembly itself). Since

```
1  assembly->set_expression(elements_expression
2  (
3    AllElementsT(),
4    group
5    (
6      _A = _0, _T = _0,
7      compute_tau(u, nu_eff, u_ref, lit(tau_ps), lit(tau_su), lit(tau_bulk)),
8      element_quadrature
9      (
10       _A(p    , u[_i]) += transpose(N(p) + tau_ps*u_adv*nabla(p)*0.5) * nabla(u)[_i]
11                           + tau_ps * transpose(nabla(p)[_i]) * u_adv*nabla(u),
12       _A(p    , p)     += tau_ps * transpose(nabla(p)) * nabla(p) / rho,
13       _A(u[_i], u[_j]) += transpose((tau_bulk + 1./3.*nu_eff)*nabla(u)[_i]
14                           + 0.5*u_adv[_i]*(N(u) + tau_su*u_adv*nabla(u))) * nabla(u)[_j
                                 ],
15       _A(u[_i], u[_i]) += nu_eff * transpose(nabla(u)) * nabla(u)
16                           + transpose(N(u) + tau_su*u_adv*nabla(u)) * u_adv*nabla(u),
17       _A(u[_i], p)     += transpose(N(u) + tau_su*u_adv*nabla(u)) * nabla(p)[_i] / rho,
18       _T(p    , u[_i]) += tau_ps * transpose(nabla(p)[_i]) * N(u),
19       _T(u[_i], u[_i]) += transpose(N(u) + tau_su*u_adv*nabla(u)) * N(u)
20     ),
21     system_rhs += -_A * _x,
22     _A(p) = _A(p) / theta,
23     system_matrix += invdt() * _T + theta * _A
24 )));
```

Listing 4.14: The assembly of the PSPG/SUPG stabilized incompressible Navier-Stokes equations.

only one linear system is involved, the code is integrated into the framework in the same way as in listing 4.10. The value of the coefficients τ depend on local element properties, as explained in section 3.1.1. We calculate them using a user-defined terminal `compute_tau`, passing a reference to a double for each coefficient (line 7). On line 13, we use indices `_i` and `_j` to create a nested loop over the dimension of the problem, filling all $A_{u_i u_j}$ blocks. The right hand side from equation (3.3) is built by applying the element matrix A to the current element unknowns $\boldsymbol{x}_e$, represented by `_x`. On line 22, we divide by θ to avoid the use of the θ scheme on the continuity equation, improving stability. Finally, on line 23 we write the system matrix as in the left hand side of equation (3.3).

4.4 Performance analysis

In this section we discuss the results of some performance tests, using the application examples from the previous section. Table 4.1 lists our system

	Poisson and Chorin	PSPG/SUPG (per machine)
CPU(s)	Intel i7-2600	Two Intel Xeon E5520
RAM	16 GB	24 GB
Operating system	Fedora 18	CentOS 6.2
Compiler	GCC 4.7.2	GCC 4.8.0
Trilinos version	11.4.1	11.2.3

Table 4.1: System characteristics for the performance tests.

characteristics, where the SUPG/PSPG test uses a different system because it concerns a large-scale test run on our cluster.

4.4.1 Poisson problem

Our first performance test concerns the Poisson problem. The element equations (4.9) can easily be calculated analytically when using linear shape functions, allowing a comparison between manually coded versions, a code using a virtual function interface to the shape functions and code generated using our language. Additionally, we will compare with a specialized user-defined terminal containing the manually coded version and with DOLFIN [98] from the FEniCS project. Problem (4.9) is completed with boundary conditions and a source term identical to the Poisson demo case from FEniCS:

$$f = -6 \qquad \text{over } \Omega \tag{4.15}$$

$$u = 1 + x^2 + 2y^2 \qquad \text{over } \Gamma \tag{4.16}$$

When using linear shape functions, the solution for the discrete system captures the analytical solution up to machine precision at the nodes. As an illustration, the code for the specialized user-defined terminal is presented in listing 4.15. The terminal `assemble_triags` can then be used to directly assemble the linear system as shown on line 41. The manually coded version uses the same algorithm, but here we also loop over elements directly, avoiding the use of Proto entirely.

We first run a test on the unit square, divided into 1000 parts in both the x and y direction. Each square cell is divided into two triangles and first order shape functions from the Lagrange family are used. Table 4.2 summarizes the results, with labeling as follows: "Proto" is the code using our EDSL (see listing 4.10); "Specialized" is the user-defined terminal from listing 4.15; "Manual" is the manually coded assembly loop; "Virtual" is the code using the virtual function interface to the shape functions as it is available in Coolfluid 3 and finally "DOLFIN" is the code generated by the FEniCS project demo. All timings represent the average of 10 assembly runs. Due to the simplicity of the Poisson problem, the insertion into the global sparse matrix structure can actually be more expensive than the evaluation of the element integrals. We run the benchmark using both the Trilinos backend

```
// Specialized code for triangles
struct PoissonTriagAssembly
{
  typedef void result_type;
  // Functor that takes: source term f, Linear system lss, element matrix and vector acc
  template<typename FT, typename LSST>
  void operator()(const FT& f, LSST& lss, math::LSS::BlockAccumulator& acc) const
  {
    typedef mesh::LagrangeP1::Triag2D ElementT;
    // Get the coordinates of the element nodes
    const ElementT::NodesT& nodes = f.support().nodes();

    // Compute normals
    ElementT::NodesT normals;
    normals(0, XX) = nodes(1, YY) - nodes(2, YY);
    // ... repetitive code omitted
    // Jacobian determinant
    const Real det_jac = normals(2, YY)*normals(1, XX) - normals(1, YY)*normals(2, XX);
    const Real c = 1. / (2.*det_jac);

    // Indices of the nodes of the current element
    acc.neighbour_indices(f.support().element_connectivity());

    for(Uint i = 0; i != 3; ++i)
      for(Uint j = 0; j != 3; ++j)
        acc.mat(i, j) = c * (normals(i, XX)*normals(j, XX) + normals(i, YY)*normals(j,
             YY));

    // Get the values of the source term
    const Real f0 = f.value()[0];
    // ... f1 and f2
    acc.rhs[0] = (2*f0 + f1 + f2);
    // ... acc.rhs[1] and acc.rhs[2]
    acc.rhs *= det_jac/24.;
    lss.matrix().add_values(acc);
    lss.rhs().add_rhs_values(acc);
  }
};
// Create an terminal that can be used as a function in a proto expression
static MakeSFOp<PoissonTriagAssembly>::type const assemble_triags = {};
// Usage example:
assembly->set_expression(elements_expression
(
  boost::mpl::vector1< mesh::LagrangeP1::Triag2D>(),
  assemble_triags(f, system_matrix, m_block_accumulator)
));
```

Listing 4.15: Code for the specialized user-defined terminal, valid for linear shape functions over a triangle.

(using an Epetra CRS matrix) and a "dummy" matrix —i.e. not storing any data— to properly time the assembly procedure. As seen from table 4.2, the overhead of the matrix insertion is about 0.3 s in Coolfluid 3 and 0.8 s in DOLFIN, i.e. at least of the order of the time it takes to compute the element matrix itself. When comparing the timings for the dummy matrix, we see that the generic Proto code —which uses second order Gauss quadrature— is more than 5 times slower than the manually coded version. The difference between the specialized and the manual versions is much smaller. Since the specialized code still uses the Proto element looping mechanism, we can

	Dummy matrix		Epetra matrix	
	Wall clock	Relative	Wall clock	Relative
Proto	0.32 s	1	0.61 s	1
Proto Specialized	0.069 s	0.22	0.35 s	0.57
Manual	0.054 s	0.17	0.34 s	0.56
Virtual	2.82 s	8.81	3.18 s	5.21
DOLFIN	0.31 s	0.97	1.13 s	1.85

Table 4.2: Linear system assembly times (wall clock time and timing relative to Proto) for the Poisson problem on the unit square, using first order triangle shape functions on a 1000x1000 grid.

conclude that its inherent overhead is small. We confirmed this by profiling the assembly with gperftools[1], generating the call graphs shown in figure 4.3. Each graph starts off in the base class `TimedAction`. On the left, the generic Proto code is seen to be mostly inlined into the element looper, with only 10% of the time spent in calls to shape functions. For the specialized and manual versions, these calls are no longer made and everything is inlined, either in the Proto element loop or in the concrete implementation of the manual loop. The overhead in the generic Proto code is due to the extra matrix operations involved in the second order quadrature. For the user-defined terminal, some small overhead remains, due to index conversions in the Proto data structure that are used in the case that the expression is only used on part of the mesh.

The virtual function implementation performs much worse than any other method. Like Proto, it allows writing generic code for all elements, but does so by using virtual function calls and dynamically allocated matrices. This makes the method much slower than the Proto code. A separate test comparing dynamically and statically sized matrices (size 4x4) from the Eigen library shows a slowdown by a factor of about 8 for dynamic matrices when running matrix-matrix and matrix-vector multiplications, reinforcing the results from table 4.2.

In DOLFIN, the right hand side and the matrix are computed using separate assembly loops. The presented timing is the total time spent on assembly, divided by the number of assembles executed. For this particular case, Proto and DOLFIN result in the same performance. This is surprising, since the more advanced integration routines in DOLFIN detect here that first order quadrature is sufficient, while our Proto code assumes the worst case and applies second order integration.

The above observation leads us to perform the test in 3D, using a unit cube with 100 segments in each direction and built up of tetrahedra. We only compare the Proto and DOLFIN versions, since this code can be applied in 3D without modification. The effect of the quadrature order is obvious here,

[1] `http://code.google.com/p/gperftools`

	Dummy matrix		Epetra matrix	
	Wall clock	Relative	Wall clock	Relative
Proto, default	3.29 s	1	5.51 s	1
Proto, hexahedra	4.60 s	1.40	5.65 s	1.03
Proto, 1st order	0.81 s	0.25	2.64 s	0.48
DOLFIN	1.22 s	0.37	5.05 s	0.92

Table 4.3: Linear system assembly times (wall clock time and timings relative to the default Proto implementation) for the Poisson problem on the unit cube, using first order tetrahedron or hexahedron shape functions on a 100x100x100 grid.

with our second order quadrature being almost three times slower than the first order quadrature performed by DOLFIN. To check if this is the only effect, we temporarily forced our code to use first order quadrature, i.e. using one quadrature point instead of four. The speedup is as expected, but we do emphasize that it is only obtained after modification of the integration code: we do not have a method for determining the integration order based on the terms appearing in the equations. Instead, our integration system assumes there is a mass matrix term in the equation, and proceeds to choose the integration order based on the shape function. If the performance penalty is significant, as is mostly the case with simple problems, it is possible to use a user-defined terminal to override the integration method, or even to avoid numerical integration altogether.

We also include some results for hexahedral elements, where the second order quadrature is necessary. The element matrix dimension is also doubled, resulting in longer computation times for the matrix operations. We see that this is compensated here when inserting into the sparse matrix, due to the lower number of elements (each hexahedron represents 6 tetrahedra).

4.4.2 Chorin's method

In Chorin's method, there are a total of 6 different assembly loops to be run: one for each system matrix, and one for each right hand side. Even though the matrices only need to be assembled once for the simulation, we present timings here for comparison purposes. Table 4.4 summarizes the results. We see that except for the auxiliary matrix assembly, DOLFIN is faster every time, with a very large discrepancy for the correction matrix assembly. This is due to an optimization in DOLFIN, which recognizes the coefficients of the element matrix can easily be precomputed, avoiding quadrature. Our code does not allow this optimization to happen automatically, although it is of course possible to add a user-defined terminal. In the auxiliary matrix, the same term appears, but here it is divided by Δt, causing DOLFIN to apply quadrature.

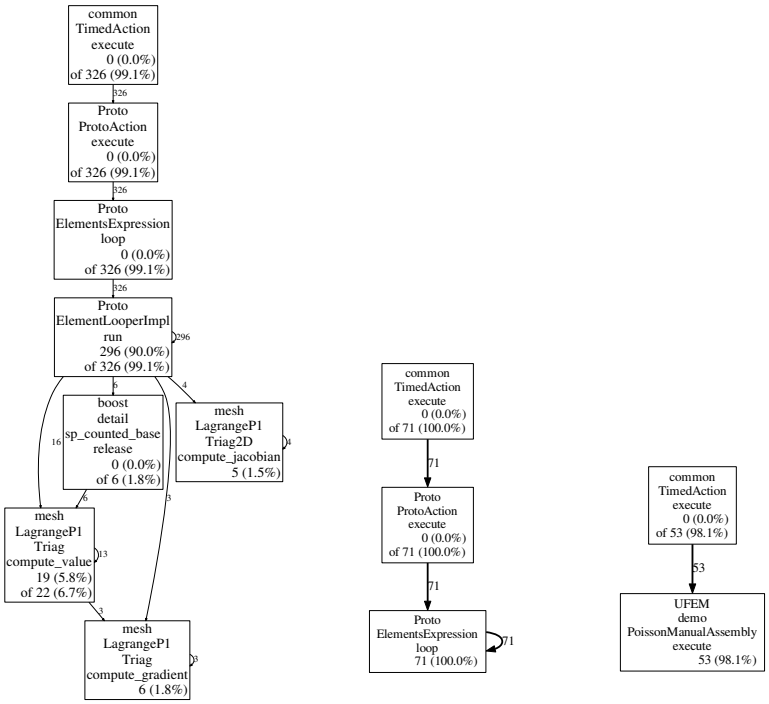


Figure 4.3: Call graphs of optimized code for the Poisson element matrix computation, from left to right: the generic Proto code, the Proto code with user-defined terminal and the manually coded version.

	Dummy matrix			Epetra matrix		
	Proto	DOLFIN		Proto	DOLFIN	
	Wall	Wall	Relative	Wall	Wall	Relative
Aux. matrix	4.33 s	10.15 s	2.34	7.17 s	17.98 s	2.51
p matrix	0.28 s	0.19 s	0.67	0.53 s	0.75 s	1.42
Corr. matrix	2.38 s	0.22 s	0.09	5.23 s	8.09 s	1.55
Aux. RHS	3.12 s	1.17 s	0.375	3.18 s	2.32 s	0.73
p RHS	0.85 s	0.40 s	0.47	0.86 s	0.85 s	0.99
Corr. RHS	1.23 s	0.35 s	0.28	1.26 s	1.59 s	1.26

Table 4.4: Assembly times for each step in Chorin's method, compared between our Proto expressions and DOLFIN. Relative is the DOLFIN timing in multiples of the Proto timing.

The Proto-generated code is currently sub-optimal for the assemblies of the right hand sides. This is due to some missed chances for matrix reuse:

the advection operation in equation (4.12), for example, is calculated once for every component. While this effect is significant when we eliminate the influence of the linear system, it is much less apparent when looking at the results for Epetra matrices and vectors. This leads us to conclude that our performance level is adequate for practical use and the element matrix and vector calculations will not be a dominant factor in the total solution time.

4.4.3 Channel flow simulation

In the last performance test, we take a look at a practical example, using the PSPG/SUPG stabilized Navier-Stokes formulation from listing 4.14. The test problem is the flow between two infinite flat plates, i.e. a 3D channel flow with two periodic directions. We initialize the flow using a laminar solution with centerline Reynolds number of 11250 with respect to the channel half-height. We apply periodic boundary conditions in the stream- and span-wise directions and a no-slip condition at the walls. The average timings for the first 100 timesteps (initial Courant number: 0.57) are presented in table 4.5. We ran the test using hexahedra and tetrahedra, where the

# CPU	Element	Assembly	Solution	$\frac{\text{Solution}}{\text{Assembly}}$
32	Hexa	8.97 s	90.90 s	10.14
	Tetra	7.69 s	73.06 s	9.51
	Tetra Specialized	2.73 s	70.95 s	25.99
64	Hexa	4.89 s	48.33 s	9.88
	Tetra	4.14 s	40.59 s	9.81
	Tetra Specialized	1.45 s	40.15 s	27.67
128	Hexa	3.05 s	32.91 s	10.47
	Tetra	2.58 s	54.53 s	21.13
	Tetra Specialized	0.99 s	46.32 s	46.70

Table 4.5: Assembly and solution times for the coupled PSPG/SUPG stabilized Navier-Stokes equations (listing 4.14) on a 3D channel flow with 128 hexahedra (tetrahedralized in the tetra cases) in each direction.

test on tetrahedra also used a specialized code wrapped into a user-defined terminal ("Tetra Specialized" in the table). The linear system was solved using the Belos Block GMRES method from Trilinos, preconditioned using ML algebraic multigrid. We tweaked the settings to obtain the fastest possible solution time. We see that the solution of the system takes about 10 times as long as its assembly using our EDSL. This shows that even for the relatively complicated assembly expressions of equations (3.6) - (3.13), our language can be used to assemble the system efficiently. Any further optimization should first focus on the linear system solution before the assembly will become a bottleneck.

The user-defined code for tetrahedra results in a further speedup factor of 2.5. In this case, the code was reused from a previous version of the solver, written only for tetrahedra. A domain specific language can also assist in developing hand-tuned code, however: using the language we can first easily specify the generic formulation, and then check the element matrices of manually coded optimizations against the automatically generated matrices.

4.5 Conclusion and future work

We presented a domain specific language for the implementation of finite element solvers, embedded in C++. The language mirrors mathematical notation faithfully, allowing users to focus on the modeling of the physical problem. It is set apart from other work in this area by the use of the Boost.Proto library, which allows using concise grammars to describe and extend the functionality of the language. This mechanism was explained in detail using a stand-alone example. The addition of user defined terminals allows using hand-optimized code when possible, while staying within the automated framework for element looping.

We also analyzed the performance, demonstrating acceptable abstraction overhead when compared to manual implementations and FEniCS. A large scale test with the PSPG/SUPG method for the incompressible Navier-Stokes equations showed that assembly took up to 10% of the linear system solution time.

Possible directions for future development include changes to the numerical integration framework, to better deduce the required quadrature order. On a more technical level, some parts of the code could be simplified by using new features of the C++11 standard, such as variadic templates and automatic return type deduction. Better error handling can also be looked into. One final interesting possibility is the investigation of expression optimization techniques. Using grammars, it is theoretically possible to scan the expressions for recurring matrix products, and calculate each product only once, for example.

Part II

Validation

Chapter 5

Reference cases

5.1 Fluid model

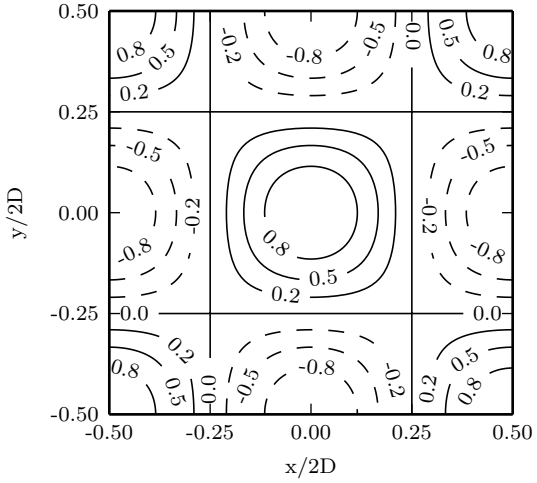
5.1.1 Taylor-Green vortices

We first apply the flow solver to the Taylor-Green periodic vortices, advected by a constant velocity field. The advantage of this test case is that it is a time dependent problem with an analytical solution in closed form. It is therefore ideal to perform some numerical tests regarding the accuracy of the method. The velocity components, the pressure and the vorticity as a function of spatial coordinates and time are:

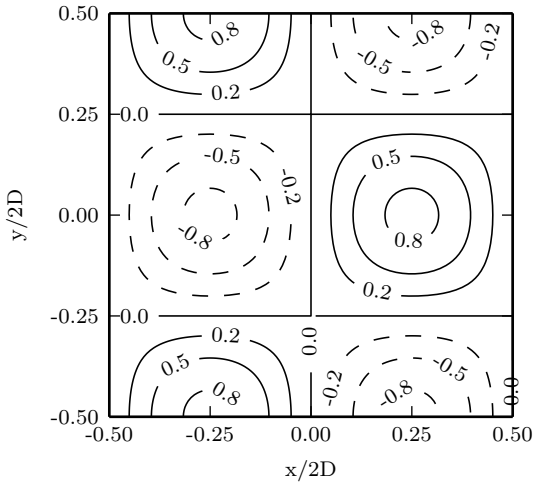
$$
\begin{aligned}
u =& U_a - V_s \cos\left(\frac{\pi}{D}(x - U_a t)\right) \sin\left(\frac{\pi}{D}(y - V_a t)\right) e^{-\frac{2\nu\pi^2}{D^2}t} \\
v =& V_a + V_s \sin\left(\frac{\pi}{D}(x - U_a t)\right) \cos\left(\frac{\pi}{D}(y - V_a t)\right) e^{-\frac{2\nu\pi^2}{D^2}t} \\
p =& -\frac{V_s^2}{4}(\cos(2\frac{\pi}{D}(x - U_a t)) + \\
& \cos(2\frac{\pi}{D}(y - V_a t)))e^{-\frac{4\nu\pi^2}{D^2}t} \\
\omega =& \frac{2V_s\pi}{D}\cos(\frac{\pi}{D}(x - U_a t))\cos(\frac{\pi}{D}(y - V_a t))e^{-\frac{4\nu\pi^2}{D^2}t}
\end{aligned}
$$

This flow field represents two-dimensional, periodic vortices with diameter D and initial maximal swirl velocity V_s, advected by the advection velocity (U_a,V_a), and dissipating due to the kinematic viscosity ν. We use the following values in our tests (based on [49]): $D = 0.5$ m, $V_s = 1$ m/s, $U_a = 0.3$ m/s, $V_a = 0.2$ m/s and $\nu = 0.001$ m^2/s. The flow field is visualized in figure 5.1, using contours of the dimensionless vorticity ω/ω_0 and dimensionless time $tV_s/(2D)$. From these images, it is clear that the vortex centers move along the advection velocity vector. Viscosity redistributes the vorticity until it uniformly reaches zero everywhere in the domain, as indicated by the decrease in vorticity magnitude in the figures.

For the numerical simulations, we initialize the flow with the analytical solution at time t = 0 and set periodic boundary conditions in both directions. Since this determines the pressure only up to a constant, we impose the pressure in the center of the domain, setting it equal to the analytical solution at every time step. In a first test, we will determine the effect of



(a) $t = 0$



(b) $t = 2.5$

Figure 5.1: Contours of dimensionless vorticity, at dimensionless time 0 (a) and 2.5 (b).

the number of inner iterations M, using a grid of 64x64 quadrilaterals that are triangulated for the triangle element tests. In figure 5.2, the error of the segregated solution is compared to the fully coupled solution, defining the relative difference with the coupled solution as, for the x-component of the

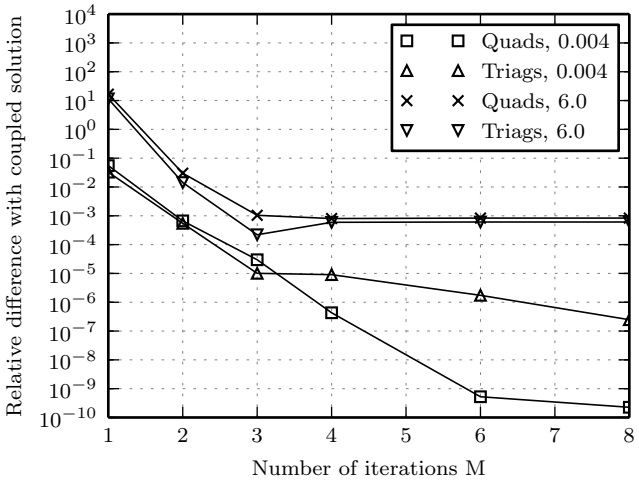


Figure 5.2: Comparison between the fully coupled solution of the linear system and the current method, as a function of the number of inner iterations and at two dimensionless times 0.004 (i.e. after one time step) and 6.0.

velocity:

$$\frac{\max_\Omega |u_c - u_s|}{\max_\Omega |u_c - u_{th}|} \tag{5.1}$$

Here, u_c is the solution of the coupled system of equations, u_s is the segregated solution and u_{th} is the analytical solution. For $M = 1$ and at time 6, the difference between the segregated and the coupled solution is about 10 times greater than the error between the fully coupled solution and the analytical solution, i.e. the absolute error is an order of magnitude greater. When we increase the number of iterations to two, the difference between both methods is two orders of magnitudes smaller than the absolute error, i.e. the difference is negligible and increasing the number of iterations further is not necessary. The difference flattens off after 4 iterations. At the first time step (time 0.004), the iterative technique converges towards the coupled solution as the number of iterations increases. Again, the difference with the coupled solution decreases with two orders of magnitude when using two iterations instead of one. We conclude from these observations that two inner iterations offer a good balance between computational cost and accuracy. This is no surprise: in [46] the authors point out that the term including the effect of the mass matrix on the acceleration in the right hand side of the velocity system only contributes from the second iteration onwards, since we initialize the acceleration to zero each time step.

From figure 5.2, it is clear that the difference between the two methods

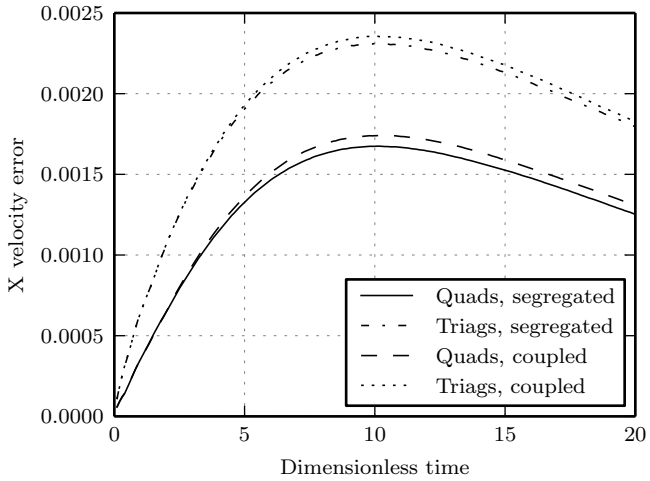


Figure 5.3: Maximum error over the domain for the x-component of the velocity, for triangles and quadrilaterals and using the coupled and segregated solution method using two inner iterations.

is greater at time 6 than after the first time step. This effect is better illustrated in figure 5.3, where we have plotted the errors $\max_\Omega |u_c - u_{th}|$ and $\max_\Omega |u_s - u_{th}|$ as a function of time. Both errors reach a maximum around time 10, and so does the difference between both methods. This difference between the segregated and coupled solution remains small, varying between 2 % and 4 % of the difference with the analytical solution. This confirms that two inner iterations suffice to reproduce the results of the fully coupled solution.

Figure 5.4 shows the error for a series of meshes with $N \times N$ quadrilaterals (triangulated for the triangle results). The time step was adapted to maintain a constant Courant number of 0.32 at the start of the simulation, using $\Delta t = 0.256/N$. We calculate the error as the maximum of the absolute value of the difference with the analytical solution over the entire domain, taking the maximum of either component for the velocity error. The velocity error after one time step (dashed line) follows the second order slope, while the pressure error follows the first order slope. The errors at time 10 (i.e. near the maximum of figure 5.3) decrease with a slope between first and second order. These effects are due to the time stepping: if we lower the Courant number to 0.03, the pressures also follow the second order law, as illustrated in figure 5.4. The errors for the coupled and the segregated method overlap, further confirming the equivalence of both methods at the time steps considered here.

Figure 5.5 shows the evolution of the maximum norm of the velocity error

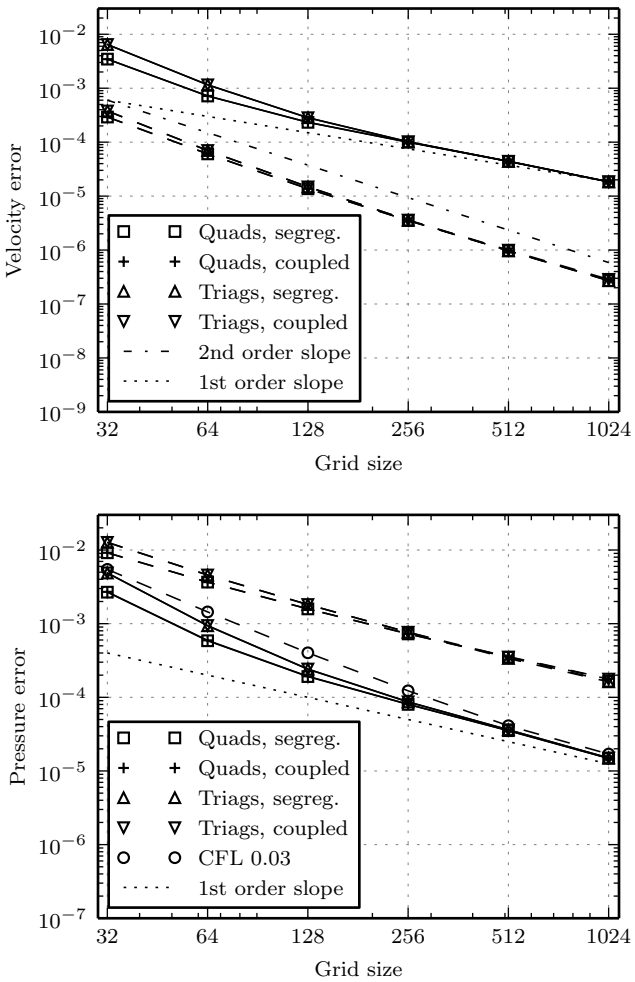


Figure 5.4: Maximum norm of the error for the velocity vector and the pressure, as a function of mesh size and with a constant Courant number of 0.32. The dashed lines connect the errors after one time step, the solid lines those at time 10. The lower plot also shows pressure error for quadrilateral elements computed at Courant number 0.03, denoted by the circles.

for increasing Courant numbers. The Courant number is computed here using the maximum velocity projection for all element edges. The segregated method becomes unstable when the Courant number reaches values close to 0.8, which is in line with the theoretical limits from [46]. Although we are using Crank-Nicolson time stepping, the errors in figure 5.5 do not decrease

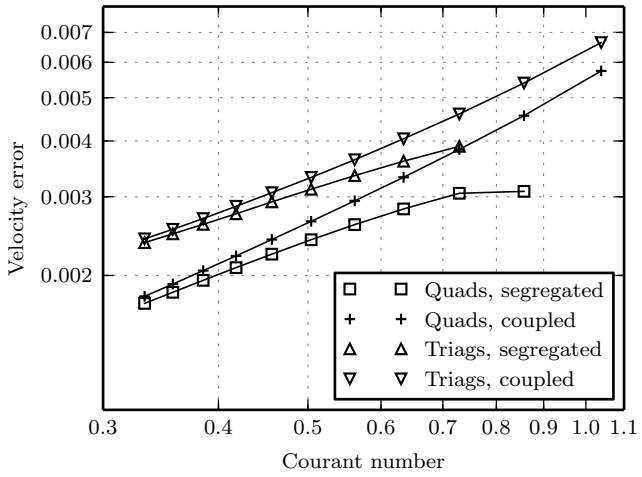


Figure 5.5: Maximum norm of the error for the velocity vector as a function of Courant number on the 64x64 grid.

along a second order slope. Further tests with the fully coupled scheme show that second order in time is only visible at Courant number 2 and higher. In [108], a similar effect is visible in the numerical tests and this is attributed to the spatial component of the error.

5.1.2 Turbulent channel flow

The plane channel flow, i.e. the flow between two infinite flat plates driven by a uniform force parallel to the plates, is one of the most widely used benchmark cases for turbulent flow simulations. The configuration is illustrated in figure 5.6, looking along the z-axis which corresponds to the spanwise direction. The acceleration due to the driving force is denoted g, h is the channel half-height and U is the average streamwise velocity, which is also the only non-zero average velocity component.

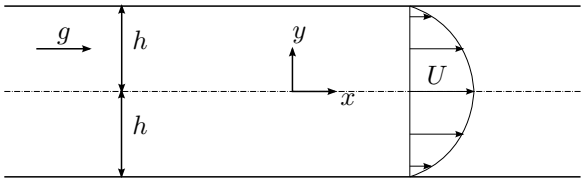


Figure 5.6: Problem configuration for the channel flow

The Reynolds number is usually defined in terms of the half-height and friction velocity, i.e. $\mathrm{Re}_\tau \equiv u_\tau h/\nu$. For $\mathrm{Re}_\tau = 180$, reference DNS data is

available from [109]. These results were later extended to $\mathrm{Re}_\tau = 395$ and $\mathrm{Re}_\tau = 590$ in [110]. Their data is available online[1] and is the standard reference for channel flows at the given Reynolds numbers.

The turbulent channel flow is a challenging test for Large Eddy Simulation [111]. Although we are not attempting any turbulence modeling in this work, it is interesting to see how our implementation performs in the case of direct numerical simulation. The purpose of our simulations is twofold. First, we want to asses the accuracy of the chosen solution method using a turbulent flow and gain insight in the dissipative nature of the solution scheme. For a sufficiently fine mesh, the reference results should be recovered. Second, DNS results with particles in the flow are available from e.g. [10], so we need an accurate turbulent flow field as a basis for particle model validation.

To keep the mesh resolution requirements reasonable, and to match with available results including particles, we only consider $\mathrm{Re}_\tau = 180$.

Solution method

The simulation was performed using meshes with different resolutions, following the characteristics listed in table 5.1. The first column in this table lists the number of nodes in each direction, not counting the periodic nodes in the x and z directions. The parameter b defines the grading by using the

$N_x \times N_y \times N_z$	Δx^+	$\Delta y^+\vert_{y=0}$	Δz^+	b
$32 \times 65 \times 32$	70.69	11.06	23.56	0.9544
$64 \times 65 \times 64$	35.34	11.06	11.78	0.9544
$128 \times 129 \times 128$	17.67	5.422	5.890	0.9500
$256 \times 257 \times 256$	8.846	2.711	2.945	0.9500

Table 5.1: Properties of the different meshes.

formula from [50] to compute the y-coordinates of the nodes:

$$y = \frac{h}{b} \tanh\left(\xi \tanh^{-1} b\right) \tag{5.2}$$

Here, ξ is uniformly spaced over the interval $[-h, h]$. The domain size is the same as in [110] and [50], i.e. $4\pi h$ in the x-direction and $4\pi h/3$ in the z-direction. Our coarsest mesh matches the coarsest mesh from [50] exactly. The resolution of $128 \times 129 \times 128$ was also used in [110], but they used Chebyshev points in the wall-normal direction, resulting in a different mesh grading.

The simulations were carried out using the semi-implicit, segregated solver described in section 3.1.2. For stabilization we used the formulation based on the metric tensor (equations (3.20) to (3.25)), using the standard $c_1 = 1$

[1] `http://turbulence.ices.utexas.edu/MKM_1999.html`

and $c_2 = 16$. Table 5.2 lists the time steps used for the different simulations, together with the time over which statistics were collected. The dimensionless time step is defined as $\Delta t^+ \equiv \Delta t u_\tau^2/\nu$. The flow was forced using a

Mesh	Δt^+	Iterations	Flow-throughs	CFL
$32 \times 65 \times 32$	0.324	270000	601	0.19
$64 \times 65 \times 64$	0.130	124000	110	0.095
$128 \times 129 \times 128$	0.0648	160000	71	0.14
$256 \times 257 \times 256$	0.0324	90000	20	0.15

Table 5.2: Time steps and statistic collection durations.

constant body force, computed from the target friction Reynolds number $\mathrm{Re}_\tau = 180$. We initialized using the laminar solution, with up to 30% perturbation from a uniform random distribution. This is the same technique as used in [50], but it is in contrast with [110], where a constant mass flux is imposed by adjusting the mean pressure gradient. The advantage of using a constant driving force is that no adjustment procedure is needed. The disadvantage is that it takes many iterations for the flow to decelerate to the fully turbulent state. To mitigate this problem, we only applied the above initialization on the coarsest mesh, and initialized the finer simulations using interpolation of the turbulent solution on the coarse grid. All averaging was done both in time and in the homogeneous (i.e. parallel to the walls) directions.

Results

Profiles of the mean velocity, RMS velocity fluctuations and the shear stress are presented in figures 5.7, 5.8 and 5.9, respectively. Throughout the figures, the solid black lines represent the reference results from [110], the solid gray line are the results on our finest mesh and the dashed and dotted lines are the coarser meshes. Each graph represents the profiles for both the top and bottom half of the channel, overlaid on the same graph for comparison. Except for the finest mesh, both sides overlap, indicating good statistical convergence.

From figure 5.7, it is immediately obvious that coarser meshes lead to an over-prediction of the mean velocity near the center of the channel. This is consistent with the results from e.g. [57], although in [50] the mean velocity is lower than the reference results [110], using an identical $32 \times 64 \times 32$ mesh. This is due to the adjustment using the parameter c_1 in equation (3.23), which was set to 16 in [50]. The finest mesh result visually overlaps with the reference DNS.

The over-prediction of the maximum $\sqrt{u'u'}$ on coarse meshes, seen in figure 5.8, is consistent with other results, including [50, 57, 112]. Again,

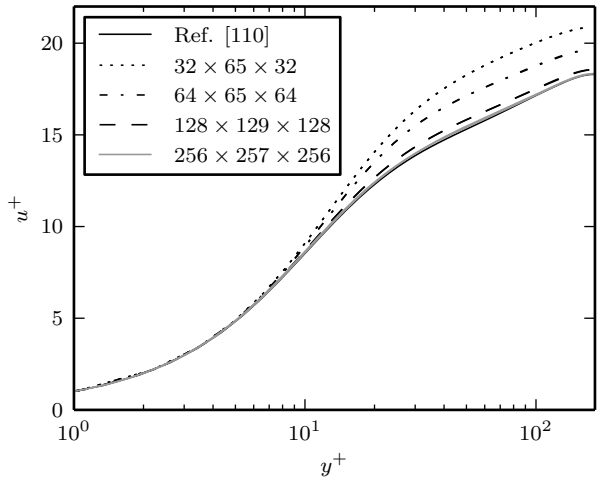


Figure 5.7: Mean velocity profiles

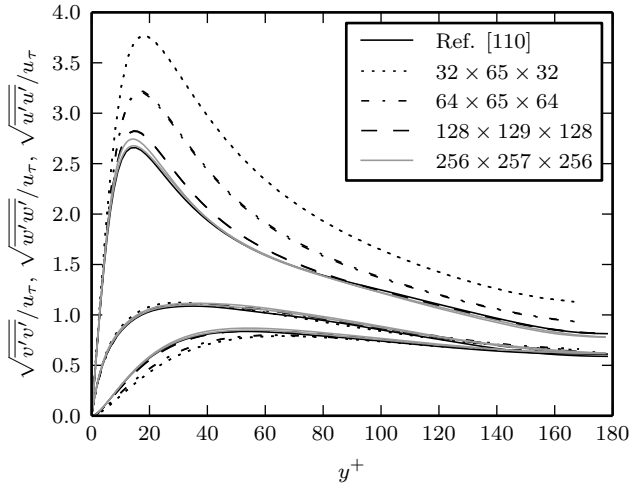


Figure 5.8: RMS velocity fluctuations, from top to bottom: $\sqrt{\overline{u'u'}}/u_\tau$, $\sqrt{\overline{w'w'}}/u_\tau$, $\sqrt{\overline{v'v'}}/u_\tau$

the finest mesh produces the correct result, but there is a slight difference between the top and bottom half of the channel. This is due to an incomplete statistical convergence, since we were only able to collect statistics for 20 flow-through times in this case, corresponding already to more than a full month of simulation using 4 nodes on the VKI cluster (see table 3.1).

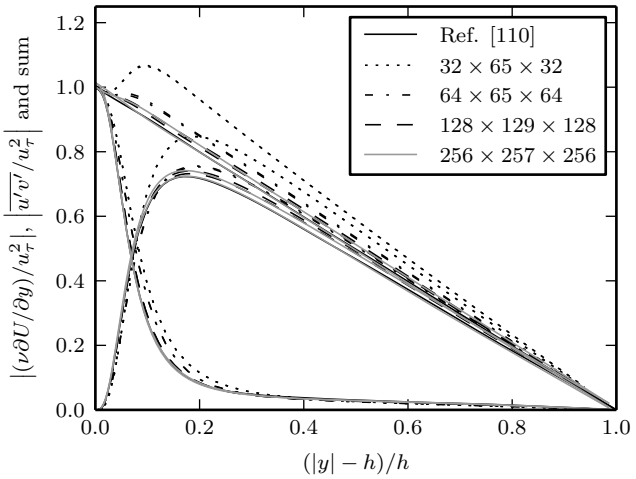


Figure 5.9: Absolute values of the shear stresses on both sides of the channel, i.e. the mean shear stress ($\left|(\nu\partial U/\partial y)/u_\tau^2\right|$, near unity at the wall), the Reynolds shear stress ($\left|\overline{u'v'}/u_\tau^2\right|$, zero at the wall and the center) and their sum.

In figure 5.9 an over-prediction of the Reynolds shear stress is observed for coarse meshes. This results in an apparent violation of the stress balance, which can be shown theoretically to be exactly linear (e.g. [113]). The simulations are only a numerical approximation, however, and the resulting errors may cause deviations from the stress balance. It is clear, however, that the imbalance in the stresses is greatly reduced as the mesh is refined.

The energy spectra presented in figures 5.10 to 5.12 were computed using the Fast Fourier transform on the streamwise and spanwise velocity autocorrelation functions. As expected, the finest mesh results are close to the reference spectra, but they are still quite noisy due to the incomplete convergence of the statistics. In most cases, the spectra for the $128 \times 129 \times 128$ mesh drop off sooner than the reference DNS, indicating that our finite element discretization is more dissipative than the spectral/Chebyshev method used in [110].

5.2 Particle model

5.2.1 Taylor-Green vortices

In [71], the preferential concentration of bubbles in the Taylor-Green vortex flow was studied, using equation (3.50), except for the locally-implicit

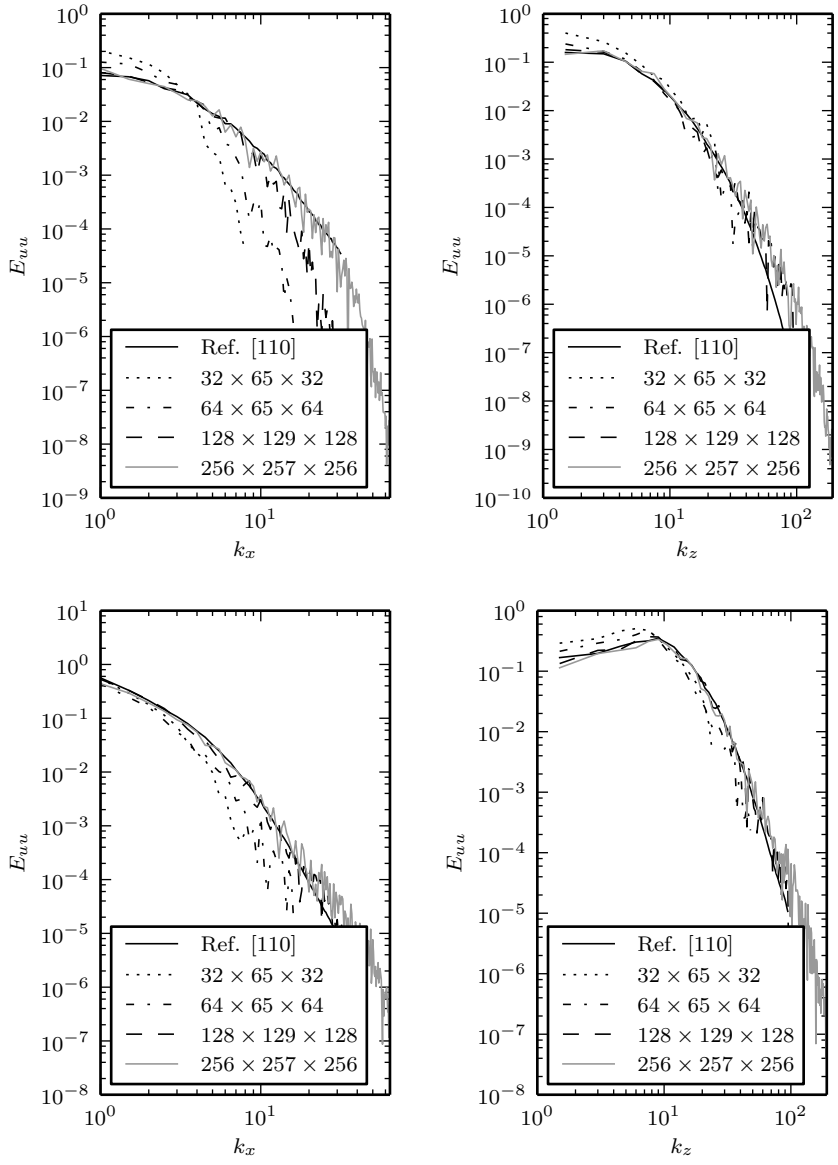


Figure 5.10: Energy spectra for $u'u'$. Left: streamwise direction, right: spanwise direction. Top: Centerline, bottom: $y^+ \approx 5$

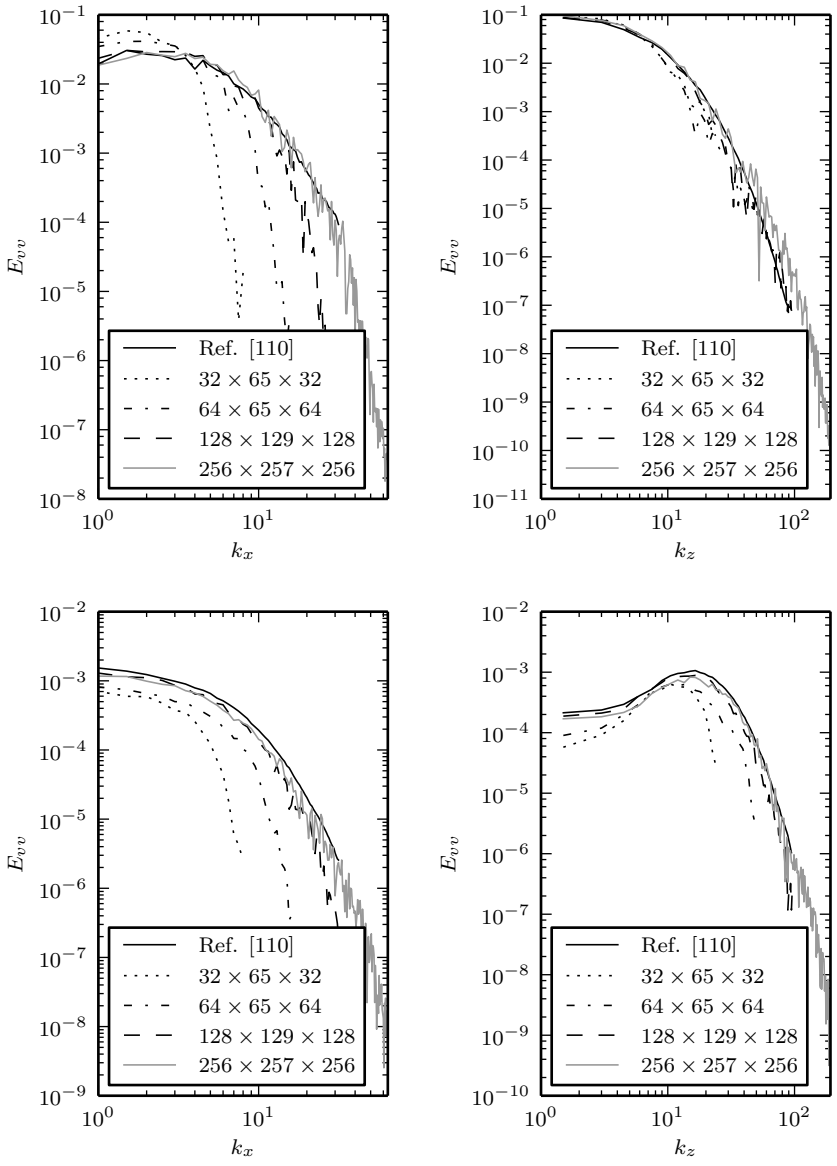


Figure 5.11: Energy spectra for $v'v'$. Left: streamwise direction, right: spanwise direction. Top: Centerline, bottom: $y^+ \approx 5$

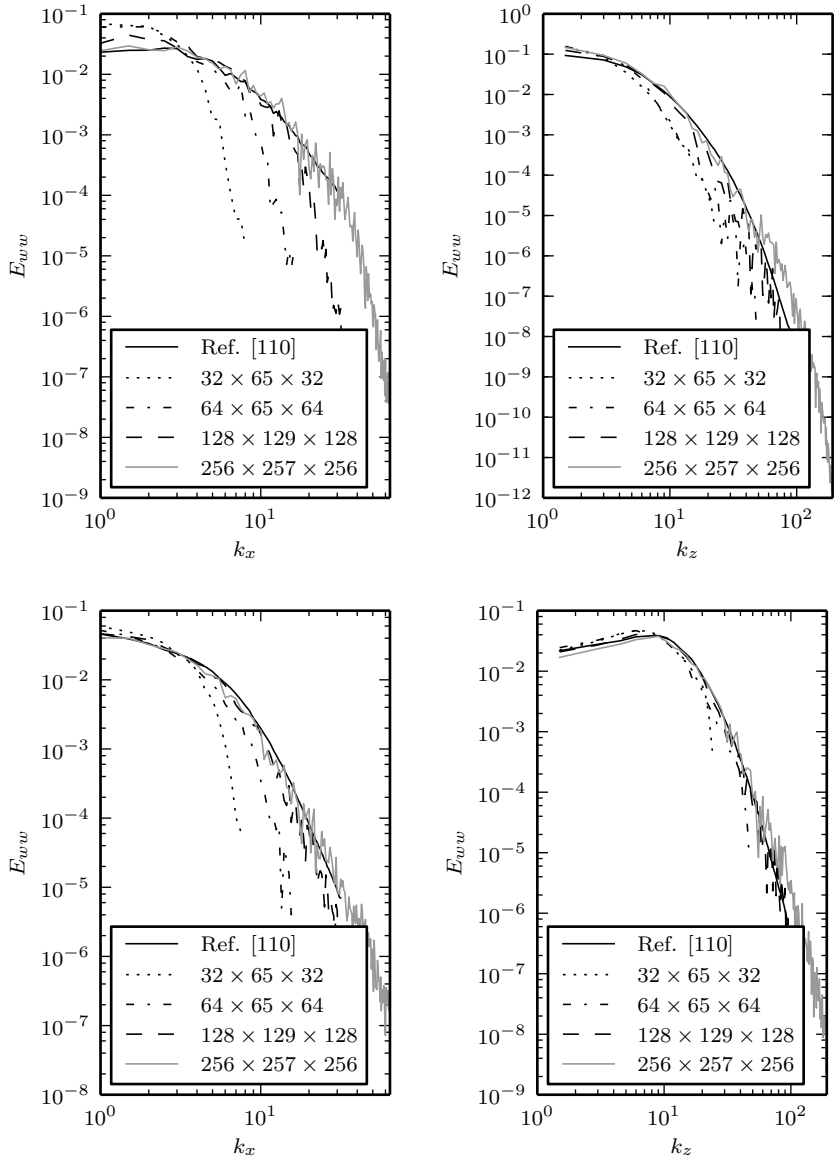


Figure 5.12: Energy spectra for $w'w'$. Left: streamwise direction, right: spanwise direction. Top: Centerline, bottom: $y^+ \approx 5$

factor. We repeat the same test here, to verify the correctness of our implementation. We adapt the parameters to match the study [71], so we set $U_a = V_a = 0$, $V_s = 1/4\pi$ m/s and $\nu = 1/5000$ m^2/s. For the particle phase, we set $\tau = 1/4$ s and β is set to 0 for solid particles and 3 for bubbles. To stay coherent with the reference data, we use the non-dimensional time $4\pi V_s t/2D$.

Before moving on to the particle concentration field, we examine the particle velocity itself. Due to the explicit nature of equation (3.50) and the simple analytical solution of the flow field, it is possible to compute an analytical solution of the particle velocity field and use that to investigate the errors resulting from different approximations (discussed in section 3.2.1) of the velocity gradient. The analytical solution was obtained using computer algebra software and is omitted here for the sake of brevity. Figure 5.13 shows the evolution of the error as we increase the grid size from 32x32 to 512x512. The grid is graded in the vertical direction, to assess the effect of a non-uniform mesh. The complete finite element formulation (equation (3.55)) yields the most accurate result. The node-based formulation of equation (3.56) has an error that is almost twice as large, but still follows the second order slope. Finally, the finite element formulation that makes use of a lumped mass matrix is almost an order of magnitude less accurate than the other methods. Given the large gain in performance and the small cost in accuracy, the node-based formulation is chosen for this work. Figure 5.14

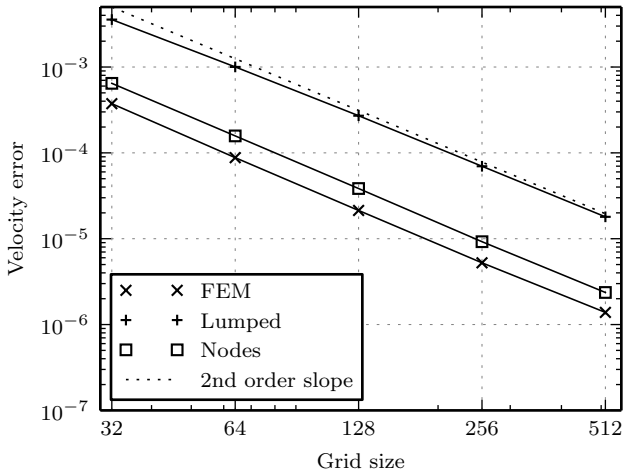


Figure 5.13: Maximum norm of the particle velocity error over the domain when compared with the analytical solution, at non-dimensional time 2

shows the particle velocity field, using vectors and contours of the X-velocity.

The contours for the Y-velocity are not shown, but could be obtained by rotating the figure 90° counter clockwise. Due to the particle inertia, solid particles migrate out of the vortices. Bubbles tend to accumulate near the vortex centers, due to the effect of the fluid acceleration force. These effects can be seen in the velocity vector field, where the vectors circling the central vortex are directed more towards the center for bubbles than for solid particles.

The velocity vectors also indicate that the particle velocity field is not divergence-free. Around the point $(1/4, 1/4)$ in figure 5.14a for example, the horizontal vectors are shorter than the vertical vectors at an equal distance from this point, resulting in a non-symmetric distribution of the particles. This effect is reflected in the particle concentration distributions, shown in figure 5.15. For both bubbles and solid particles, the concentration field shows stretched patches at the corners between vortices, which are the result of the different magnitudes for the horizontal and vertical velocity components. As expected, the concentration field for the bubbles shows a maximum near the vortex centers, while the solid particles migrate away from the vortex centers. The streamlines also show the particles spiraling outwards for solid particles and inwards for bubbles. To do a quantitative analysis of the concentration field, it is possible to obtain an analytical solution of the particle concentration evolution with time, as was also done in [71]. We extend the analysis to be applicable for all values of β and we retain the locally implicit term. The equation is obtained by observing that the concentration reaches an extremum at the vortex center, so its gradient reaches zero. Neglecting Brownian motion, we obtain the following equation for the concentration:

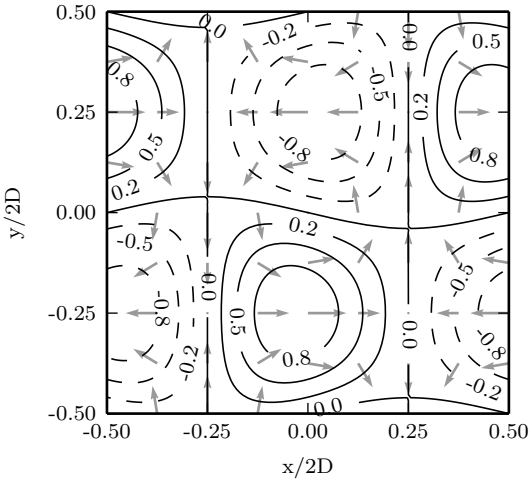
$$\frac{\mathrm{d}c}{\mathrm{d}t} + c(t)\left(\nabla \cdot \boldsymbol{v}\right)_{x=0,y=0} = 0 \tag{5.3}$$

Filling in the expression for the divergence of the Taylor-Green particle velocity field, evaluated at the origin, yields the time evolution of the concentration at the vortex center:

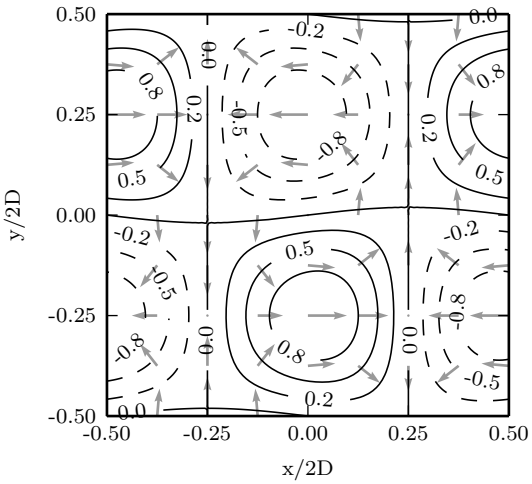
$$\begin{aligned} c(t) = c_0 \exp\Bigg[(\beta-1)\left(\frac{1}{2\nu\pi^2\tau} + \frac{1}{D^2}\right)\Bigg(4\nu\pi^2 t \\ - D^2 \log\left(D^2 \exp\left(\frac{4\nu\pi^2 t}{D^2}\right) + (\pi V_s \tau)^2\right)\Bigg)\Bigg] \\ \cdot \left(D^2 + (\pi V_s \tau)^2\right)^{(\beta-1)(D^2/(2\nu\pi^2\tau)+1)} \end{aligned} \tag{5.4}$$

We can compare this with the result from [71], adapted to our notation and generalized for any β:

$$c(t) = c_0 \exp\left[\frac{V_s^2(\beta-1)\tau}{2\nu}\left(1 - \exp\left(-\frac{4\nu\pi^2 t}{D^2}\right)\right)\right] \tag{5.5}$$



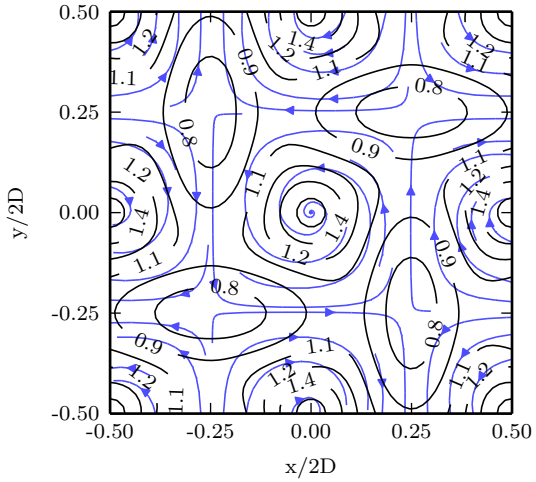
(a) Bubbles ($\beta = 3$)



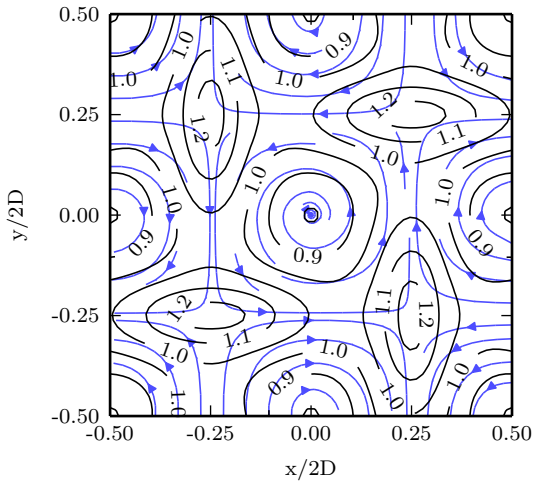
(b) Solid particles ($\beta = 0$)

Figure 5.14: Velocity vectors and contours of the X-velocity for the particles, normalized by V_s, at non-dimensional time 2.

Figure 5.16 plots both equations, together with the numerical result on a 64x64 uniform mesh. Both analytical solutions and the numerical solution appear to be very close together. The excellent agreement can be explained by the small particle relaxation time ($\tau = 1/4$) which satisfies the hypothesis of the equilibrium Euler model very well. When we increase the relaxation



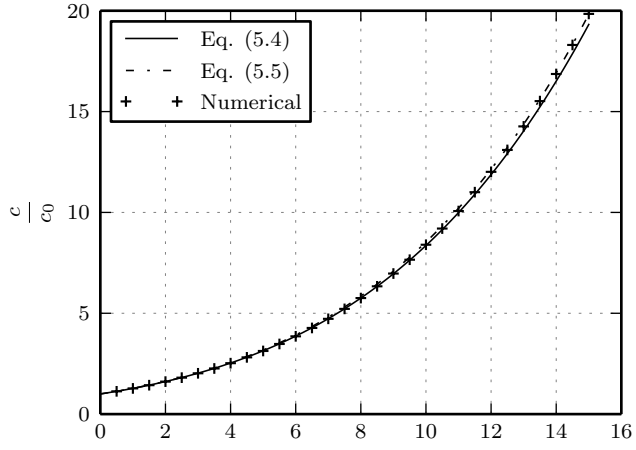
(a) Bubbles ($\beta = 3$)



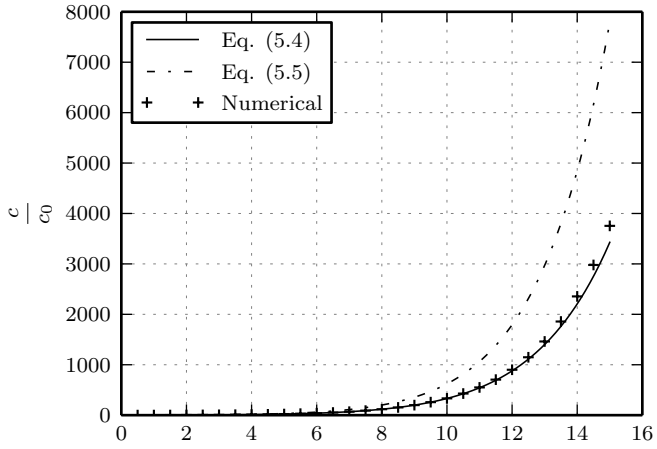
(b) Solid particles ($\beta = 0$)

Figure 5.15: Particle streamlines and volume fraction contours, normalized by the initial uniform volume fraction and shown at non-dimensional time 2.

time to $\tau = 3/4$, the simpler analytical model (5.4) predicts peak concentrations that are more than twice as large as the values from the locally implicit approach. This indicates that the higher accuracy of the approximation $\mathrm{d}\boldsymbol{v}/\mathrm{d}t \approx \mathrm{d}\boldsymbol{u}/\mathrm{d}t$ becomes more important as τ increases. At this



(a) $\tau = 1/4$, mesh size 64



(b) $\tau = 3/4$, mesh size 512

Figure 5.16: Particle volume fraction at the domain center for bubbles ($\beta = 0$), relative to the initial concentration c_0, for relaxation times $\tau = 1/4$ and $\tau = 3/4$

relatively high relaxation time, steep concentration gradients appear in the solution, and a mesh size of 512 was necessary to avoid oscillations and negative concentrations.

When increasing τ, we should mind the basic hypothesis of the equilibrium Euler approach, namely that the particle velocity field can be represented as

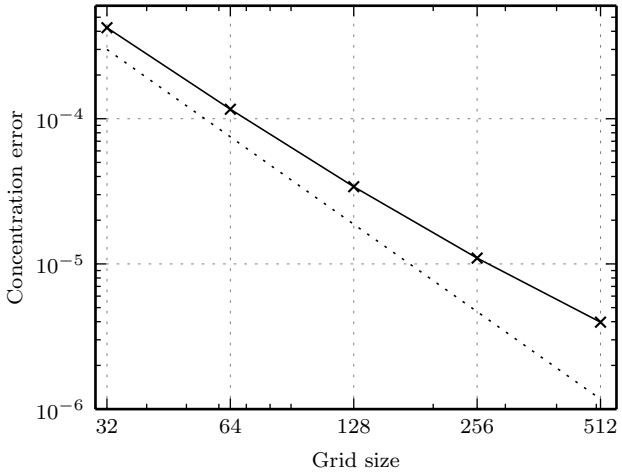


Figure 5.17: Error of the bubble volume concentration at the vortex center at time 0.1, normalized by the initial concentration. Evolution as a function of uniform mesh size, with the second order slope as reference (dotted line).

a unique vector field over the domain. In [17], a rigorous condition was established to verify that this is the case. The condition is that all eigenvalues of the matrix

$$\frac{1}{2}\left(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^{\mathrm{T}}\right) \tag{5.6}$$

are greater than $-1/\tau$ at every node in the domain. We have verified that this condition is satisfied for both relaxation times used in the current test.

In figure 5.17, we plot the evolution of the error at the vortex center as a function of the mesh size. For each grid, the simulation was run up to time 0.1 and the time step was adjusted to keep a constant Courant number as $3.2/N$, where N is the number of grid points. The expected second order slope is almost recovered, with a slight deviation that is likely due to errors introduced by the time discretization.

5.2.2 Burgers vortex

In [89], the collision rates of rain drops are studied using the same collision kernel as we describe in table 3.3, using the Burgers vortex as fluid flow field. The method is based on the calculation of particle trajectories, so collision rates in [89] are only visualized on top of particle trajectories. Still, it is possible to compare the collision rate fields that we obtain with the figures from [89] to verify the correctness of our implementation.

The Burgers vortex is often used to approximate a turbulent structure. The velocity components of the flow field are given by:

$$u = -\frac{r_0^2 y \omega_0}{2(x^2+y^2)} \left(1 - e^{-\frac{x^2+y^2}{r_0^2}}\right) \tag{5.7}$$

$$v = \frac{r_0^2 x \omega_0}{2(x^2+y^2)} \left(1 - e^{-\frac{x^2+y^2}{r_0^2}}\right) \tag{5.8}$$

Here, ω_0 is the vorticity at the center of the vortex and r_0 is a parameter related to the size of the vortex. In this validation of the particle collision model, we directly impose these values as a steady flow field and no fluid computation is performed.

Three particle sizes are considered, the properties of which are listed in table 5.3 in the presence of a weak and strong vortex. The Stokes number

Diameter (μm)	τ_p (s)	Weak ($\omega_0 = 18$ s^{-1}, $r_0 = 1$ cm) St$_w$	Strong ($\omega_0 = 180$ s^{-1}, $r_0 = 1/3$ cm) St$_s$
20	1.3×10^{-3}	2.5×10^{-3}	2.5×10^{-2}
40	5.2×10^{-3}	1.0×10^{-3}	1.0×10^{-2}
80	2.1×10^{-2}	4.0×10^{-2}	4.0×10^{-1}

Table 5.3: Particle relaxation time and Stokes numbers for the weak and strong vortices.

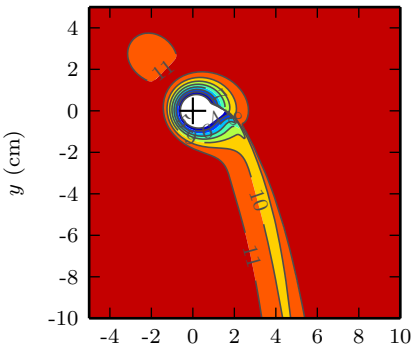
is defined as $0.11\omega_0\tau_p$, where the factor 0.11 appears because this number is defined at the location of the maximum flow velocity. To simulate the effect of gravity, an acceleration of 10 m/s^2 is imposed for the body force on the particles. The initial condition and boundary condition at the top are a uniform concentration for each particle size.

Figures 5.18 and 5.19 show the resulting collision rates. The collision rate is defined as:

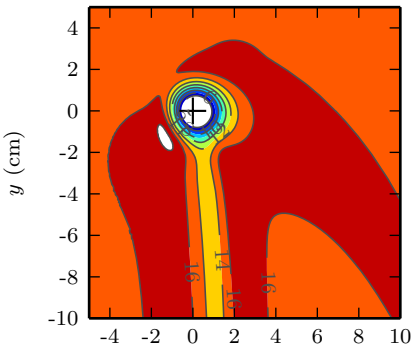
$$\frac{n_\alpha(\boldsymbol{x},t) n_\gamma(\boldsymbol{x},t)}{n_{\alpha 0} n_{\gamma 0}} \beta_{\alpha\gamma} \tag{5.9}$$

This is the collision kernel multiplied with the product of the particle number concentrations, normalized by the initial (uniform) concentration. The collision kernel $\beta_{\alpha\gamma}$ is the sum of equation (3.68) and the appropriate expression from table 3.3. In [89], results are only presented along particle trajectories, and large gaps are present. Overall, however, our contours appear to match with their values.

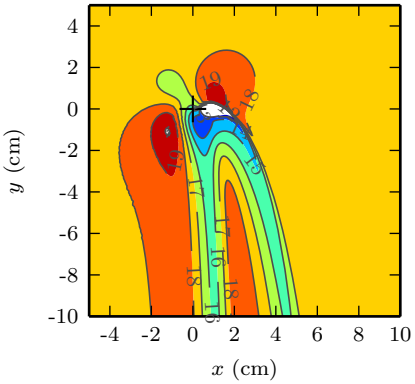
It is also interesting to compare the collisions between particles of different sizes with the collisions between particles of the same size. Figure 5.20 shows the collision rate among particles with $d_p = 20$ µm . In contrast with figures



(a) Collision rate between $d_p = 20$ µm and $d_p = 40$ µm , in 10^{-11} $\mathrm{m^3 s^{-1}}$

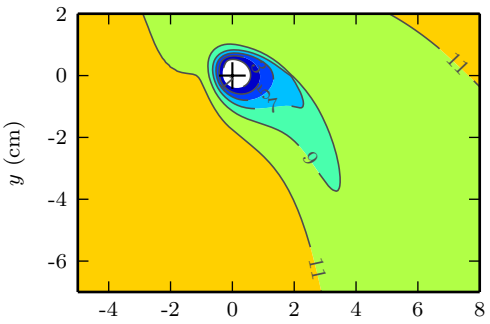


(b) Collision rate between $d_p = 20$ µm and $d_p = 80$ µm , in $10^{-10}\mathrm{m^3 s^{-1}}$

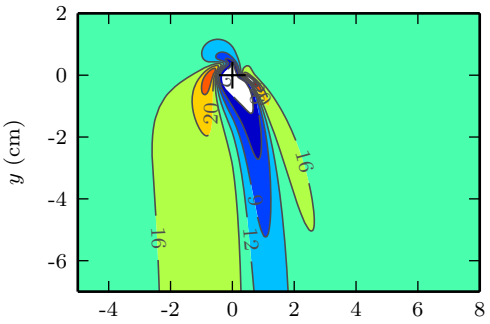


(c) Collision rate between $d_p = 40$ µm and $d_p = 80$ µm , in $10^{-9}\mathrm{m^3 s^{-1}}$

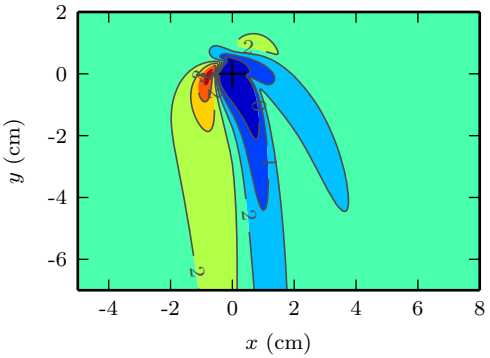
Figure 5.18: Collision rates for the weak vortex



(a) Collision rate between $d_p = 20$ µm and $d_p = 40$ µm , in 10^{-11} m^{3}s^{-1}



(b) Collision rate between $d_p = 20$ µm and $d_p = 80$ µm , in 10^{-10}m^{3}s^{-1}



(c) Collision rate between $d_p = 40$ µm and $d_p = 80$ µm , in 10^{-9}m^{3}s^{-1}

Figure 5.19: Collision rates for the strong vortex

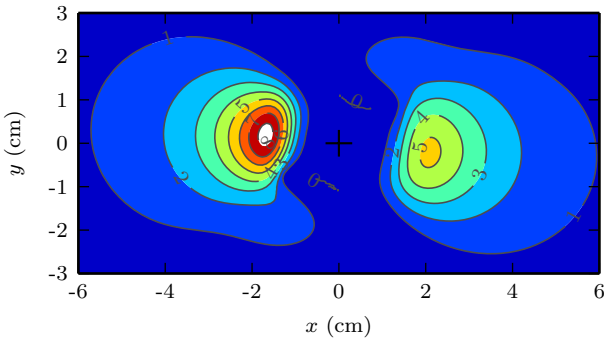
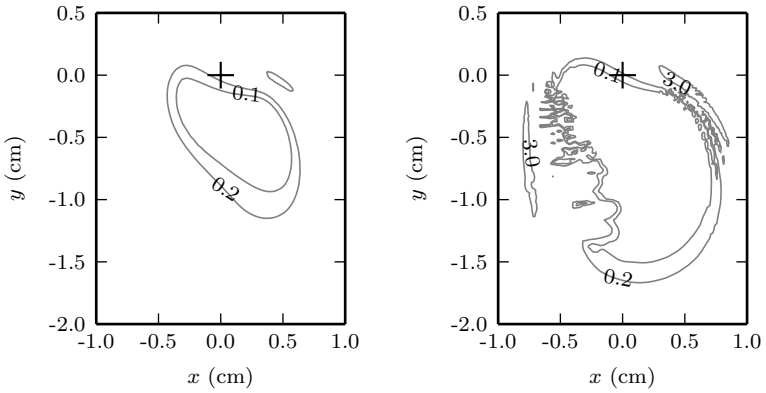


Figure 5.20: Collision rate among 20 µm particles for the weak vortex, in $10^{-14}\mathrm{m}^3\mathrm{s}^{-1}$

5.18 and 5.19, the collision rate far from the vortex is zero, since the flow is almost uniform and gravity has no effect on the collision rate of particles of the same size. The collision rate is also 3 orders of magnitude smaller than the rate between the 20 µm and 40 µm particles.

Oscillations

To obtain the above results, the crosswind diffusion (equation (3.57)) had to be used, with $a_0 = 0.1$ for the weak and $a_0 = 1$ for strong vortex. Figure 5.21 illustrates the effect of the stabilization for the strong vortex. Without



(a) $a_0 = 1$ (b) $a_0 = 0$

Figure 5.21: Contours of n/n_0 for the 80 µm particles after 0.06 s in the strong vortex.

stabilization ($a_0 = 0$), pronounced spurious oscillations are visible, and the solution deteriorates before steady state is reached. With stabilization ($a_0 = 1$), the field is smoother but the values of the concentration are less extreme, as indicated by the smaller contours. This illustrates that the stabilization requires adjustment to find an appropriate value for a_0 that results in neither too little nor too much dissipation.

5.2.3 Turbulent channel flow

In [10], DNS and LES of particle laden channel flow is performed, using Lagrangian tracking for the particles. Averaged concentration profiles are presented for a number of particle relaxation times $\tau_p^+ \equiv \tau_p u_\tau^2/\nu$, ranging from 1 to 25. Since the fundamental assumption is that the particle relaxation time is small with respect to the fluid time scale, we only compare using $\tau_p^+ = 1$ and also include additional results for $\tau_p = 0.1$.

In channel flow, preferential concentration near the walls occurs as the particle time scale approaches the fluid time scale [114]. When this happens, local concentration gradients also increase, and the use of stabilization such as crosswind diffusion becomes necessary. The difficulty here is a correct choice of the parameter a_0: it should be large enough to eliminate the oscillations, but not so large as to destroy the correct evaluation of the preferential concentration effect.

To perform our computations, we set the initial particle concentration to 1. The flow field was initialized using the last field from the statistically converged turbulent channel flow from section 5.1.2. We then observed the time history of the concentration near the wall (at $y^+ = 0.5$ and $y^+ = 4.5$) and averaged over the homogeneous directions once a statistically steady state for the concentration was reached. Figure 5.22 summarizes the results. For the small particle relaxation time $\tau_p^+ = 0.1$, the concentration remains almost uniform across the channel. Large gradients do not appear, and the crosswind diffusion was not switched on. Also, no negative concentrations appeared so the clipping to zero never occurred.

For $\tau_p^+ = 1$, the particle relaxation time is closer to the Kolmogorov time scale of the fluid, resulting in the appearance of preferential concentration. This is also accompanied with steep concentration gradients, necessitating the use of crosswind diffusion. For $a_0 = 0.005$ (i.e. the black lines) the normalized concentration n/n_0 reaches almost 3 near the wall on the fine mesh. This is still well below the value of 4 obtained in [10]. On both meshes, the centerline concentration has increased to 1.5. This is an effect of the clipping of the concentration to avoid negative values, since starting from a uniform unit concentration it is impossible to obtain a higher average value over the entire channel span. The fact that the clipping has such a large effect indicates that the stabilization is insufficient.

The gray lines correspond to $a_0 = 0.05$. Here, the centerline concentration remains near 1 as expected, but the effect of preferential concentration is

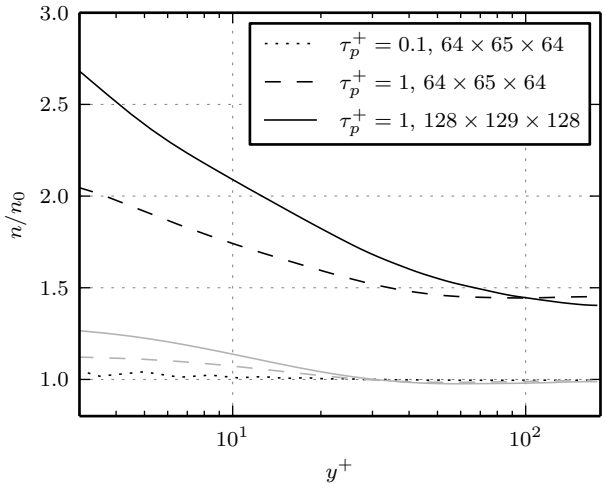


Figure 5.22: Particle concentration profiles. Black lines: $a_0 = 0.005$, gray lines: $a_0 = 0.05$. For $\tau_p^+ = 0.1$, no crosswind diffusion was needed.

also much less pronounced.

We conclude that while the basic trend of preferential concentration is visible, the numerical values are far from DNS reference results from e.g. [10]. The current formulation of the model is therefore only valid for small values of the relaxation times, though this may be sufficient for the study of nanoparticles, depending on the exact size of the particles and the fluid time scales. To extend the domain of applicability, more advanced stabilization techniques are required, such as the flux-corrected-transport (FCT) method used in [115].

Chapter 6

Experimental validation

The objective of the experiments is to study the feasibility of measuring particle size distributions and their evolution in time in interesting flow configurations, and to determine if the resulting data is appropriate for model validation. While within the scope of this work it is not possible to carry out a fully quantitative validation, we will demonstrate that the applied techniques for measuring and post-processing open the way to future experiments that could be used to perform a detailed validation and fine-tuning of the model. One of the important effects that we wish to capture is the coagulation of liquid or solid particles in air. So far, but a few experiments that attempt to measure this phenomenon have been published. One example is [63], where the authors study coagulation of DEHS (Di-Ethyl-Hexyl-Sebacat) droplets in grid generated turbulence.

In our experiments, we generate a flow field that is easy to reproduce numerically and study the dispersion and coagulation of particles. With the available particle generation techniques, a sufficient particle concentration for coagulation could only be obtained in low-speed flow, so we expect Brownian motion to be the dominant factor in the coagulation process.

6.1 Experimental setup

The experimental setup is presented in figure 6.1. The test section consists of a cube with an edge length of 56 cm and an inlet and outlet slot of 1 cm $\times$ 50 cm, located at 3 cm from the bottom and the top of the box, respectively. The configuration is such that a circulation zone should appear, which we hope will promote the residence time of the particles and give them enough time to coagulate. The air inlet is conditioned using a device similar to a wind tunnel, with a settling chamber followed by a convergent duct. The system is fed using an electronically regulated fan, with settings ranging from 0 to 50 in intervals of 0.1. These settings are linked to air velocities at the inlet of the test chamber. To facilitate optical access for PIV measurements, the test chamber is made of plexi-glass. The setup as it was used during the flow field measurements is depicted in figure 6.2.

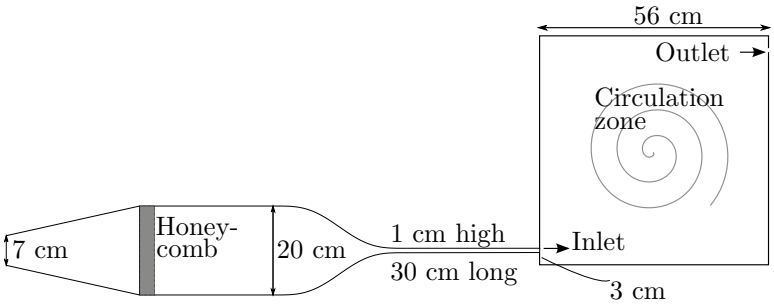


Figure 6.1: Schematic of the experimental setup.

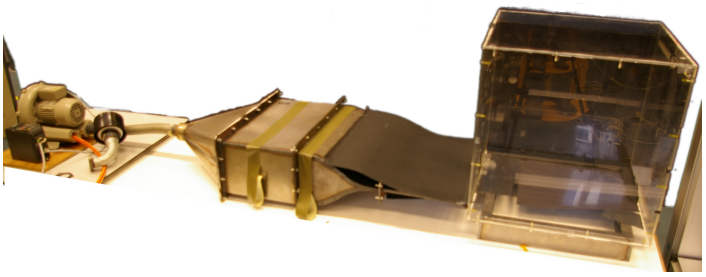


Figure 6.2: Photograph of the experimental setup.

6.2 Flow measurements

The flow field is characterized using 2D PIV measurements, with the purpose of providing validation data for the fluid model and to characterize the inlet conditions for the simulations based on all experiments using the inlet depicted in figure 6.1. Most measurements will be focused around the center plane, where the flow should be approximately two-dimensional, justifying the use of 2D PIV. We used a commercial system provided by Dantec, a complete description can be found in [116]. The measurements were performed in three planes, shown together with the coordinate system in figure 6.3. Plane A is a horizontal cut across the inlet, plane B provides a detailed view of the inlet flow at the center plane and plane C provides a global view of the center plane. Camera and laser sheet were positioned using a traverse, allowing placement with an accuracy of 0.5 mm.

For each considered fan setting and camera position, image pairs were processed using the "Adaptive Correlation" method in Dantec Dynamic-Studio 3.31. We used an initial window size of 64×64 pixels, refined to 32×32 pixels with 50% overlap. The default validation settings were used, i.e. vectors with a magnitude that deviate more than 50 % from the value of the average of the surrounding vectors are discarded. Table 6.1 lists the dif-

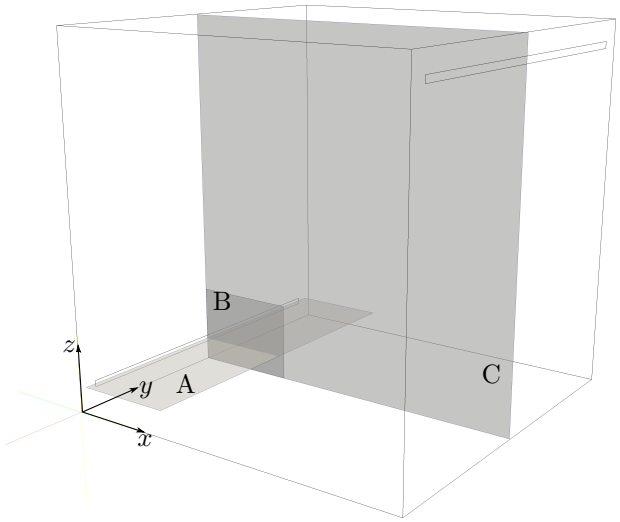


Figure 6.3: Coordinate system and locations of the measurements.

ferent fan settings with averaged flow quantities and image capture settings. The x-component of the velocity is indicated by U_c for the time-averaged centerline velocity and U_m for the mean velocity over the z-direction at the center plane. The Reynolds number Re_m is defined with respect to the inlet height D and U_m. The time difference Δt between the PIV frames differs

Fan setting	U_c (m/s)	U_m (m/s)	Re_m	Δt_A (μs)	Δt_B (μs)	Δt_C (μs)
2	0.44	0.3	209		750	
5	0.77	0.57	397		400	
10	1.21	0.94	648	400	250	8000
20	2.19	1.76	1217	250	150	3000
30	3.22	2.63	1824	200	100	2000
50	5.26	4.35	3008		50	

Table 6.1: Fan settings with velocity and time between PIV frames.

between measurements due to the differences in physical size of the measurement area. Table 6.2 details the image sizes, number of image pairs taken and the average validation rate. The significantly lower rate for plane A is due to some reflections and background noise that could not be eliminated. The number of image pairs for planes A and B corresponds to the size of the image buffer. While the software allows combining several measurement series into one, using just one buffer was more convenient and we assume

Plane	x-size (mm)	$y-$ or z-size (mm)	Vector spacing (mm)	# images	Validation rate (%)
A	130	99	1.57	284	69.8
B	57	43	0.70	284	99.6
C	227	174	2.77	852	95.0

Table 6.2: Properties of the measurement windows for the different planes.

the estimate of the mean velocity is satisfactory in the high-velocity region near the entrance.

The flow was seeded using olive oil droplets, generated by a Laskin nozzle.

6.2.1 Horizontal plane (A)

The measurements in the horizontal plane serve to determine the inlet velocity in the spanwise direction and to verify the flow symmetry. Figure 6.4 shows the velocity profiles that we obtained. It is clear that the symmetry

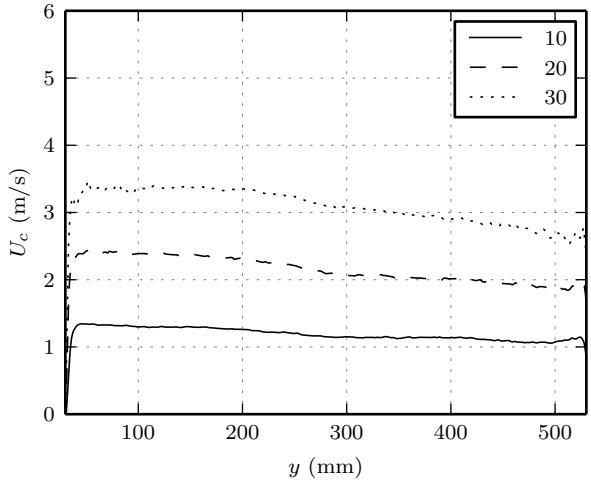


Figure 6.4: Horizontal velocity at the centerline of the inlet slot for different fan settings.

is not respected. Closer examination of the 30 cm inlet duct (see figure 6.1) indicated that it was not manufactured accurately, and the inlet height varied between 0.8 cm and 1.2 cm instead of the prescribed 1.0 cm. In [117], the same inlet device was used to study plane offset jets and the inlet was replaced by a more accurately constructed 1 m duct. Figure 6.5 shows the resulting profiles, indicating that symmetry in the horizontal plane is satis-

factory. For the measurements that we present here, the PIV results were obtained using the short 30 cm inlet, while the later particle experiments were done using an improved 1 m inlet.

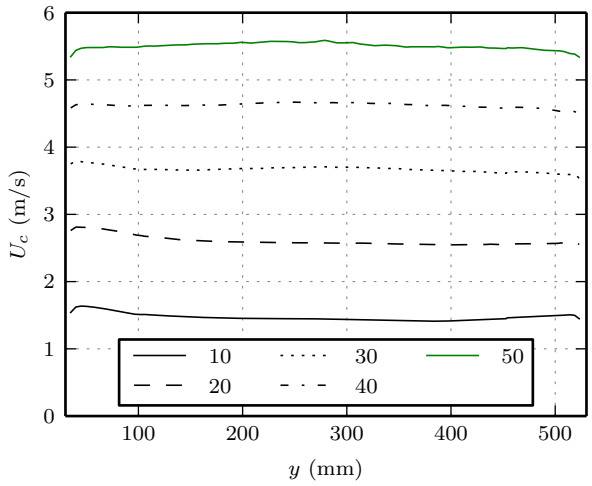


Figure 6.5: Horizontal velocity at the centerline of the inlet slot.

6.2.2 Inlet detail (B)

The detailed measurements at the inlet help to quantify the velocity profile for the purpose of defining the boundary conditions for the numerical model. The measurements were taken in a 58 mm x 44 mm plane, resulting in 16 velocity vectors across the inlet height. These measurements also allow a characterization of the recirculation zone near the inlet, resulting in flow configuration similar to an offset jet (figure 6.6). The offset jet was studied

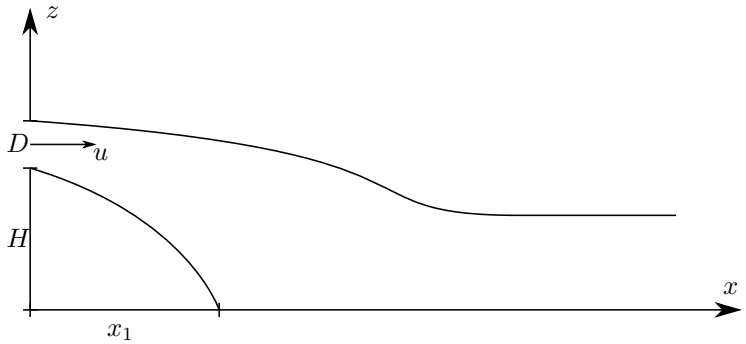


Figure 6.6: Schematic of the offset jet flow at the inlet.

in detail in [118, 119]. Even though our configuration is similar, additional experiments carried out in [117] showed that the reattachment lengths do not match with a pure offset jet flow, probably due to the confinement of the flow in the box. Figure 6.7 compares both situations for the same value of H and D, indicating a lower reattachment length in the case of a pure offset jet.

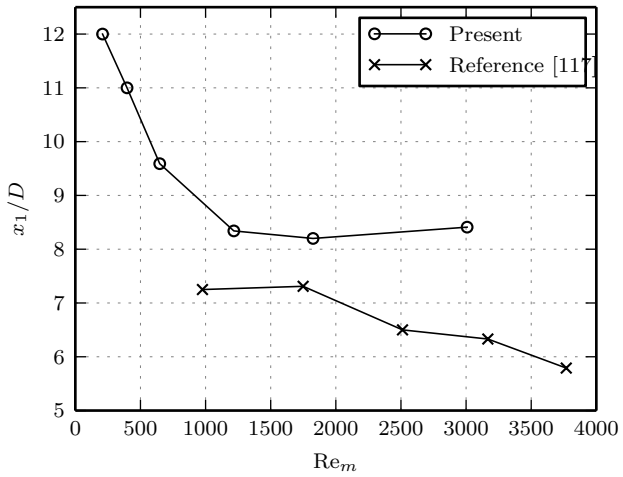
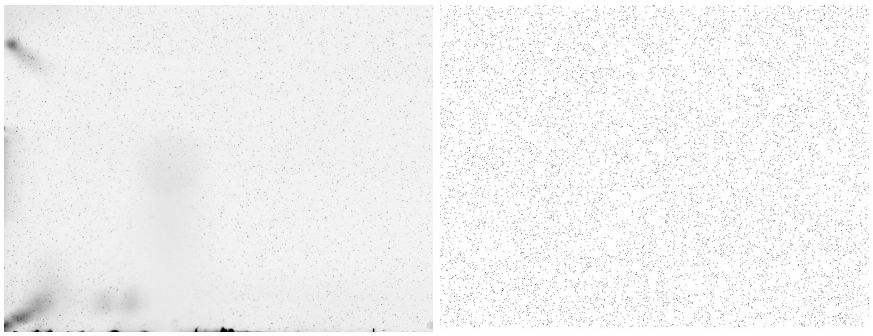
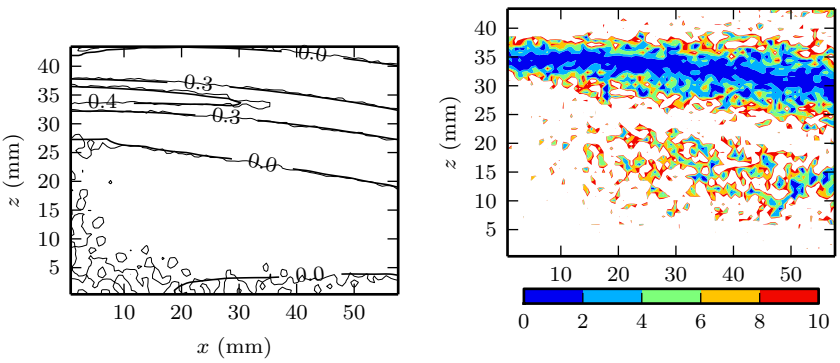


Figure 6.7: Comparison of the reattachment length x_1 between the present measurements and a pure offset jet with the same offset ratio H/D, normalized by the height of the inlet slot D.

Before presenting the inlet velocity profiles, we will now use the detailed images near the inlet to quantify the errors of the PIV system using artificial images. We make use of [120] to generate the images. The idea is to generate artificial particle images, taking into account the optical properties of the system. The particle positions are obtained using a known displacement field, which we generate here using a 2D laminar simulation of the region near the inlet. The result of the PIV algorithm when applied to the artificial image can then be compared with the known displacement, resulting in an estimate of the error. Figure 6.8 shows a comparison of a real image and a synthetic one (colors are inverted for clarity). The artifacts visible in the real image are due to light reflections that could not be eliminated completely. In figure 6.8c, the velocity contours from the simulation and from PIV on the synthetic images are overlaid, while the relative error is shown in figure 6.8d. The error is larger in the shear layer, where it can locally exceed 10%. Further from the jet, the relative error can no longer be calculated accurately, since the velocity is close to 0 m/s. Some noise is present in the lower left corner in the experimental results, due to background and



(a) Real image (b) Synthetic image



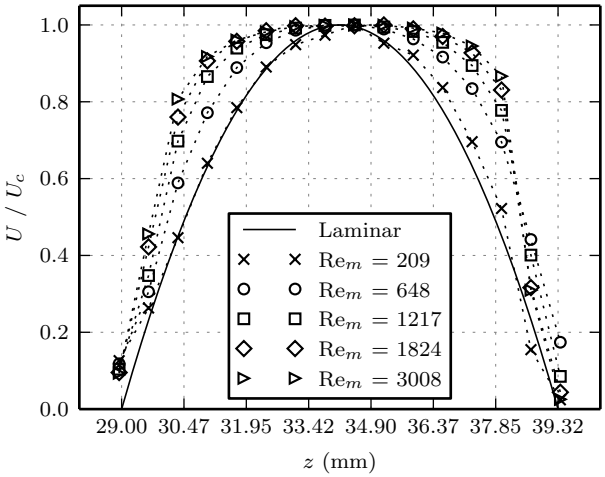
(c) x-velocity contours (m/s). (d) Relative velocity error (%)

Figure 6.8: Real and synthetic images (inverted for clarity) and the velocity field from simulation (thick lines) and PIV on the synthetic images (thin lines), as well as the relative error.

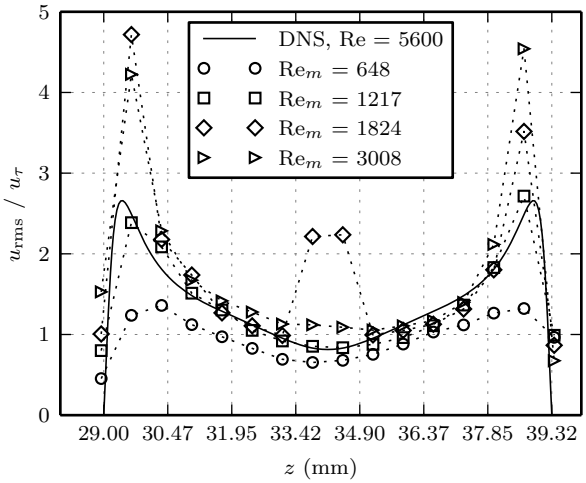
reflections that could not be eliminated.

The error analysis represents an ideal situation: we only compute the error due to the correlation procedure. Errors due to the placement of the laser sheet and particles entering and leaving the sheet are not accounted for. It is for this simple test also assumed that the particles follow the flow perfectly. In future work, more detailed error analysis should be possible, using 3D simulations and the particle concentration model to account for all these effects.

We will now discuss the vertical velocity profile across the inlet. The measured profile can be used to impose a boundary condition for the numerical simulations. Figure 6.9 shows the average velocity profile as well as the RMS value of the fluctuation. Measurements were taken at 3 mm from the inlet due to reflections. The velocity profiles are normalized using



(a) Mean velocity



(b) RMS values

Figure 6.9: Mean x-velocity profile across the inlet (top) and RMS fluctuations (bottom)

the centerline velocity (values listed in table 6.1) and the solid line indicates the laminar profile. Only fan setting 02 matches this profile closely. For the other profiles, we need to find out if they are flatter due to the onset of turbulence, or if we are simply dealing with a developing laminar profile. In

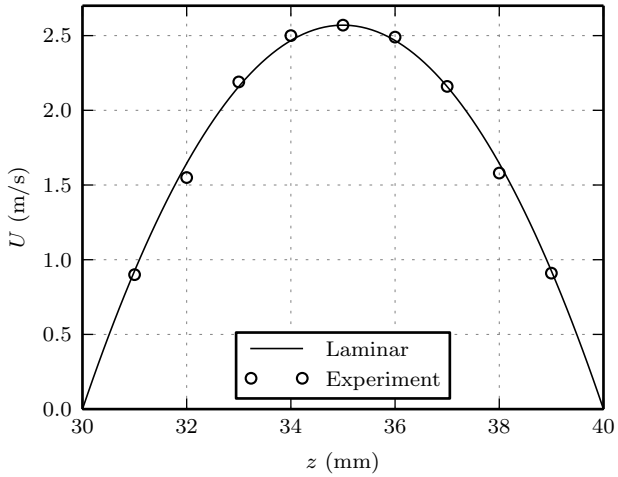


Figure 6.10: Velocity profile for fan setting 20 from [117], compared with the laminar profile. Average over 600 measurements.

[121], a correlation is given to compute the development length:

$$\frac{L}{D} = \left(0.631^{1.6} + (0.0442\mathrm{Re}_m)^{1.6}\right)^{\left(\frac{1}{1.6}\right)} \tag{6.1}$$

According to this equation, the development length for fan setting 10 should be 28.7 cm, but we see that the laminar profile is not reached. Some fluctuations are present in figure 6.9b, but at this low Reynolds number the flow should be laminar. To determine if fully developed turbulence exists, we compare with the results of the DNS from [109], at $\mathrm{Re}_m = 5600$. This is considered to be a low Reynolds number for turbulent channel flow, so we can expect that the flow will remain laminar at our much lower values, i.e. at least up to fan setting 20. The measured RMS values are much higher, however, and also far from symmetric, indicating that this statistic is not converged. For the two highest Reynolds numbers, a fully developed flow would present RMS values below those from the DNS, so this indicates that the flow is not yet developed.

Later results from [117] reconstructed the profile by making measurements in horizontal planes at different heights near the inlet, using the 1 m inlet. The resulting profile for fan setting 20 is shown in figure 6.10. The result is clearly laminar, so for the purpose of particle experiments —done at fan settings below 20— we can use a laminar boundary condition for the inlet.

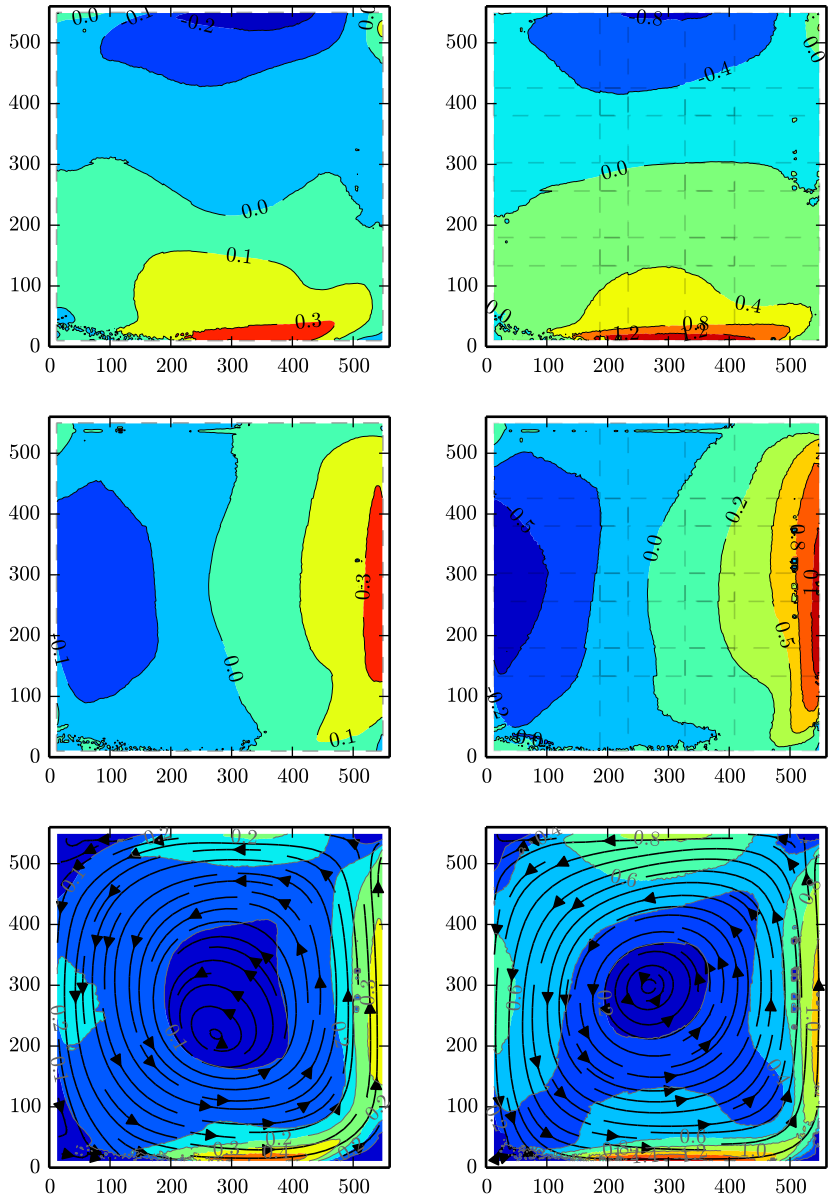


Figure 6.11: op to bottom: x-velocity , z-velocity and streamlines over velocity magnitude. Left: setting 10, right 30. Coordinates in mm.

	Coarse	Medium	Fine
Minimum cell size (mm)	5	2.5	1.8
Maximum cell size (mm)	30	27.5	19.8
Number of nodes	824560	4285050	8069527
time step (segregated) (s)	5×10^{-4}	2.5×10^{-4}	2.5×10^{-4}
time step (coupled, $\theta = 1$) (s)	0.01	0.01	-
time step (coupled, $\theta = 0.5$) (s)	2×10^{-3}	5×10^{-4}	-

Table 6.3: Statistics for the different meshes.

6.2.3 Center plane (C)

The measurements taken in the center plane are presented in figure 6.11. Because of the size of the measurement plane, each image is a combination of 3 measurements in the x direction and 4 measurements in the y direction. The borders of the images are indicated by the dashed lines. In the areas of overlap, the average of all available measurements was taken. To improve convergence of the statistics for this measurement, each window consists of 3 acquisitions of 284 images, so the corner overlap between 4 camera position averages a total of 12 acquisitions or 3408 images. Only a few small discontinuities are visible near the image boundaries, suggesting that this number of acquisitions is sufficient to obtain a converged average.

In the streamline plots, the existence of a circulation zone is evident, with the center moving more towards the box center for the higher fan setting. This kind of circulation zone should promote particle residence time, which should give particles more time to coagulate. We will investigate this in section 6.3.

6.2.4 Numerical model validation

The measurements of the velocity field at the center plane serve as validation data for the fluid simulations. The objective is to verify that the main features of the flow are reproduced on a mesh that is small enough to obtain a solution in a reasonable time frame. Unlike the channel flow simulations, we are not after a completely accurate description of the flow, since —as will be clear from the particle experiments— we expect Brownian coagulation to dominate, thus limiting the effect of the flow on the particle size distribution. We also limit the simulation to fan setting 10, since at higher settings no coagulation could be observed at all.

Table 6.3 lists the properties of the different meshes that we used for the simulations. A fully unstructured tetrahedral mesh was generated using Gmsh[1], applying a grading in the z-direction so the mesh is finer near the

[1] `http://www.geuz.org/gmsh`

floor and the ceiling of the box. The listed cell sizes are the minimum and maximum imposed by the grading function, but due to the unstructured nature of the meshing process some deviations from these sizes are likely.

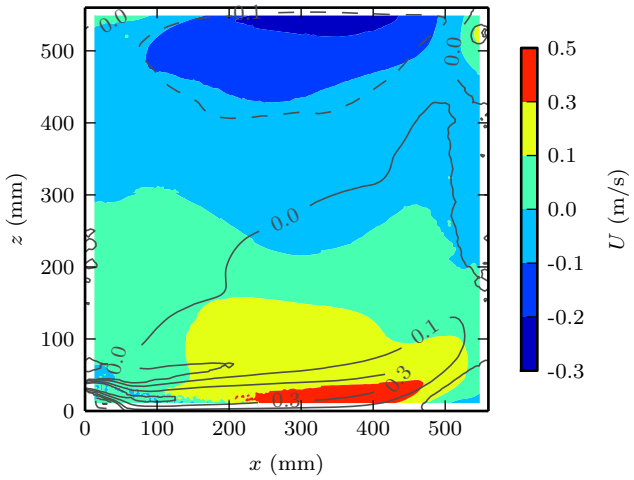


Figure 6.12: Contours of mean x-velocity. Filled contours represent the experimental values, the lines the numerical simulation.

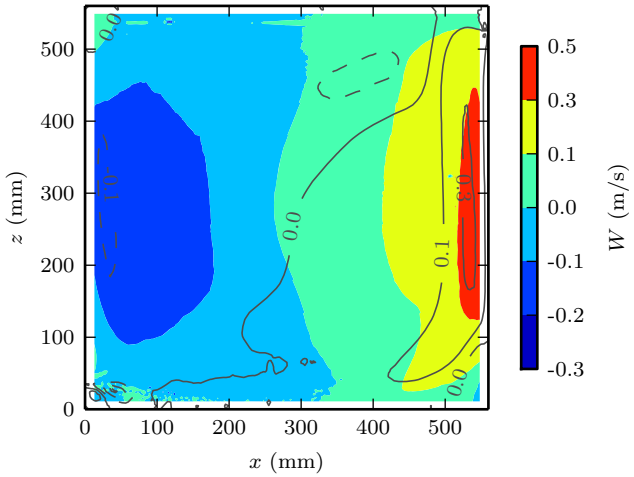


Figure 6.13: Contours of mean z-velocity. Filled contours represent the experimental values, the lines the numerical simulation.

Figures 6.12 and 6.13 present contours of the velocity components on the

coarse mesh using the implicit Euler method. Overall, the main features of the flow are captured. A more detailed comparison is possible using the

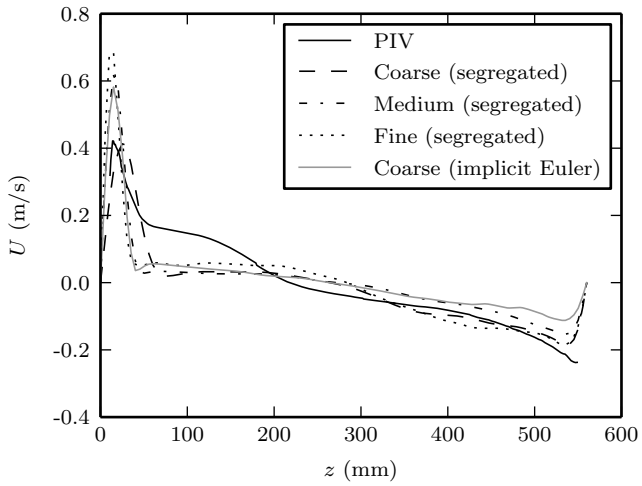


Figure 6.14: Profiles for the mean x-velocity along the vertical centerline (x = 280 mm)

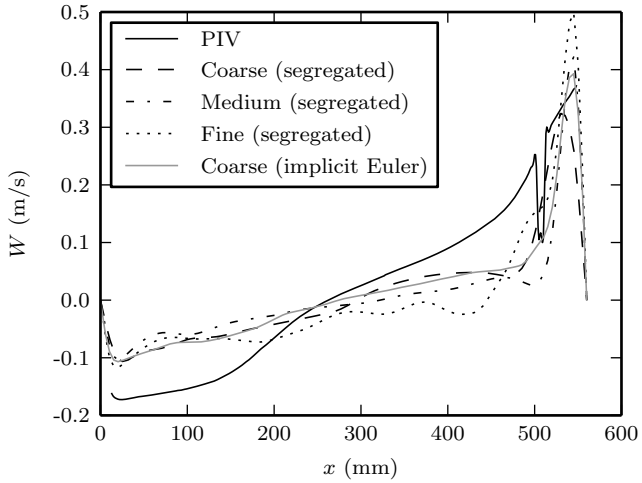


Figure 6.15: Profiles for the mean y-velocity along the horizontal centerline (z = 280 mm)

centerline profiles of figures 6.14 and 6.15. Although the overall shape of the profile is reproduced, the value of the velocities deviate significantly from

the measurements. Moreover, refining the grid does not show improvement, leading even to larger errors for the W-profile. When using a coarse DNS, it is not guaranteed that the error decreases monotonically [122], so the present results suggest that even finer meshes are needed to improve the results.

To use the computing resources more efficiently, it is useful to compare the coupled and the segregated approach. Due to the explicit treatment of the advection term, the segregated solver imposes a severe restriction, especially since the mesh near the inlet is fine and the velocity is high. Table 6.3 lists the time steps that could be used. The Crank-Nicolson scheme also becomes unstable in this case if the time steps are larger than the values in the table. Per time step, the segregated method was between 5 and 8 times faster than the coupled methods, but since we could use 20 times longer time steps using the implicit Euler method it was the fastest method overall. Using 64 CPUs, the solution on finest mesh took 30 s per time step using the segregated solver, so it was impractical to run this case using the available computing power.

6.3 Particle behavior

6.3.1 PDA measurements

In a first test, the ability to observe particle coagulation in the box was checked using the Dantec PDA system available at the RMA. The objective of the experiment is to compare the size distributions of particles directly from the Laskin nozzle with particles that went through the box.

The measurement of the output of the Laskin nozzle is shown in figure 6.16. The distributions overlap, confirming repeatability of a measurement of the same size distribution.

Particles were injected into the fan inlet and measured at the box outlet. To perform the measurement, we switched on the fan and particle generator, injecting particles through the fan. We then waited until the system reached a steady state, i.e. when there was no longer a visual change in concentration in the box. At this point, the particle generator was switched off and PDA measurements were taken for 400 s, resulting in a signal that indicates the particle size vs. time in the measurement volume. Figure 6.17 shows the result for fan setting 10. As the measurement time increases, no new particles had the opportunity to enter the box, since the particle injection is stopped at the start of the measurement. We would then expect that the measured distributions contain larger and larger particles as time passes. The measured distributions show a decrease of the number of small particles and an increase of the large ones, indicating that this is indeed the case.

We must note that the accuracy of the PDA system for particles this small is not guaranteed, but the difference with the output from the Laskin nozzle indicates that the effect observed in figure 6.17 is significant.

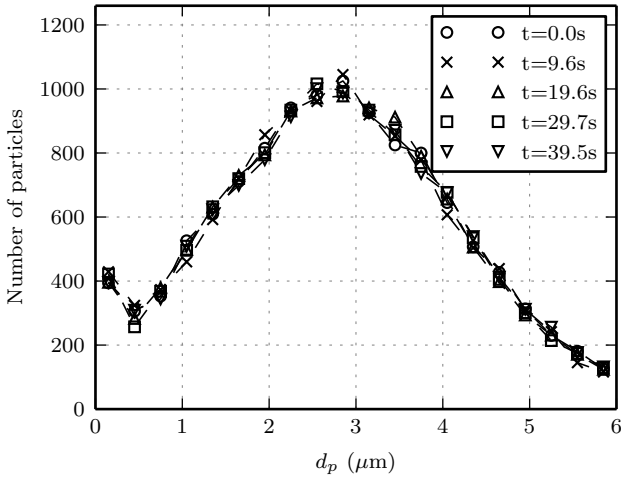


Figure 6.16: Same as figure 6.17, but measured directly at the Laskin nozzle outlet.

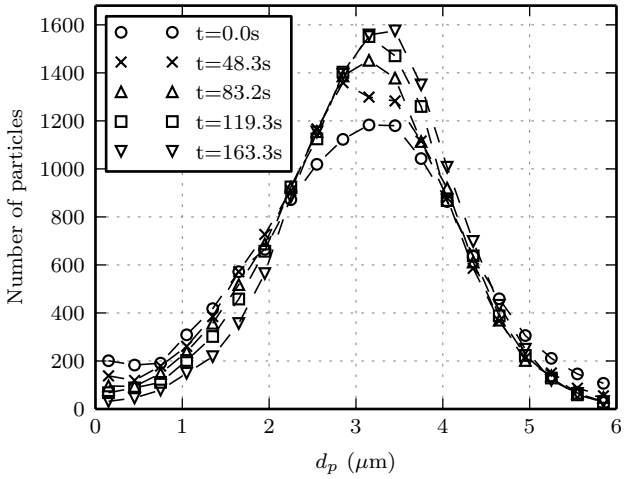


Figure 6.17: Time evolution of the particle size histogram. Times are distributed so that each curve corresponds to a sample of 12000 particles passing through the measurement volume. Measurement for fan setting 10.

6.3.2 Multi-Wavelength Light Extinction

The setup and processing system for the Multi-Wavelength Light Extinction (MWLE) technique is developed by Horváth et al. [65] at the VKI. The working principle of the technique is illustrated in figure 6.18. The incident light beam is scattered and absorbed by the particles, resulting in an attenuated light beam. The spectrum of the attenuated light is then compared to the spectrum in the absence of particles, from which the transmittance spectrum of the particle cloud is obtained. This can then be used to derive the size distribution using an inverse method. The obtained result represents an average along the entire intersection between the beam and the particle cloud. When using a combination of UV and halogen light sources it is possible to measure particles from about 20 nm to 2000 nm. This makes the technique suitable to measure particles much smaller than what is possible using PDA. The output of the processing algorithm is a distribution function for the particle volume fraction as a function of particle size, i.e. it has units [1/m] and integrating a segment of this function yields the volume fraction of particles with sizes between the integration boundaries.

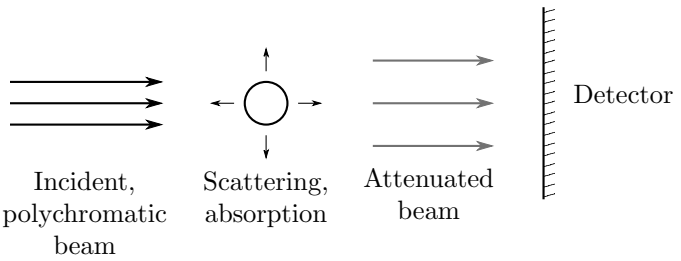


Figure 6.18: Mult-wavelength light extinction working principle.

Calibrated sphere measurements

Measurements of calibrated polystyrene particles were performed, dispersing one particle size for each measurement to obtain a monodisperse flow. The objective of these tests is to verify the accuracy of the technique with respect to the predicted particle sizes. Figure 6.19 summarizes the result, with the expected mean diameter indicated by the vertical dotted lines. The error is less than 5 % on the measured particle size peak value.

Measurements without flow

The purpose of the first experiment was to check the possibility of observing coagulation with the available particle generator and measurement system. To this end, we sealed the box as much as possible and filled it with particles during a set time interval. After the filling phase, we measured the size

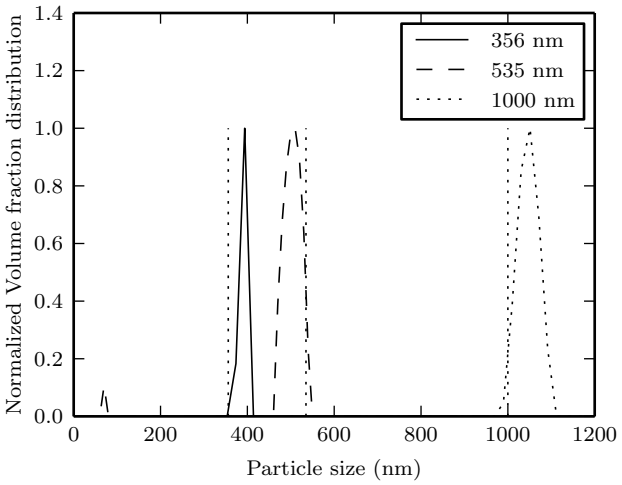


Figure 6.19: Size distributions for three sizes of calibrated polystyrene particles.

distributions at different time intervals. The MWLE technique operates along a line, so we took measurements over the entire span of the outlet. Measuring just outside the box circumvents the UV-blocking effect of the plexi-glass.

The aerosol generator used is a Topas ATM 220, capable of generating number concentrations in excess of 10^8 particles / cm^3. It can generate both solid particles and droplets. We first used solid KCl particles. In all tests, particles were injected for a 5 minute period. The fan was then activated after a certain settling time, and the distributions were measured immediately as the particles started leaving the box. This procedure leaves the settling period as the main opportunity for particles to coagulate. By activating the fan, fresh air is introduced, so the flow is diluted. This means that during the evacuation phase, coagulation is less likely to occur, also because this phase is much shorter than the settling time. Figure 6.20 shows the resulting size distribution for settling times of 0, 10 and 20 minutes. Particle coagulation would be expressed by a shift to the right in the size distribution, and we do indeed observe a very small shift here. The result is hardly convincing, however.

Using KCl and the available aerosol generator, it was not possible to get a denser particle cloud to further enhance the coagulation, so in a second attempt DEHS oil droplets were injected into the box. The particle generator operates on pressurized air, and the amount of droplets can be controlled by adjusting the inlet pressure. In figures 6.21 and 6.22, the size distributions for injector pressures of 0.5 and 5 bar are shown. In both cases,

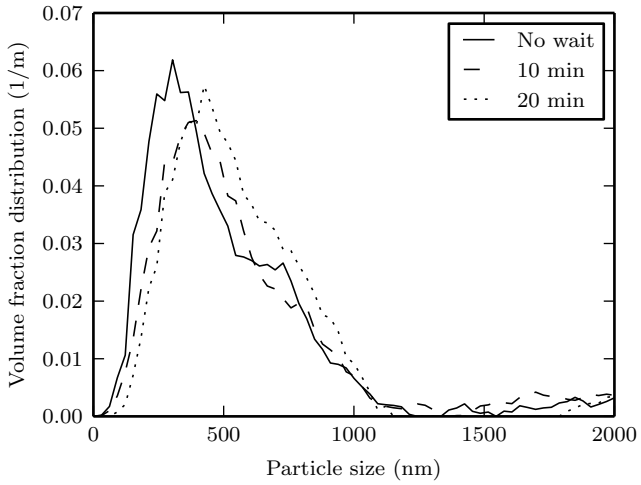


Figure 6.20: Size distributions for KCl, for three different settling times.

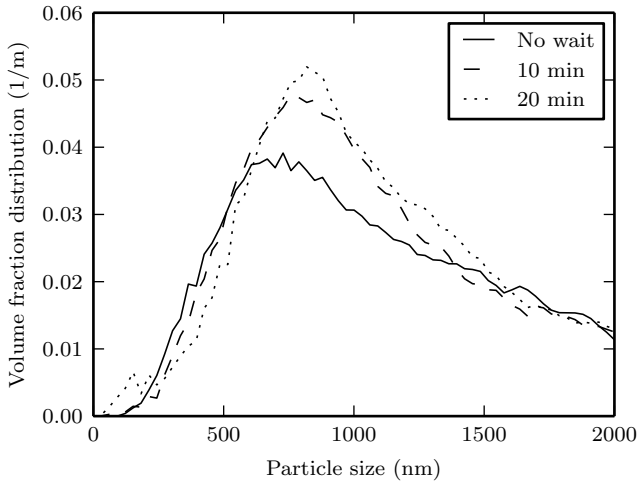


Figure 6.21: Size distribution for DEHS, with an injector pressure of 0.5 bar and measured 20 s after fan on.

an effect from coagulation is observed, but the shift to the right is much more pronounced in the 5 bar case, due to the higher particle concentration. Note also that the distribution becomes significantly narrower for the higher pressure. For the 5 bar test, we measured the size distributions at regular time intervals, starting 15 s after switching on the fan and stopping after

120 s. The limits of this time interval were dictated by the limitations of the measurement technique. If the flow is too dense (i.e. right after switching on the fan) then multiple scattering will disturb the measurement. Conversely, after 120 s the flow becomes so dilute that an accurate measurement is no longer possible. Figure 6.22 shows the volume fraction distributions for the first and last measurements. The volume fraction decreased by a factor of 10, but the shape of the size distribution remains the same, with the peak value remaining in the same location. As the concentration decreases, we expect the prediction of the absolute value of the concentration to be less accurate, and it can be seen that the relative magnitudes of the peaks differ between both measurements. The initial time of 15 s was chosen so that the transmittance of the light beam was optimal to obtain an accurate result, so we expect the earlier measurements to be the most reliable. Figure 6.23 illustrates the effect of the time between the activation of the fan and the measurement. To better show the effect, we use the particle number concentration (converted following the formulas in section 3.2.2), normalized by the maximum concentration measured at the same interval between fan activation and measurement but without the settling phase. Under this normalization, we would expect the respective curves for the 10 and 20 minute settling times to overlap (i.e. all gray curves should overlap, and all black curves as well), but as can be seen there is a discrepancy between the measurements taken after 15 and 30 s on the one hand, and those taken at 90 s and 120 s on the other hand. For further validation of the numerical model, we choose the measurements after 15 s and 30 s as the most representative of the actual situation inside the box.

Measurements with flow

A second series of measurements was taken with a flow in the box. After some experimentation, it was found that moving the air outlet to the bottom of the box while injecting particles from the top generated the longest particle residence times. The new configuration is shown in figure 6.24. Different measurement positions were used, depending on the fan setting. Positions 1 and 2 could be used at fan setting 10, measuring continuously through the box:

1. Start fan at desired speed (10 or higher)
2. Start injection and timer
3. Measure repeatedly inside the box

At fan settings 4 and 5, the flow became to dense and we had to measure in point 3:

1. Start fan at desired speed
2. Start injection and timer

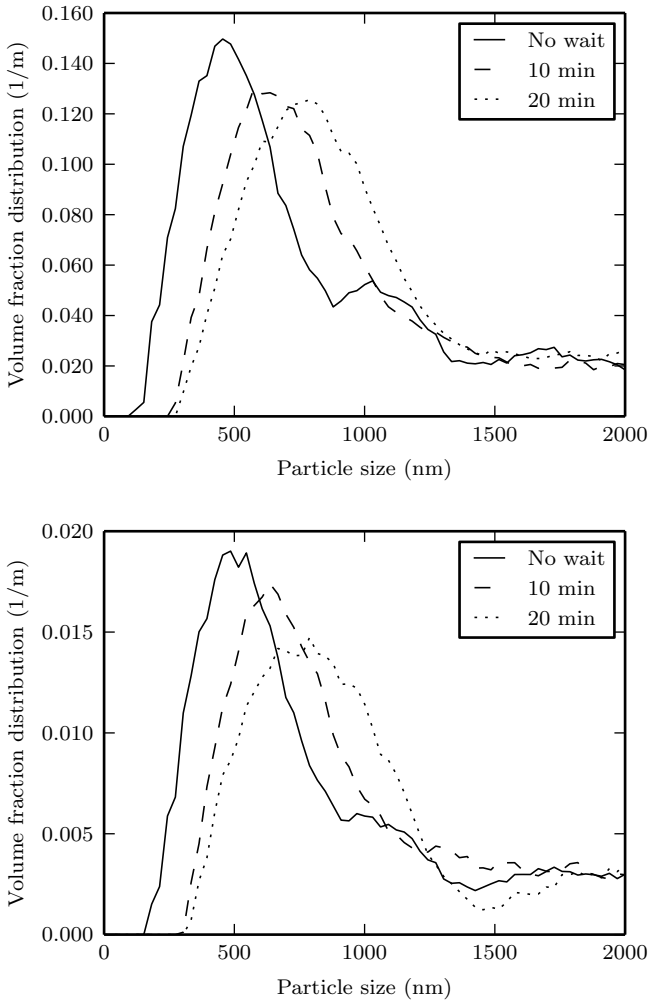


Figure 6.22: Size distribution for DEHS, with an injector pressure of 5 bar and measured 15 s (top) and 120 s (bottom) after fan on.

3. Measure initial concentration
4. Stop particle injection after X minutes
5. Measure the outflow repeatedly

For fan settings 4 and 5, the results are shown in figures 6.25 and 6.26. As before, the density of the particles prohibits measurements during and immediately after injection. At these low fan settings, best results were

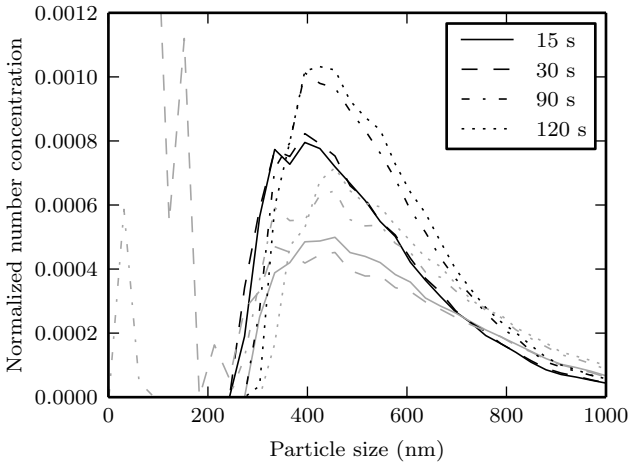


Figure 6.23: Illustration of the spread on the measurements taken at different times after the fan was switched off. The black lines: 10 minutes settling time, gray lines: 20 minutes settling time.

obtained when measuring 3 minutes after the end of the injection, so this is what is presented in the graphs. Because of the different injection durations, absolute values of the concentrations varied, so the distributions are normalized using the maximum value. At both fan settings, the effect of coagulation becomes apparent after 5 minutes of injection, but it seems to evolve to a steady state, since even the 48 minute injection at fan setting 5 does not differ much from the result after 10 minutes. This can be explained by the fact that there is a permanent flow through the system, both of fresh air and new particles for as long as the injection is active. Even though the difference in fan speed is small, there is a visible difference in the offsets after 10 minutes. We also did this test at fan setting 10, but there the effect of coagulation is no longer visible in the distributions.

Due to the lower number density inside the box at fan setting 10, it was possible to measure while the particle injection was active. We took advantage of this to measure the concentration evolution in points 1 and 2 of figure 6.24. Figure 6.27 presents the volume fraction distribution at 3 different times. The number concentration in point 2 (located in the recirculation zone under the inlet jet) increases more slowly than the concentration in point 1. This is logical, since we inject particles from the top, and the clean air coming in through the inlet slot separates point 2 from this injection.

With the current setup, it is only possible to observe coagulation at fan settings far below 10. At this low speed, it can be expected that coagulation is still dominated by the effect of Brownian motion and that the flow itself

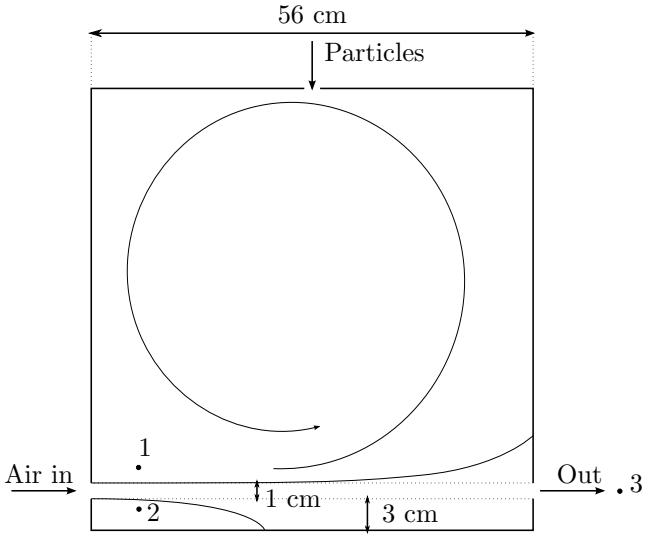


Figure 6.24: Experimental setup, after moving the inlet to the bottom to increase particle residence time. Points 1 and 2 are at 5.3 cm from the left wall. Point 1 is at 4.8 cm from the floor, point 2 at 1.0 cm.

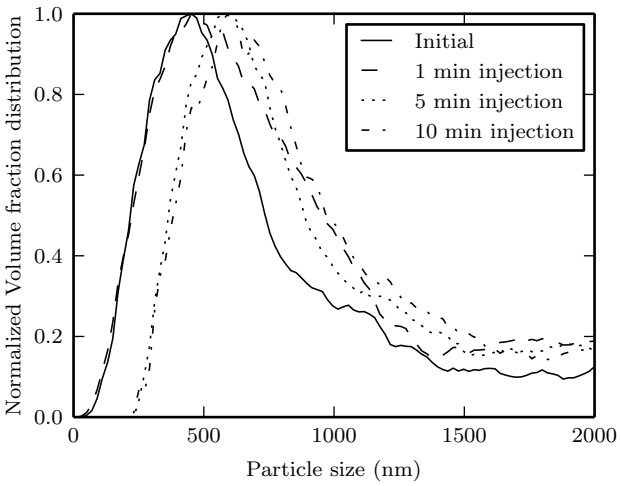


Figure 6.25: Normalized (with respect to maximum) volume fraction in point 3, for fan setting 4 and for different injection times.

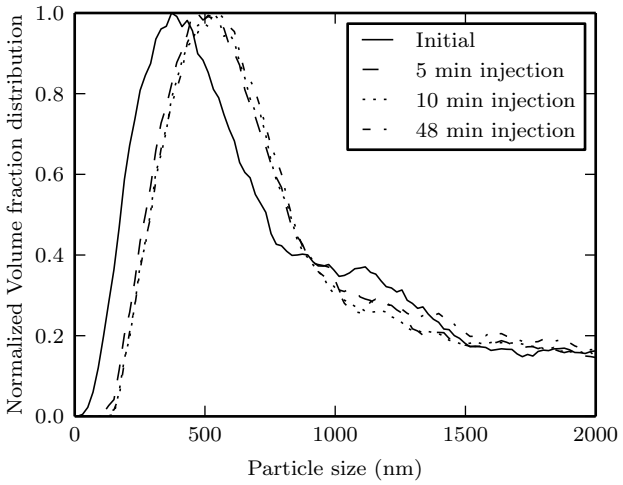


Figure 6.26: Normalized (with respect to maximum) volume fraction in point 3, for fan setting 5 and for different injection times.

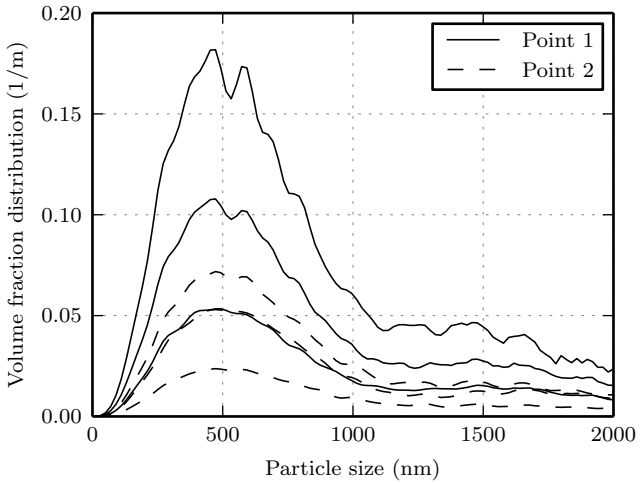


Figure 6.27: Evolution of the volume fraction distribution above and below the inlet jet (points 1 and 2), measured at times 30 s, 60 s and 120 s (lines ordered from bottom to top at each point)

has little effect. The limiting factor here is the flow rate from the aerosol generator: if the fan speed is increased too much, the aerosol is diluted down to a level so that coagulation can no longer be measured during the time

the particles spend in the box. To maintain a higher velocity while still

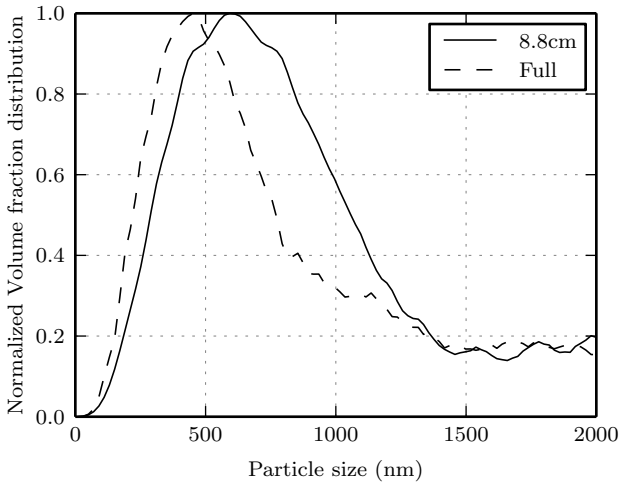


Figure 6.28: Measurement after 10 minutes in the original box compared to a measurement in the 8.8 cm box after 4 minutes of injection and 2.5 minutes of waiting before the measurement.

having a high particle concentration, we reduced the span of the box to 8.8 cm, and then repeated the measurement at fan setting 10. In figure 6.28, we compare the normalized distributions after 10 minutes and 4 minutes for both boxes. Even with the 8.8 cm span, the concentration was too high to do a measurement immediately after the injection, so we waited 2.5 minutes before measuring the size distribution. Even though the injection time was shorter in the smaller box, there is a clear shift to the right in the distribution.

Generator output

To characterize the inlet conditions for later numerical simulation, we also measured the output of the aerosol generator directly. Figure 6.29 shows the distribution function for the volume fraction for 4 different measurement series. From this, we computed a number density distribution function (figure 6.30), since this is a required input for the computation. Even though there is little difference between the measured volume fractions, the transformation to a number concentration distribution shows considerable spread, especially for the smaller particles. As in figure 6.23, this is caused by the division by d_p^3. Since this number becomes very small as $d_p \to 0$, the number concentration in figure 6.30 does not reach zero for $d_p = 0$ nm as would be expected.

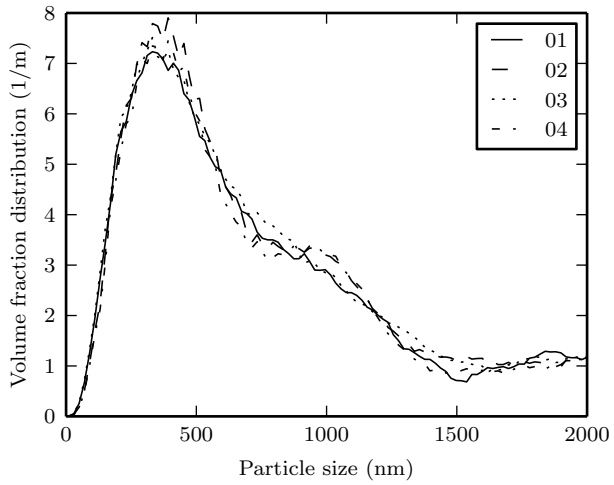


Figure 6.29: Measured volume fraction distribution at the outlet of the aerosol generator, using an input pressure of 5 bar.

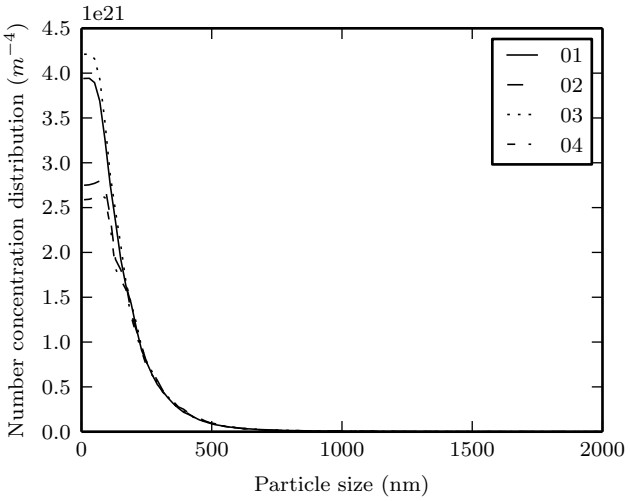


Figure 6.30: Number concentration, transformed from figure 6.29.

For the numerical algorithm, we need an estimation of the number concentration distribution, but from the above measurements it is not clear which result should be used. In an attempt to clarify this, we will fit a log-normal distribution to the measurements, and check for consistency between the volume fraction and number concentration results. Since there is little

spread on the measurements, we will only do this for the first series. Figure

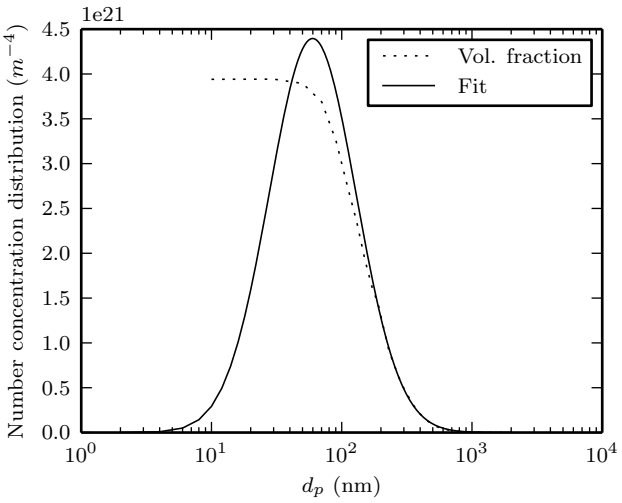


Figure 6.31: Fitted curve for the number concentration distribution, ignoring particles smaller than 200 nm.

6.31 shows the result of the fit, where we used the number concentration computed from the volume fraction (dotted line in the figure) to compute the fit. Due to the large errors on this transformation for small particle sizes, we only used particles larger than 200 nm to compute the fit. For clarity, the particle size axis is logarithmic. Figure 6.32 shows the result of transforming the fit to a volume fraction distribution, together with the corresponding measurement. The shapes are very similar, indicating that the log-normal distribution is a good approximation for the measured distributions. We conclude that the above fitting procedure is suitable to compute the approximations of the distributions as needed for the simulations.

6.3.3 Numerical model validation

The data we presented above can be used to validate the numerical model to a certain extent. Although the measurements without flow were primarily intended to investigate the possibility of observing coagulation at all, the provided data is useful to check the Brownian coagulation kernel.

Brownian coagulation

We will now apply the procedure from section 3.2.2 to model the situation in the box without flow. This should correspond to Brownian coagulation,

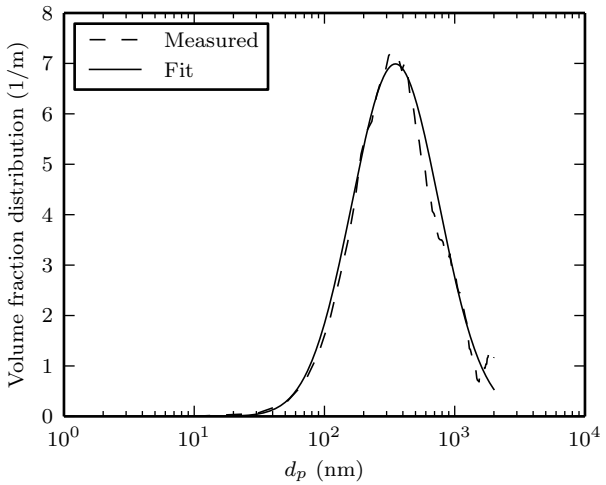


Figure 6.32: Transformation of the fit from figure 6.31 to a volume fraction distribution, compared to the corresponding measurement.

so we only need to fit a log-normal distribution to the initial concentration, and then the time evolution of the distribution should be reproduced.

As previously mentioned, the purpose of the measurements without flow was merely to determine the occurrence of coagulation. Due to the density of the particles, it was impossible to measure the concentration inside the box directly. For the numerical simulation, it is this concentration that is needed to obtain the correct coagulation rate. By scaling up the number concentration measured after 15 s with no settling (solid line in figure 6.22, top) we could obtain an initial condition from which coagulation rate that matches the experiment could be found. Figure 6.33 shows the result of the simulation together with the measurements. For reference, we also show the result using the analytical formulas from [93] as also used in section 3.2.2. To obtain these results, we scaled up the measured initial concentration by a factor of 3, assuming that this is a good approximation of the initial concentration inside the box. We then fitted a log-normal distribution with $\sigma_g = 1.59$ and $\mu_g = 281$ nm and computed the Dirac delta approximation to use the DQMOM to track the time evolution of the distribution. Since there is no flow, the computation was performed on a 1x1 2D quadrilateral mesh. The results are scaled down again by a factor of 3 to compare with the measurements. Note that for the DQMOM, we applied the Cunningham correction, which explains why here the peak value of the DQMOM is below the results using [93], i.e. exactly the opposite of what was obtained in section 3.2.2 without the Cunningham correction. Since the initial geometric standard deviation was found to be less than 2, there would be very little

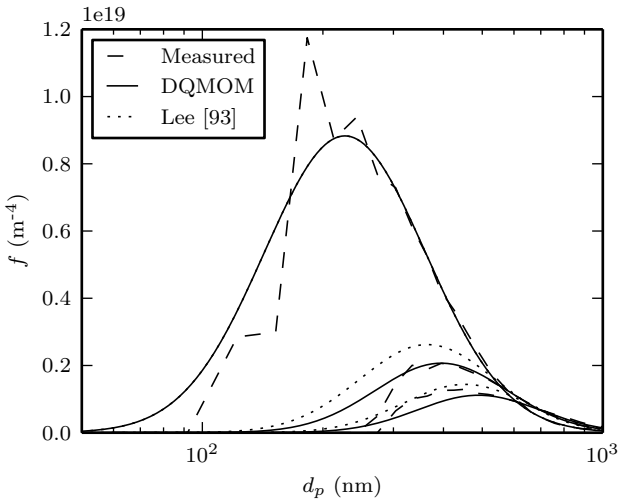


Figure 6.33: The size distributions without settling, and after 10 and 20 minutes (from top to bottom).

difference between the DQMOM and Lee's formulas if we neglected the Cunningham correction (see figure 3.10). For these small particles, however, the Cunningham correction must be applied.

The model reproduces the measured distributions fairly well, assuming that the fit for the initial concentration is indeed correct. This then also confirms that what we observed was indeed mainly Brownian coagulation. In would be useful to do a more quantitative experiment, where the concentration inside the coagulation container could be measured directly, so no guesses regarding the initial conditions would be needed.

Part III

Conclusion

Chapter 7

Conclusion and further work

7.1 Conclusions

The objective of this work was to develop a framework for the numerical study of dispersed particle-laden flow. In addition to this, we investigated appropriate methods for model validation. We mainly targeted applications in the context of air pollution, where the particle mass loading is low, but the number concentrations can be high and particle coagulation may have an effect on the particle size distribution. The focus was on particles smaller than 10 µm.

A study of available literature pointed out that in this context, Eulerian models are most suitable. The main restriction of these models is that the particle relaxation time must not be too high, which is indeed the case for the small particles under consideration.

In recent years, techniques have been developed to allow using the finite element method in flow problems, using equal-order interpolation for the velocity and the pressure. Due to the solid mathematical foundation and extensibility to higher order interpolations, this is an attractive method to use. It is also applicable to a wide variety of problems, including the Eulerian particle transport model.

Since the application of the finite element method to a given problem always follows the same scheme, we decided to first make it as easy as possible to write a finite element solver. To this end, we developed an Embedded Domain Specific Language. It allows model developers to implement a finite element model by writing code that closely resembles the weak form of the mathematical equations. Parts of the element matrix can be specified directly if needed. A unique feature is the possibility for the user to extend the language with his own functions. We also believe it is the first language for the finite element method to use the Boost Proto library, which simplifies the development of this kind of embedded language. This work is implemented in the open source Coolfluid 3 project [7], ensuring future availability of the code and encouraging collaboration.

As a first step in the development of the fluid model, the existing Coolfuid 2 solver for triangles and tetrahedra [49] was implemented using our embedded domain specific language. This enabled the solver for quadrilateral, hexahedral and prism elements. Adding additional element types requires

implementing the shape functions and related operations in the Coolfluid 3 framework, but no modification of the solver code is necessary.

Since the current work focuses on DNS to obtain an instantaneous fluid velocity for the particle model, we then explored methods to accelerate the solution of the flow equations when small time steps need to be imposed. The result was an adaptation of the predictor-multicorrector algorithm [46] to the SUPG/PSPG/Grad-Div stabilized equations, resulting in a considerable gain in efficiency in the context of DNS.

Given the small particle relaxation times, an Eulerian method was chosen. Starting from the particle equation of motion, it is possible to extrapolate the particle velocity from the fluid, eliminating the need to solve an extra set of momentum transport equations. In this formulation, it is possible to increase the accuracy using a locally implicit method [18]. Even though the additional cost of this method is small, we have shown on the Taylor-Green vortex case that it leads to an improved prediction of the particle concentration. The locally implicit model was also formulated so that it can be applied to both bubbles and solid particles.

To quantify the effect of particle coagulation, we implemented the recent Direct Quadrature Method of Moments (DQMOM) [39] to solve the population balance equations. This method tracks the weights and abscissa of a Dirac delta function approximation of the particle size distribution. Both the weights and abscissa are transported using advection-diffusion equations, so each time and position is associated with the appropriate Dirac delta approximation of the size distribution. This means that only a limited number (typically 3 or less) of delta functions are needed to get an accurate description of the moments of the size distribution.

So far, we found no previous use of the DQMOM combined with a direct numerical simulation of the flow. To make this possible, we derived a collision kernel (also previously derived in [89]) that makes no assumptions regarding the statistics of the fluid velocity. The resulting integral needs to be solved numerically in general, but we also derived an analytical expression that is valid in 2D and certain other flow configurations. Furthermore, a geometrical interpretation was given to the equation for the collision kernel, using the eigenvalues of the rate of strain tensor for the particle velocity.

Finally, we explored some possibilities for experimental validation. The basis for the experiments was a cube with a rectangular inlet and outlet, resulting in a simple flow configuration for numerical modeling. To quantify the flow, 2D PIV measurements were performed and it was found that a laminar velocity profile could be imposed at the inlet. We also discovered that the flow was too complex for direct numerical simulation using the available computing resources, resulting in a poor agreement between the computed and measured velocity fields, although the main features of the flow were recovered.

The main experimental challenge was the detection of coagulation in a complex flow, using non-intrusive methods. For this aspect, we were able to

use the Multiple Wavelength Light Extinction technique, newly developed at the von Karman Institute for Fluid Dynamics. In a static test (i.e. without flow) Brownian coagulation was observed and could be reproduced using the numerical model. We did have to make an assumption regarding the initial number concentration, since it was impossible to measure this directly due to the high particle number density. The capacity to measure particle size distributions directly in the flow was also demonstrated.

7.2 Further work

An obvious and necessary extension for the flow model is turbulence modeling. We believe that using Large Eddy Simulation would be most appropriate, since it provides the most accurate instantaneous velocity field for the particle model. The challenge here is to find a turbulence model that interacts appropriately with the numerical diffusion already imposed by the stabilization terms. The models based on the Variational Multiscale Method seem like a good candidate here. The concentration transport equation and the collision kernels will also require modification when turbulence modeling is added.

The existing channel simulations can be repeated for higher Reynolds numbers to provide reference data for the development of the turbulence models. We can also leverage the strength of the finite element method, and try to solve the channel problem on a fully unstructured grid, where the grid refinement is not restricted to the wall-normal direction.

When computing the particle concentration transport, steep concentration gradients present a numerical difficulty. The solution can deteriorate even if the particle relaxation time is less than the theoretical limit of the Eulerian method. Implementation of more advanced stabilization techniques, such as the flux-corrected transport method [115] seem promising.

Another challenge is the matching of the numerical and experimental size distributions. So far, we used the log-normal approximation to match both, but a more general approach would be needed if the experimental distribution does not closely match a theoretical continuous distribution.

The applied measurement techniques can be used in additional experiments. A Taylor-Couette flow is a good candidate, since it would allow studying the particles in a closed environment at variable levels of turbulence. As the particle time scale approaches the fluid time scale, effects of fluid turbulence levels on the size distribution evolution may be observed and used as validation data for simulations.

Until now, we considered that each collision leads to coagulation. More detailed experiments could be conducted with different kinds of particles, to observe changes in the coagulation rate with respect to the particle properties and provide data to refine existing models.

7.3 Future developments

7.3.1 Language development

Further improvements to the embedded domain specific language presented in this work are possible. In 2011, a new C++ standard was released with changes that can make the code behind our language more concise and maintainable. At the same time, more effort should be spent on making the language more user friendly, with better error reporting as a major concern.

On the functional level, the system to mix different interpolation orders could be simplified. Some advanced features such as expression optimization and the option to choose different back-ends such as the GPU can also be explored.

Finally, the possibility of extending the language to methods closely related to the finite element method, such as the residual distribution method or the discontinuous Galerkin method can be investigated.

7.3.2 Performance improvements

In the segregated solver, the solution time of the pressure system is the dominant factor. Currently, we interface with the MUMPS direct solver through the Trilinos interface, which does not allow the use of the symmetric solver in MUMPS. A direct interface between Coolfluid 3 and MUMPS would allow us to use the symmetric solver, halving the memory use and hopefully improving performance and scalability. For larger problems, even the symmetric MUMPS solver will not suffice. For this situation, more efficient iterative methods are needed.

Large problems require a parallel solution of the problem. For now, we use the classic MPI parallel communication layer, combined with automated grid partitioning. Modern alternatives to MPI, such as Charm++[1] or HPX[2], may be useful to obtain better scaling.

7.3.3 Numerical improvements

So far, we limited the work to first order elements of the Lagrange family. An extension to higher order elements may lead to more accurate results, but this needs a lot of further research. The stabilization terms may need modification, so we expect that it is not simply a matter of adding new element types and shape functions.

The current time stepping algorithm is fairly primitive. Methods such as the generalized-α method used in [50] should be expored, again with the objective to achieve a higher accuracy and possibly allow larger time steps.

[1] `http://charm.cs.uiuc.edu/research/charm`
[2] `http://stellar.cct.lsu.edu`

7.3.4 Extending the applicability

The current solver is limited to incompressible, isothermal flow. Other flow regimes should be considered. As a first step, variable density flow using the Boussinesq approximation should be fairly straightforward. After that, inclusion of the full energy equation and later an extension to compressible flow would be useful.

The particle model currently covers only a small part of the full spectrum of dispersed flow. A natural extension would be the inclusion of two-way coupling, i.e. to consider the effect of particles on the flow field. Broadening of the applicable size range is also a possible research goal, where first the particle velocity could be included in the phase space. For the largest particles, the only option would be to add Lagrangian particle tracking.

7.3.5 Potential applications

As it stands, the model is limited to low Reynolds number flows, due to the lack of turbulence modeling. The current model should be seen as a basis for further model development. Since it relies on DNS data, particle behavior is not influenced by uncertainties inherent in turbulence modeling, and simulation data from our model can be used as reference data for turbulence model development.

In incompressible flow, problems of air pollution would be a typical application. As the model is extended to include more flow regimes, applications inside turbo machinery, for example to study the effect of volcanic ash, could also be targeted. Finally, soot formation in combustion should also be a possible application, since the DQMOM can easily be extended to include several particle growth mechanisms.

Chapter 8

Bibliography

[1] C. A. Pope and D. W. Dockery. "Health Effects of Fine Particulate Air Pollution: Lines that Connect". In: *J. Air & Waste Manage. Assoc.* 56 (2006), pp. 709–742.

[2] K. Donaldson, V. Stone, A. Seaton, and W. MacNee. "Ambient particle inhalation and the cardiovascular system: potential mechanisms." In: *Environ. Health Perspect.* 109 Suppl 4 (2001), pp. 523–7.

[3] K. Donaldson, A. Seaton, et al. "A short history of the toxicology of inhaled particles". In: *Part Fibre Toxicol* 9.1 (2012), p. 13.

[4] N. Riemer, M. West, R. A. Zaveri, and R. C. Easter. "Simulating the evolution of soot mixing state with a particle-resolved aerosol model". In: *Journal of Geophysical Research: Atmospheres (1984–2012)* 114.D9 (2009).

[5] N. Bukowiecki, P. Zieger, E. Weingartner, Z. Jurányi, M. Gysel, B. Neininger, B. Schneider, C. Hueglin, A. Ulrich, A. Wichser, et al. "Ground-based and airborne in-situ measurements of the Eyjafjallajökull volcanic aerosol plume in Switzerland in spring 2010". In: *Atmospheric Chemistry and Physics* 11.19 (2011), pp. 10011–10030.

[6] V. P. Reshetin and J. L. Regens. "Simulation modeling of anthrax spore dispersion in a bioterrorism incident". In: *Risk Analysis* 23.6 (2003), pp. 1135–1145.

[7] T. Quintino, W. Deconinck, B. Janssens, T. Bányai, Q. Gasper, et al. *Coolfluid 3*. http://coolfluid.github.com. 2012.

[8] C. Clayton, S. Martin, and T. Yutaka. *Multiphase Flows with Droplets and Particles*. CRC, 1997.

[9] A. Renoux and D. Boulaud. *Les aérosols, physique et métrologie*. Ed. by. Lavoisier. 1998.

[10] J. G. M. Kuerten. "Subgrid modeling in particle-laden channel flow". In: *Physics of Fluids* 18.2, 025108 (2006), p. 025108.

[11] R. W. Johnson. *Handbook of fluid dynamics*. Crc Press, 1998.

[12] M. Yu, J. Lin, H. Jin, and Y. Jiang. "The verification of the Taylor-expansion moment method for the nanoparticle coagulation in the entire size regime due to Brownian motion". In: *Journal of Nanoparticle Research* 13.5 (2011), pp. 2007–2020.

[13] Z. Li and H. Wang. "Drag force, diffusion coefficient, and electric mobility of small particles. I. Theory applicable to the free-molecule regime". In: *Phys. Rev. E* 68.6 (2003), p. 061206.

[14] G. W. M. S. R. K. Jung Hyeun Kim and D. Y. H. Pui. "Slip Correction Measurements of Certified PSL Nanoparticles Using a Nanometer Differential Mobility Analyzer (Nano-DMA) for Knudsen Number From 0.5 to 83". In: *Journal of Research of the National Institute of Standards and Technology* 110.1 (2005).

[15] H. Jung, G. W. Mulholland, D. Y. Pui, and J. H. Kim. "Re-evaluation of the slip correction parameter of certified PSL spheres using a nanometer differential mobility analyzer (NDMA)". In: *Journal of Aerosol Science* 51 (2012), pp. 24–34.

[16] S. Balachandar and J. K. Eaton. "Turbulent dispersed multiphase flow". In: *Annual Review of Fluid Mechanics* 42 (2010), pp. 111–133.

[17] J. Ferry and S. Balachandar. "A fast Eulerian method for disperse two-phase flow". In: *International Journal of Multiphase Flow* 27.7 (2001), pp. 1199–1226.

[18] J. Ferry, S. L. Rani, and S. Balachandar. "A locally implicit improvement of the equilibrium Eulerian method". In: *International journal of multiphase flow* 29.6 (2003), pp. 869–891.

[19] R. O. Fox, F. Laurent, and M. Massot. "Numerical simulation of spray coalescence in an Eulerian framework: direct quadrature method of moments and multi-fluid method". In: *Journal of Computational Physics* 227.6 (2008), pp. 3058–3088.

[20] S. Elghobashi and G. C. Truesdell. "Direct simulation of particle dispersion in a decaying isotropic turbulence". In: *Journal of Fluid Mechanics Digital Archive* 242.-1 (1992), pp. 655–700.

[21] C. Béghein, Y. Jiang, and Q. Y. Chen. "Using large eddy simulation to study particle motions in a room". In: *Indoor Air* 15.4 (2005), pp. 281–290.

[22] M. Boivin, O. Simonin, and K. D. Squires. "On the prediction of gas–solid flows with two-way coupling using large eddy simulation". In: *Physics of Fluids* 12.8 (2000), p. 2080.

[23] B. Vreman, B. J. Geurts, N. Deen, J. Kuipers, and J. Kuerten. "Two- and four-way coupled Euler–Lagrangian large-eddy simulation of turbulent particle-laden channel flow". In: *Flow, Turbulence and Combustion* 82.1 (2009), pp. 47–71.

[24] O. Simonin. "Statistical and continuum modelling of turbulent reactive particulate flows". In: *Theoretical and experimental modelling of particulate flow*. von Karman Institute Lecture Series. 2000.

[25] P. Nerisson. "Modélisation du transfert des aérosols dans un local ventilé". PhD thesis. Université de Toulouse, 2009.

[26] L. Zaichik, N. Drobyshevsky, A. Filippov, R. Mukin, and V. Strizhov. "A diffusion-inertia model for predicting dispersion and deposition of low-inertia particles in turbulent flows". In: *International Journal of Heat and Mass Transfer* 53.1 (2010), pp. 154–162.

[27] M. R. Maxey and J. J. Riley. "Equation of motion for a small rigid sphere in a nonuniform flow". In: *Physics of fluids* 26 (1983), p. 883.

[28] L. S. Hughes, G. R. Cass, J. Gone, M. Ames, and I. Olmez. "Physical and Chemical Characterization of Atmospheric Ultrafine Particles in the Los Angeles Area". In: *Environmental Science & Technology* 32.9 (1998), p. 1153.

[29] W. W. Nazaroff. "Indoor particle dynamics". In: *Indoor Air* 14.s7 (2004), pp. 175–183.

[30] D. Ramkrishna. *Population balances: Theory and applications to particulate systems in engineering*. Academic press, 2000.

[31] H. Zhao, C. Zheng, and M. Xu. "Multi-Monte Carlo approach for general dynamic equation considering simultaneous particle coagulation and breakage". In: *Powder Technology* 154.2-3 (2005), pp. 164–178.

[32] H. Zhao, A. Maisels, T. Matsoukas, and C. Zheng. "Analysis of four Monte Carlo methods for the solution of population balances in dispersed systems". In: *Powder Technology* 173.1 (2007), p. 38.

[33] H. Zhao and C. Zheng. "Correcting the multi-Monte Carlo method for particle coagulation". In: *Powder Technology* 193.1 (2009), p. 120.

[34] M. Bini and W. Jones. "Large-eddy simulation of particle-laden turbulent flows". In: *Journal of Fluid Mechanics* 614 (2008), pp. 207–252.

[35] S. Garrick, K. Lehtinen, and M. Zachariah. "Nanoparticle coagulation via a Navier–Stokes/nodal methodology: Evolution of the particle field". In: *Journal of Aerosol Science* 37.5 (2006), p. 555.

[36] V. John and C. Suciu. "Direct discretizations of bi-variate population balance systems with finite difference schemes of different order". In: *Chemical Engineering Science* 106 (2014), pp. 39–52.

[37] D. L. Marchisio, R. D. Vigil, and R. O. Fox. "Quadrature method of moments for aggregation–breakage processes". In: *Journal of colloid and interface science* 258.2 (2003), pp. 322–334.

[38] D. L. Marchisio, J. T. Pikturna, R. O. Fox, R. D. Vigil, and A. A. Barresi. "Quadrature method of moments for population-balance equations". In: *AIChE Journal* 49.5 (2003), pp. 1266–1276.

[39] D. L. Marchisio and R. O. Fox. "Solution of population balance equations using the direct quadrature method of moments". In: *Journal of Aerosol Science* 36.1 (2005), pp. 43–73.

[40] L. Zaichik, R. Mukin, L. Mukina, and V. Strizhov. "Development of a diffusion-inertia model for calculating bubble turbulent flows: Isothermal polydispersed flow in a vertical pipe". In: *High Temperature* 50.5 (2012), pp. 621–630.

[41] V. John, I. Angelov, A. Öncül, and D. Thévenin. "Techniques for the reconstruction of a distribution from a finite number of its moments". In: *Chemical Engineering Science* 62.11 (2007), pp. 2890–2904.

[42] L. De Souza, G. Janiga, V. John, and D. Thévenin. "Reconstruction of a distribution from a finite number of moments with an adaptive spline-based algorithm". In: *Chemical Engineering Science* 65.9 (2010), pp. 2741–2750.

[43] D. L. Marchisio and R. O. Fox. *Computational models for polydisperse particulate and multiphase systems*. Cambridge University Press, 2013.

[44] I. Babuška. "Error-bounds for finite element method". In: *Numer. Math.* 16.4 (1971), pp. 322–333.

[45] F. Brezzi. "On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multipliers." English. In: *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique* 8.1 (1974), pp. 129–151.

[46] A. N. Brooks and T. J. Hughes. "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations". In: *Computer methods in applied mechanics and engineering* 32.1 (1982), pp. 199–259.

[47] T. E. Tezduyar, S. Mittal, S. Ray, and R. Shih. "Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements". In: *Computer Methods in Applied Mechanics and Engineering* 95.2 (1992), pp. 221–242.

[48] M. Olshanskii and A. Reusken. "Grad-div stablilization for Stokes equations". In: *Mathematics of Computation* 73.248 (2004), pp. 1699–1718.

[49] T. Bányai, D. Vanden Abeele, and H. Deconinck. "A fast fully-coupled solution algorithm for the unsteady incompressible Navier-Stokes equations". In: *Conference on Modelling Fluid Flow (CMFF'06)*. Budapest, Hungary, 2006.

[50] A. V. Trofimova, A. E. Tejada-Martínez, K. E. Jansen, and R. T. Lahey. "Direct numerical simulation of turbulent channel flows using a stabilized finite element method". In: *Computers & Fluids* 38.4 (2009), pp. 924–938.

[51] A. E. Tejada-Martınez and K. E. Jansen. "On the interaction between dynamic model dissipation and numerical dissipation due to streamline upwind/Petrov–Galerkin stabilization". In: *Computer methods in applied mechanics and engineering* 194.9 (2005), pp. 1225–1248.

[52] M. Braack, E. Burman, V. John, and G. Lube. "Stabilized finite element methods for the generalized Oseen problem". In: *Computer methods in applied mechanics and engineering* 196.4 (2007), pp. 853–866.

[53] T. Tezduyar and S. Sathe. "Stabilization parameters in SUPG and PSPG formulations". In: *Journal of computational and applied mechanics* 4.1 (2003), pp. 71–88.

[54] T. J. Hughes. "Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods". In: *Computer methods in applied mechanics and engineering* 127.1 (1995), pp. 387–401.

[55] T. J. Hughes, L. Mazzei, and K. E. Jansen. "Large eddy simulation and the variational multiscale method". In: *Computing and Visualization in Science* 3.1-2 (2000), pp. 47–59.

[56] T. J. Hughes, A. A. Oberai, and L. Mazzei. "Large eddy simulation of turbulent channel flows by the variational multiscale method". In: *Physics of Fluids (1994-present)* 13.6 (2001), pp. 1784–1799.

[57] Y. Bazilevs, V. Calo, J. Cottrell, T. Hughes, A. Reali, and G. Scovazzi. "Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows". In: *Computer Methods in Applied Mechanics and Engineering* 197.1 (2007), pp. 173–201.

[58] M. Abadie. "Contribution à l'étude de la pollution particulaire : rôle des parois, rôle de la ventilation". PhD thesis. Université de La Rochelle, 2000.

[59] J. Bouilly. "Etude de l'impact de la pollution particulaire sur la qualité de l'air intérieur en site urbain". PhD thesis. Université de La Rochelle, 2003.

[60] T. Denes. "Etude de l'impact sur les particules fines de la ventilation sur la qualité de l'air intérieur". PhD thesis. Université de La Rochelle, 2007.

[61] J. E. Hijri. "Etude de la remise en suspension particulaire, évaluation de son impact sur la qualité de l'air intérieur". PhD thesis. FSTT de Tanger en co-tutelle avec ULR, 2008.

[62] L. Olea. "Contribution à l'étude du transfert de particules solides dans le bâtiment: Rôle des fissures et ouvertures réduites". PhD thesis. Université de La Rochelle, 2009.

[63] P. Duru, D. Koch, and C. Cohen. "Experimental study of turbulence-induced coalescence in aerosols". In: *International Journal of Multiphase Flow* 33.9 (2007), p. 987.

[64] S. Laın, M. Sommerfeld, and J. Kussin. "Experimental studies and modelling of four-way coupling in particle-laden horizontal channel flow". In: *International journal of heat and fluid flow* 23.5 (2002), pp. 647–656.

[65] I. Horvath, E. Pescini, P. Colinet, and M. R. Vetrano. "Application de la technique spectroscopique d'extinction à un écoulement de nanoparticules". In: *13ième Congrès Francophone de Techniques Laser*. 2012.

[66] I. T. Horvath, M. R. Vetrano, and P. Colinet. "Validation of the Multi-Wavelength Light Extinction technique for the characterization of solid nanoparticles". In: *European Conference on Liquid Atomization and Spray Systems (ILASS 2013)*. 2013.

[67] T. A. Zang. "On the rotation and skew-symmetric forms for incompressible flow simulations". In: *Applied Numerical Mathematics* 7.1 (1991), pp. 27–40.

[68] S. Mittal. "On the performance of high aspect ratio elements for incompressible flows". In: *Comput. Methods Appl. Mech. Engrg.* 188 (2000), pp. 269–287.

[69] T. Heister and G. Rapin. "Efficient augmented Lagrangian-type preconditioning for the Oseen problem using Grad-Div stabilization". In: *International Journal for Numerical Methods in Fluids* 71.1 (2013), pp. 118–134.

[70] Y. Ling, M. Parmar, and S. Balachandar. "A scaling analysis of added-mass and history forces and their coupling in dispersed multiphase flows". In: *International Journal of Multiphase Flow* 57 (2013), pp. 102–114.

[71] O. Druzhinin and S. Elghobashi. "Direct numerical simulations of bubble-laden turbulent flows using the two-fluid formulation". In: *Physics of Fluids* 10 (1998), p. 685.

[72] V. Armenio and V. Fiorotto. "The importance of the forces acting on particles in turbulent flows". In: *Physics of Fluids* 13.8 (2001), pp. 2437–2440.

[73] A. Daitche and T. Tél. "Memory effects are relevant for chaotic advection of inertial particles". In: *Physical Review Letters* 107.24 (2011), p. 244501.

[74] K. Guseva, U. Feudel, and T. Tél. "Influence of the history force on inertial particle advection: Gravitational effects and horizontal diffusion". In: *Physical Review E* 88.4 (2013), p. 042909.

[75] A. Daitche. "Advection of Inertial Particles in the Presence of the History Force: Higher Order Numerical Schemes". In: *Journal of Computational Physics* 254 (2013), pp. 93–106.

[76] M. Van Hinsberg, J. ten Thije Boonkkamp, and H. Clercx. "An efficient, second order method for the approximation of the Basset history force". In: *Journal of Computational Physics* 230.4 (2011), pp. 1465–1478.

[77] E. Calzavarini, R. Volk, M. Bourgoin, E. Lévêque, J.-F. Pinton, F. Toschi, et al. "Acceleration statistics of finite-sized particles in turbulent flow: the role of Faxén forces". In: *Journal of Fluid Mechanics* 630 (2009), p. 179.

[78] H. Homann, J. Bec, et al. "Finite-size effects in the dynamics of neutrally buoyant particles in turbulent flow". In: *Journal of Fluid Mechanics* 651 (2010), p. 81.

[79] V. John and P. Knobloch. "On spurious oscillations at layers diminishing (SOLD) methods for convection–diffusion equations: Part I–A review". In: *Computer Methods in Applied Mechanics and Engineering* 196.17 (2007), pp. 2197–2215.

[80] V. John and P. Knobloch. "On spurious oscillations at layers diminishing (SOLD) methods for convection–diffusion equations: Part II–Analysis for P1 and Q1 finite elements". In: *Computer Methods in Applied Mechanics and Engineering* 197.21 (2008), pp. 1997–2014.

[81] V. John, T. Mitkova, M. Roland, K. Sundmacher, L. Tobiska, and A. Voigt. "Simulations of population balance systems with one internal coordinate using finite element methods". In: *Chemical Engineering Science* 64.4 (2009), pp. 733–741.

[82] R. P. Feynman, R. B. Leighton, and M. Sands. *The Feynman Lectures on Physics, Desktop Edition Volume I*. Vol. 1. 2013.

[83] F. Laurent, M. Massot, and P. Villedieu. "Eulerian multi-fluid modeling for the numerical simulation of coalescence in polydisperse dense liquid sprays". In: *Journal of Computational Physics* 194.2 (2004), pp. 505–543.

[84] P. Saffman and J. Turner. "On the collision of drops in turbulent clouds". In: *Journal of Fluid Mechanics* 1.01 (1956), pp. 16–30.

[85] L.-P. Wang, A. S. Wexler, and Y. Zhou. "Statistical mechanical descriptions of turbulent coagulation". In: *Physics of Fluids (1994-present)* 10.10 (1998), pp. 2647–2651.

[86] Y. Zhou, A. S. Wexler, and L.-P. Wang. "On the collision rate of small particles in isotropic turbulence. II. Finite inertia case". In: *Physics of Fluids* 10.5 (1998), p. 1206.

[87] L.-P. Wang, A. S. Wexler, and Y. Zhou. "On the collision rate of small particles in isotropic turbulence. I. Zero-inertia case". In: *Physics of Fluids* 10.1 (1998), p. 266.

[88] R. Mei and K. C. Hu. "On the collision rate of small particles in turbulent flows". In: *Journal of Fluid Mechanics* 391 (1999), pp. 67–89.

[89] R. J. Hill. "Geometric collision rates and trajectories of cloud droplets falling into a Burgers vortex". In: *Physics of Fluids (1994-present)* 17.3 (2005), p. 037103.

[90] L. I. Zaichik and V. M. Alipchenkov. "The coagulation of aerosol particles in turbulent flow". In: *High Temperature* 46.5 (2008), p. 666.

[91] M. Williams. "A unified theory of aerosol coagulation". In: *Journal of Physics D: Applied Physics* 21.6 (1988), p. 875.

[92] L. I. Zaichik and V. M. Alipchenkov. "The effect of Brownian motion on collisions between aerosol particles in turbulent flow". In: *High Temperature* 46.4 (2008), p. 502.

[93] K. Lee. "Change of particle size distribution during Brownian coagulation". In: *Journal of Colloid and Interface Science* 92.2 (1983), pp. 315–325.

[94] N. Park, J. Y. Yoo, and H. Choi. "Discretization errors in large eddy simulation: on the suitability of centered and upwind-biased compact difference schemes". In: *Journal of Computational Physics* 198.2 (2004), pp. 580–616.

[95] G. Lindauer and A. Castleman Jr. "Behavior of aerosols undergoing Brownian coagulation and gravitational settling in closed systems". In: *Journal of Aerosol Science* 2.2 (1971), pp. 85–91.

[96] E. Niebler. "Proto: a compiler construction toolkit for DSELs". In: *Proceedings of the 2007 Symposium on Library-Centric Software Design - LCSD '07*. LCSD '07. Montreal, Canada: ACM, 2007, pp. 42–51.

[97] H. G. Weller. "A tensorial approach to computational continuum mechanics using object-oriented techniques". In: *Computers in Physics* 12.6 (1998), p. 620.

[98] A. Logg and G. N. Wells. "DOLFIN: Automated finite element computing." In: *ACM Trans. Math. Softw.* 37.2 (2010).

[99] M. S. Alnæs and K.-A. Mardal. "On the efficiency of symbolic computations combined with code generation for finite element methods". In: *ACM Transactions on Mathematical Software* 37.1 (2010), p. 1.

[100] C. Prud'homme. "A domain specific embedded language in C++ for automatic differentiation, projection, integration and variational formulations". In: *Scientific Programming* 14.2 (2006), pp. 81–110.

[101] O. Zienkiewicz and R. Taylor. *The Finite Element Method*. Ed. by F. Edition. Butterworth Heinemann, 2000.

[102] G. Guennebaud, B. Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[103] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.

[104] M. A. Heroux and J. M. Willenbring. *Trilinos Users Guide*. Tech. rep. SAND2003-2952. 2003.

[105] R. Johnson, R. Helm, J. Vlissides, and E. Gamma. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[106] K. Iglberger, G. Hager, J. Treibig, and U. Rüde. "Expression Templates Revisited: A Performance Analysis of Current Methodologies." In: *SIAM J. Scientific Computing* 34.2 (2012).

[107] A. J. Chorin. "Numerical solution of the Navier-Stokes equations". In: *Mathematics of computation* 22.104 (1968), pp. 745–762.

[108] R. Codina, J. Principe, O. Guasch, and S. Badia. "Time dependent subscales in the stabilized finite element approximation of incompressible flow problems". In: *Computer Methods in Applied Mechanics and Engineering* 196.21 (2007), pp. 2413–2430.

[109] J. Kim, P. Moin, and R. Moser. "Turbulence statistics in fully developed channel flow at low Reynolds number". In: *Journal of Fluid Mechanics* 177 (1987), pp. 133–166.

[110] R. D. Moser, J. Kim, and N. N. Mansour. "Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$". In: *Physics of Fluids* 11.4 (1999), pp. 943–945.

[111] U. Piomelli and P. Sagaut. "Large Eddy Simulation. Theory and Applications". In: von Karman Institute Lecture Series. 2012.

[112] Y. Morinishi and O. V. Vasilyev. "A recommended modification to the dynamic two-parameter mixed subgrid scale model for large eddy simulation of wall bounded turbulent flow". In: *Physics of Fluids* 13.11 (2001), pp. 3400–3410.

[113] H. Tennekes and J. L. Lumley. *A first course in turbulence*. MIT press, 1972.

[114] J. G. M. Kuerten and A. W. Vreman. "Can turbophoresis be predicted by large-eddy simulation?" In: *Physics of Fluids* 17.1, 011701 (2005), p. 011701.

[115] V. John and E. Schmeyer. "On finite element methods for 3d time-dependent convection-diffusion-reaction equations with small diffusion". In: *BAIL 2008-Boundary and Interior Layers*. Springer, 2009, pp. 173–181.

[116] E. Recker, W. Bosschaerts, and P. Hendrick. "Numerical and experimental study of a hydrogen gas turbine combustor using the jet in cross-flow principle". PhD thesis. Royal Military Academy, 2012.

[117] R. N. Habiyambere. "Experimentele studie van de karakteristieken van Offset jet flows". MA thesis. Royal Military Academy, 2013.

[118] P. Rajesh Kanna and M. K. Das. "Numerical simulation of two-dimensional laminar incompressible offset jet flows". In: *International Journal for Numerical Methods in Fluids* 49.4 (2005), pp. 439–464.

[119] A. Nasr and J. Lai. "A turbulent plane offset jet with small offset ratio". In: *Experiments in Fluids* 24.1 (1998), pp. 47–57.

[120] B. Lecordier and J. Westerweel. "The EUROPIV synthetic image generator (SIG)". In: *Particle image velocimetry: recent improvements*. Springer, 2004, pp. 145–161.

[121] F. Durst, S. Ray, B. Ünsal, and O. Bayoumi. "The development lengths of laminar pipe and channel flows". In: *Journal of Fluids Engineering* 127.6 (2005), pp. 1154–1160.

[122] J. Meyers and P. Sagaut. "Is plane-channel flow a friendly case for the testing of large-eddy simulation subgrid-scale models?" In: *Physics of Fluids* 19.4 (2007), pp. 48105–49900.